AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Engineering

Lab Report

Experiment # 04

Experiment Title: Study of a Digital Timer using the millis() function of Arduino to avoid problems associated with the delay() function and power measurement in different power reduction modes while the circuit is in sleep modes.

Date of Perform:	17 March 2025	**Date of Submission:**	10 April 2025
Course Title:	Microprocessor and Embedded Systems Lab		
Course Code:	EE4103	**Section:**	P
Semester:	Spring 2024-25	**Degree Program:**	BSc in CSE
Course Teacher:	**Prof. Dr. Engr. Muhibul Haque Bhuyan**		

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case Study is my/our original work; no part has been copied from any other student's work or any other source except where due acknowledgment is made.
3. No part of this Assignment/Case Study has been written for me/us by any other person except where such collaboration has been authorized. by the concerned teacher and is acknowledged in the assignment.
4. I/we have not previously submitted or am submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared, and archived to detect plagiarism.
6. I/we permit a copy of my/our marked work to be retained by the Faculty Member for review by any internal/external examiners.
7. I/we understand that Plagiarism is the presentation of the work, idea, or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offense that may lead to expulsion from the University. Plagiarized material can be drawn from, and presented in, written, graphic, and visual forms, including electronic data, and oral presentations. Plagiarism occurs when the origin of the source is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or copy my/our work.

** Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group # 01

Sl No	Name	ID	PROGRAM	SIGNATURE
1.	Md. Saikot Hossain	23-51242-1	BSc in CSE	
2	Md. Mosharof Hossain Khan	23-51259-1	BSc in CSE	
3	Rimal Banik	23-51260-1	BSc in CSE	
4	Md. Rahidul Islam	23-51269-1	BSc in CSE	
5	Rahat Ahmed	21-44911-2	BSc in CSE	

Faculty use only		
FACULTY COMMENTS	**Marks Obtained**	
	Total Marks	

Contents

Objectives	3
Apparatus	3
Circuit Diagram	3-4
Experimental Output Results	4-10
Simulation Output Results	11-15
Answer to Question	15-27
Discussion	27-28
Conclusion	28
References	28

Objectives:

The objectives of this experiment are to

a) Study the application of the millis() function, a built-in Timer of Arduino.
b) Build a digital timer to turn on several LEDs every minute sequentially on an Arduino Microcontroller Board.
c) Implement a sequential LED light pattern control system using an input switch and an Arduino Microcontroller Board.
d) Measure the amount of power and energy saved using power down or power save mode.

Apparatus:

For experiment 4(a)

1. Arduino IDE 2.3.5
2. Arduino Microcontroller board.
3. LED lights (Red, Green, and Yellow- each 2)
4. Six 100 Ω resistors and one 10 kΩ resistor
5. One tilt switch
6. Jumper wires

For experiment 4(b)

1. Arduino Board
2. Breadboard
3. Resistors
4. Switches
5. LED
6. USB Ammeter
7. Jumper Wires
8. Laptop or PC with Arduino IDE installed in it.

Circuit Diagram:

For experiment 4(a):

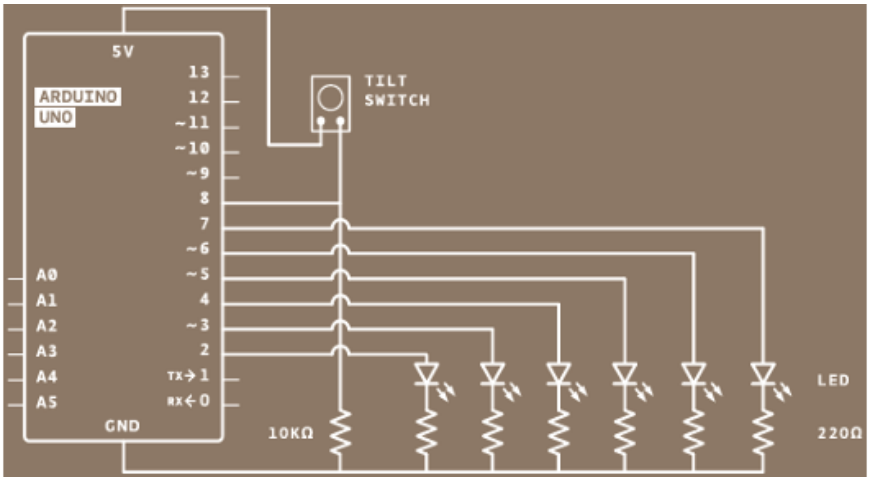
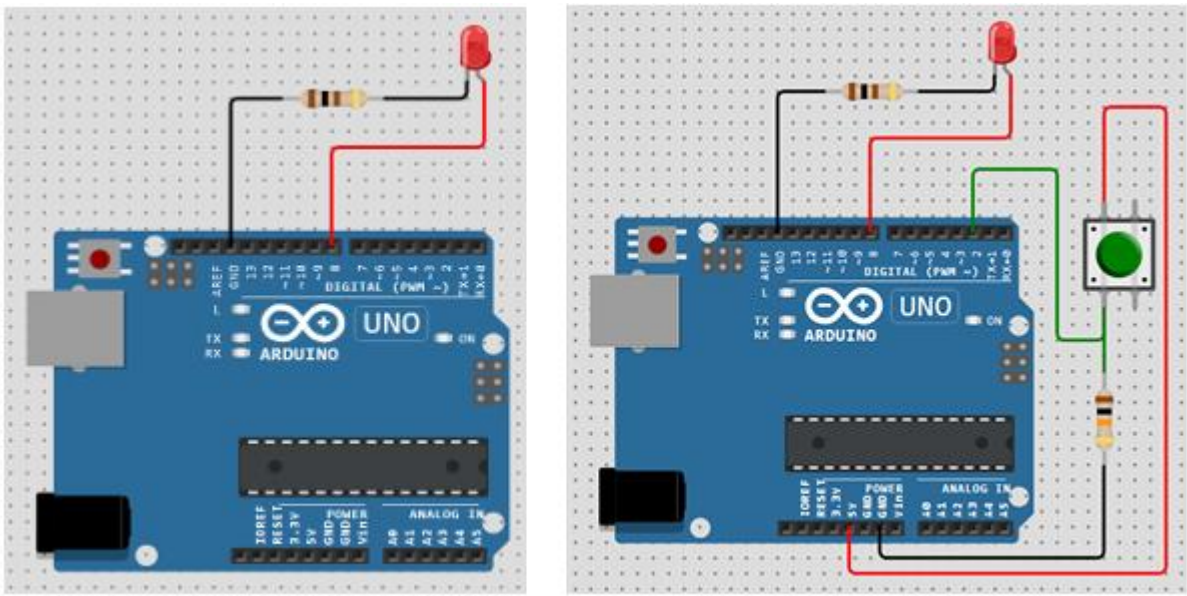


Figure 1: Experimental Setup of an LED Light Pattern Control System using a Tilt switch and an Arduino Microcontroller Board.

For experiment 4(b):



(a) LED state change without any switch; (b) LED state change with switch:

Figure 2: Power Measurement during different states of LED

Experimental Output Results:

a) **LED Light Pattern Control System using a Tilt switch and an Arduino Microcontroller Board:**

