AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Engineering

Lab Report

Experiment # 10

Experiment Title: Familiarization with the Raspberry Pi.

Date of Perform:	12 May 2025	**Date of Submission:**	19 May 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:

The objectives of this experiment are to-

a) Familiarize the students with the Raspberry Pi.
b) Make an LED blink using the Raspberry Pi and its time.sleep() function.
c) Control the LEDs' ON/OFF using the input push switch.
d) Implement a traffic light control system.

Apparatus:

1. Activated Raspberry pi
2. LEDs (GREEN,YELLOW AND RED)
3. Push switch
4. Resistor (220 Ω and 10kΩ)
5. Breadboard
6. Jumper wires

Circuit Diagram:

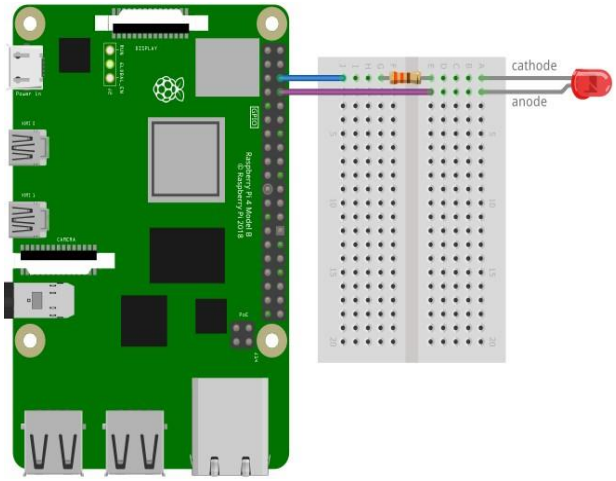

Figure-01: Circuit for LED blink

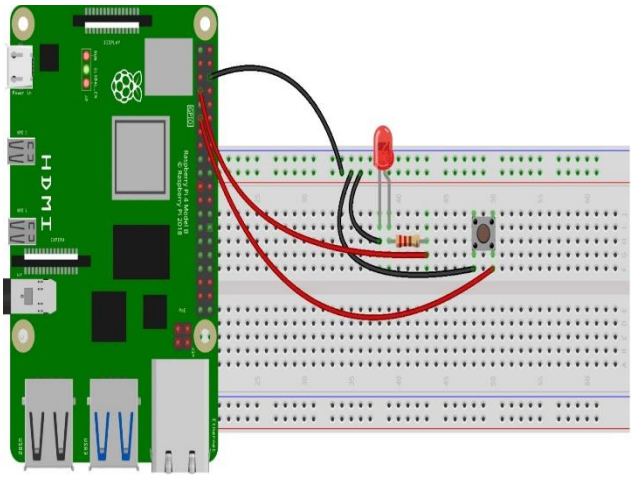

Figure-02:Circuit for control LED with Button

Code Explanation:

Code-01: LED Blinking

```
import RPi.GPIO as GPIO          # RPi.GPIO library will allow us to control the GPIO pins.
import time                      # time library contains the sleep()
GPIO.setmode(GPIO.BCM)           # BCM pin numbering is used
GPIO.setwarnings(False)          # to disable warnings
GPIO.setup(14, GPIO.OUT)         # to set GPIO14 as an output
GPIO.output(14, GPIO.HIGH)       # to specify the GPIO 14 as HIGH
print "LED is ON"                # show message to Terminal.
time.sleep(2)                    # for two seconds
GPIO.output(14, GPIO.LOW)        # to specify the GPIO 14 as LOW
print "LED is OFF"               # show message to Terminal.
```

Explanation: This Python script is designed to control an LED connected to a Raspberry Pi's GPIO pin. It uses the RPi.GPIO library, which allows the Raspberry Pi to interface with its GPIO pins. At the beginning of the code, **GPIO.setmode(GPIO.BCM)** sets the pin numbering system to use the Broadcom SOC channel numbers (e.g., GPIO14), ensuring consistent reference to pins across different models. **GPIO.setwarnings(False)** disables warnings about pins that might already be in use, avoiding unnecessary alerts during repeated script runs. The **GPIO.setup(14, GPIO.OUT)** line configures pin 14 as an output pin, making it suitable for sending signals to connected components like an LED. **GPIO.output(14, GPIO.HIGH)** turns the LED on by supplying a HIGH voltage level, and the message "LED is ON" is printed. The program then pauses for 2 seconds using **time.sleep(2),** allowing the LED to remain lit. After the delay**, GPIO.output(14, GPIO.LOW)** switches the pin to LOW, turning the LED off, followed by printing "LED is OFF".

Code-02: LED controlling with a push button switch

```
from gpiozero import LED              # imports LED functions from the gpiozero library
from gpiozero import Button           # imports Button functions from the gpiozero library
led = LED(4)                          # declare the GPIO in pin 4 for LED output and store it in a variable named led.
button = Button(17)                   # declare the GPIO in pin 17 for Button input and store it in a variable named button.
while True:                           # initiate an infinite while loop
button.wait_for_press()               # use the built-in function of the button to wait till press
led.on()                              # turn on the LED
button.wait_for_release()             # use the built-in function of the button to wait till release
led.off()                             # turn off the LED
```

Explanation: This Python script demonstrates how to control an LED using a push button with the **gpiozero** library, which simplifies GPIO operations on the Raspberry Pi. The program begins by importing the **LED** and **Button** classes from the **gpiozero** library, allowing direct use of their built-in methods. An LED is connected to GPIO pin 4 and assigned to the variable led, while a push button is connected to GPIO pin 17 and assigned to the variable button. The script then enters an infinite **while** loop, which continuously monitors the state of the button. Inside the loop, **button.wait_for_press()** causes the program to pause until the button is pressed. Once pressed, the **led.on()** function turns on the LED. The program then waits for the button to be released using **button.wait_for_release(),** after which the LED is turned off using **led.off().**

Code-03: Simple Traffic Control System

```
import RPi.GPIO as GPIO
import time

# setting-up the raspberrypi pins
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(14, GPIO.OUT)  # Green LED
GPIO.setup(15, GPIO.OUT)  # Yellow LED
GPIO.setup(18, GPIO.OUT)  # Red LED

while (True):    # Forever Loop
# for green led
GPIO.output(14, GPIO.HIGH)
print("Green LED is ON")
time.sleep(1.0)    # wait 1 sec
GPIO.output(14, GPIO.LOW)
print("Green LED is OFF")
time.sleep(1.0)    # wait for 1 sec
# for yellow led
for i in range(3):   # repeat 3 times
GPIO.output(15, GPIO.HIGH)
print("Yellow LED is ON for " + str(i+1))
time.sleep(0.5)
GPIO.output(15, GPIO.LOW)
print("Yellow LED is OFF for " + str(i+1))
time.sleep(0.5)
# for red led
GPIO.output(18, GPIO.HIGH)
print("Red LED is ON")
time.sleep(1.0)
GPIO.output(18, GPIO.LOW)
print("Red LED is OFF")
time.sleep(1.0)
```

Explanation: This Python code is written to control three LEDs connected to a Raspberry Pi using the **GPIO** library. GPIO pins 14, 15, and 18 are configured as outputs for the green, yellow, and red LEDs respectively. The program begins by setting the GPIO mode to BCM and disabling warnings. Inside an infinite loop, the green LED connected to pin 14 is turned on for 1 second and then turned off, using **GPIO.output()** and **time.sleep().** Next, the yellow LED (pin 15) blinks three times with 0.5-second intervals, implemented using a **for** loop and alternating HIGH/LOW states. Finally, the red LED on pin 18 is turned on for 1 second and then off. The loop continues indefinitely, repeating this sequence. The **print()** statements after each action help monitor LED status in the terminal during execution.

