A Mini Project Report
On
SMART COLLAR SYSTEM

Submitted By
TEAM-08

Mr. MEENUGU HANI SATWIK	**22B91A4731**
Mr. SUREDDY B D S N RISHI	**23B95A4712**
Mr. ARJILA KASI	**22B91A4704**
Mr. ANDEY LOKESH NAIDU	**23B95A4701**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

S. R. K. R ENGINEERING COLLEGE (A)

(Affiliated to JNTU, KAKINADA)

BHIMAVARAM-534204

(2024-2025)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

S. R. K. R ENGINEERING COLLEGE BHIMAVARAM

CERTIFICATE

This is to certify that this is a bonafide work on "**SMART COLLAR SYSTEM**" for and has been submitted by Mr. MEENUGU HANI SATWIK (**22B91A4731**), Mr. SUREDDY RISHI (**23B95A4712**), Mr.ARJILA KASI (**22B91A4704**), Mr.ANDEY LOKESH (**23B954A701**), as a **IOT** laboratory report, in partial fulfilment of the requirements for theaward of the Degree of Bachelor of Technology in Computer Science and Engineering, during the academic year 2024-2024. The candidate worked right under my Supervision and guidance.

Lecturers In-Charge

Mr.V.S.R.K RAJU

Ms. Neelima

(Assistant professor)

Department of CSE

Head of the Department

Dr. B.H.V.S.R.RAJU

(PROFESSOR**)**

TABLE OF CONTENTS

LAB PROGRAMS

Program - I

Start Raspberry Pi and try various Linux commands in the command terminal window: ls, cd, touch, mv, rm, man, mkdir, rmdir, tar, gzip, cat, more, less, ps, sudo, cron, chown, chgrp, ping, etc

1. ls

Command: ls

Description: Lists files and directories in the current directory.

- ✓ ls -l: Long listing format (with permissions, size, etc.)
- ✓ ls -a: Shows hidden files too

---

2. cd

Command: cd <directory>

Description: Changes the current directory.

- ✓ cd ..: Goes back one directory
- ✓ cd ~: Goes to home directory

---

3. touch

Command: touch <filename>

Description: Creates a new empty file.

- ✓ Example: touch file1.txt

---

4. mv

Command: mv <source> <destination>

Description: Moves or renames a file or directory.

- ✓ Example: mv file1.txt folder/
- ✓ Rename: mv old.txt new.txt

5. rm

Command: rm <filename>

Description: Removes (deletes) a file.

- ✓ rm -r foldername/: Removes directory and contents
- ✓ rm -i file.txt: Prompts before deleting

---

6. man

Command: man <command>

Description: Opens the manual/helps with a command.

- ✓ Example: man ls

---

7. mkdir

Command: mkdir <directory-name>

Description: Creates a new directory (folder).

- ✓ Example: mkdir new_folder

---

8. rmdir

Command: rmdir <directory-name>

Description: Removes an empty directory.

- ✓ Example: rmdir new_folder

---

9. tar

Command: tar -cvf archive.tar file1 file2

Description: Archives files into .tar format.

- ✓ Extract: tar -xvf archive.tar

10. gzip

Command: gzip <filename>

Description: Compresses a file into .gz format.

- ✓ Decompress: gunzip file.gz

---

11. cat

Command: cat <filename>

Description: Displays the content of a file.

- ✓ Example: cat file.txt

---

12. more

Command: more <filename>

Description: Views file content one screen at a time (scroll forward only).

- ✓ Example: cat file.txt

---

13. less

Command: less <filename>

Description: Similar to more, but allows scrolling up and down.

- ✓ Example: cat file.txt

---

14. ps

Command: ps

Description: Lists running processes.

- ✓ ps aux: Shows all running processes with details

---

15. sudo

Command: sudo <command>

Description: Runs a command as a superuser (admin privileges).

- ✓ Example: sudo apt update

---

16. cron

Command: crontab -e

Description: Edits cron jobs (scheduled tasks).

- ✓ crontab -l : Lists current user’s cron jobs

---

17. chown

Command: chown <user>:<group> <filename>

Description: Changes ownership of a file or directory.

- ✓ Example: sudo chown pi:pi file.txt

---

18. chgrp

Command: chgrp <group> <filename>

Description: Changes the group ownership of a file.

Example: chgrp developers project.txt

---

19. ping

Command: ping <hostname or IP>

Description: Checks network connectivity.

- ✓ Example: ping google.com

Program-II

Run some Python programs:

1. **Read your name and print the hello message with your name**

   Program:

```
name = input("Enter your name: ")
print("Hello", name)
```

   Output:

```
Enter your name: Avinash
Hello Avinash
```

2. **Read two numbers and print their sum, difference, product and division.**

   Program:

```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("Sum:", num1 + num2)
print("Difference:", num1 - num2)
print("Product:", num1 * num2)
if num2 != 0:
    print("Division:", num1 / num2)
else:
    print("Cannot divide by zero")
```

   Output:

```
Enter first number: 10
Enter second number: 5
Sum: 15
Difference: 5
Product: 50
Division: 2.0
```

3. **Word and character count of a given string**

Program:

```
text = input("Enter a string: ")
print("Words:", len(text.split()))
print("Characters:", len(text))
```

Output:

```
Enter a string: Hello world from CRX
Words: 4
Characters: 20
```

4. **Area of a given shape (rectangle, triangle, and circle), reading the shape and the appropriate values form standard input**

Program:

```
shape = input("Enter shape (rectangle, triangle, circle): ")
if shape == "rectangle":
    l = float(input("Enter length: "))
    b = float(input("Enter breadth: "))
    print("Area:", l * b)
elif shape == "triangle":
    b = float(input("Enter base: "))
    h = float(input("Enter height: "))
    print("Area:", 0.5 * b * h)
elif shape == "circle":
    r = float(input("Enter radius: "))
    print("Area:", 3.14 * r * r)
else:
    print("Unknown shape")
```

Output:

```
Enter shape (rectangle, triangle, circle): circle
Enter radius: 5
Area: 78.5
```

5. **Print a name 'n' times, where name and n are read from standard input, using for and while loops**

Program:

```
name = input("Enter a name: ")
n = int(input("How many times? "))
print("Using for loop:")
for i in range(n):
    print(name)
print("Using while loop:")
i = 0
while i < n:
    print(name)
    i += 1
```

Output:

```
Enter a name: Gojo
How many times? 3
Using for loop:
Gojo
Gojo
Gojo
Using while loop:
Gojo
Gojo
Gojo
```

6. **Handle Divided by Zero Exception.**

Program:

```
try:
    x = int(input("Enter numerator: "))
    y = int(input("Enter denominator: "))
    print("Result:", x / y)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Output:

```
Enter numerator: 10
Enter denominator: 0
You can't divide by zero!
```

7. **Print the current time 10 times with an interval of 10 seconds.**

Program:

```
import time
for i in range(10):
    print("Time now:", time.ctime())
    time.sleep(10)
```

Output:

```
Time now: Sun Apr 21 18:24:00 2025
Time now: Sun Apr 21 18:24:10 2025
...
(repeats 10 times, spaced 10 seconds apart)
```

8. Read a file line by line and print the word count of each line.

Program:

```
filename = input("Enter filename: ")
with open(filename, "r") as f:
    for line in f:
        print("Words in line:", len(line.split()))
```

Output:

```
Enter filename: sample.txt
Hello world
This is Itachi
Gojo rocks       (sample.txt)
Words in line: 2
Words in line: 3
Words in line: 2
```

Program-III

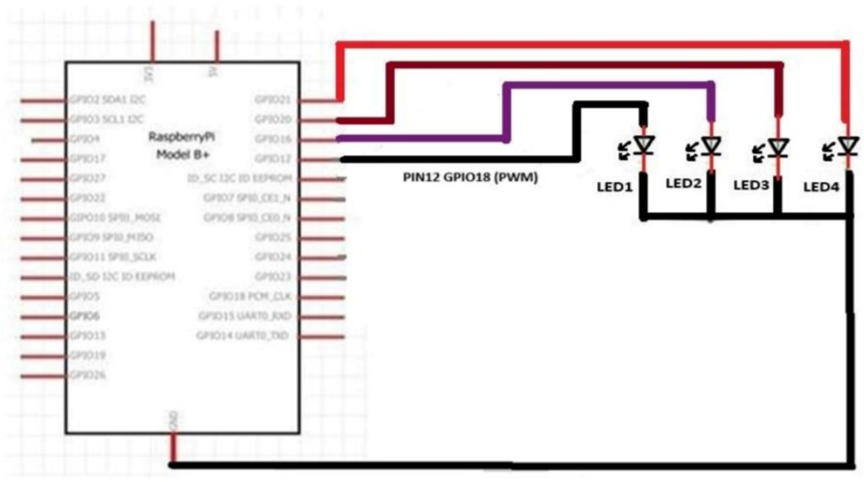
AIM:

Light an LED through a Python program.

Description:

This program blinks an LED connected to GPIO pin 26 of a Raspberry Pi. It turns the LED ON and OFF every second using the RPi.GPIO library. The LED blinks in a loop until the user stops the program, after which the GPIO is cleaned up.

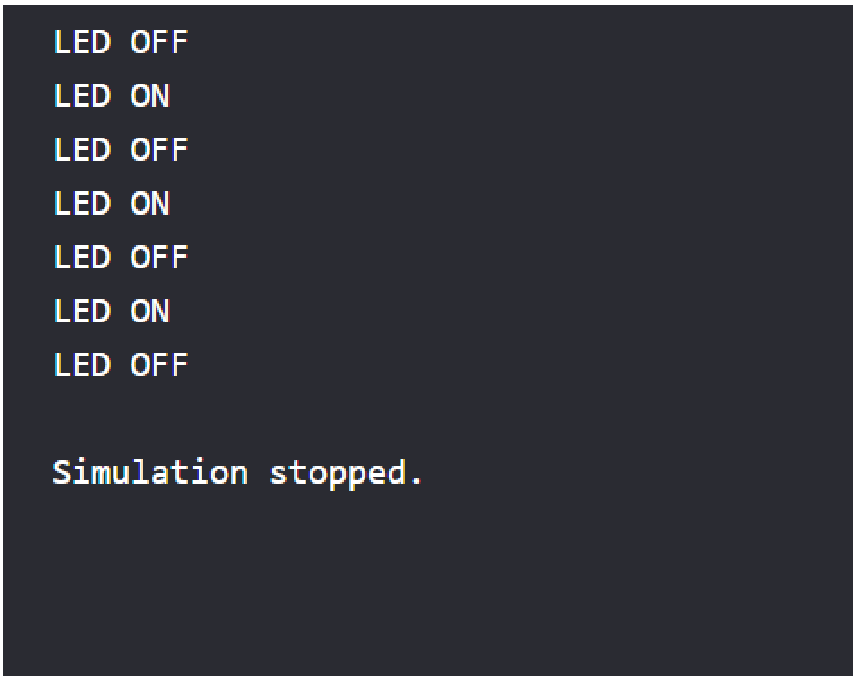
Circuit Diagram:



Program:

```
import time
import RPi.GPIO as gpio
gpio.setwarnings(False)
gpio.setmode(gpio.BOARD)
gpio.setup(26, gpio.OUT)
try:
    while(1):
        gpio.output(26,0)
        print("LED OFF")
        time.sleep(1)
        gpio.output(26,1)
        print("LED ON")
        time.sleep(1)
       #gpio.cleanup()
    except KeyboardInterrupt:
        gpio.cleanup()
        exit
```

Output:



```
LED OFF
LED ON
LED OFF
LED ON
LED OFF
LED ON
LED OFF

Simulation stopped.
```

Program-IV

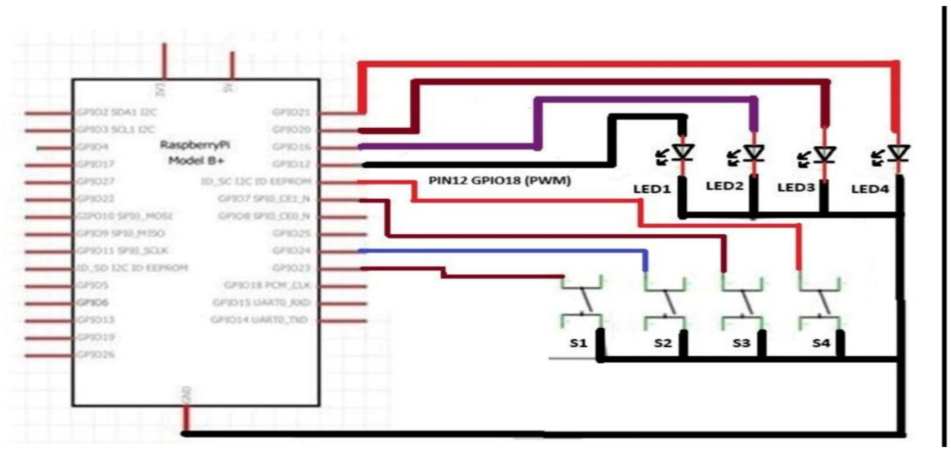
AIM:

Get input from two switches and switch on the corresponding LEDs.

Description:

This Python program demonstrates how to control two LEDs using two switches connected to a Raspberry Pi. It sets up GPIO pins 26 and 24 as outputs for the LEDs, and pins 40 and 38 as inputs for the switches. When a switch is pressed, the corresponding LED lights up briefly. The program uses interrupts to detect the switch presses in real-time, without continuously checking the input state. This approach is efficient and ensures a quick response to user interaction. The GPIO pins are properly cleaned up when the program is interrupted, preventing any pin conflicts during future use.

Circuit Diagram:

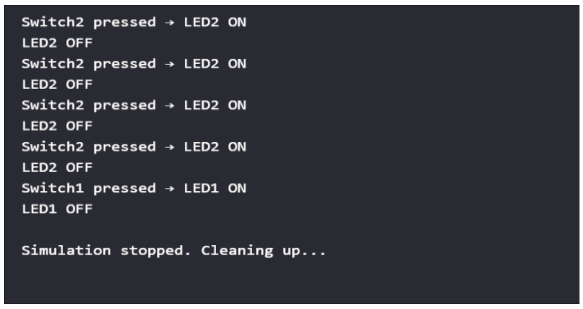


Program:

```
import time
import RPi.GPIO as gpio
gpio.setwarnings(False) gpio.setmode(gpio.BOARD)
led2 = 24      # pin is connected to LED and it should be OUT
led1 = 26       # pin is connected to LED and it should be OUT
switch2 = 38   # pin is connected to SWITCH and it should be IN
switch1= 40     # pin is connected to SWITCH and it should be IN
gpio.setup(led1,gpio.OUT,initial=0)
gpio.setup(led2,gpio.OUT,initial=0)
gpio.setup(switch1,gpio.IN)
gpio.setup(switch2,gpio.IN)
def glow_led(event):
    if event == switch1 :
        gpio.output(led1, True)
        time.sleep(0.2)
        gpio.output(led1, False)

    elif event == switch2 :
         gpio.output(led2, True)
         time.sleep(0.2)
         gpio.output(led2, False)
gpio.add_event_detect(switch1, gpio.RISING , callback = glow_led, bouncetime = 1)
gpio.add_event_detect(switch2, gpio.RISING , callback = glow_led, bouncetime = 1)
try:
    while(True):
         #to avoid 100% CPU usage
         time.sleep(1)
except KeyboardInterrupt:
    gpio.cleanup()
```

Output:



```
Switch2 pressed → LED2 ON
LED2 OFF
Switch2 pressed → LED2 ON
LED2 OFF
Switch2 pressed → LED2 ON
LED2 OFF
Switch2 pressed → LED2 ON
LED2 OFF
Switch1 pressed → LED1 ON
LED1 OFF