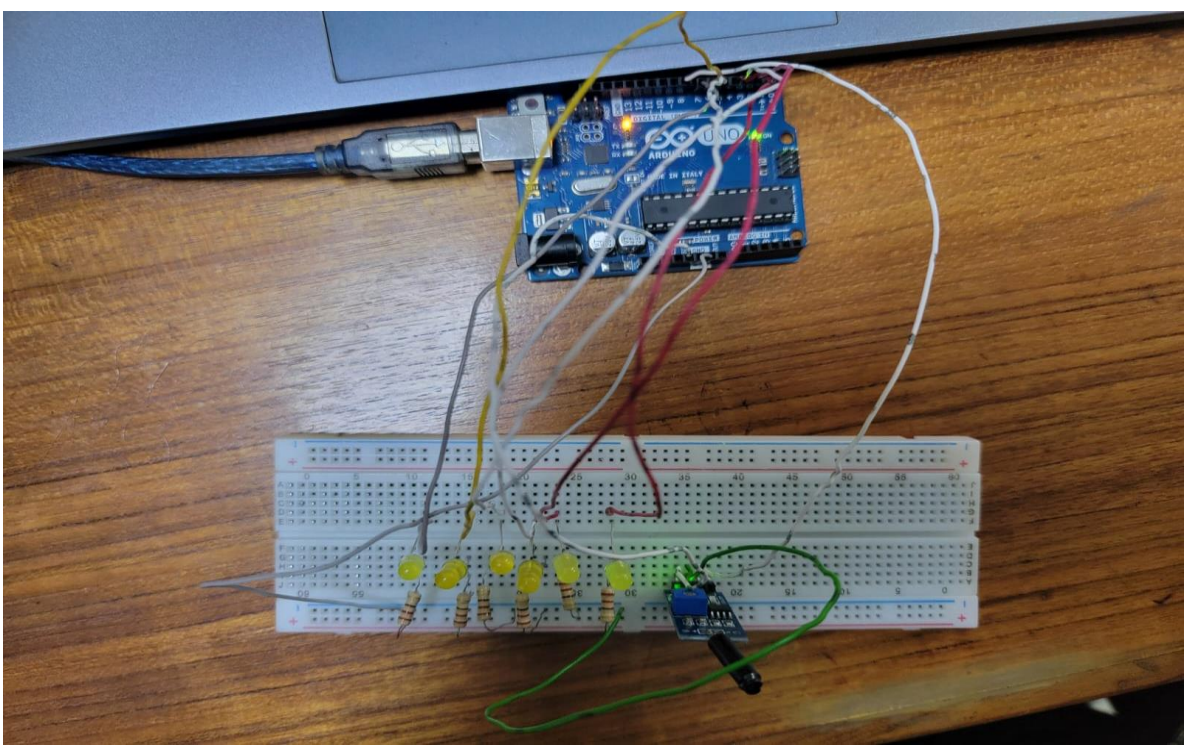


Figure 3: All Led off state

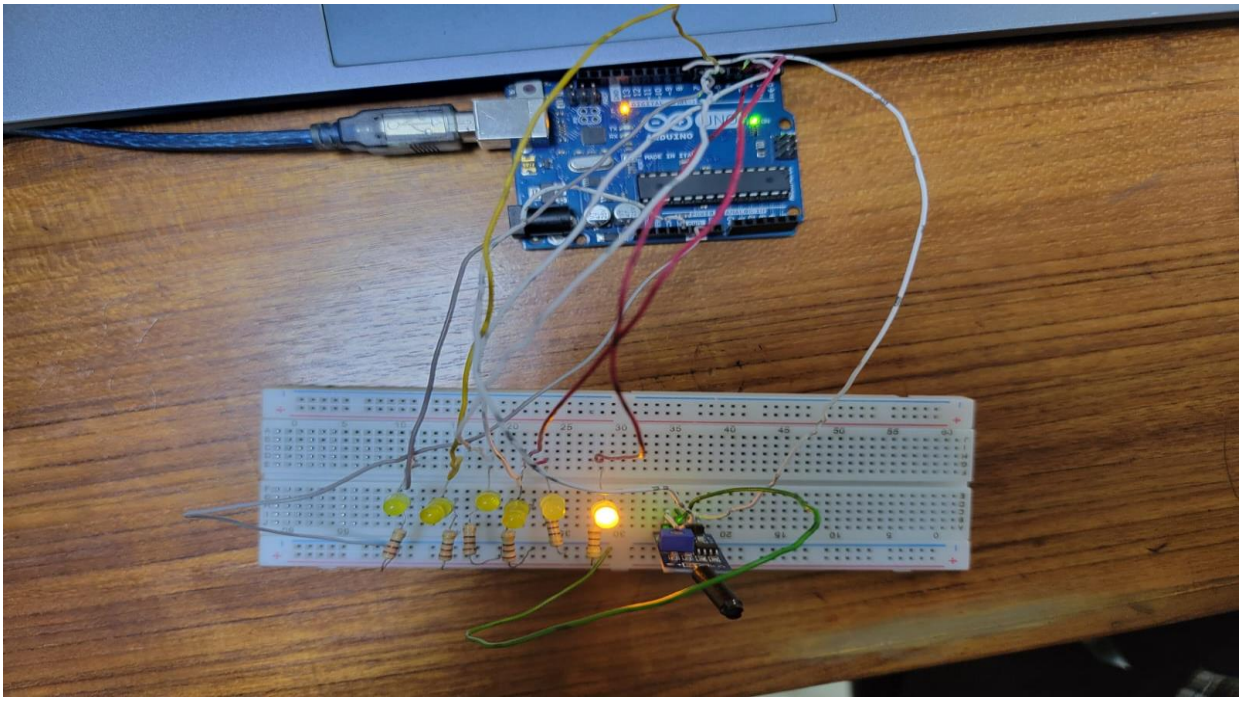


Figure 4: First one Led on state

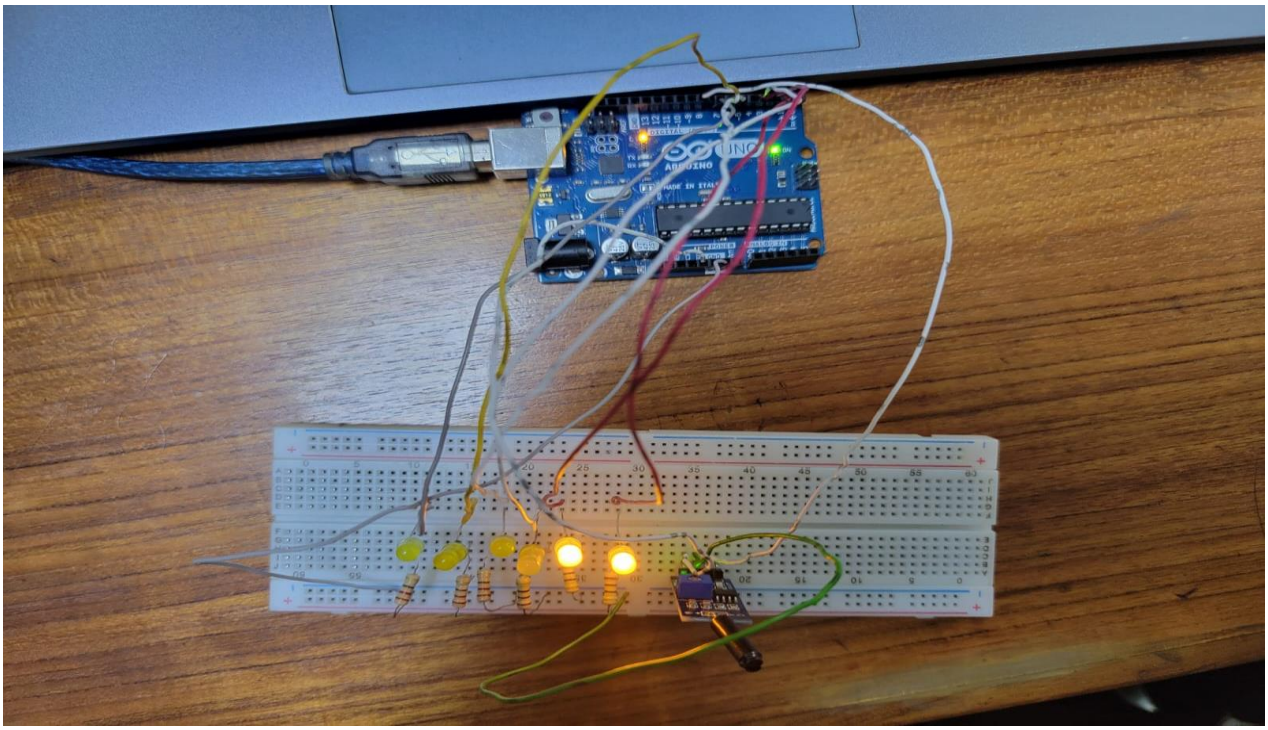


Figure 5: First two led on state

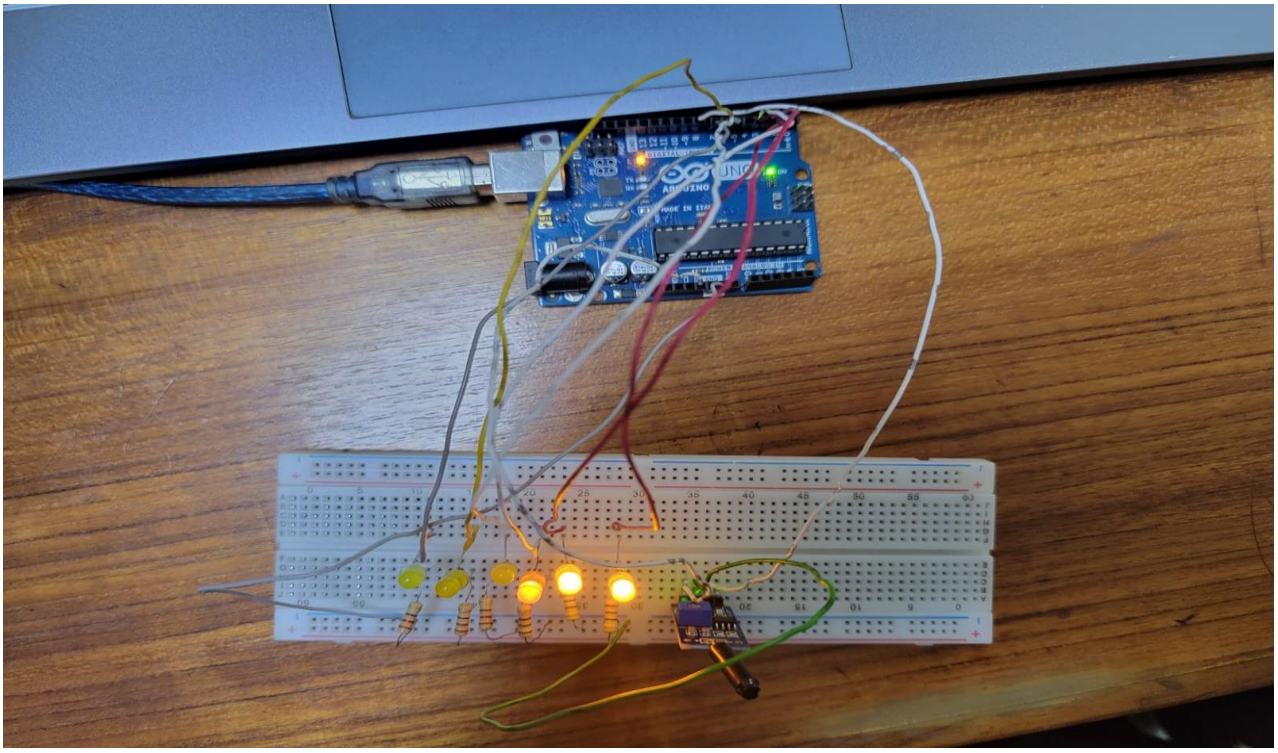


Figure 6: First three led on state

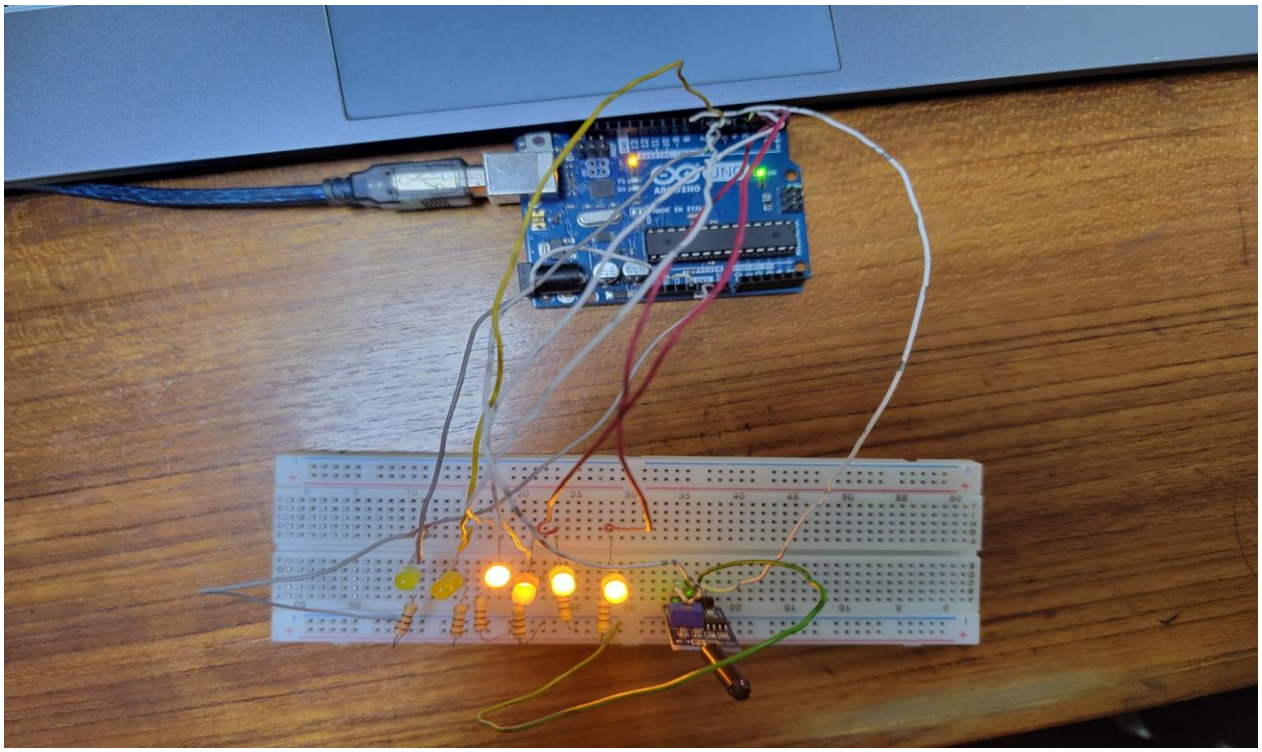


Figure 7: First four led on state

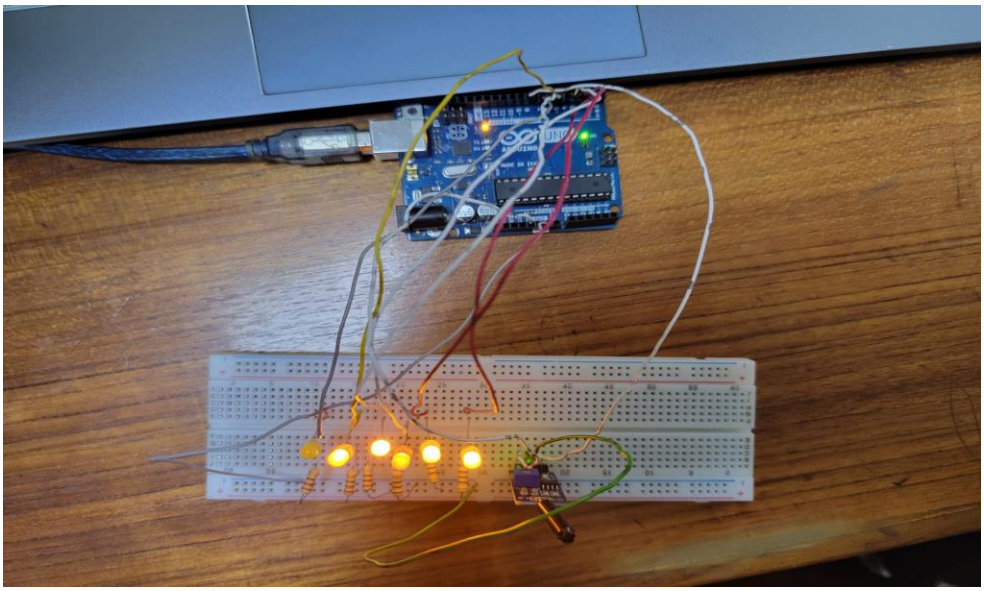


Figure 8: First five led on state

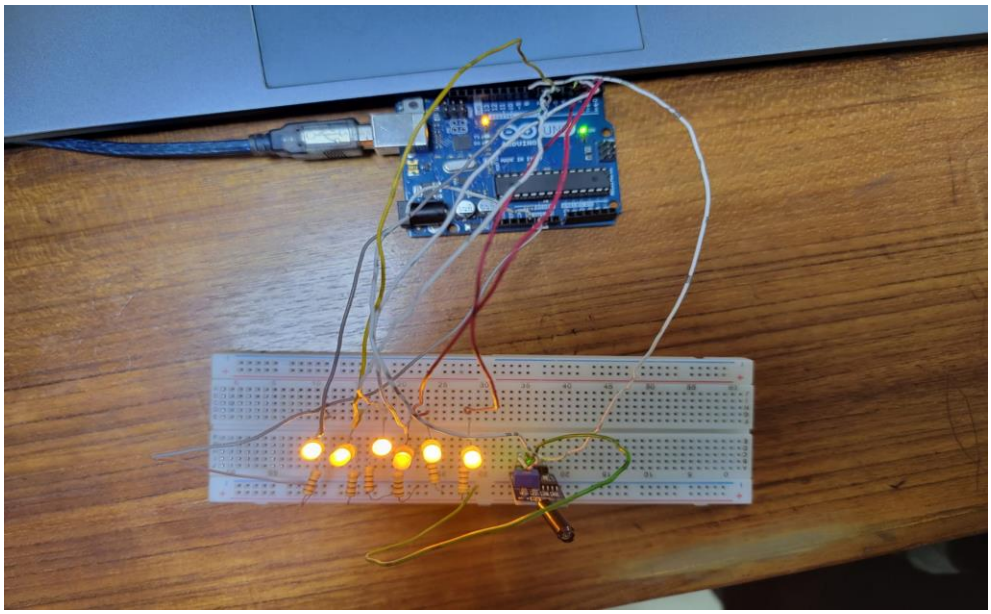


Figure 9: All led on state

Explanation 4(a): In this experiment, an LED pattern control system was implemented using the millis() function on an Arduino Uno. The hardware setup included the Arduino Uno board, a tilt switch for orientation detection, six yellow LEDs (labeled D1 to D6), and 100Ω current-limiting resistors for each LED. The LEDs were connected to six digital output pins of the Arduino through the resistors to ensure safe current flow. The tilt switch was connected to one of the Arduino's digital input pins and served as the trigger for detecting the tilt of the device.The program was written and tested in Arduino IDE version 2.3.5, where the logic was designed to use the millis() function for non-blocking timing. After verification, the compiled HEX file was exported and loaded into the Proteus simulation environment to visualize the circuit behavior. The logic in the code continuously checks the state of the tilt switch. When a tilt is detected, the Arduino responds by turning ON all six LEDs simultaneously. Once the device is returned to a level or upright position, the LEDs turn OFF. This approach avoids using the delay function and ensures better timing accuracy and responsiveness in the LED pattern control, which is particularly important in applications requiring real-time interaction.

b)Power measurement in different power reduction modes while the circuit is in sleep modes:

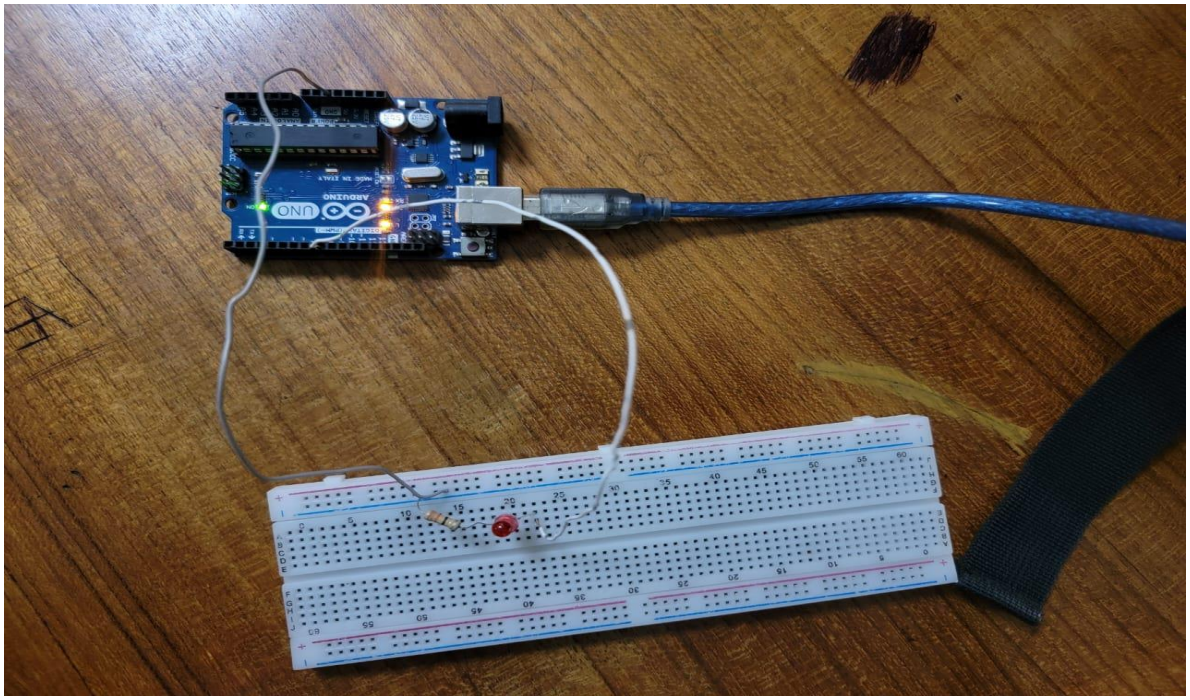


Figure 10: Led in sleep mode

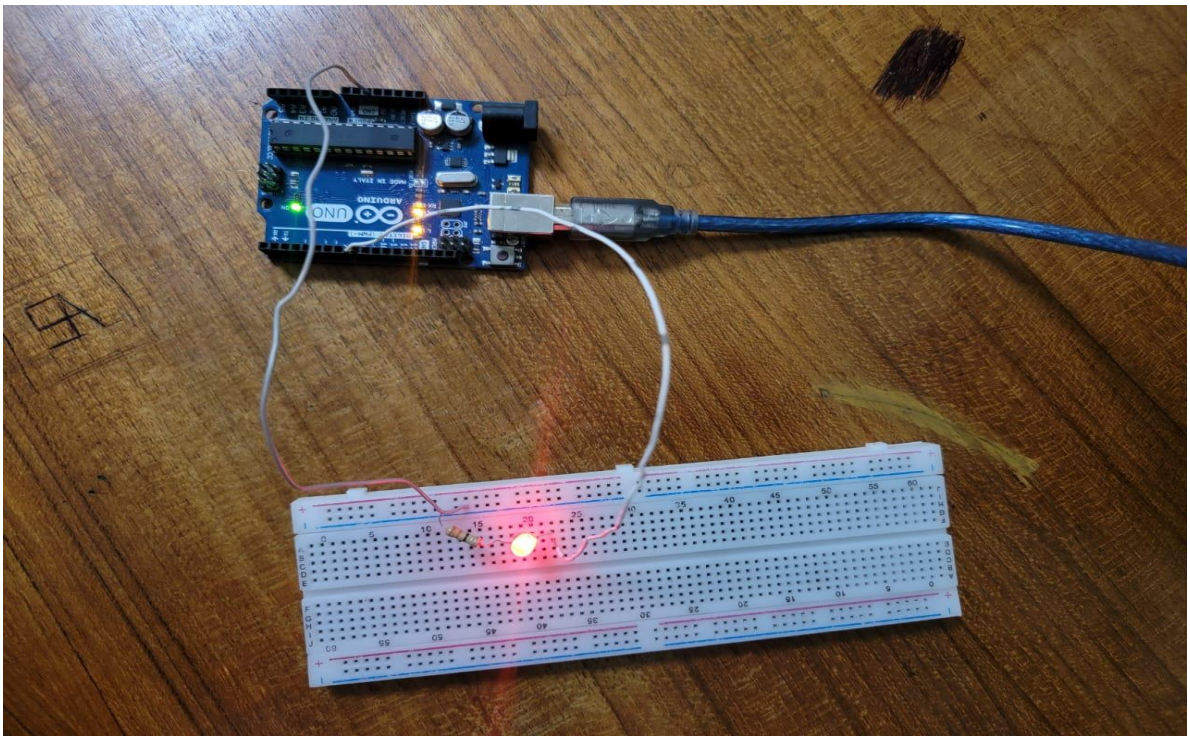


Figure 11: Led in awake mode

Figure 12: Low power

Figure 13: Shutting down time power

Figure 14: High power

Explanation 4(b):