Experimental Output Results:

1.Blink test:

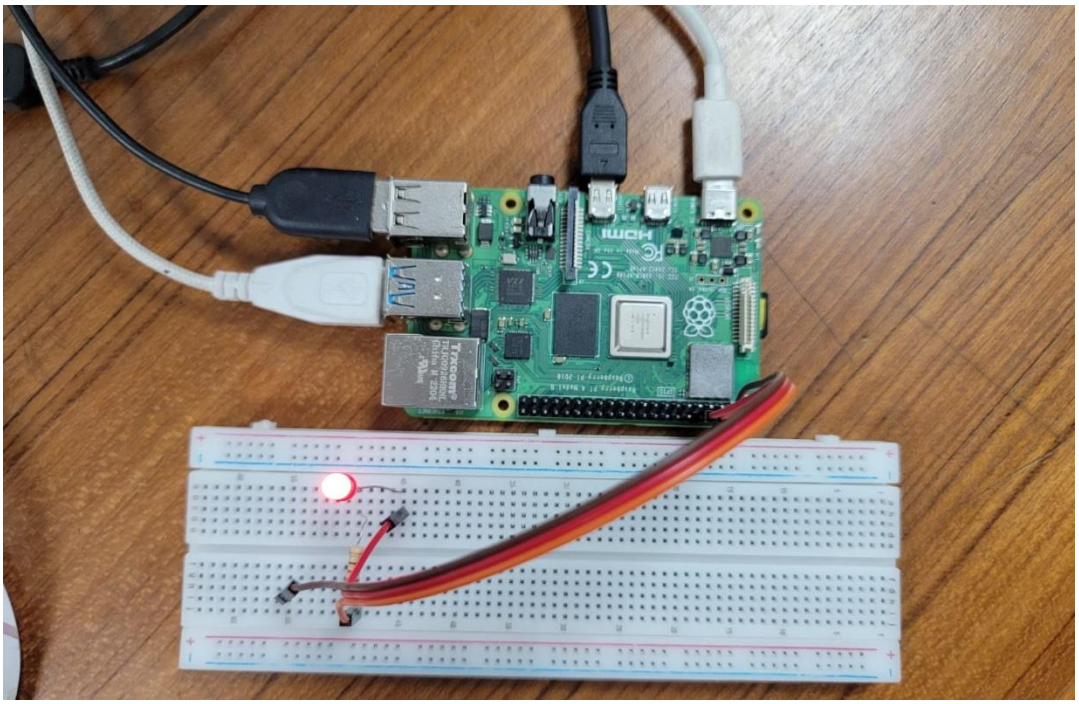

Figure-03:LED blink test(When LED is on)

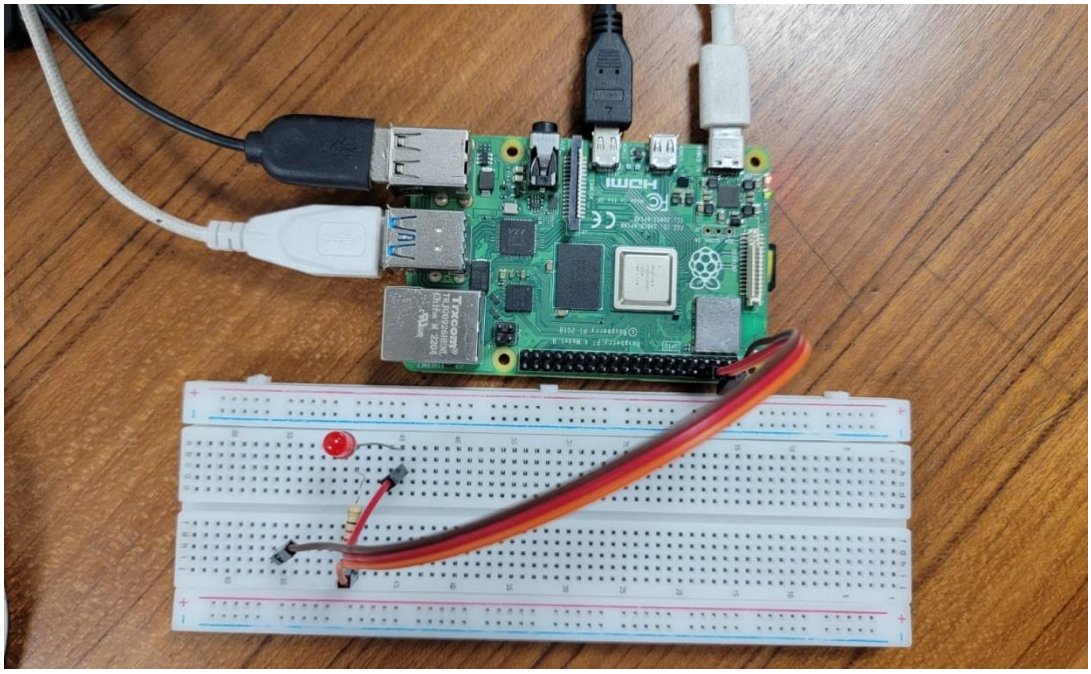

Figure-04:LED blink test(When LED is off)

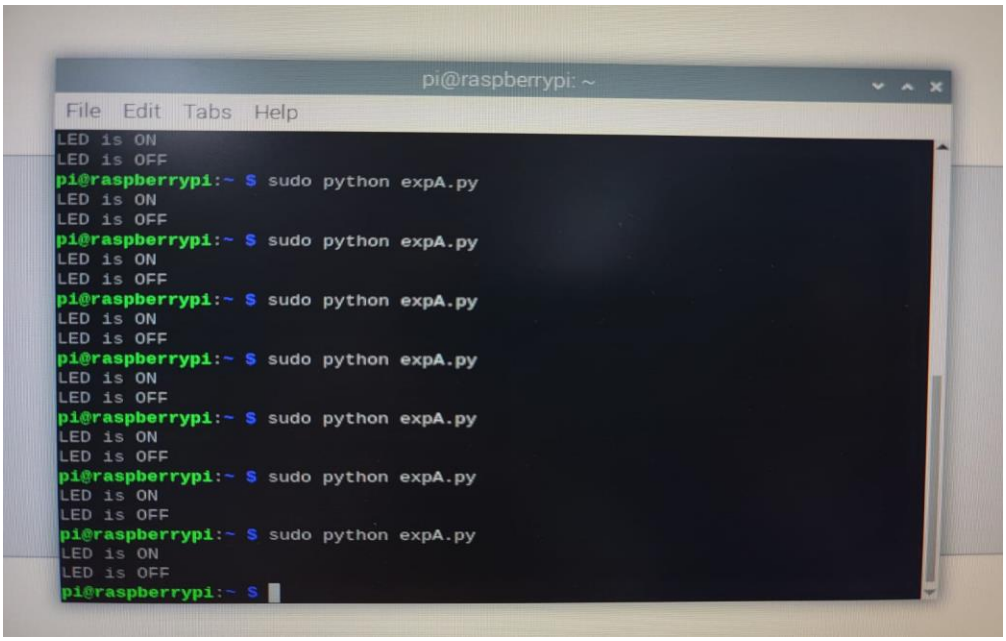

Figure-05:LED blink test(Display comment on terminal)

Explanation: In this setup, a single LED is controlled using a Raspberry Pi and the RPi.GPIO library. The circuit is built by connecting a standard LED to GPIO **pin 14** of the Raspberry Pi, with an appropriate current-limiting resistor in series. The Raspberry Pi is programmed to operate in BCM pin numbering mode, which maps physical pins to their GPIO designations. To ensure smooth operation during repeated code executions, warnings are disabled using the **GPIO.setwarnings(False)** function. Once the GPIO pin is configured as an output, the LED is turned ON by sending a HIGH signal to pin 14. A message stating "LED is ON" is printed in the terminal for user confirmation. After a delay of **2** seconds using **time.sleep(2),** the LED is turned OFF by setting the pin to LOW, followed by the message "LED is OFF."