Simulation stopped. Cleaning up...
```

Program-V

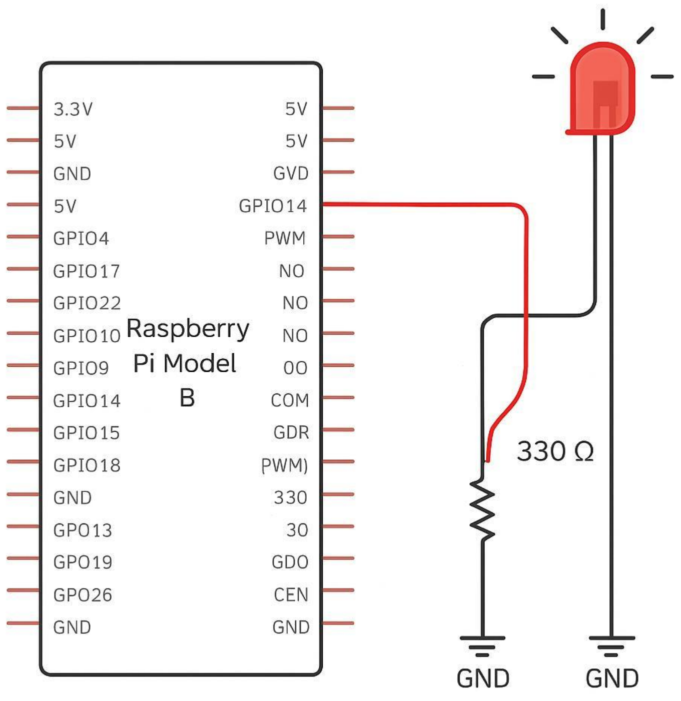
AIM:

Flash an LED at a given on time and off time cycle, where the two times are taken from a file.

Description:

This Python program controls an LED's blinking pattern on a Raspberry Pi based on on-time and off-time values read from an external text file (led_times.txt). GPIO pin 26 is used to control the LED. The program reads the file's key-value pairs (e.g., on=1.0, off=0.5) to blink the LED accordingly. It updates the pattern by changing the file values, handles file reading errors, and cleans up GPIO on interruption.

Circuit Diagram:



Program:

import time

# Setup gpio.setwarnings(False) gpio.setmode(gpio.BOARD)

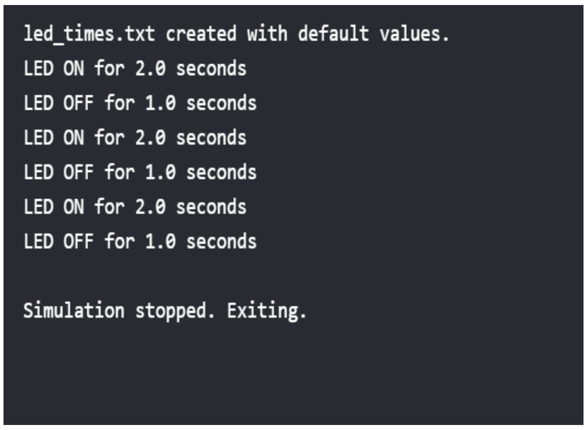
led1 = 26 # Using same pin as before gpio.setup(led1, gpio.OUT, initial=0)

# Function to read on/off times from a file def read_times(led_times):

try:

```
    with open(led_times, 'r') as f:
        for line in f:
            key, value = line.strip().split('=')
            if key == 'on':
               on_time = float(value)
            elif key == 'off':
              off_time = float(value)
    except Exception as e: print(f"Error reading file: {e}")
        return on_time, off_time
# Main flashing loop
try:
    while True:
        on, off = read_times('led_times.txt')
        gpio.output(led1, True)
        time.sleep(on)
        gpio.output(led1, False)
        time.sleep(off)
 except KeyboardInterrupt:
    gpio.cleanup()
    print("\nGPIO cleaned up. Exiting.")
```

Output:



```
led_times.txt created with default values.
LED ON for 2.0 seconds
LED OFF for 1.0 seconds
LED ON for 2.0 seconds
LED OFF for 1.0 seconds
LED ON for 2.0 seconds
LED OFF for 1.0 seconds

Simulation stopped. Exiting.
```

Program-VI

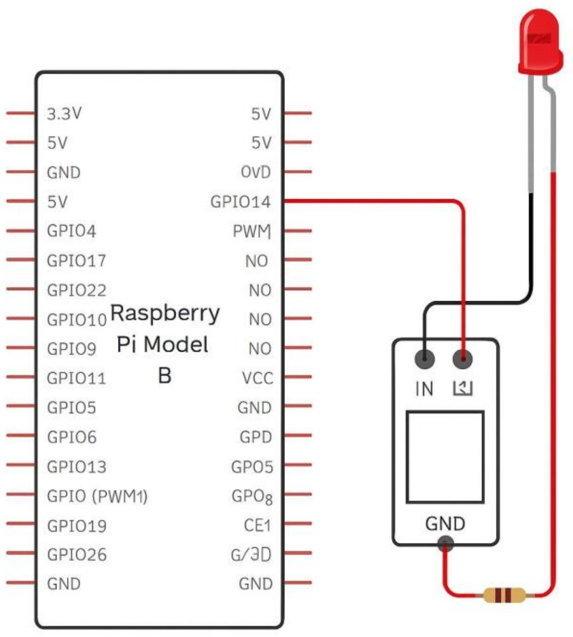
AIM:

Flash an LED based on cron output (acts as an alarm).

Description:

This Python program controls an LED (or relay) using cron jobs, acting as a simple alarm system. It takes a command-line argument (on or off) to control the LED on GPIO pin 26 of the Raspberry Pi. When scheduled with cron, it can automatically switch the LED on or off at specified times. The program validates the argument, configures the GPIO pin, and disables warnings before executing the command.

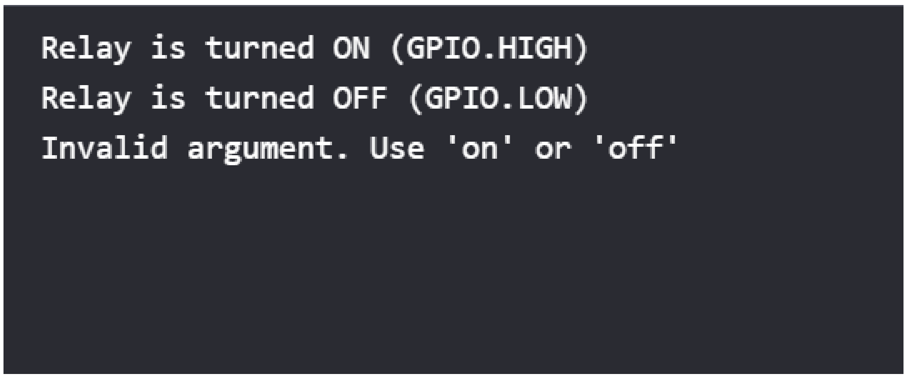
Circuit Diagram:



Program:

```
import RPi.GPIO as GPIO import sys
RELAY_PIN = 26 # BOARD pin number
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
 GPIO.setup(RELAY_PIN, GPIO.OUT)
if len(sys.argv) != 2:
    print("Usage: python3 relay_control.py [on|off]")
    sys.exit(1)
command = sys.argv[1].lower()
if command == "on":
    GPIO.output(RELAY_PIN, GPIO.HIGH) # Relay ON
elif command == "off":
    GPIO.output(RELAY_PIN, GPIO.LOW) # Relay OFF
else:
     print("Invalid argument. Use 'on' or 'off'")
     sys.exit(1)
```

Output:



```
Relay is turned ON (GPIO.HIGH)
Relay is turned OFF (GPIO.LOW)
Invalid argument. Use 'on' or 'off'
```

Program-VII

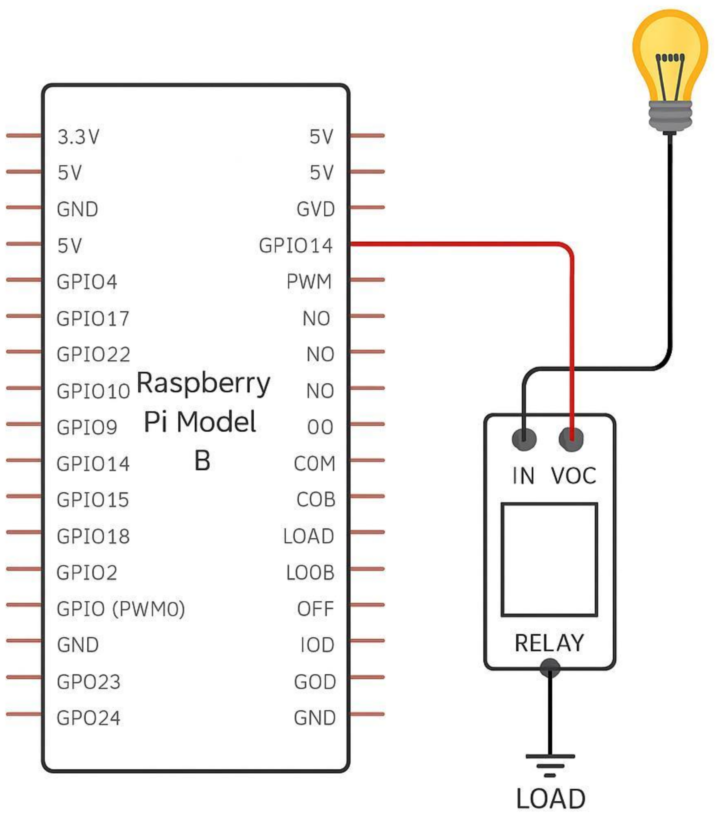
AIM:

Switch on a relay at a given time using cron, where the relay's contact terminals are connected to a load.

Description:

This Python program controls a relay connected to GPIO pin 26 on a Raspberry Pi, allowing automation of electrical loads. It accepts a command-line argument (on or off) to activate or deactivate the relay. The program verifies the input and sets the GPIO output accordingly. By scheduling the script with cron, the relay can be controlled at specific times, enabling home or industrial automation.

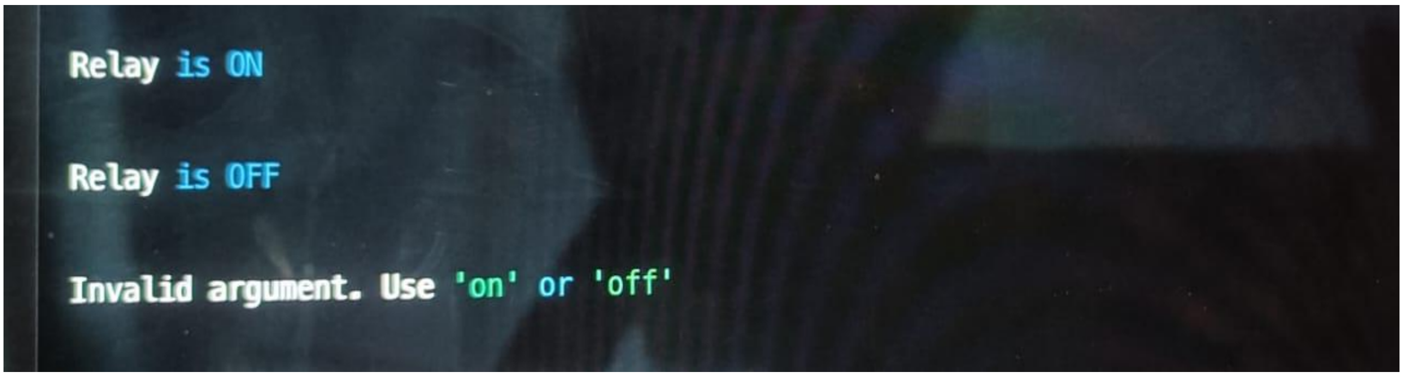
Circuit Diagram:



Program:

```
import RPi.GPIO as GPIO import sys
RELAY_PIN = 26 # BOARD pin number
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(RELAY_PIN, GPIO.OUT)
if len(sys.argv) != 2:
    print("Usage: python3 relay_control.py [on|off]")
    sys.exit(1)
command = sys.argv[1].lower()
if command == "on":
    GPIO.output(RELAY_PIN, GPIO.HIGH) # Relay ON
elif command == "off":
     GPIO.output(RELAY_PIN, GPIO.LOW) # Relay OFF
else:
     print("Invalid argument. Use 'on' or 'off'")
sys.exit(1)
```

Output:



```
Relay is ON

Relay is OFF

Invalid argument. Use 'on' or 'off'
```

Program-VIII

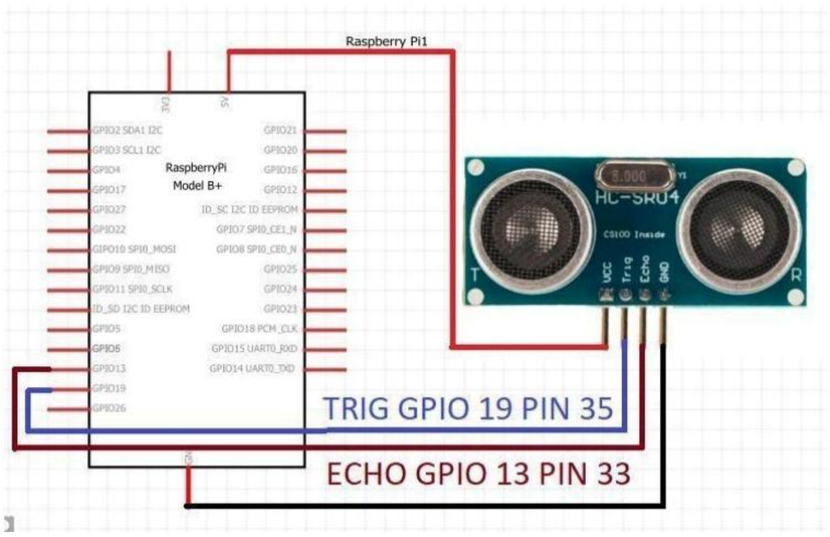
AIM:

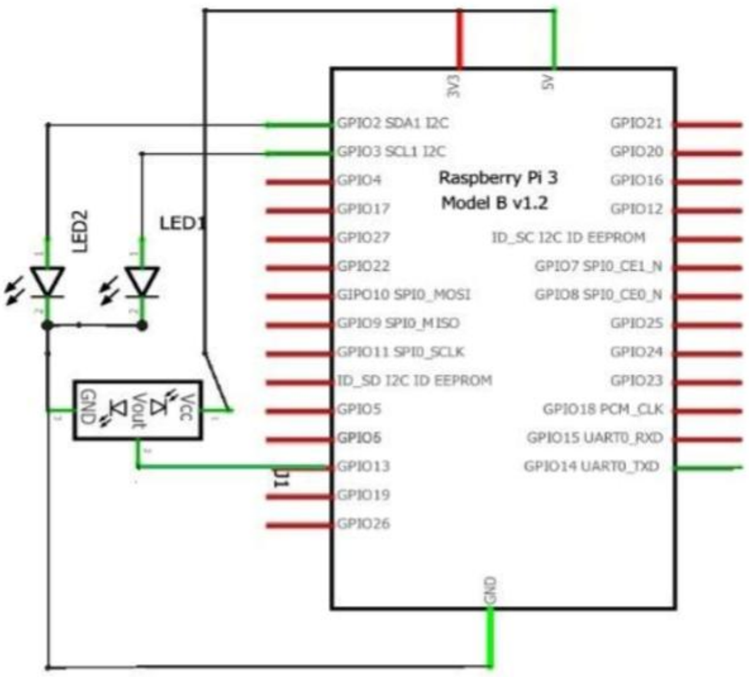
Get the status of a bulb at a remote place (on the LAN) through the web. The student should have hands-on experience in using various sensors like temperature, Humidity, smoke, light, etc., and should be able to use a control web camera, network, and Relays connected to the Pi.

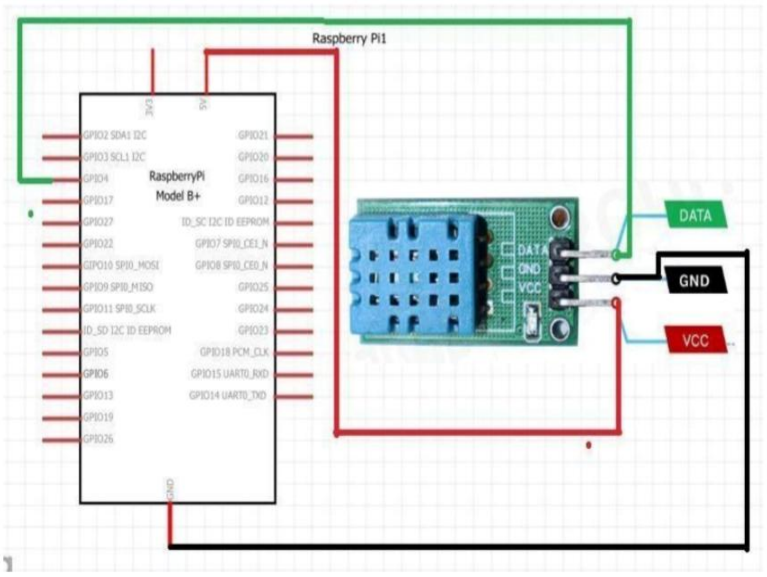
Description:

This project enables remote monitoring of a bulb or device status over a LAN, while offering hands-on experience with various sensors like ultrasonic, IR, DHT11, LDR, and a buzzer. The Raspberry Pi collects data: ultrasonic for distance, IR for object detection, DHT11 for temperature and humidity, and LDR for light intensity. Based on sensor inputs, actions like triggering a buzzer for fire, darkness, or obstacles are taken. The data can be displayed on a web interface for real-time monitoring, creating a smart IoT-based alerting system.

Circuit Diagram:







Program:

```
import RPi.GPIO as GPIO
import time
try:
    GPIO.cleanup()  # added cleanup at the start.
    GPIO.setmode(GPIO.BCM) # Use BCM numbering
    TRIG =19
    ECHO =13
```

```
BUZZER =26 #pin 37 in BOARD mode is pin 26 in BCM mode.
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(BUZZER, GPIO.OUT)
while True:
    GPIO.output(TRIG, False)
    print("Waiting For Sensor To Settle")
    time.sleep(2)
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)
    if 2 < distance < 50:
        print(f"Distance: {distance - 0.5} cm")
        GPIO.output(BUZZER, True)
        print("Buzzer ON")
        time.sleep(1)
    else:
        print("Out Of Range")
        GPIO.output(BUZZER, False)
         print("Buzzer OFF") time.sleep(1)
```

```
except KeyboardInterrupt:
    print("Program stopped by User")
except Exception as e:
    print(f"An error occurred: {e}")
finally:
    GPIO.cleanup()
```

-- IR Sensor

```
import RPi.GPIO as GPIO
import time
sensor1 = 11
sensor2 = 13
buzzer = 36
GPIO.setmode(GPIO.BOARD)
GPIO.setup(sensor1,GPIO.IN)
GPIO.setup(sensor2,GPIO.IN)
GPIO.setup(buzzer,GPIO.OUT)
GPIO.output(buzzer,False)
print ("IR Sensor Ready. ")
try:
    while True:
        if GPIO.input(sensor1):
            GPIO.output(buzzer,True)
            print("Object Detected-1")
        while GPIO.input(sensor1):
            time.sleep(0.3)
        else: GPIO.output(buzzer,False)
         if GPIO.input(sensor2):
             GPIO.output(buzzer,True)
             print("Object Detected-2")
           while GPIO.input(sensor2):
               time.sleep(0.3)
           else: GPIO.output(buzzer,False)
except KeyboardInterrupt:
    GPIO.cleanup()
```

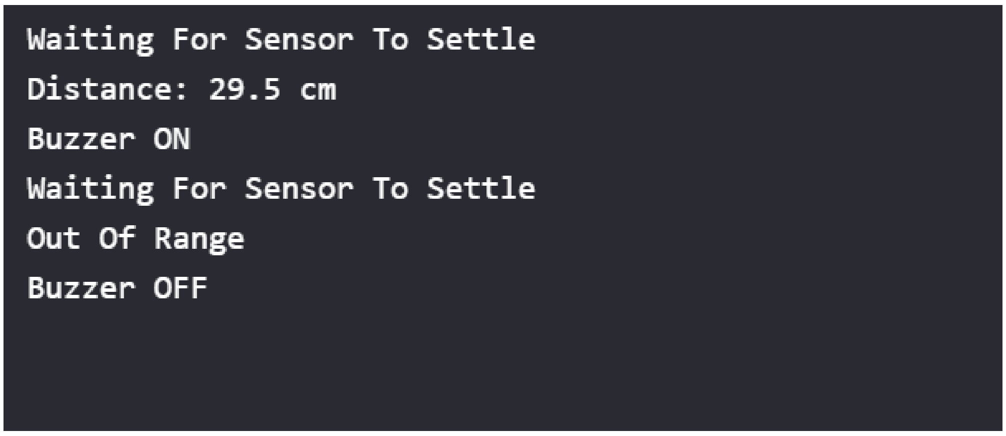
-- TEMP

```
import RPi.GPIO as GPIO
import time
import Adafruit_DHT # Library for DHT sensor
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
# Define Sensor and Buzzer Pins
TRIG = 35  # Ultrasonic Trigger Pin
ECHO = 33 # Ultrasonic Echo Pin
DHT_PIN = 4 # Temperature Sensor Pin (DHT11/DHT22)
LDR_PIN = 11 # Light Sensor (LDR) Pin (Using as Digital Input)
BUZZER = 36 # Buzzer Pin
# Setup GPIO Pins
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(LDR_PIN, GPIO.IN) # Light Sensor as Input
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.output(BUZZER, GPIO.LOW) # Start with buzzer off
# Define Constants
TEMP_THRESHOLD = 30 # Temperature limit for fire detection (in Celsius)
DIST_THRESHOLD = 50 # Distance limit for obstacle detection (in cm)
LIGHT_THRESHOLD = 0 # LDR gives HIGH in light, LOW in darkness
```

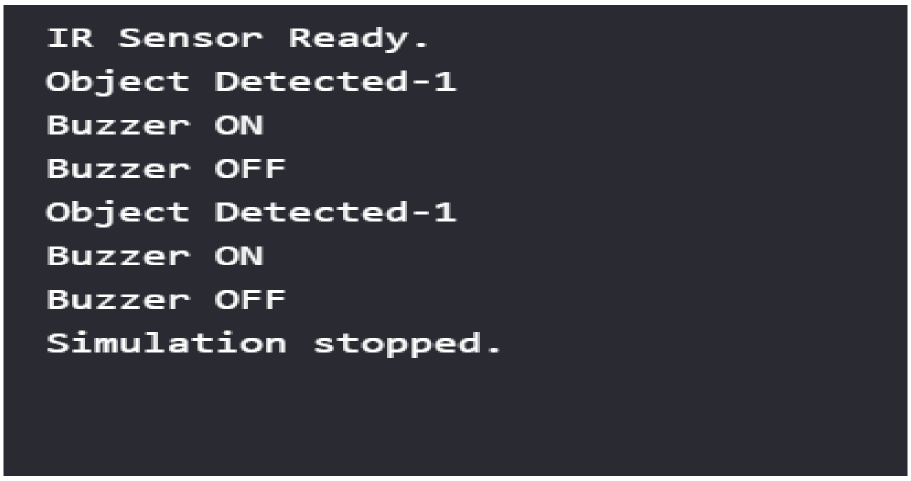
```
# Function to measure distance using Ultrasonic Sensor
def measure_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001) # Send 10us pulse
    GPIO.output(TRIG, False)
# Wait for the echo to start (HIGH)
    while GPIO.input(ECHO) == 0:
        start_time = time.time() # Start timer when echo is LOW
# Wait for the echo to return (HIGH to LOW)
     while GPIO.input(ECHO) == 1:
         stop_time = time.time() # Stop timer when echo is HIGH
# Calculate distance
elapsed_time = stop_time - start_time
distance = (elapsed_time * 34300) / 2 # Convert time to distance (cm)
return round(distance, 2)
# Function to read temperature from DHT sensor
def read_temperature():
    sensor = Adafruit_DHT.DHT11 # Use DHT22 if available
    humidity, temperature = Adafruit_DHT.read_retry(sensor, DHT_PIN)
    return round(temperature, 1) if temperature else None
# Function to check light level using LDR
def check_light():
    return GPIO.input(LDR_PIN) # HIGH = Light, LOW = Darkness
# Main loop
try:
    while True:
        distance = measure_distance()
        temperature = read_temperature()
        light_status = check_light()
       print(f"Distance: {distance} cm | Temperature: {temperature}°C | Light: {'Bright' if light_status == 0 else 'Dark'}")
    # Fire Detection
    if temperature and temperature > TEMP_THRESHOLD:
        print(" ● Fire Detected! Activating Buzzer...")
       GPIO.output(BUZZER, GPIO.HIGH)   # Fire alarm ON
       time.sleep(2) # Delay for fire alert
       GPIO.output(BUZZER, GPIO.LOW) # Turn OFF buzzer after the alert
     # Obstacle Detection
    elif distance < DIST_THRESHOLD:
        print("´'_· ▲` Obstacle Detected! Activating Buzzer...")
        GPIO.output(BUZZER, GPIO.HIGH) # Obstacle alert ON
        time.sleep(1) # Delay for obstacle alert
        GPIO.output(BUZZER, GPIO.LOW) # Turn OFF buzzer after the alert