In this experiment, a simple hardware setup was used to measure the power consumption of an Arduino UNO under different operating conditions. The setup included an Arduino UNO board, one LED, and a current-limiting resistor (220Ω). The LED's anode was connected to digital pin 13 of the Arduino, while the cathode was connected to ground through the resistor. A multimeter was used to measure the current drawn from the power supply. A basic LED blink program was uploaded to the Arduino, which turned the LED ON (pin HIGH) and OFF (pin LOW) with a 1-second delay in between. Current readings were taken during both the HIGH and LOW states. When the LED was ON, higher current consumption was observed due to the LED drawing power. When the LED was OFF, the current decreased, as only the Arduino's internal circuits were active. To further analyze power-saving techniques, the code was modified to put the Arduino into sleep mode during the OFF state using the AVR sleep library. This significantly reduced power consumption, demonstrating the effectiveness of sleep mode in minimizing energy usage for low-power applications. The power was calculated using the formula $P = V \times I$, and the experiment successfully illustrated how power consumption varies based on the microcontroller's operational state.

Simulation Output Results:

Simulation 4(a)

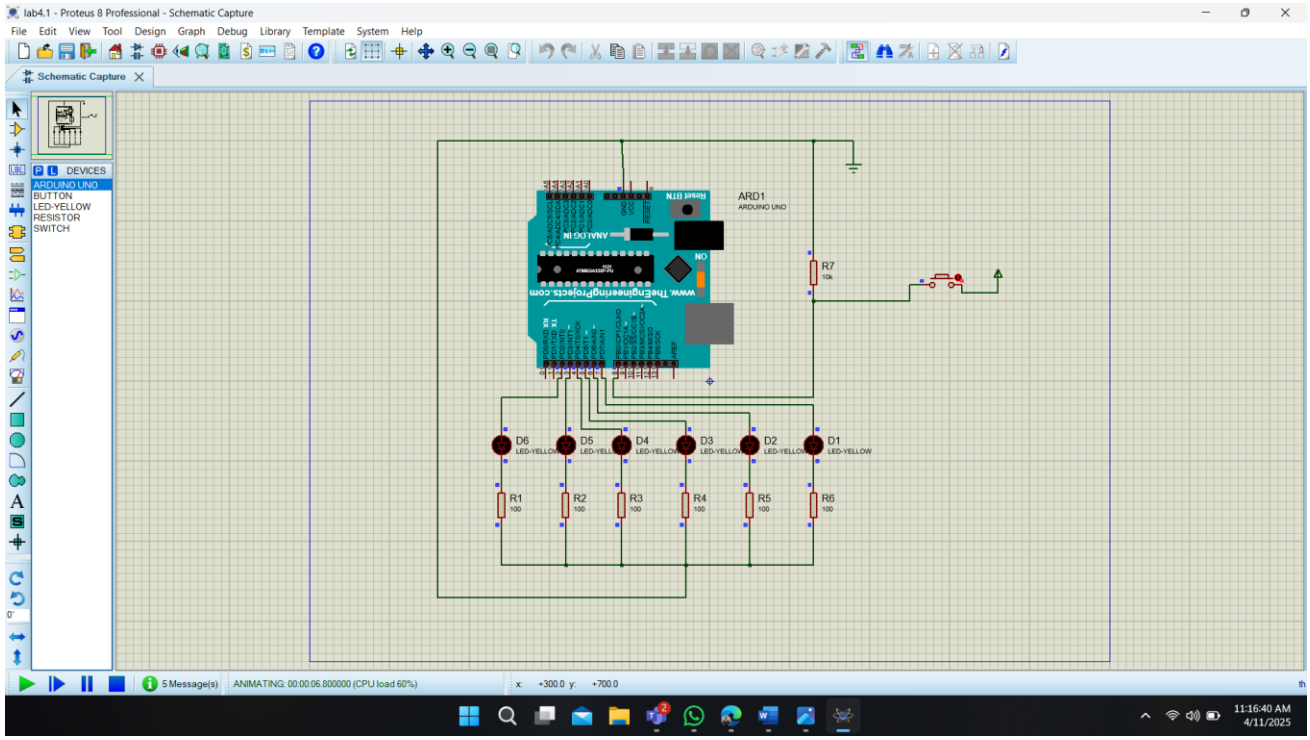


Figure 15: All LED off state

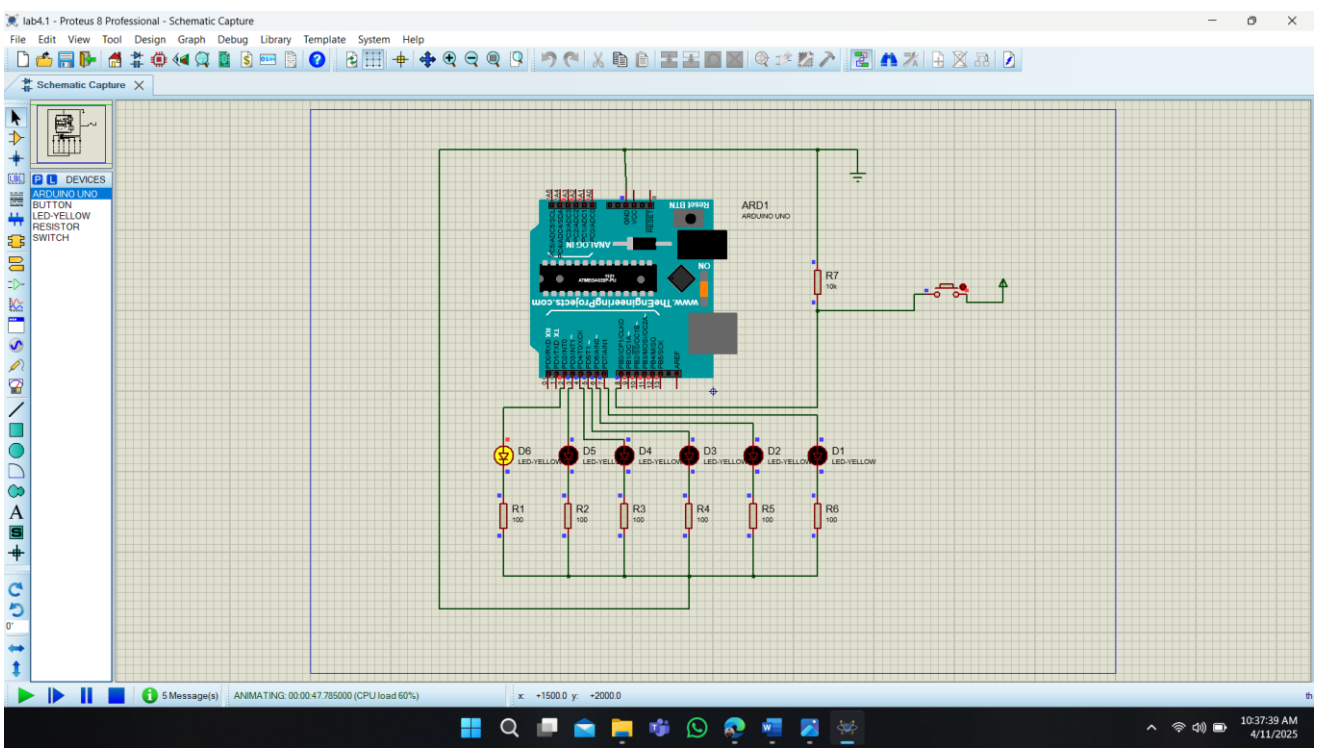


Figure 16: First one LED on state

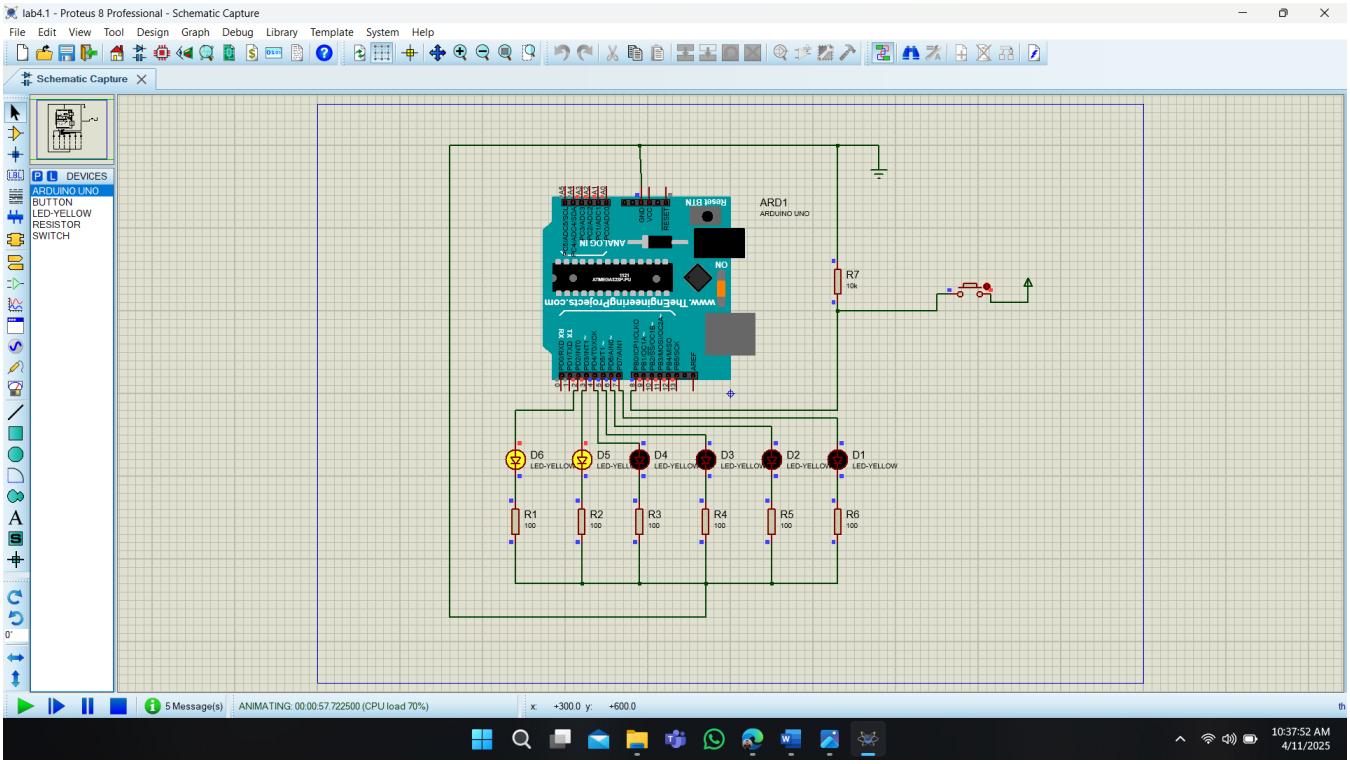


Figure 17: First two LED on state

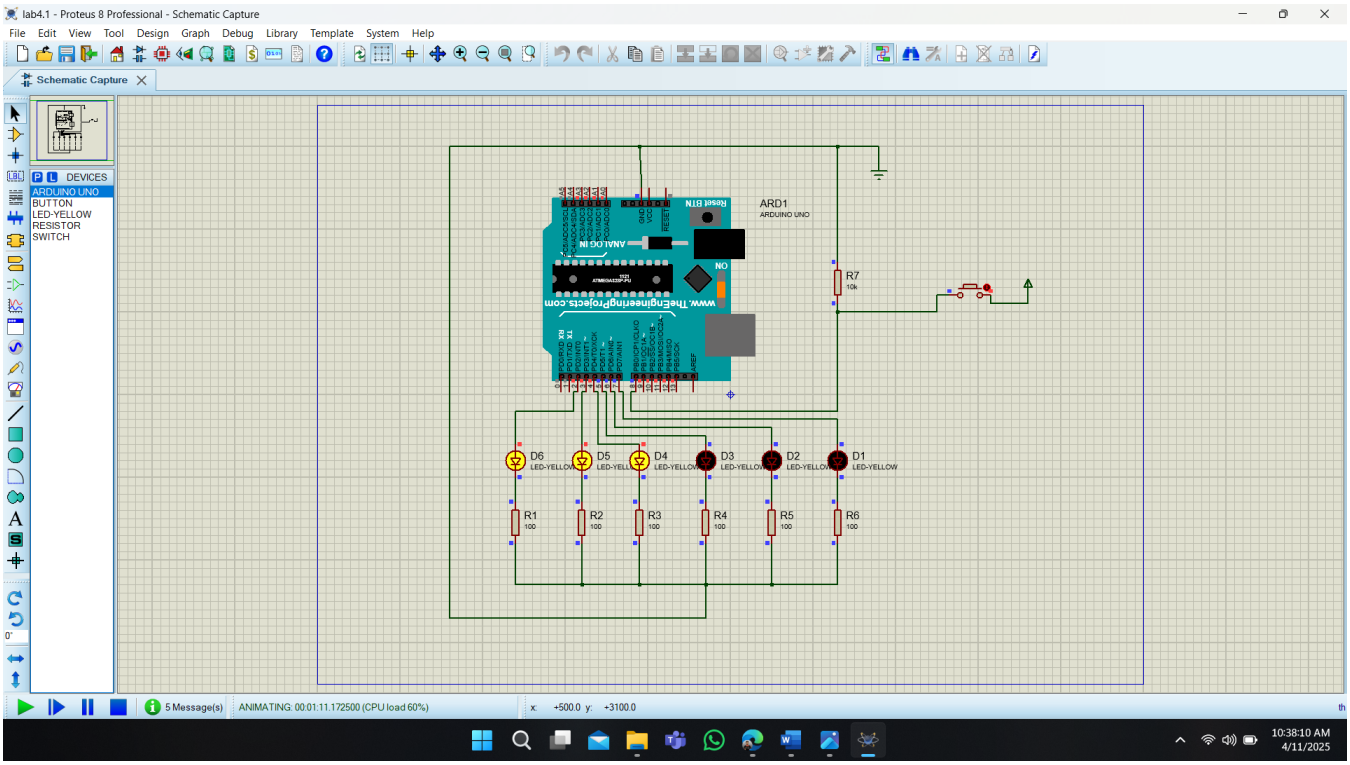


Figure 18: First three LED on state

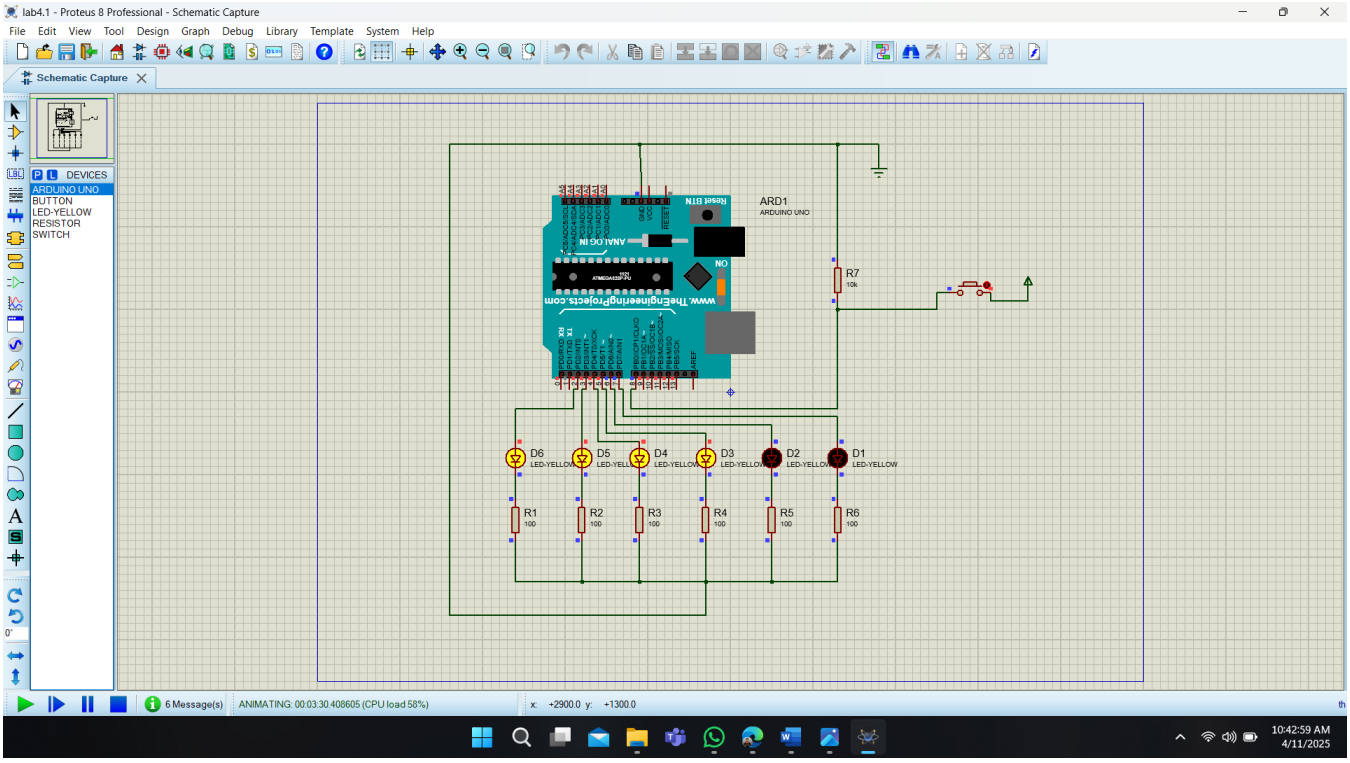


Figure 19: First four LED on state

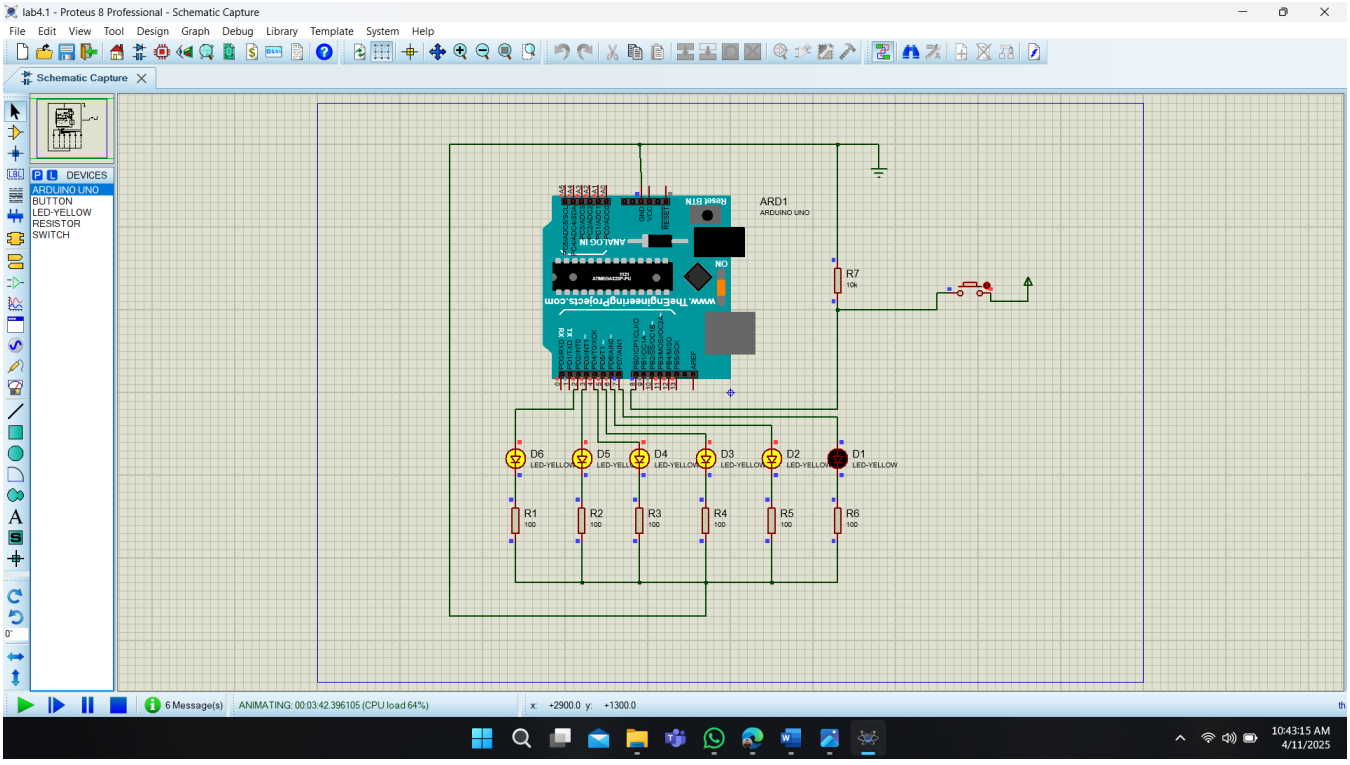


Figure 20: First five LED on state

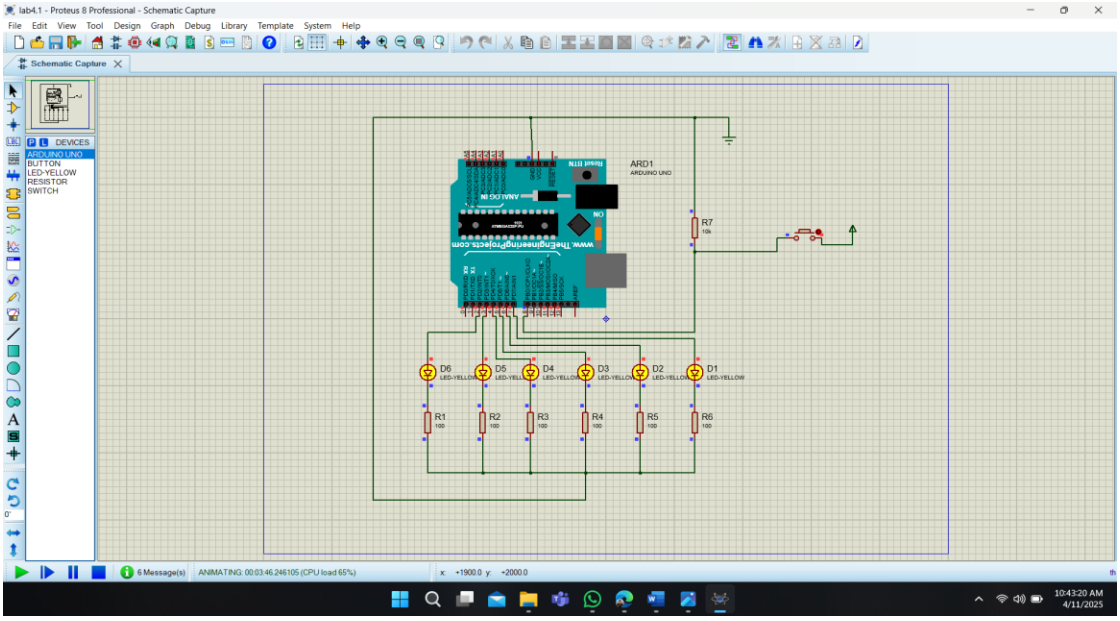


Figure 21: All LED on state

Explanation 4(a): In the simulation phase of the experiment, the entire circuit setup was recreated using the Proteus Design Suite. The components used in the simulation included the Arduino Uno board, a tilt switch module, six yellow LEDs, and six 100Ω resistors-exactly matching the hardware configuration. First, the circuit was designed