2.LED controlling with a push button switch:

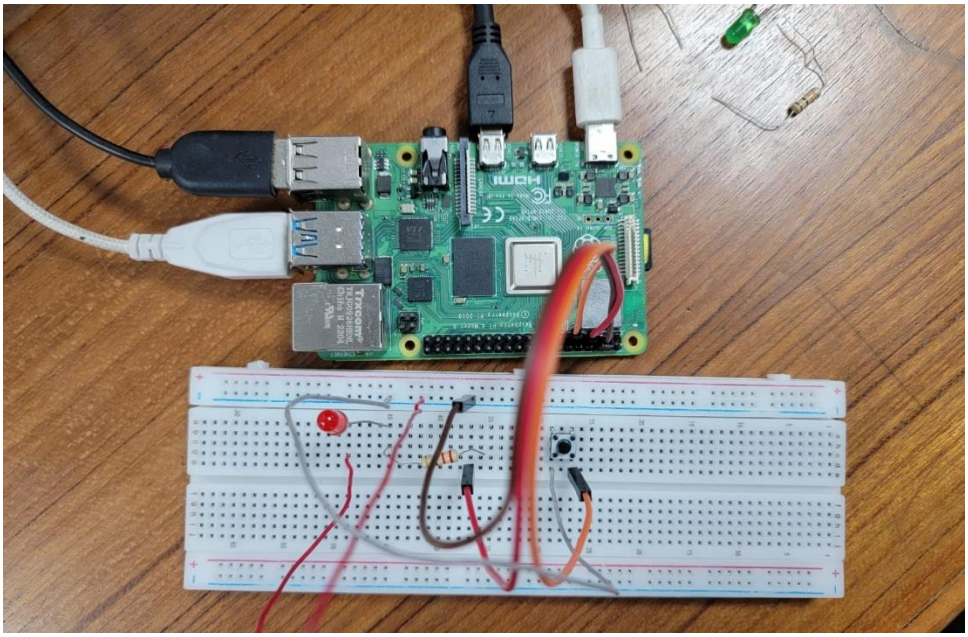

Figure-06: push button LED activation (Button not pressed LED OFF)

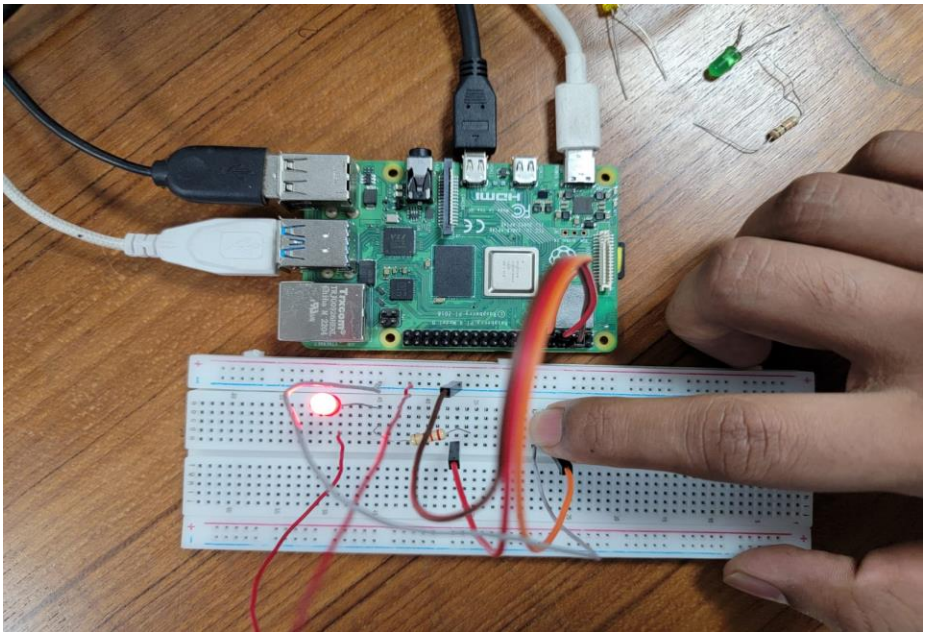

Figure-07: push button LED activation (Button pressed LED ON)

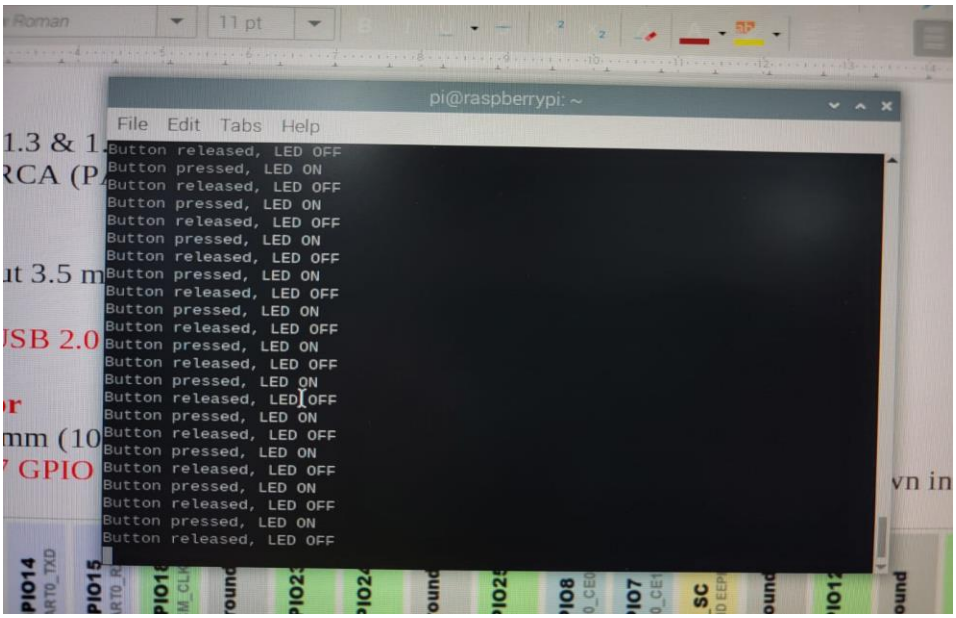

Figure-08: push button LED activation (Display comment on terminal)

Explanation: In this setup, a push-button switch is used in conjunction with an LED and a Raspberry Pi to demonstrate simple digital input and output interaction using the gpiozero library. The LED is connected to GPIO **pin 4** and controlled through the LED class, while the push-button is attached to GPIO **pin 17** and managed using the Button class. Both components are initialized in the code and stored in respective variables led and button. The system runs an infinite loop where the Raspberry Pi waits for the button to be pressed using the wait_for_press() method. When the button is pressed, the LED is turned on immediately by calling led.on(). The system then waits until the button is released using wait_for_release(), after which the LED is turned off by invoking led.off(). This cycle continues indefinitely. During testing, each time the button is pressed, the LED lights up, and it switches off upon release.

3.Simple Traffic Control System:

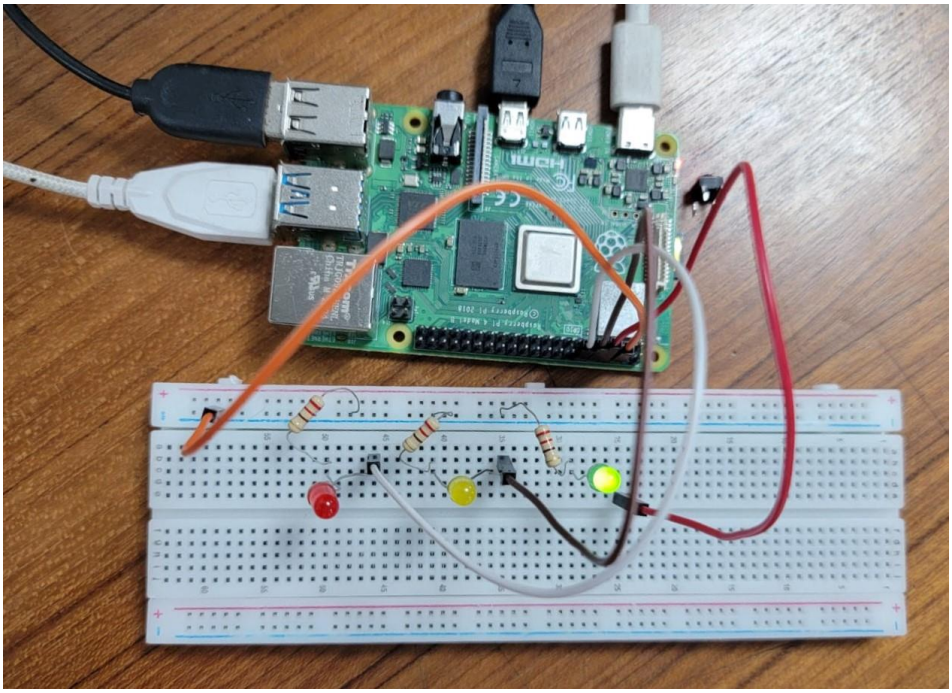

Figure-09:Traffic Control system(When Green LED is on)