     # Darkness Detection (Triggering Alert)
    elif light_status == 0: # Light is dark
        print(" F Low Light Detected! Activating Buzzer...")
        GPIO.output(BUZZER, GPIO.HIGH) # Darkness alert ON
        time.sleep(1) # Delay for darkness alert
        GPIO.output(BUZZER, GPIO.LOW) # Turn OFF buzzer after the alert
    else:
        GPIO.output(BUZZER, GPIO.LOW) # Turn OFF buzzer when no conditions are met
        time.sleep(1) # Delay before next reading
except KeyboardInterrupt:
    print("Exiting... Cleaning up GPIO")
    GPIO.cleanup()
```

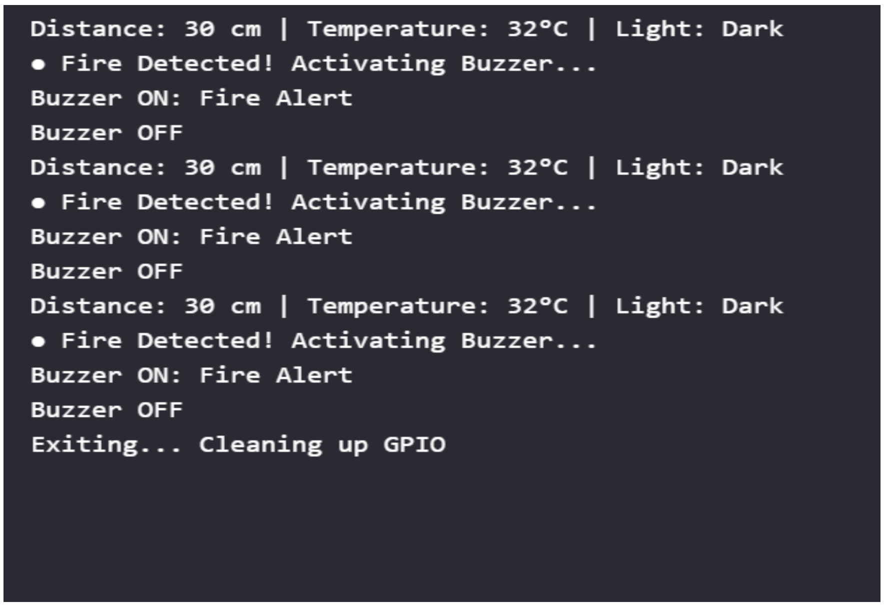
Output:



```
Waiting For Sensor To Settle
Distance: 29.5 cm
Buzzer ON
Waiting For Sensor To Settle
Out Of Range
Buzzer OFF
```



```
IR Sensor Ready.
Object Detected-1
Buzzer ON
Buzzer OFF
Object Detected-1
Buzzer ON
Buzzer OFF
Simulation stopped.
```