by placing and wiring the virtual components in Proteus. The anodes of the LEDs were connected to six digital output pins of the virtual Arduino Uno, and their cathodes were connected to ground through the resistors. The tilt switch was connected to one of the Arduino's digital input pins to simulate orientation detection.The program written in Arduino IDE using the millis() function was compiled to generate a .hex file. This HEX file was then imported into the Arduino Uno component within Proteus by setting it in the "Program File" section of the Arduino's properties. Once the simulation was started, the tilt switch's behavior was simulated by toggling its state during runtime. When a tilt was simulated (switch closed), all six LEDs turned ON simultaneously as per the logic in the code. When the switch was returned to its original state (open), the LEDs turned OFF.

Simulation 4(b)

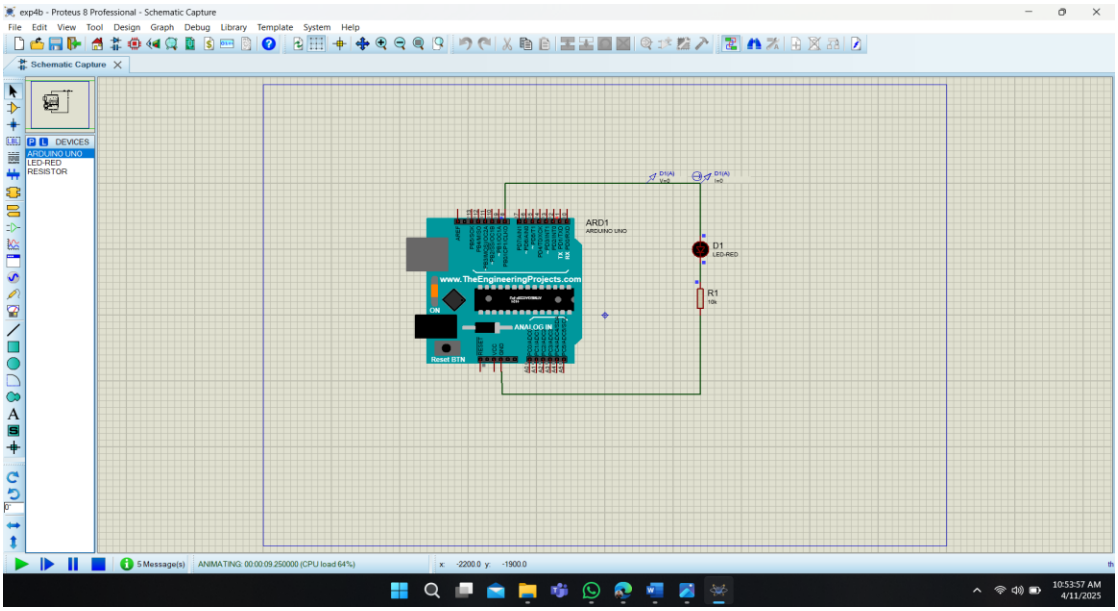


Figure 22: Low power mode

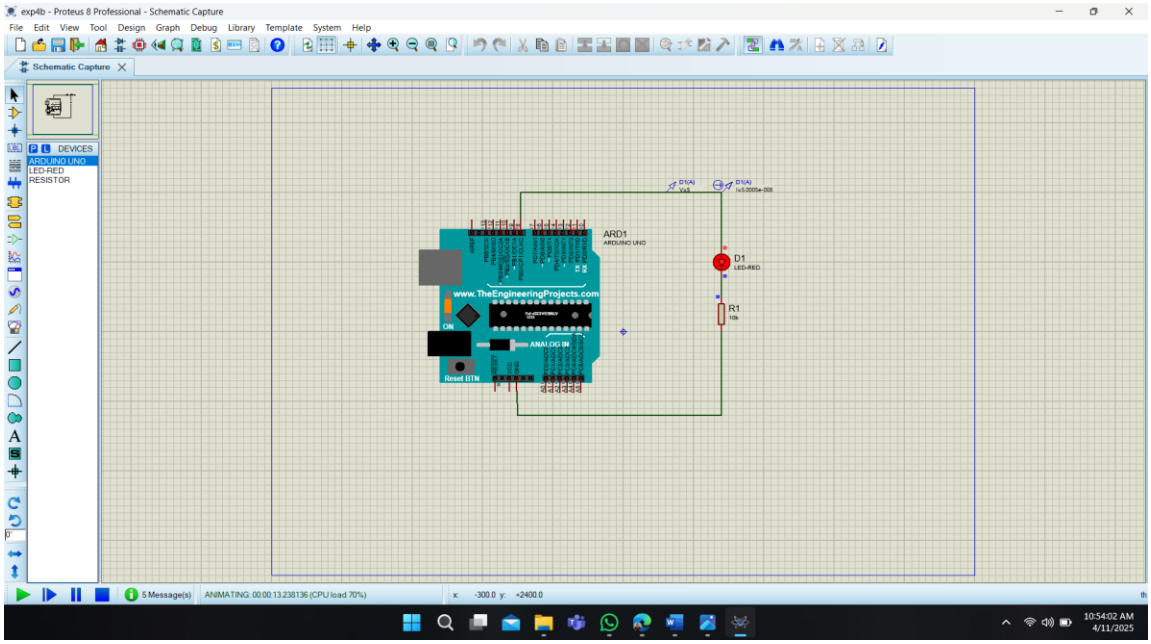


Figure 23: High power mode

<u>Explanation 4(b):</u>In the simulation part of the experiment, Proteus Design Suite and Arduino IDE were used to replicate and analyze the behavior of the hardware circuit virtually. The Arduino UNO board was simulated in Proteus along with a virtual LED and a 220Ω resistor, connected in the same configuration as the physical setup. The LED's anode was connected to digital pin 13 of the Arduino UNO in the simulation, and the cathode was connected to ground through the resistor. The LED blink code was written and compiled in the Arduino IDE, and the generated .hex file was imported into the Arduino UNO component in Proteus. When the simulation was run, the LED was observed turning ON and OFF at 1-second intervals, just like in the physical circuit. This visual behavior confirmed that the code was functioning correctly. Additionally, Proteus provided tools to measure current and voltage in real-time, allowing us to analyze the current flow through the circuit during different states without the need for physical instruments. The simulation also allowed for the testing of sleep mode functionality by observing reduced current flow when the Arduino entered low-power states.