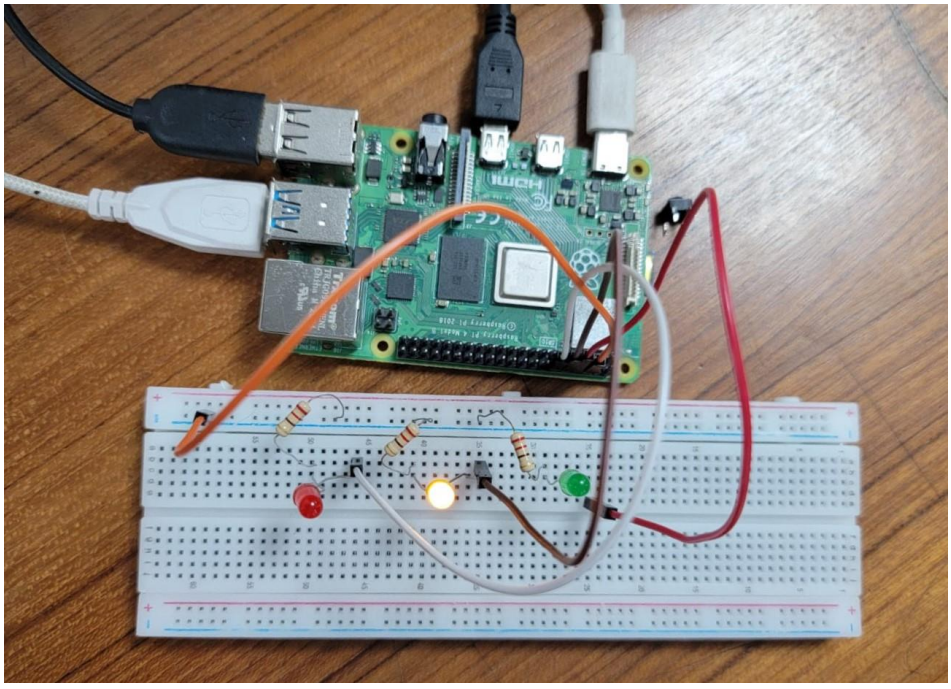

Figure-10: Traffic Control system(When Yellow LED is on)

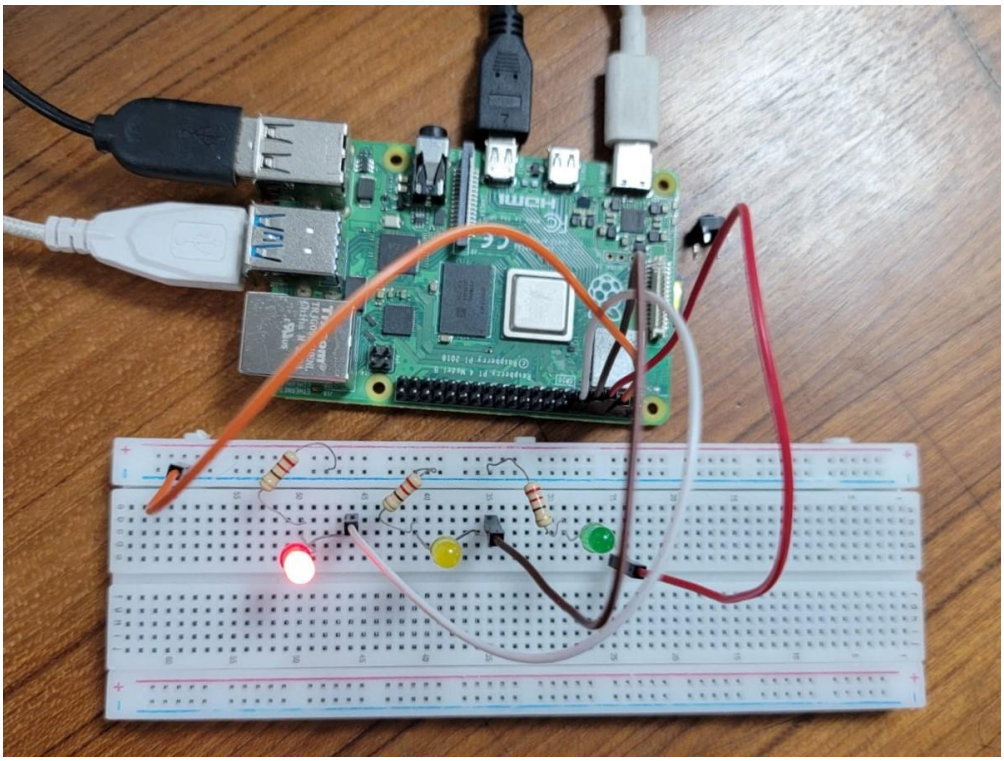

Figure-11: Traffic Control system(When RED LED is on)

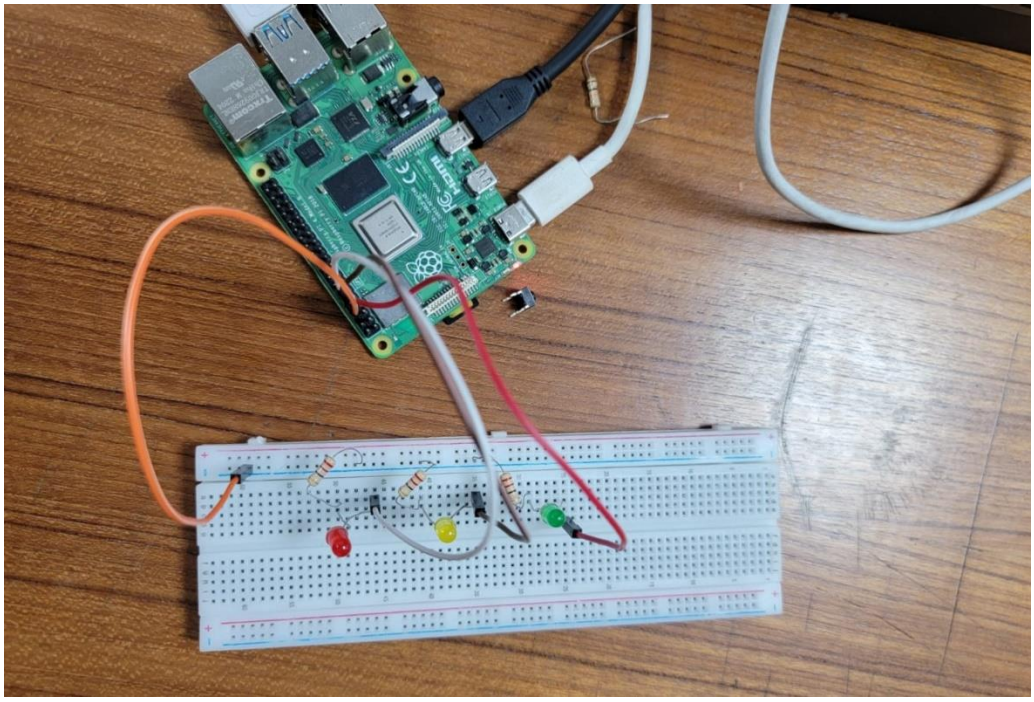

Figure-12: Traffic Control system(When LEDs are off)