```
Distance: 30 cm | Temperature: 32°C | Light: Dark
● Fire Detected! Activating Buzzer...
Buzzer ON: Fire Alert
Buzzer OFF
Distance: 30 cm | Temperature: 32°C | Light: Dark
● Fire Detected! Activating Buzzer...
Buzzer ON: Fire Alert
Buzzer OFF
Distance: 30 cm | Temperature: 32°C | Light: Dark
● Fire Detected! Activating Buzzer...
Buzzer ON: Fire Alert
Buzzer OFF
Exiting... Cleaning up GPIO
```

(CASE STUDY)
SMART COLLAR SYSTEM

TABLE OF CONTENTS

ABSTRACT

In today's tech-driven world, pet ownership has been revolutionized by advancements in Internet of Things (IoT). This project presents a Smart Collar for Pets using IoT, aimed at enhancing the safety, health, and overall well-being of domestic animals, particularly dogs and cats. The system is designed to track real-time location via GPS, monitor health parameters such as heart rate and temperature, and provide behavior insights through a mobile application. Additionally, it utilizes geofencing, cloud-based data storage, and alert systems to ensure that pet owners are constantly aware of their pet’s status. With real-time alerts and cloud integration, the smart collar offers peace of mind to pet owners and improves the care provided to pets. This report explores the design, implementation, benefits, and potential enhancements of the proposed system.

LITERATURE SURVEY

Study	Technology Used	Strength	Weakness
Whistle Go Explore	GPS+ Accelerometer	Accurate location tracking	Lacks health monitoring
Fi Smart Dog Collar	GPS+Motion Sensors	Fitness tracking, long battery life	No temperature or heart rate monitoring
FitBark Health Monitor	Accelerometer + Bluetooth	Lightweight, behavioral analysis	No GPS or physiological data
Singh et al. (IEEE)	IoT + Vital Sensors + Cloud	Real-time health monitoring, cloud integration	Network dependency
Reddy & Kumar (Springer)	Multi-sensor system + Wi-Fi	Early disease detection, scalable	Cost increases with sensor complexity

EXISTING SYSTEM

In recent years, several commercial and research-based smart pet collars have been developed, aiming to assist pet owners in tracking their pets' activities and locations. Notable devices include Whistle Go Explore, Fi Smart Collar, and Tractive GPS Tracker. These devices have demonstrated the value of wearable technology in pet care, particularly in areas of GPS tracking and basic fitness monitoring.

However, a critical analysis of these existing systems reveals multiple limitations in terms of functionality, cost-effectiveness, and adaptability to various socio-economic and geographic conditions, especially in developing countries like India.

Key Features of Existing Systems:

- **Whistle Go Explore**: Offers GPS tracking, activity monitoring, and escape alerts. It requires a subscription for data services and is more suited to urban areas with consistent connectivity.
- **Fi Smart Collar**: Known for its sleek design and step counting. However, it lacks the ability to measure health vitals such as heart rate or body temperature.
- **Tractive GPS Tracker**: Provides accurate location tracking and virtual fence features, but does not support any health or behavior monitoring features.

Common Limitations of Existing Solutions:

- Lack of Real-Time Health Monitoring: Most commercial collars do not include integrated sensors to monitor critical physiological parameters like heart rate, body temperature, or activity anomalies.
- High Cost and Subscription Models: These devices are often expensive and come with recurring costs for GPS and data services, making them less accessible to pet owners in rural or low-income settings.
- Minimal Customization and Limited Local Relevance: Commercial solutions are often designed for Western markets and may not cater to the specific needs of Indian pet owners, such as support for local veterinary practices or regional connectivity issues.
- Dependency on Constant Internet Connectivity: Many of the existing systems require continuous internet access for real-time tracking and alert generation, which can be a significant challenge in remote or underdeveloped areas.
- No Veterinary Diagnostics Integration: There is no scope in the current market solutions to integrate with veterinary services for diagnostics or treatment recommendations based on the collected health data..

.

PROPOSED SYSTEM

The proposed system is a low-cost, high-functionality smart collar designed to monitor a pet's location and health metrics in real-time. It consists of multiple sensors integrated with a microcontroller (ESP32) that processes and sends data to a cloud platform. A mobile application fetches and displays this data to the pet owner.

KEY FEATURES INCLUDE:

- **Real-time GPS Tracking**: Accurate location data using the Neo-6M GPS module.
- **Health Monitoring**: Heart rate via MAX30102 sensor, and body temperature using DS18B20.
- **Motion Tracking**: Accelerometer MPU6050 to detect abnormal behavior.
- **Geofencing Alerts**: Warns the owner if the pet leaves a defined safe zone.
- **Cloud Database**: Stores real-time and historical data on Firebase or ThingSpeak.
- **Mobile Application**: Displays real-time metrics, historical data trends, and sends alerts.

ADVANTAGES OVER EXISTING SYSTEMS:

- Affordable and Custom-Built for Local Conditions: Unlike high-cost commercial products, this system is developed with low-cost components and can be easily manufactured or customized for local use.

- End-to-End Health and Location Monitoring: Provides both vital health monitoring and GPS-based safety tracking in a single device.

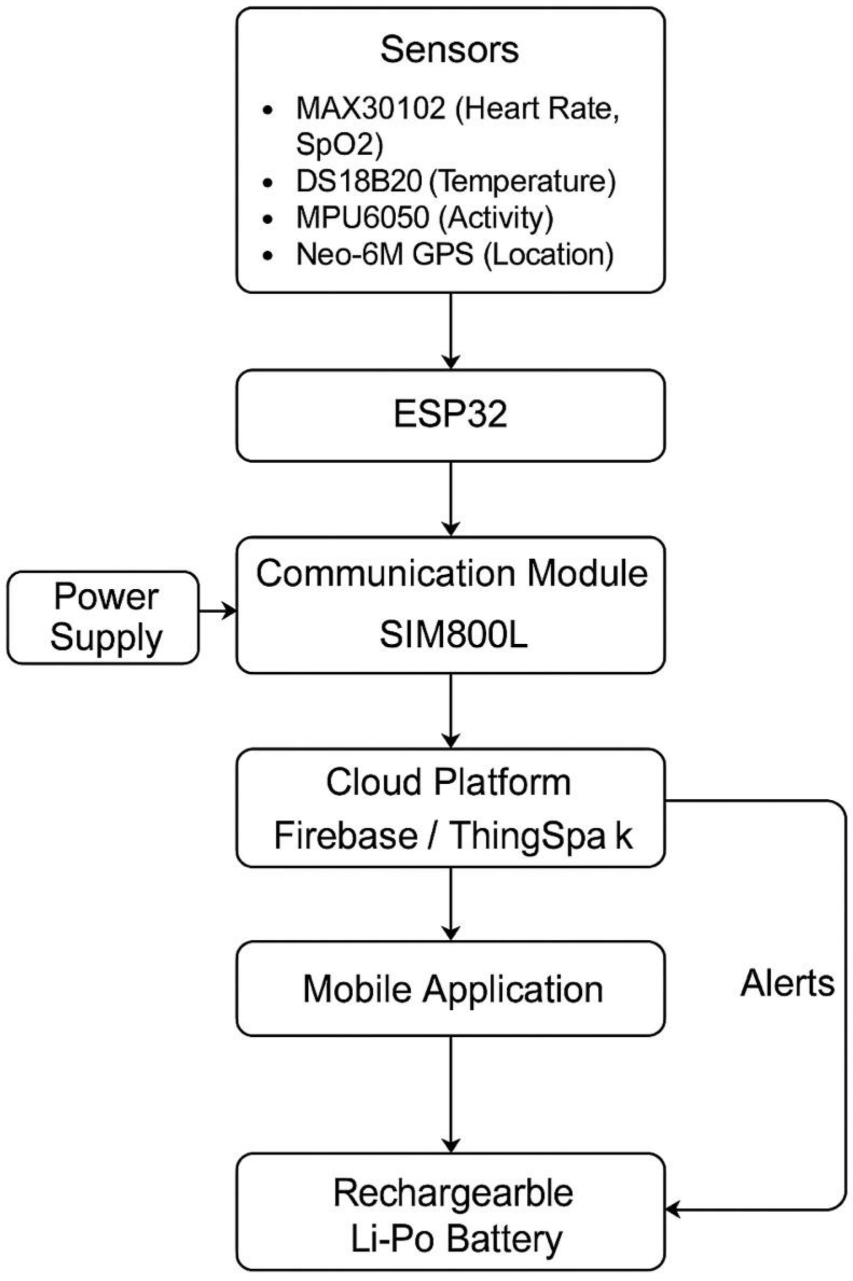
- Cloud-Based Alerts and Analysis: Enables real-time alerts for abnormalities and supports long-term health trend analysis, which is missing in most commercial systems.

- Low Power and Rural Connectivity Support: Supports GSM connectivity, enabling use in low-infrastructure environments with limited internet access.

This system empowers pet owners with timely information, facilitates preventive healthcare, and improves pet safety.

ARCHITECTURE

FLOW CHART:


Sensors
MAX30102 (Heart Rate, SpO2)
DS18B20 (Temperature)
MPU6050 (Activity)
Neo-6M GPS (Location)
ESP32
Power Supply
Communication Module SIM800L
Cloud Platform Firebase / ThingSpa k
Alerts
Mobile Application
Rechargearble Li-Po Battery

The architecture of the smart collar system includes various interconnected hardware and software components working together to collect, process, transmit, and display pet data.

Block Diagram Components:

Sensors:

- MAX30102 for heart rate and SpO2 detection.
- DS18B20 for temperature sensing.
- MPU6050 for activity recognition.
- Neo-6M GPS for location tracking.

Microcontroller: ESP32, chosen for its built-in Wi-Fi and Bluetooth capabilities.

Communication Module: SIM800L for GSM connectivity in areas without Wi-Fi.

Power Supply: Rechargeable Li-Po battery with a Battery Management System.

Cloud Platform: Firebase or ThingSpeak to store and analyze data.

Mobile Interface: Built using Flutter or MIT App Inventor.

Workflow:

1. Sensors continuously collect data from the pet.

2. ESP32 processes and formats the data.

3. Data is transmitted via Wi-Fi or GSM to the cloud.

4. The mobile application retrieves and visualizes the data.

5. Alerts are generated based on pre-set conditions.

REQUIREMENTS

Hardware Components:

- ESP32 Development Board
- GPS Module Neo-6M
- MAX30102 Heart Rate Sensor
- DS18B20 Digital Temperature Sensor
- MPU6050 Accelerometer and Gyroscope
- SIM800L GSM Module (optional)
- Li-Po Battery with Charging Module
- 3D-printed or adjustable pet collar mount
- Software Requirements:
- Arduino IDE (programming microcontroller)
- Firebase / ThingSpeak (cloud services)
- Flutter or MIT App Inventor (mobile app development)
- Embedded C or Arduino-based coding

Connectivity Requirements:

- Internet (Wi-Fi or GSM network)
- Smartphone with the companion app installed

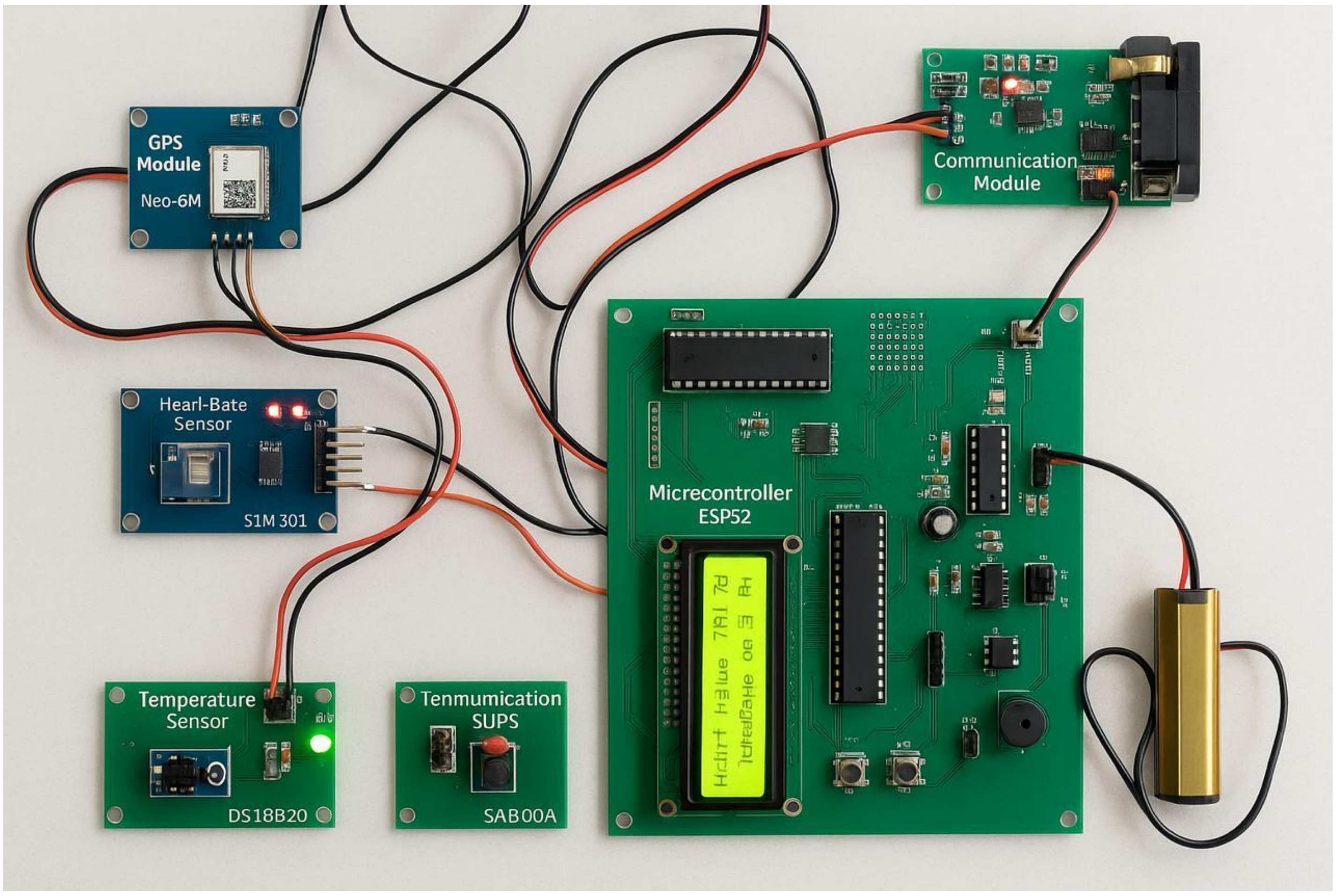
PROGRAM