Answers to the Questions:

<u>Answer to the question no 1</u>

Code of a LED light pattern control system with millis() function:

```
const int SwitchPin = 8;
unsigned long PreviousTime = 0;
int SwitchState = 0;
int PrevSwitchState = 0;
int led = 2;
long interval = 3000;
 void setup() {
for (int x = 2; x < 8; x++) {
pinMode(x, OUTPUT);
}
```

```
pinMode(SwitchPin, INPUT);
}

void loop() {
unsigned long CurrentTime = millis();
if (CurrentTime - PreviousTime > interval) {
PreviousTime = CurrentTime;

digitalWrite(led, HIGH);
led++;
if (led == 7){
}
}
SwitchState = digitalRead(SwitchPin);
if (SwitchState != PrevSwitchState){
for (int x = 2 ; x < 8; x++) {
digitalWrite(x, LOW);
}
led = 2;
PreviousTime = CurrentTime;
}
PrevSwitchState = SwitchState;
}
```

Explanation: This Arduino code controls six LEDs connected to digital pins 2 through 7 and lights them up one by one every 3 seconds using a timing mechanism based on millis(). A button connected to pin 8 is used to reset the LEDs. In the setup() function, pins 2 to 7 are configured as outputs for the LEDs, and pin 8 is set as an input for the switch. In the loop(), the code checks if 3 seconds have passed since the last LED was lit—if so, it turns on the next LED and increments the tracking variable. If the button is pressed (detected by comparing the current and previous switch states), all LEDs are turned off, and the sequence resets to start from the first LED again. However, the code lacks a proper boundary condition after lighting the last LED (pin 7), which could lead to issues if the led variable exceeds the valid pin range; ideally, this should be addressed by resetting the led counter or stopping the sequence.

Arduino Code Low Power-Power Down Mode:

```
#include <LowPower.h>
#define LED_RED 8
void setup() {
Serial.begin(9600);
pinMode(LED_RED, OUTPUT);
// SMCR = 0b00001101;
}
void loop() {
digitalWrite(LED_RED, HIGH);
delay(2000);
Serial.println("Arduino:- I am going for a Nap");
digitalWrite(LED_RED,LOW);
LowPower.idle(SLEEP_4S,
```

```
ADC_OFF, TIMER2_OFF, TIMER1_OFF,
TIMER0_OFF, SPI_OFF, USART0_OFF, TWI_OFF);
Serial.println("Arduino:- Hey I just Woke up");
delay(4000);
}
```

Explanation: This Arduino sketch demonstrates how to use the LowPower library to reduce power consumption by putting the board into idle sleep mode. An LED connected to pin 8 turns on for 2 seconds, after which the Arduino prints a message to the Serial Monitor saying it's going for a nap. Then it turns off the LED and enters a low-power idle sleep state for 4 seconds with most peripherals (like ADC, timers, SPI, USART, etc.) disabled. Once it wakes up, it prints a message indicating it’s awake and waits another 4 seconds before repeating the process. This cycle helps save energy while still allowing timed operations and serial communication when needed.

Answer to the question no 2

It has already been done, which is included in the upper section

Answer to the question no 3

Delay Mapping from ID 23-51242-1:

Code:

```
const int SwitchPin = 8;
int SwitchState = 0;
int PrevSwitchState = 0;

// LED pins from 2 to 7
int led = 2;

// Delay times for each LED in seconds, based on ID 512421
int delays[] = {5, 1, 2, 4, 2, 1}; // Corresponding to LEDs 2 to 7

unsigned long PreviousTime = 0;
bool sequenceStarted = false;

void setup() {
  for (int x = 2; x < 8; x++) {
    pinMode(x, OUTPUT);
    digitalWrite(x, LOW);
  }
  pinMode(SwitchPin, INPUT);
}

void loop() {
  unsigned long CurrentTime = millis();

  if (sequenceStarted && led < 8) {
    if (CurrentTime - PreviousTime >= delays[led - 2] * 1000UL) {
```

```
    digitalWrite(led, HIGH);
    PreviousTime = CurrentTime;
    led++;
  }
}

SwitchState = digitalRead(SwitchPin);

if (SwitchState != PrevSwitchState && SwitchState == HIGH) {
  // Reset everything when switch is pressed
  for (int x = 2; x < 8; x++) {
    digitalWrite(x, LOW);
  }
  led = 2;
  PreviousTime = millis();
  sequenceStarted = true;
}

PrevSwitchState = SwitchState;
}
```

Explanation: This Arduino program controls six LEDs connected to pins 2 through 7. A push button on pin 8 triggers the sequence. When the button is pressed, all LEDs reset, and a new lighting sequence starts. Each LED turns ON one by one after a specific delay defined in the delays[] array (in seconds). The delays are: 5, 1, 2, 4, 2, and 1 seconds respectively. The millis() function is used to track time instead of delay() to keep the program responsive. The LED sequence only starts after the button is pressed, and it won't restart until the button is pressed again.

Result:

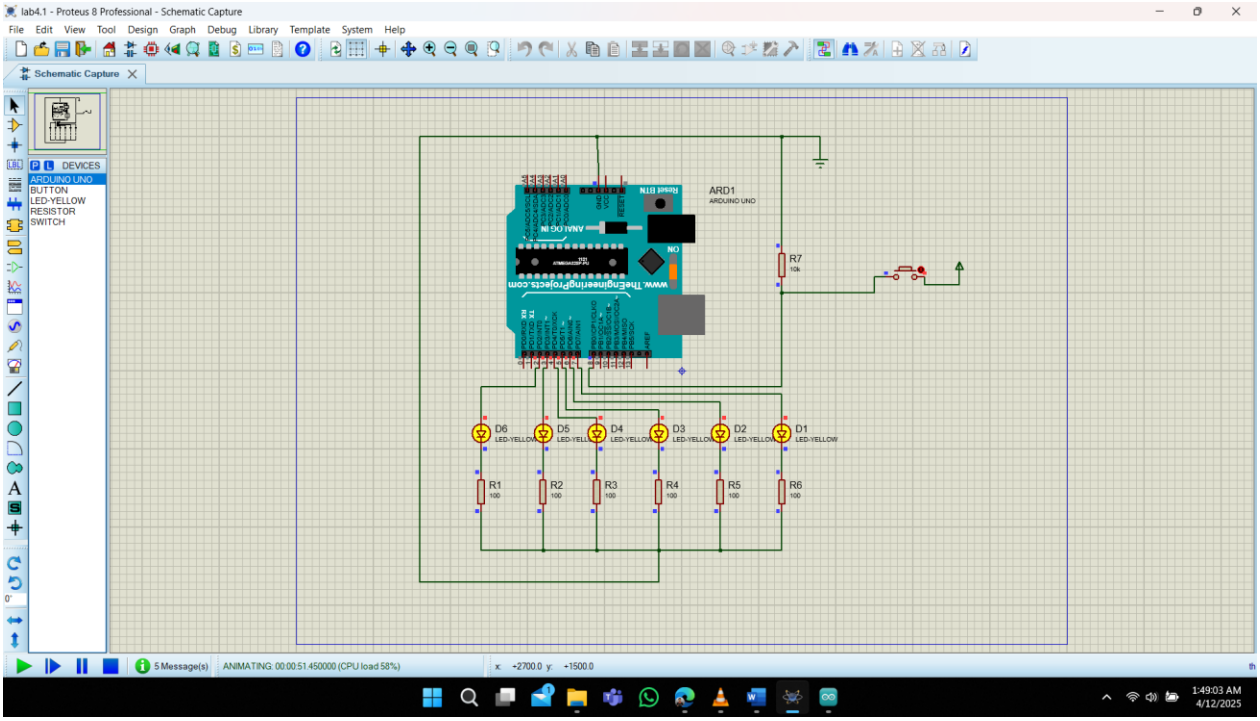


Figure 24: All LED has been on according to ID value times

Answer to the question no 4

```
const int SwitchPin = 8;
const long interval = 3000; // 3 seconds
const int firstLed = 2;
const int lastLed = 7;

unsigned long previousTime = 0;
int switchState = 0;
int prevSwitchState = 0;

bool direction = true; // true = forward (2 to 7), false = backward (7 to 2)
int currentLed = firstLed;
bool allLedsOn = false;
unsigned long allOnTime = 0;

void setup() {
  for (int x = firstLed; x <= lastLed; x++) {
    pinMode(x, OUTPUT);
    digitalWrite(x, LOW);
  }
  pinMode(SwitchPin, INPUT);
}

void loop() {
  unsigned long currentTime = millis();

  switchState = digitalRead(SwitchPin);

  // Check for button press to change direction and reset
  if (switchState != prevSwitchState && switchState == HIGH) {
    direction = !direction; // toggle direction
    resetLeds();
  }
  prevSwitchState = switchState;

  // Turn off all LEDs 3 seconds after last LED turns on
  if (allLedsOn && currentTime - allOnTime >= interval) {
    resetLeds();
  }

  // Advance LED if time has passed and all are not on
  if (!allLedsOn && currentTime - previousTime >= interval) {
    previousTime = currentTime;
    digitalWrite(currentLed, HIGH);

    if ((direction && currentLed >= lastLed) || (!direction && currentLed <= firstLed)) {
      allLedsOn = true;
      allOnTime = currentTime;
    } else {
      currentLed = direction ? currentLed + 1 : currentLed - 1;
```

```
    }
  }
}

void resetLeds() {
  for (int x = firstLed; x <= lastLed; x++) {
    digitalWrite(x, LOW);
  }
  currentLed = direction ? firstLed : lastLed;
  allLedsOn = false;
  previousTime = millis();
}
```

Explanation: This Arduino program controls six LEDs connected to digital pins 2 to 7. A push button on pin 8 is used to toggle the lighting direction of the LEDs-either from left to right (2 to 7) or right to left (7 to 2).When the button is pressed, the current direction is reversed, and the LEDs are reset. The program uses a 3-second interval (interval = 3000) between turning on each LED. After all six LEDs are ON, they remain ON for 3 seconds, then automatically turn OFF, and the sequence is ready to start again.The code uses millis() for non-blocking timing, meaning it doesn't pause the entire program like delay() would.

Answer to the question no 5:

Tilt switch is off:

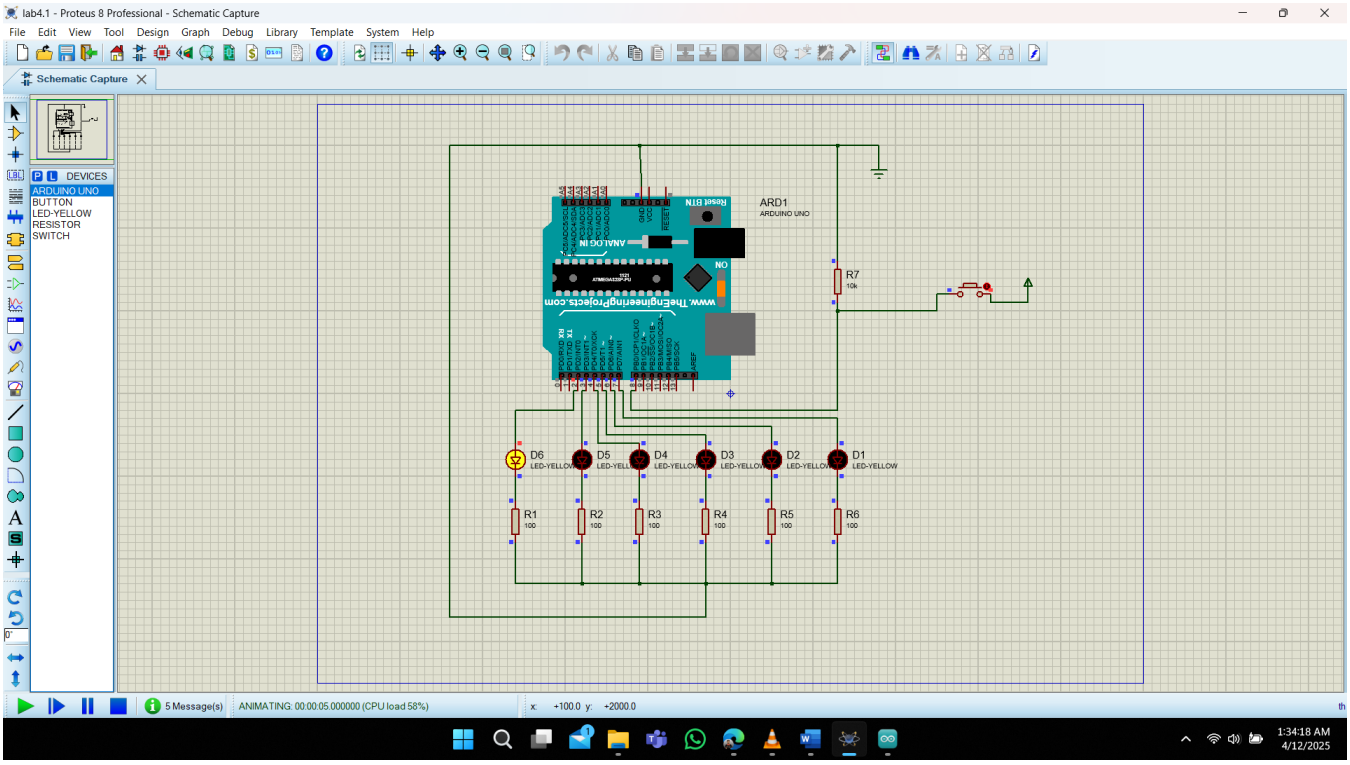


Figure 25: First one LED is on from left

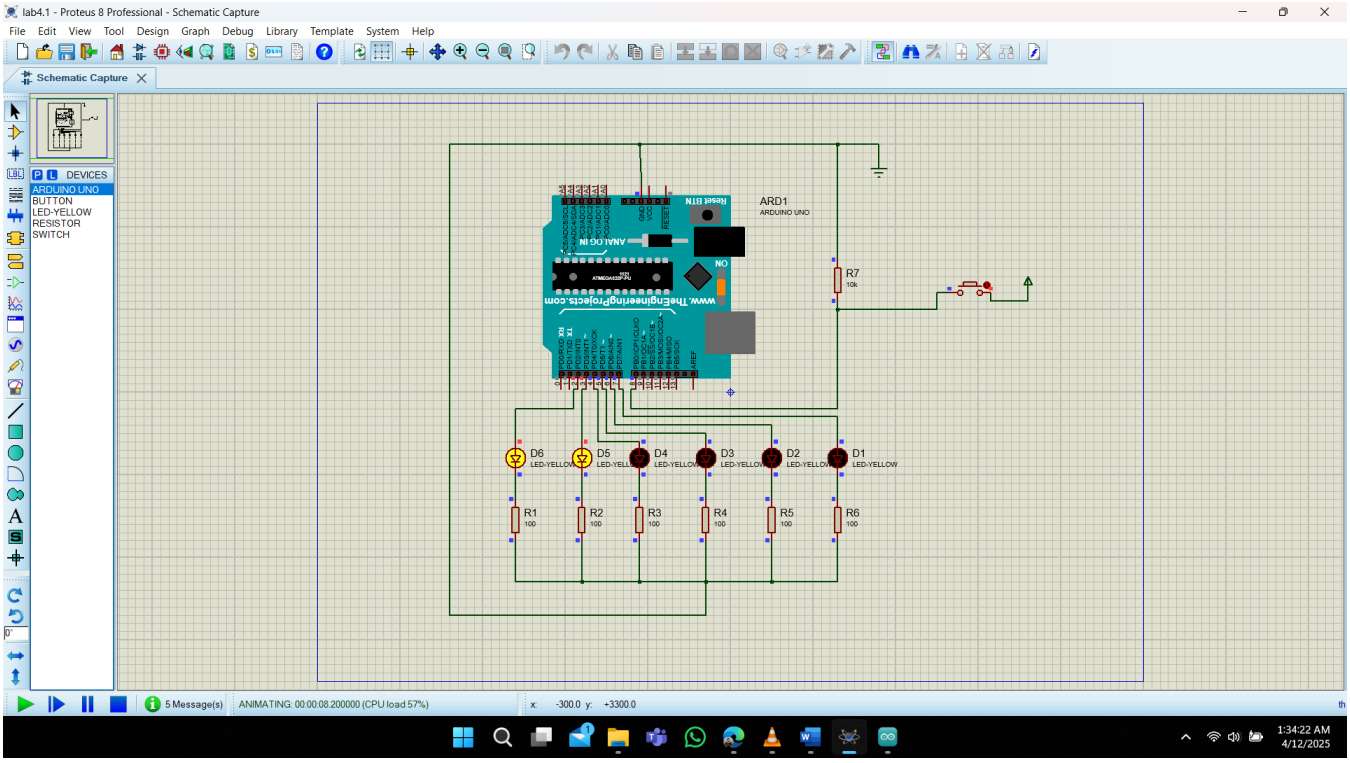


Figure 26: First two LED is on from left

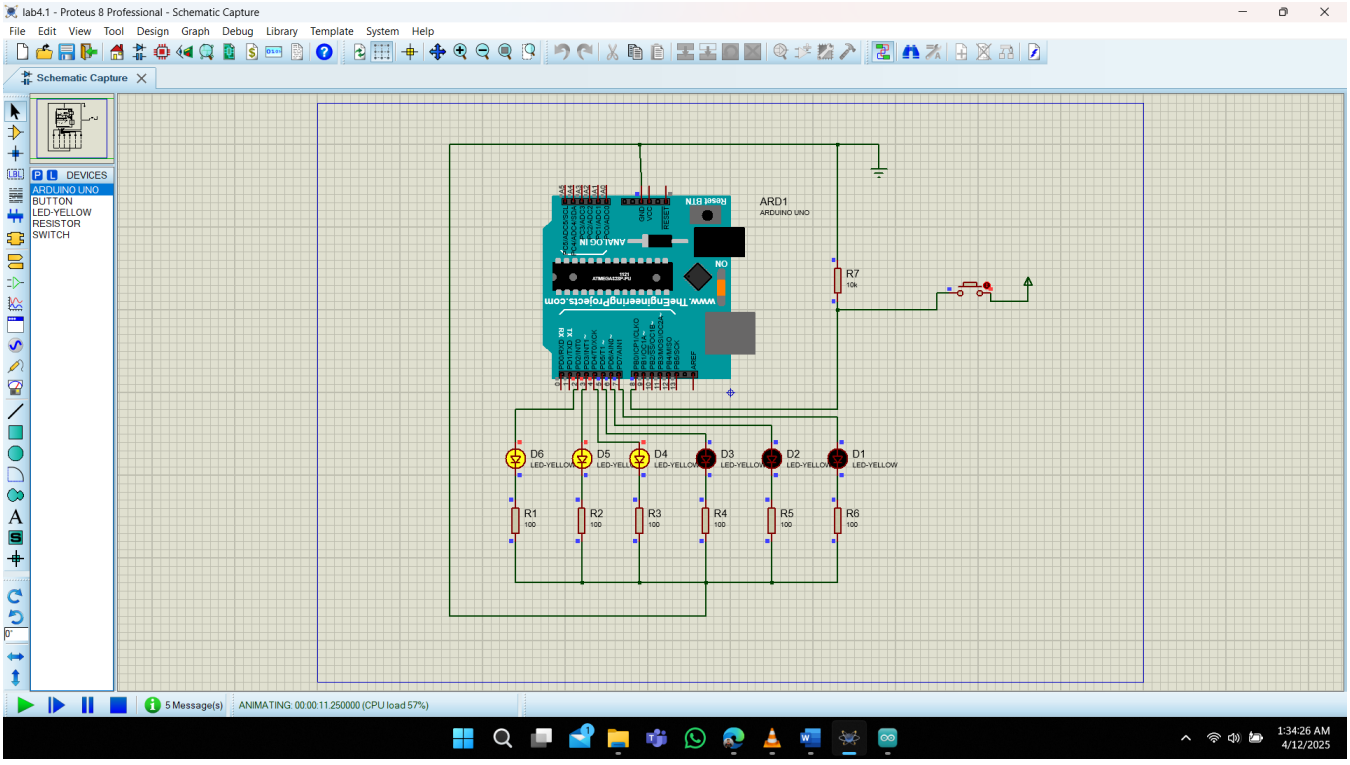


Figure 27: First three LED is on from left

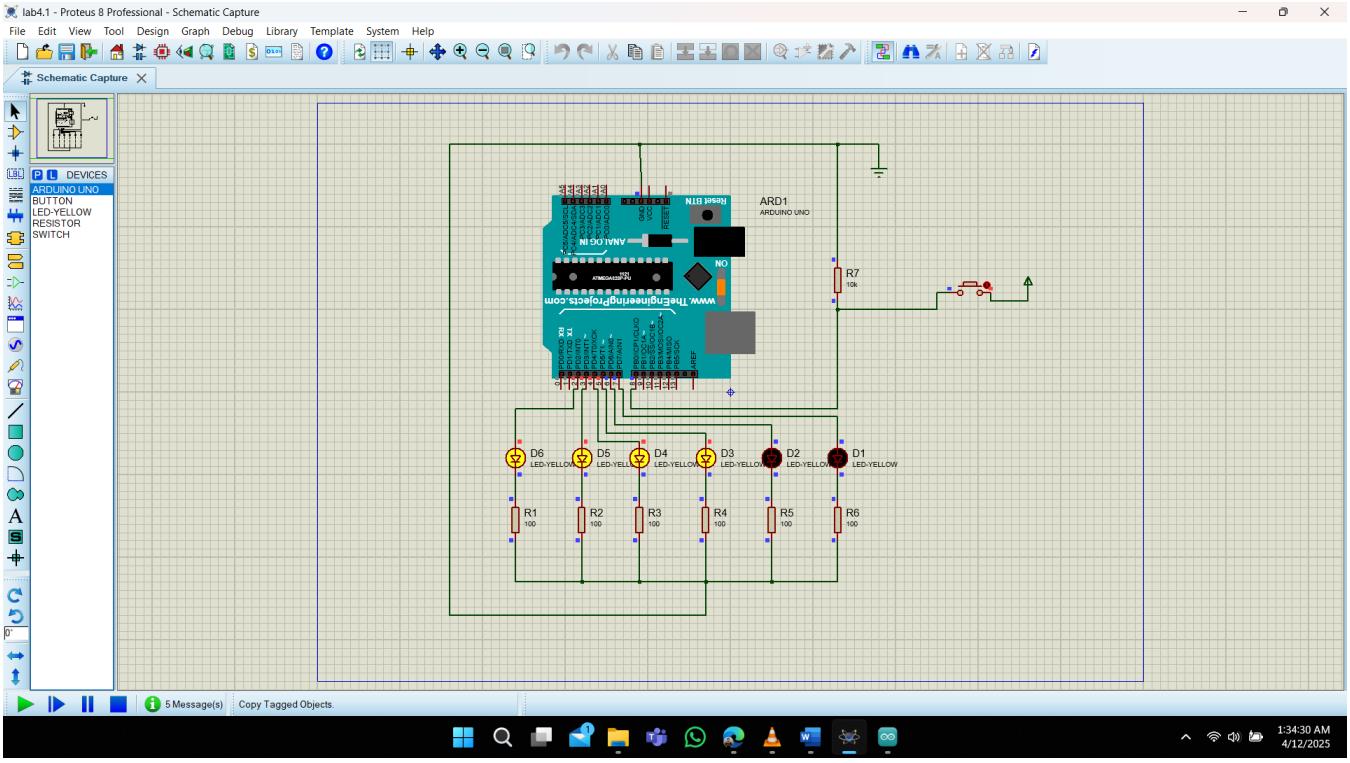


Figure 28: First four LED is on from left

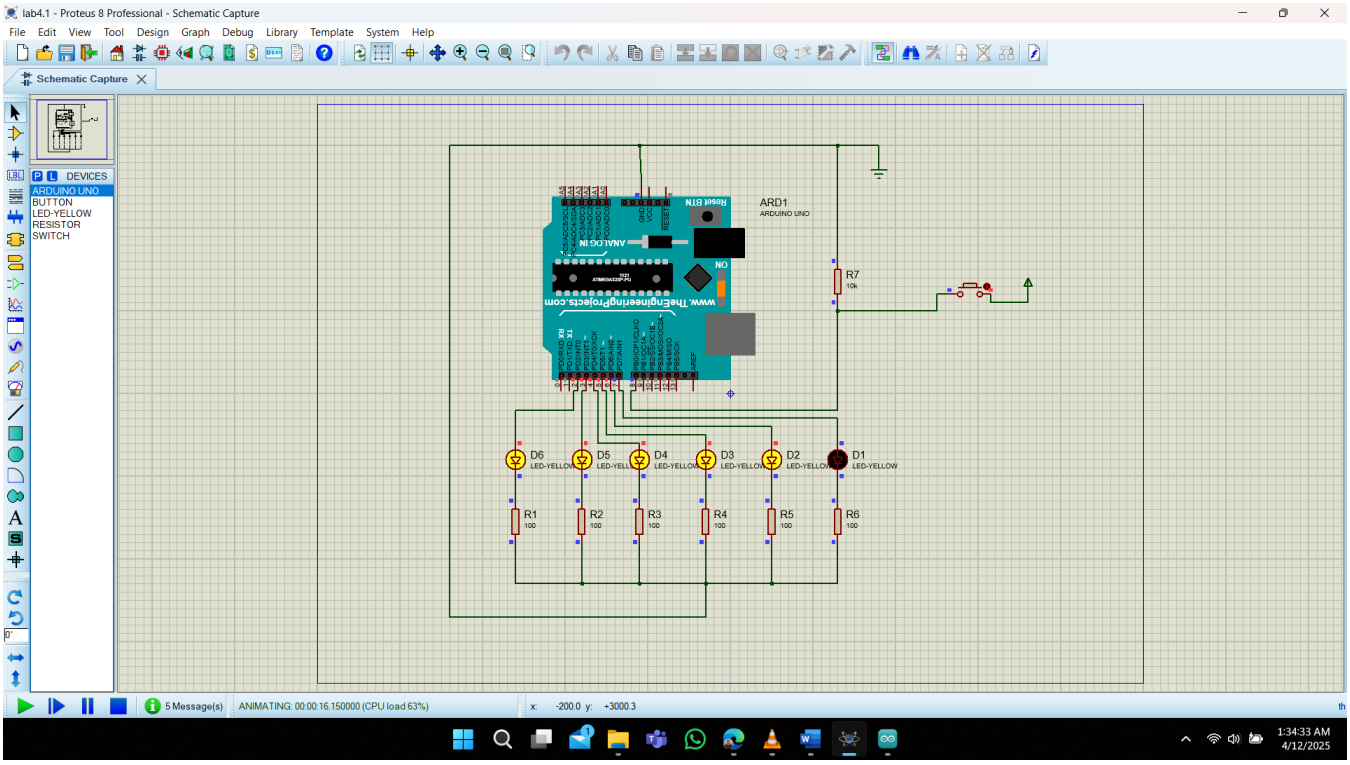


Figure 29: First five LED is on from left

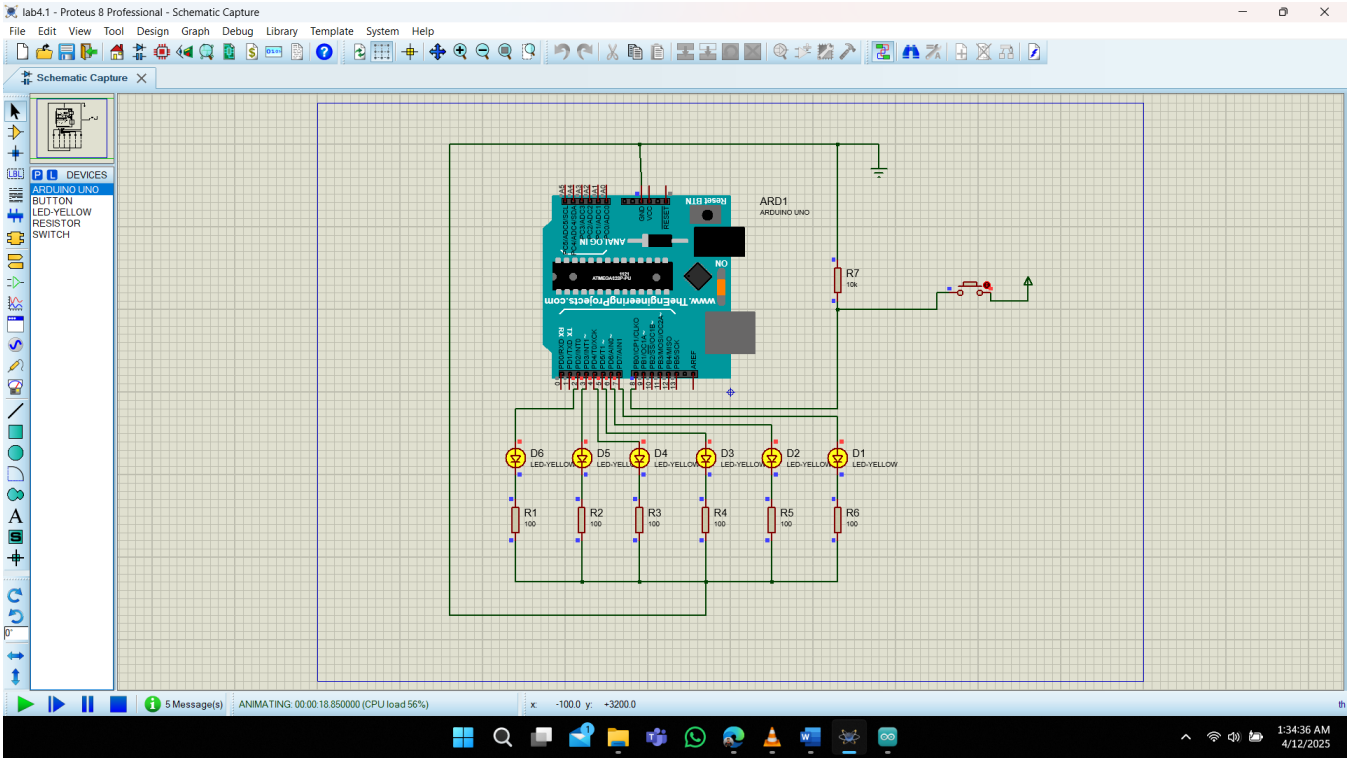


Figure 30: All LED is on from left

Tilt switch is on:

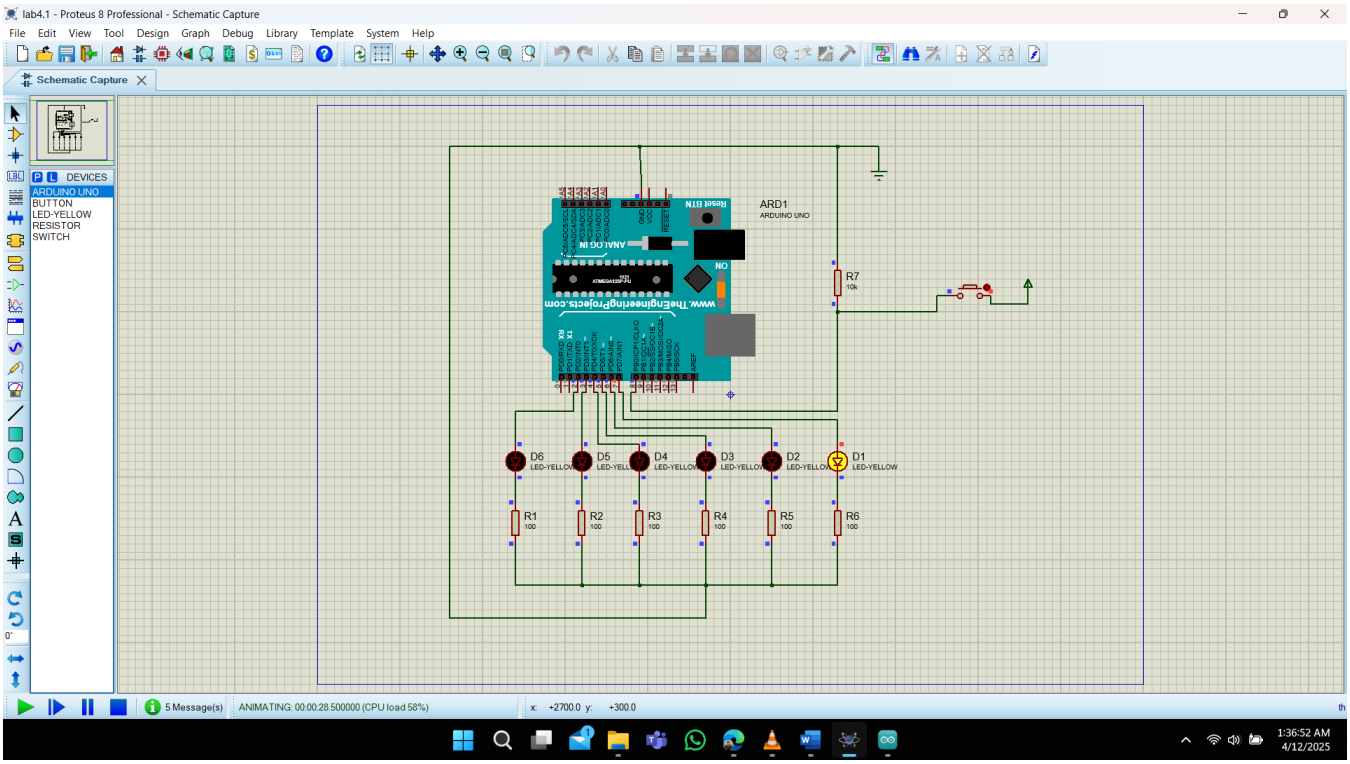


Figure 31: First one LED is on from right

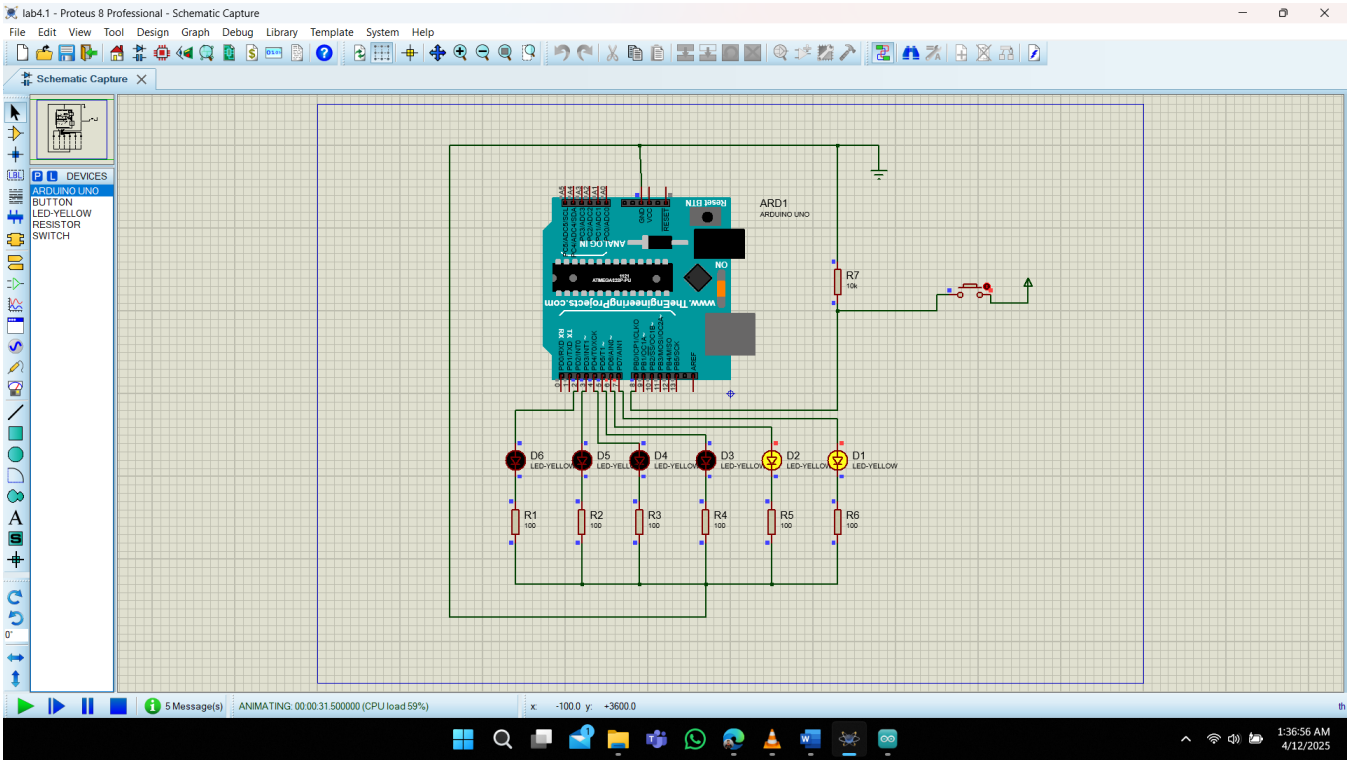


Figure 32: First two LED is on from right

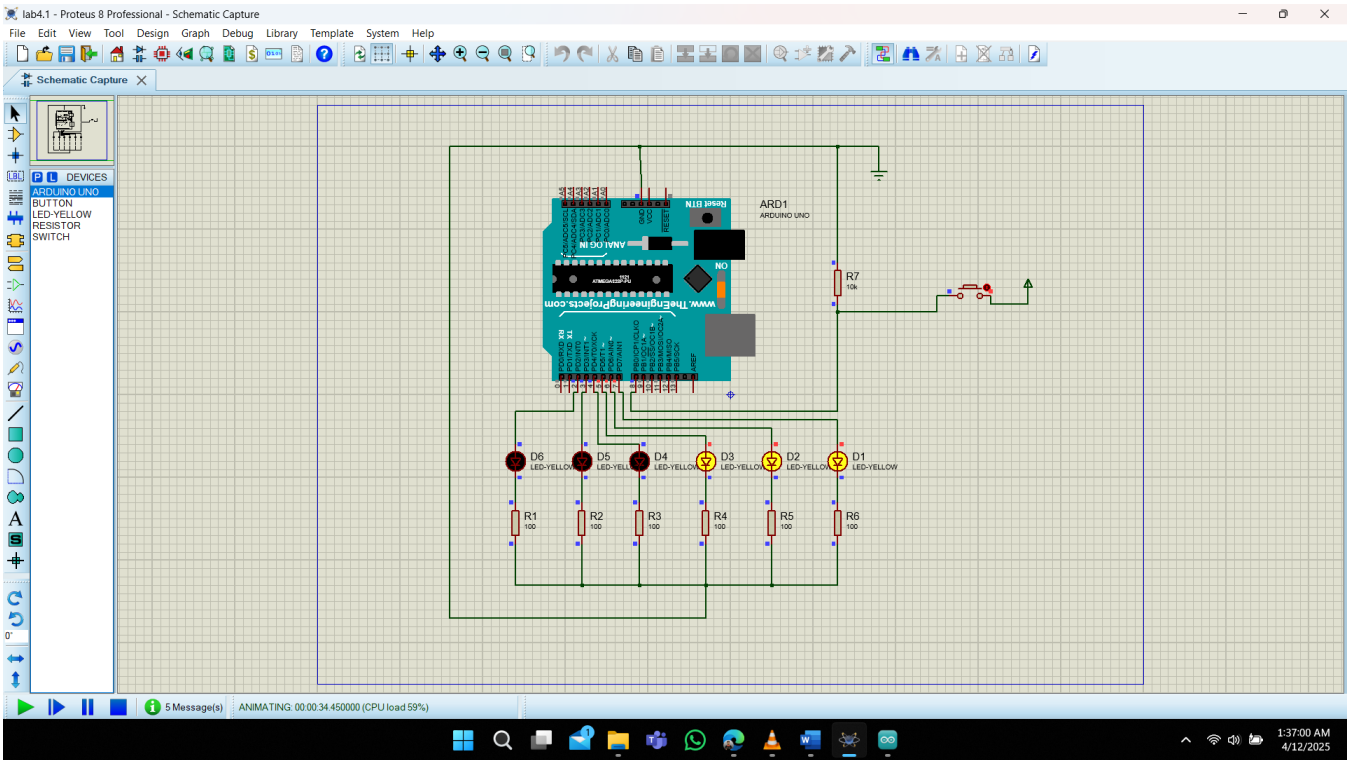


Figure 33: First three LED is on from right

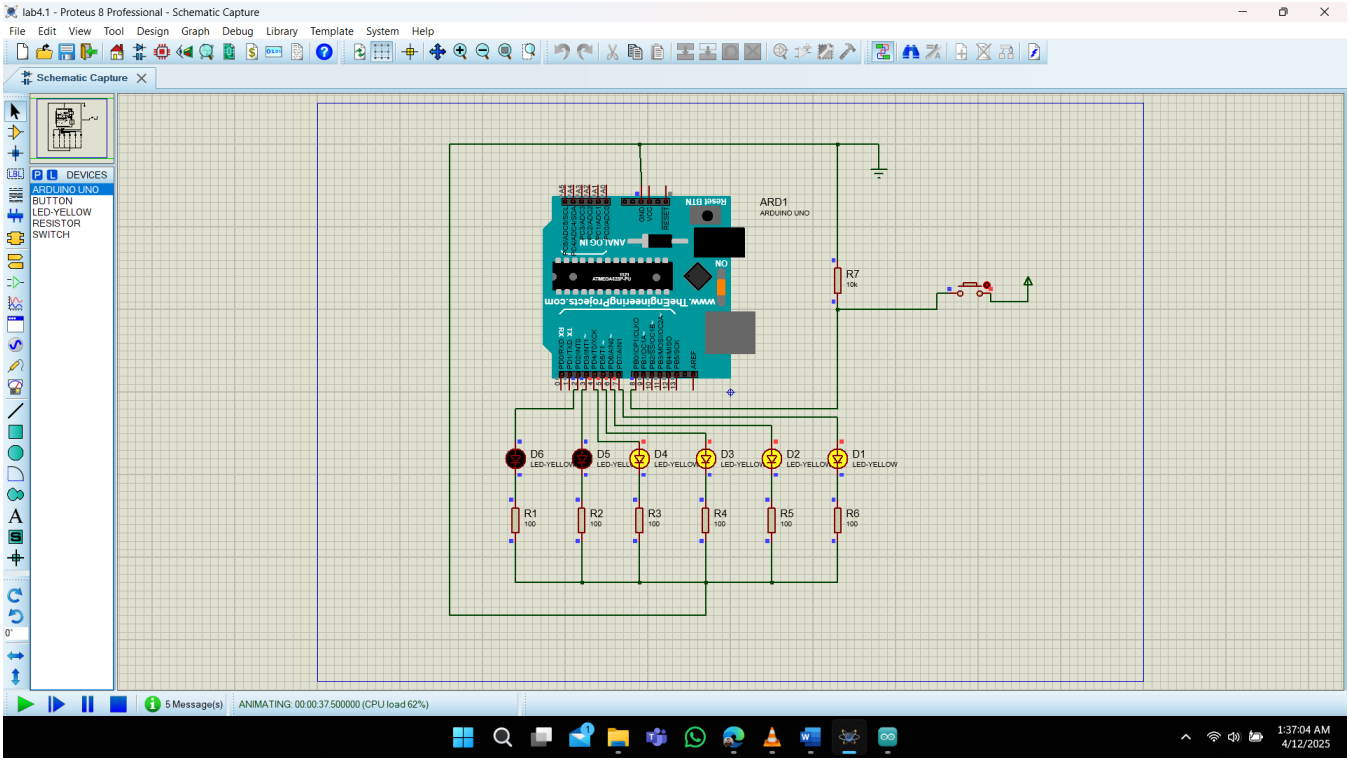


Figure 34: First four LED is on from right

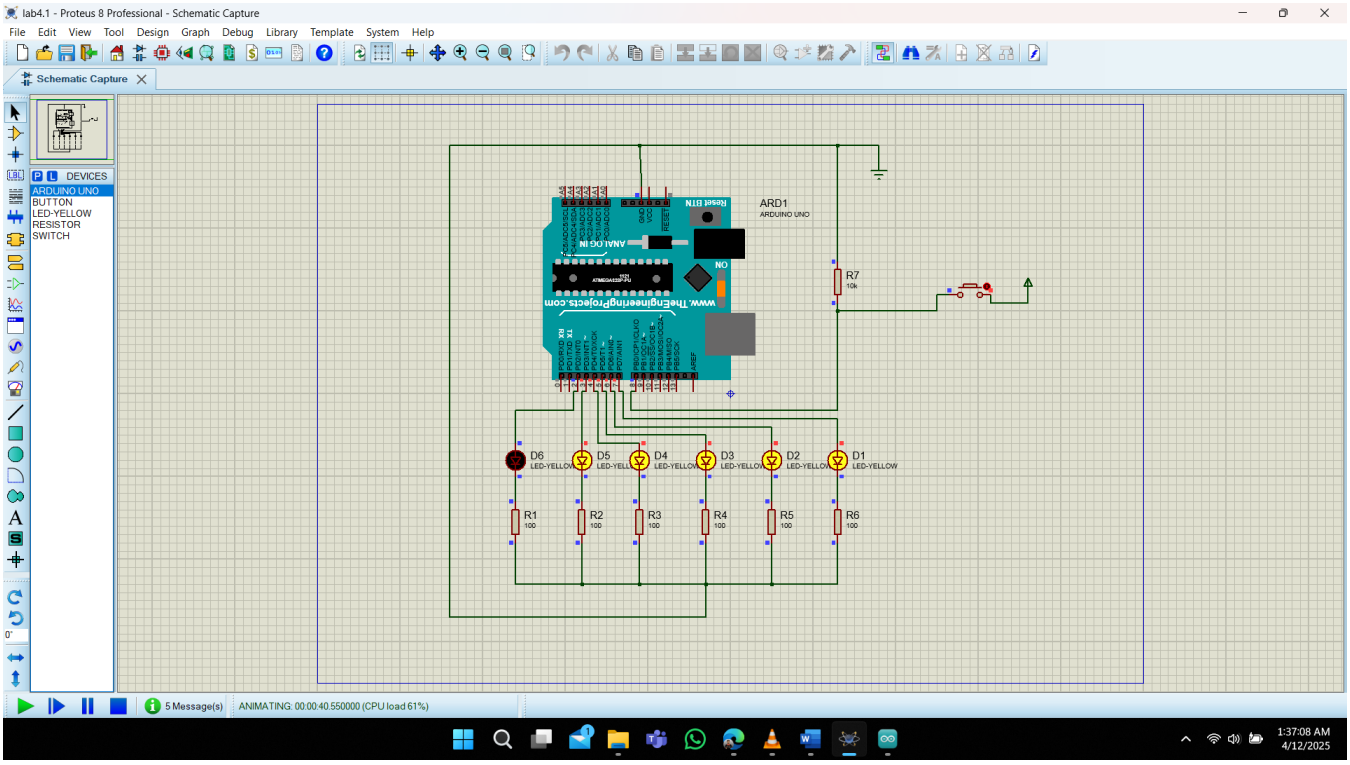


Figure 35: First five LED is on from right

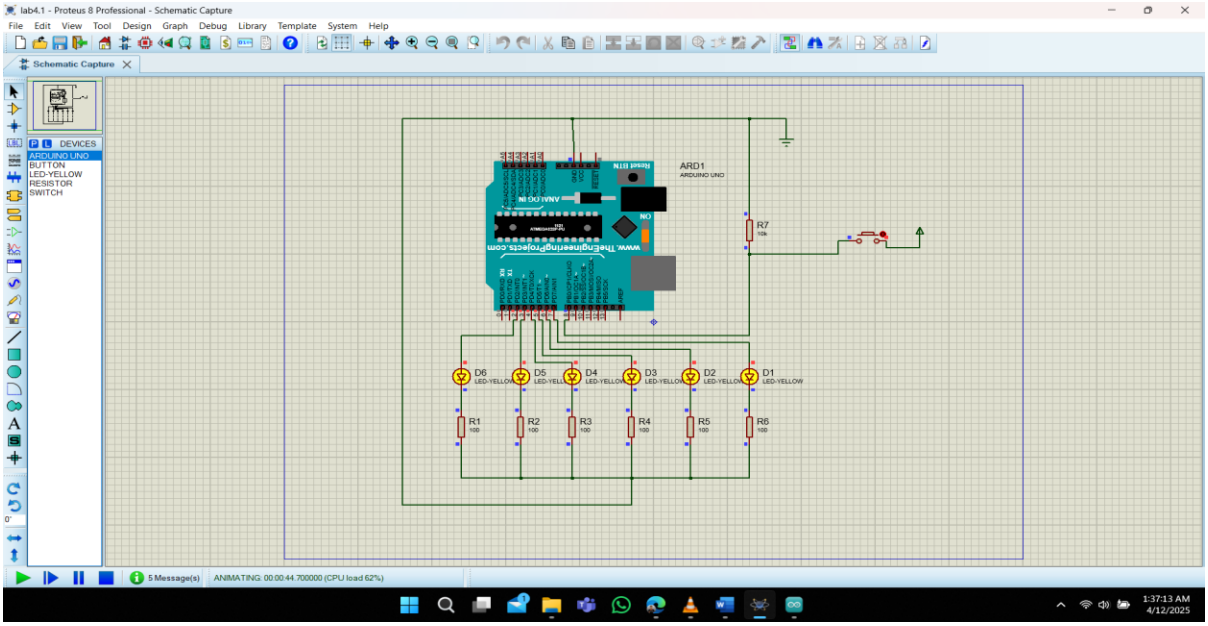


Figure 36: Al LED is on from right

Explanation: This Arduino-based simulation demonstrates a dynamic LED sequencing system where six LEDs (connected to pins 2 through 7) light up one at a time in a timed interval of 3 seconds per LED. The direction of the sequence—either from left to right or right to left—is controlled by a push-button input. Every time the button is pressed, the system reverses the direction of the LED lighting order and restarts the sequence from the appropriate end. As each LED turns on based on the direction and timing logic, once all LEDs are lit, the system enters a 3-second pause before automatically turning all LEDs off and resetting the sequence. This entire process provides a real-time interactive experience, allowing the user to influence the visual flow of the LEDs by toggling the button, making it a practical demonstration of input-triggered behavior, timing control using millis(), and directional logic in embedded systems.

Answer to the question no 6

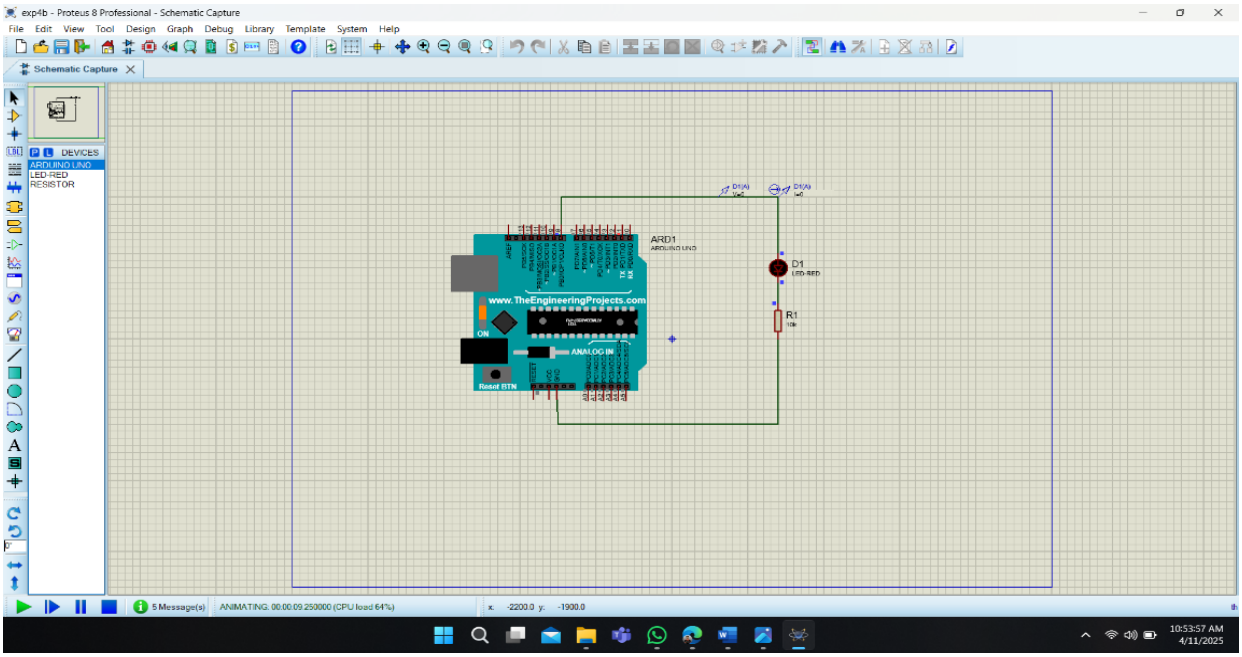


Figure 37: Arduino in sleep mode

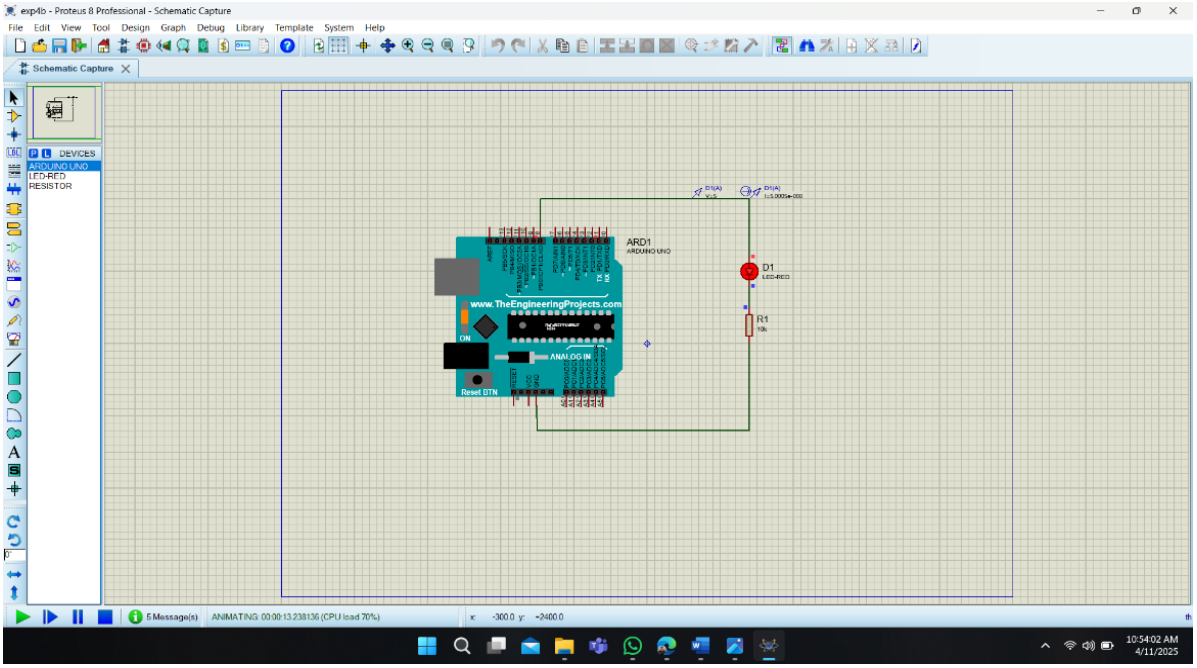


Figure 38: Arduino in awake mode

Answer to the question no 7

Compute power saving in different modes of the program:

From "Experimental Results" section we can see the voltage and current readings of the power supplied to/ requested by the Arduino UNO board to remain in an active/ low power state.
Using the formula, Power, $P = VI$ [V=Voltage, I = Current]
For high, (High) = 5.064*0.011= 0.055704 watts
For Low, (low) = 5.076 × 0.0 = 0 watts
For Reset, P = 5.066 * 0.005= 0.02533 watts
From the calculated result,it is evident that placing the Arduino UNO into a low power state is significantly more energy efficient compared to keeping it in a high power (active) state.

Discussion:

This experiment was designed to explore the advanced features of the Arduino microcontroller, specifically focusing on creating a digital timer using the millis() function and analyzing power consumption in various sleep modes. The primary aim was to address the limitations of the commonly used delay() function and implement a more efficient and non-blocking timing mechanism while also studying how the Arduino can conserve energy when idle.
In traditional Arduino programming, the delay() function is often used to create pauses in the code. However, this method is blocking, meaning the microcontroller halts all activity during the delay period. This restricts the ability to perform other tasks concurrently, such as reading sensors, updating displays, or communicating with other devices. By contrast, the millis() function returns the number of milliseconds since the program started and can be used to measure time intervals without stopping program execution. This non-blocking behavior allows for multi-tasking, making it ideal for real-time and responsive systems.