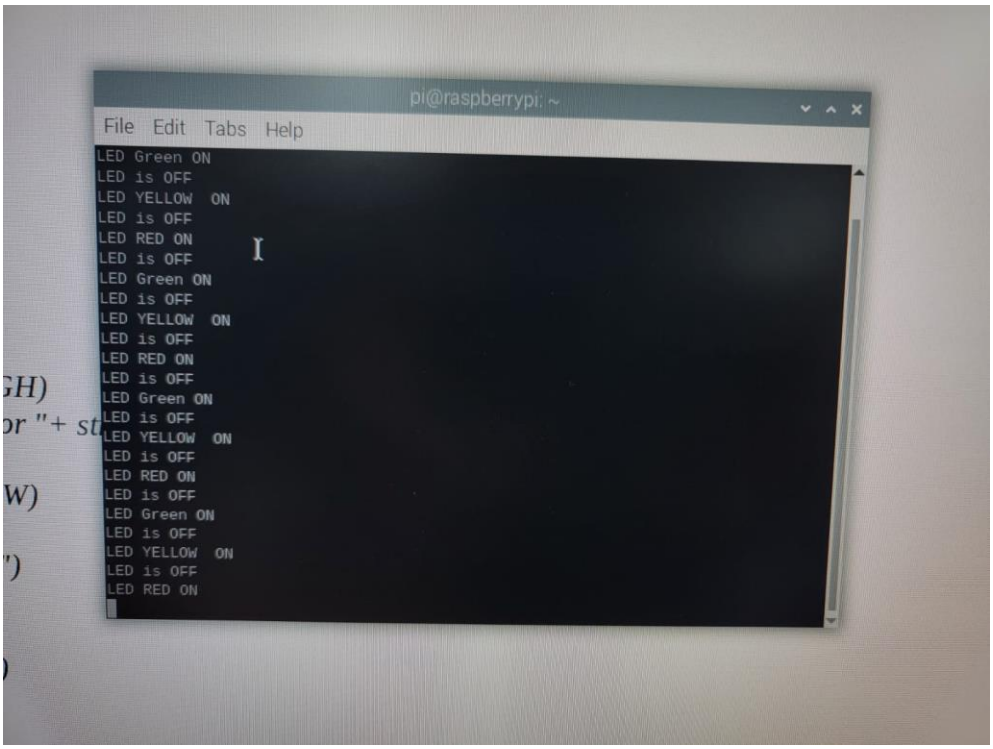

Figure-13: Traffic Control system(Display comment on terminal)

Explanation: In this setup, a simple traffic light control system is implemented using a Raspberry Pi and three LEDs representing the standard traffic signal colors: green, yellow, and red. The green LED is connected to **GPIO pin 14**, the yellow LED to **GPIO pin 15**, and the red LED to **GPIO pin 18**. These pins are set as output using the **GPIO.setup()** function. The Raspberry Pi uses the BCM (Broadcom) pin numbering system, and warnings are disabled using **GPIO.setwarnings(False)** to avoid unnecessary alerts during repeated executions.The main logic is placed inside an infinite while loop, allowing the traffic light sequence to run continuously. First, the green LED turns on for 1 second to indicate that vehicles may proceed, after which it turns off. Next, the yellow LED blinks three times, each blink lasting 0.5 seconds on and 0.5 seconds off, representing the caution signal before stopping. Finally, the red LED lights up for 1 second to signal vehicles to stop. After this, the cycle repeats indefinitely. When observed during on real hardware, the LEDs operate sequentially in a recognizable traffic light pattern. The green LED stays on steadily, the yellow LED blinks to indicate transition, and the red LED lights up solidly. This simulation effectively demonstrates a basic traffic signal using digital output pins on the Raspberry Pi and can serve as a foundation for more advanced traffic control systems involving sensors or timers.

Simulation Output Results:

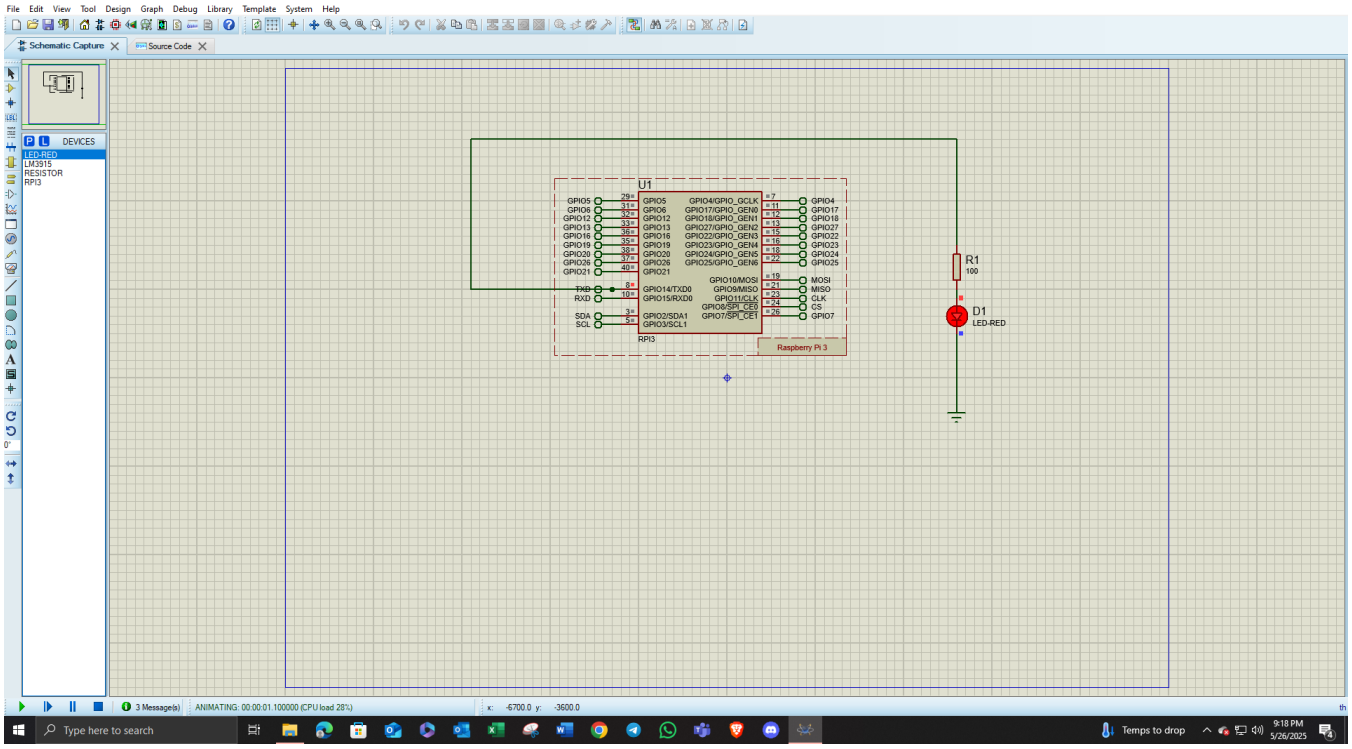

Figure-14: Simulation for LED blink test(When LED is ON)