During the experiment, we implemented a timer-based LED control using the millis() function. The system was able to blink LEDs at specific intervals while simultaneously remaining responsive to

additional tasks or inputs. This confirmed the superiority of millis() in designing efficient, event-driven programs. Moreover, we explored how to manage multiple concurrent operations without interference or time drift, which is essential in embedded systems development.
Additionally, we studied power measurement and reduction techniques using Arduino's sleep modes. By configuring the microcontroller to enter Idle, ADC Noise Reduction, Power-down, and Standby modes, we significantly reduced power consumption when the circuit was not actively performing tasks. These modes are particularly important in battery-powered applications, where energy efficiency directly affects system longevity. The Arduino was programmed to enter a sleep state and wake up periodically using interrupts or time-based conditions, thus enabling precise operation with minimal power usage.
This hands-on experiment provided valuable experience in writing efficient and power-aware code using millis() and sleep functions. It enhanced our understanding of:Timer-based task scheduling,Event-driven programming,Microcontroller power management,Practical use of hardware interrupts and sleep modes.

Here are some real life scenario and application of this experiment:
1.Battery-Operated IoT Devices: Wireless environmental sensors (temperature, humidity) that wake up at fixed intervals using millis() to transmit data, then return to sleep to conserve battery life.
2. Wearable Health Monitors: Heart-rate or step-counting devices that remain in low-power mode and wake periodically using timers, saving energy while continuously monitoring user activity.
3. Smart Irrigation Systems: Arduino-based systems that activate water pumps based on scheduled intervals using millis() and sleep during inactive periods to save energy in outdoor environments.
4. Energy-Efficient Smart Home Devices: Motion-sensitive lights or alarms that stay in sleep mode until a motion-triggered interrupt occurs, then use millis() for timing light ON duration.
5. Data Logging in Remote Locations: Weather monitoring stations powered by solar cells that log data every few minutes using millis() and remain in sleep mode between logs to maximize energy efficiency.

Conclusion:

This experiment provided valuable hands-on experience in designing and implementing an embedded system using Arduino. By utilizing the millis() function for non-blocking LED control, we gained insights into efficient programming techniques. The simulation in Proteus allowed us to test the system without physical hardware, saving time and resources. Additionally, we explored power measurement and sleep modes, which are crucial for energy-efficient designs in real-world applications like motion-triggered lighting and safety systems. Overall, this experiment enhanced our understanding of embedded systems, coding practices, and power optimization techniques.

References:

[1] https://www.arduino.cc/.
[2] ATMega328 manual
[3] https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers
[4] http://maxembedded.com/2011/06/avr-timers-timer0/