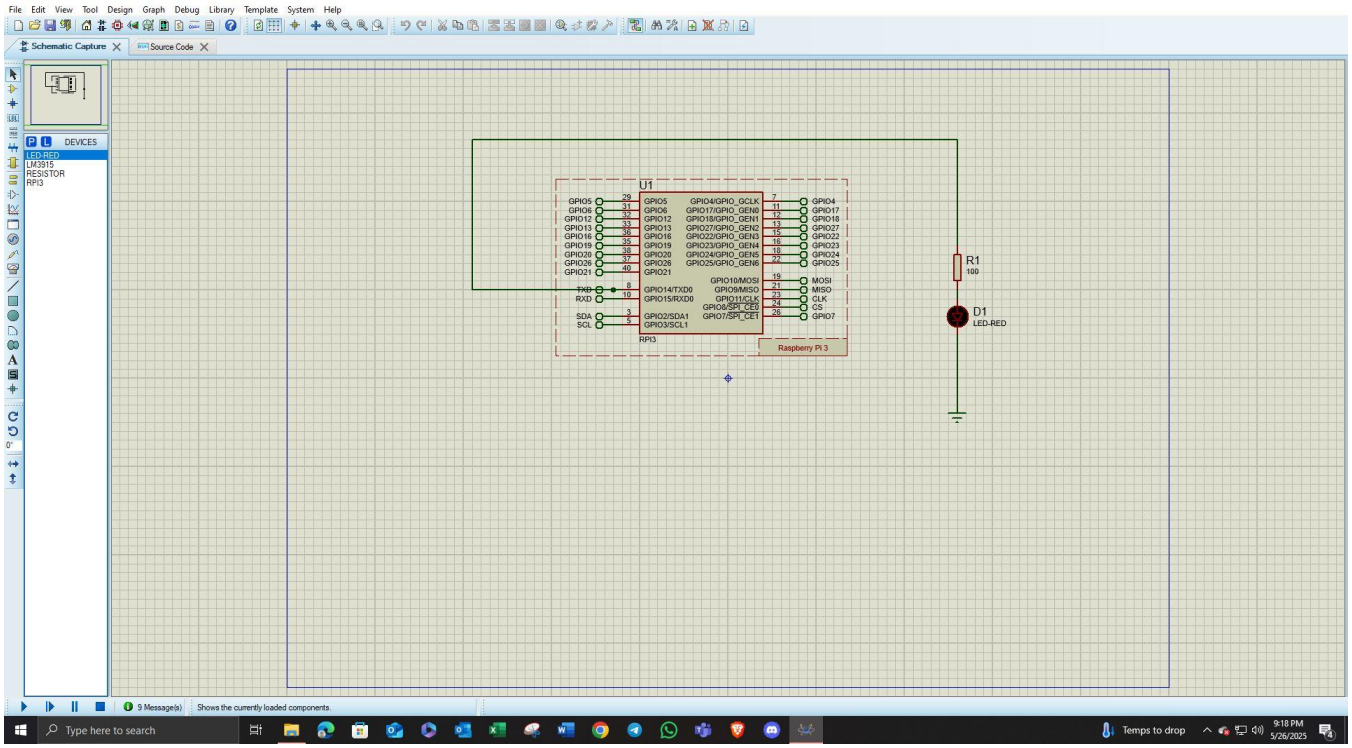

Figure-15: Simulation for LED blink test(When LED is OFF)

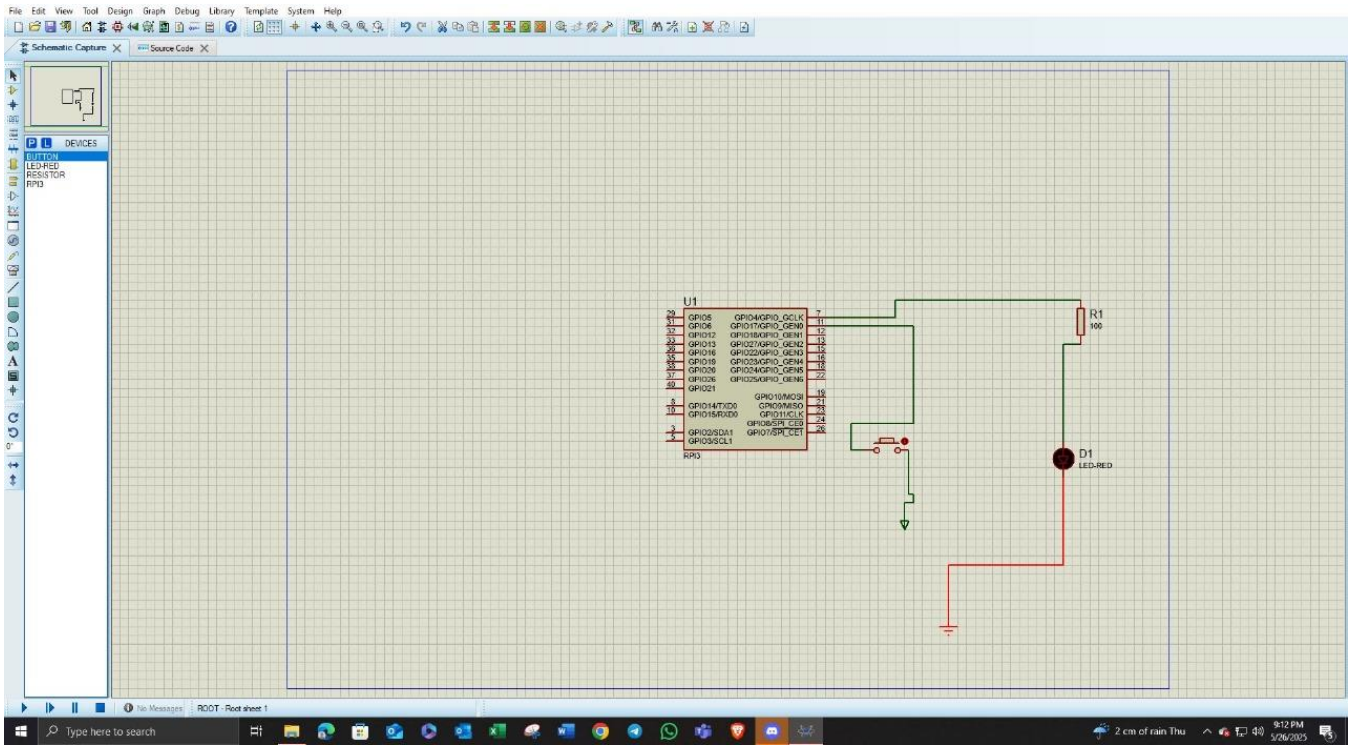

Figure-16:Simulation for push button LED activation (Button not pressed LED off)

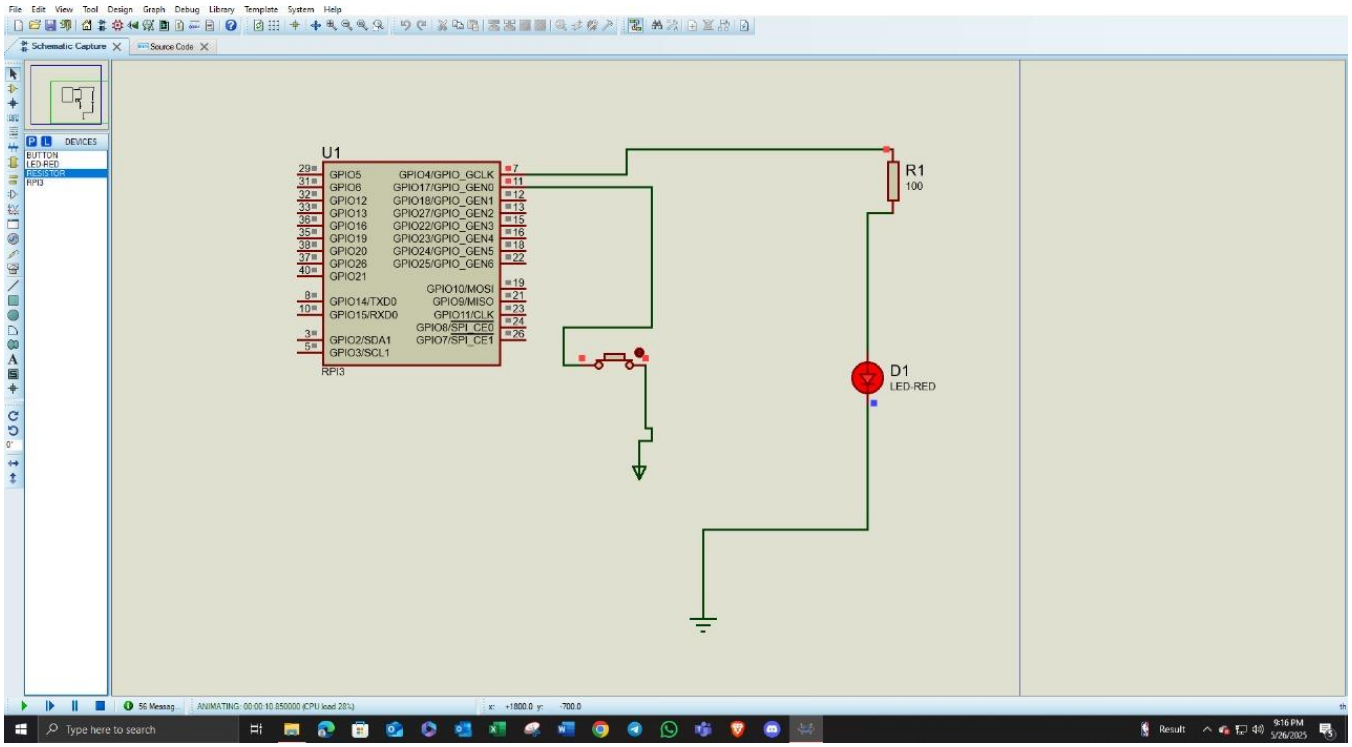

Figure-17:Simulation for push button LED activation (Button pressed LED ON)

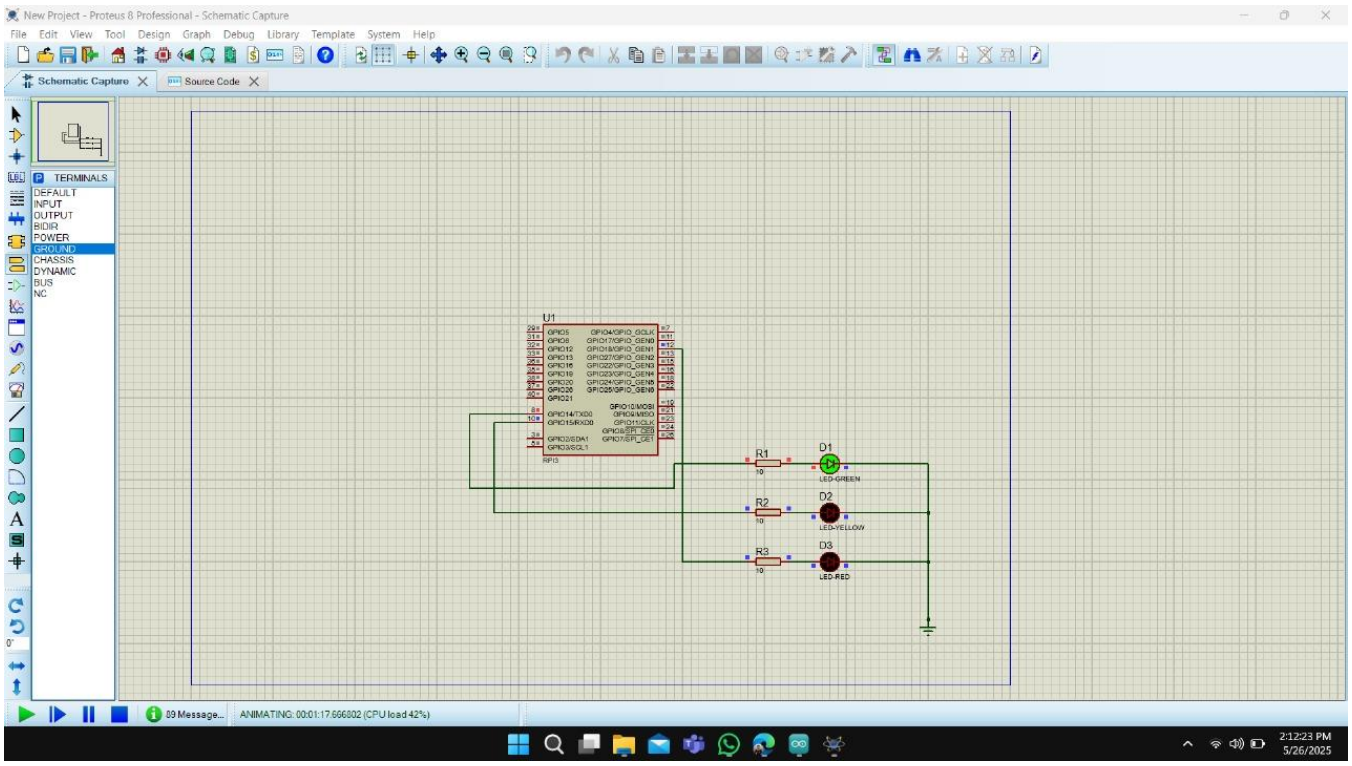

Figure-18:Simulation for Traffic Control system(When Green LED is on)

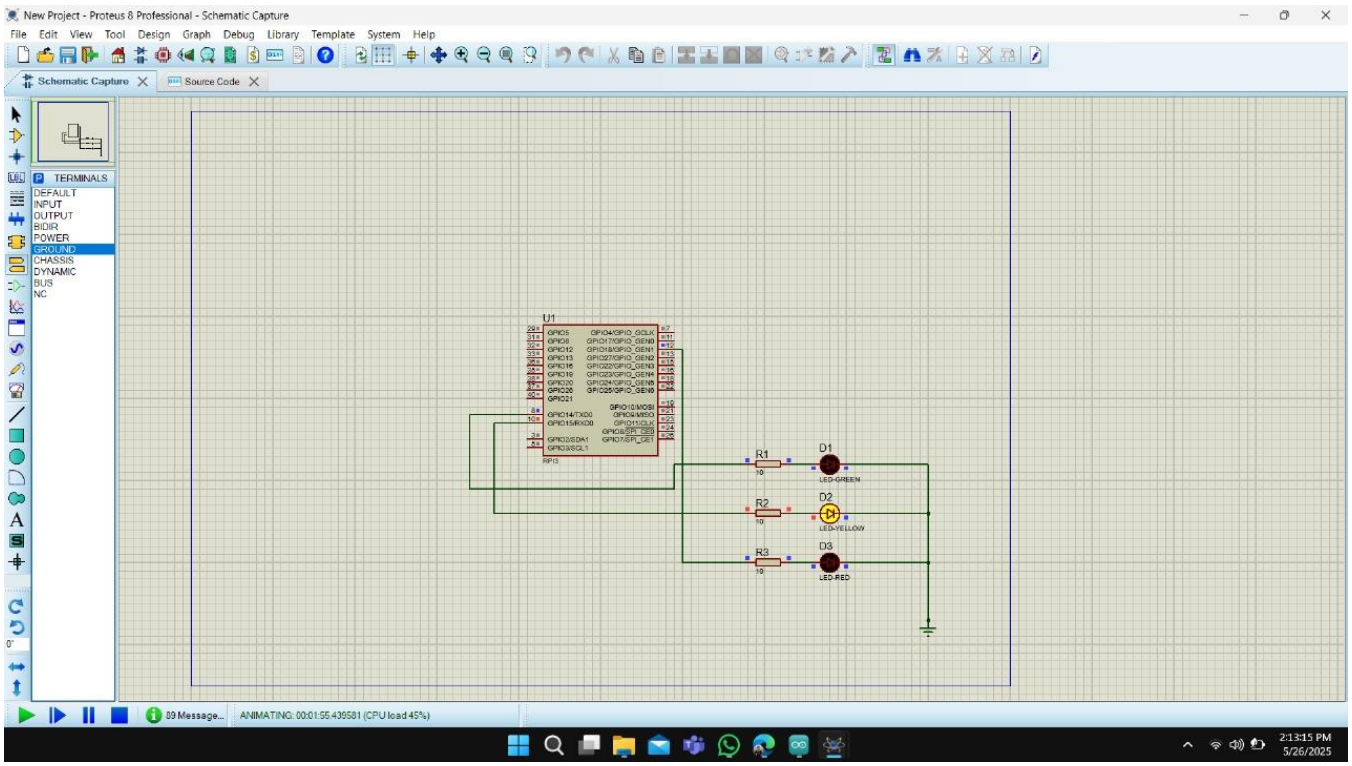

Figure-19:Simulation for Traffic Control system(When Yellow LED is on)

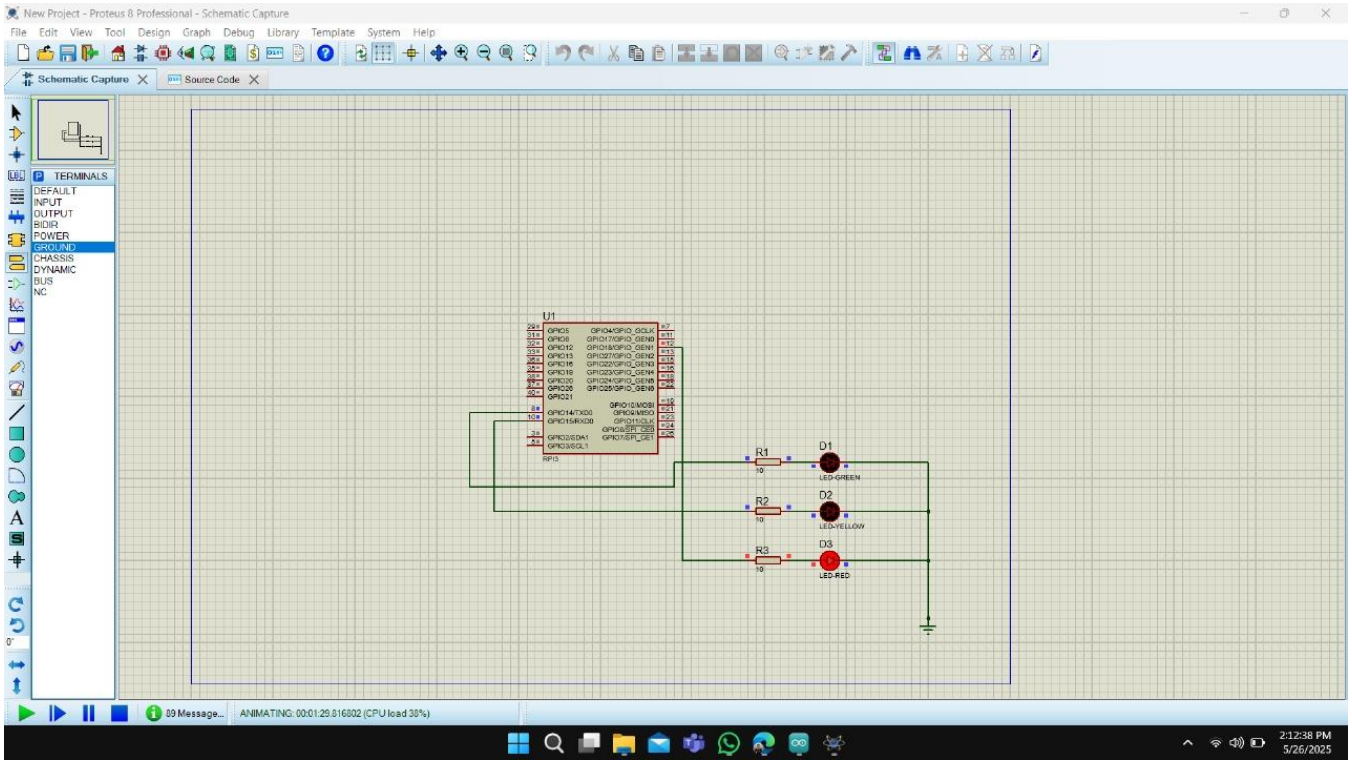

Figure-20:Simulation for Traffic Control system(When Red LED is on)

Explanation: In this simulation, a Raspberry Pi board was used to perform three tasks: LED blinking, push-button LED control, and a traffic light control system. Simulations were done using proteus, with Python code written using the gpiozero or RPi.GPIO libraries to control GPIO pins.For the **LED Blink Test**, an LED was connected to a GPIO output pin through a resistor. The program blinked the LED ON and OFF every second using time delays. This confirmed the correct functioning of GPIO output and timing control.In the **Push-Button LED Control** simulation, a push button was connected to a GPIO input pin and an LED to an output pin. When the button was pressed, the LED turned ON; when released, the LED turned OFF. This demonstrated real-time digital input handling and output response.For the **Traffic Light Control System**, three LEDs (green,yellow,red) were connected to separate GPIO pins. The LEDs were activated in a timed sequence to simulate traffic signals. Each LED stayed ON for a specific duration before the next turned ON, repeating in a continuous loop.Overall, these simulations validated the Raspberry Pi's GPIO control capabilities for automation and embedded system applications.

Discussion:

This experiment focuses on familiarizing oneself with the Raspberry Pi, exploring its capabilities and understanding how it can be used for various embedded system applications. The primary objective was to understand how to set up and interact with the Raspberry Pi, learning to use its General Purpose Input/Output (GPIO) pins, operating system (Raspberry Pi OS), and various software tools for development.

In this setup, we started by installing the Raspberry Pi OS and configuring the system for basic operations. The GPIO pins were then used to control simple devices like LEDs, buttons, and sensors, showcasing the versatility of the Raspberry Pi in interfacing with the physical world. Python programming was utilized to write simple scripts to control these devices, taking advantage of the Raspberry Pi's ease of use for scripting and automation. The experiment also highlighted the Raspberry Pi's capability to run a full Linux-based OS, allowing it to handle a wide variety of tasks, such as managing inputs/outputs, processing data, and controlling connected devices. Additionally, we explored the use of external modules like cameras and sensors, which can be easily interfaced with the Raspberry Pi, enabling it to be used for more complex projects, such as home automation, robotics, and IoT applications.

This hands-on experience provided a foundational understanding of the Raspberry Pi's functionality and potential applications. It demonstrated how this low-cost, compact platform can be used to create diverse, real-world solutions. The Raspberry Pi's flexibility, coupled with its large community and extensive libraries, makes it an excellent tool for prototyping and experimentation in the fields of embedded systems and automation.

Here are some real-life scenarios and applications of this experiment:

1.Smart Home Systems: The Raspberry Pi can control various smart devices like lights, thermostats, and security cameras, creating an intelligent and automated home environment.
Example: A Raspberry Pi-based system could adjust home lighting and temperature based on user preferences and environmental conditions.
2.IoT Applications: The Raspberry Pi can be used as an IoT gateway, collecting data from various sensors and sending it to the cloud for analysis or remote monitoring.
Example: A weather station that uses a Raspberry Pi to collect temperature, humidity, and pressure data, then sends it to a remote server for tracking and analysis.
3.Robotics: The Raspberry Pi can be used as the central controller for robotic systems, processing sensor data and controlling motors to enable autonomous movement and decision-making.
Example: A Raspberry Pi-powered robot could navigate a maze, making decisions based on sensor inputs and adjusting motor speeds using PWM.
4.Media Centers: With its ability to run media software like Kodi, the Raspberry Pi can be turned into a cost-effective media center for streaming videos, music, and other content.
Example: A Raspberry Pi could be set up as a home theater system, streaming media from a local server or the internet.

5.Security Systems: Raspberry Pi can be used to build affordable home security systems, integrating cameras, motion sensors, and alarms.
Example: A Raspberry Pi-based security system can monitor a home's perimeter, detect motion, and send alerts or activate alarms if suspicious activity is detected.
6.Personal Cloud Storage: Raspberry Pi can be transformed into a personal cloud server, enabling access to files and media from anywhere.
Example: Using software like Next cloud or ownCloud, a Raspberry Pi can serve as a private cloud storage system, allowing users to store, access, and share files securely.
7.Weather Stations: Raspberry Pi can collect data from various sensors (temperature, humidity, wind speed) to create a personal weather station.
Example: A Raspberry Pi-powered weather station can monitor local weather conditions and display them on a web interface or share data with weather forecasting models.

Conclusion:

In conclusion, this experiment provided a comprehensive introduction to the Raspberry Pi, highlighting its versatility as a low-cost, powerful platform for various embedded system applications. By exploring the use of GPIO pins, software tools, and the Raspberry Pi OS, we gained hands-on experience in interfacing external components, running scripts, and creating automated systems. The Raspberry Pi's ability to handle a wide range of tasks, from basic control of sensors and motors to more complex applications like home automation, robotics, and environmental monitoring, makes it an invaluable tool for both hobbyists and professionals. The experiment emphasized the Raspberry Pi's potential in real-world applications, demonstrating how it can be used to create cost-effective, scalable solutions for diverse needs. Overall, this exercise reinforced the importance of the Raspberry Pi in modern embedded systems, offering a gateway for exploring innovative and practical projects across various industries.

References:

[1] https://www.arduino.cc/.
[2] https://www.educba.com
[3] https://www.researchgate.net/publication/
[4] https://www.geeksforgeeks.org