```
#include <Wire.h>
#include <WiFi.h>
#include <FirebaseESP32.h>
#define WIFI_SSID "yourSSID"
#define WIFI_PASSWORD "yourPASSWORD"
#define FIREBASE_HOST "your-project.firebaseio.com"
#define FIREBASE_AUTH "yourAuthToken"
FirebaseData firebaseData;
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("."); }
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  Firebase.reconnectWiFi(true);
}
void loop() {
  float temp = readTemperature();
  float heartRate = readHeartRate();
  String location = getGPSLocation();
  Firebase.setFloat(firebaseData, "/pet/temp", temp);
  Firebase.setFloat(firebaseData, "/pet/heart_rate", heartRate);
Firebase.setString(firebaseData, "/pet/location", location);
  delay(10000);}
```

This code continuously sends temperature, heart rate, and GPS location to Firebase every 10 seconds. The readTemperature(), readHeartRate(), and getGPSLocation() functions must be implemented to read from the respective sensors.

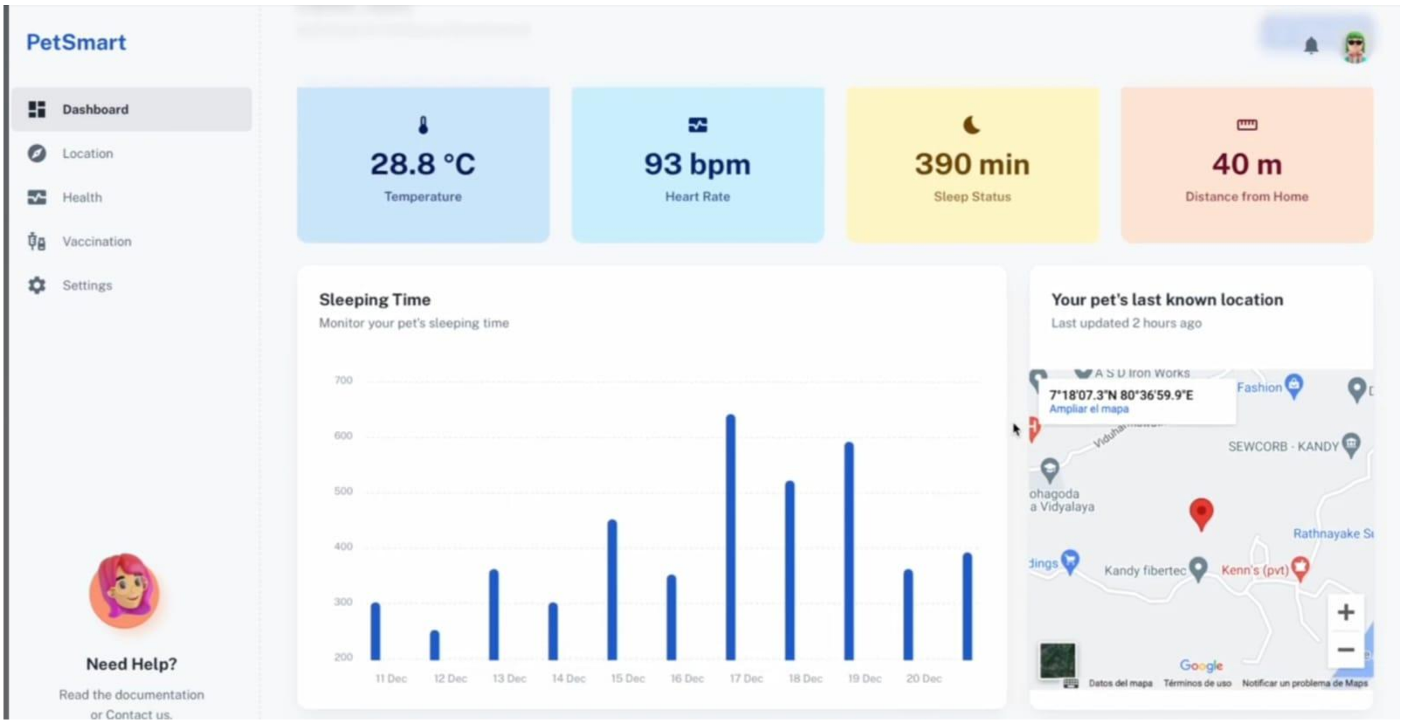
OUTPUT:

CIRCUIT DAIGRAM:



We have designed and implemented a circuit similar to the one shown, integrating GPS, heart rate, and temperature sensors with a central microcontroller. The setup also includes an LCD display and communication modules for real-time pet monitoring via IoT

THE USER DASHBOARD:



Real-Time Dashboard: Displays heart rate, body temperature, and movement status.

Live Map View: Shows pet's location using embedded Google Maps.

Alerts: Sends notifications when pet exits safe zones or abnormal vitals are detected.

Historical Data: Graphs and logs of health metrics and movements.

Owner Dashboard: Mobile app allows data review, configuration of thresholds, and historical logs.

FUTURE SCOPE

The Smart Collar for Pets using IoT opens up numerous possibilities for enhancing pet care through technology. Some potential future developments and integrations include:

AI-Based Behavior Analysis

Integrate machine learning models to detect and classify unusual behaviors such as excessive barking, licking, or signs of distress, allowing early diagnosis of health issues.

Veterinary Integration

Connect the collar's data to veterinary health systems, enabling real-time diagnostics, remote check-ups, and electronic health records for pets.

Environmental Monitoring

Add sensors to detect ambient temperature, air quality, or harmful gases (like CO2 or smoke), alerting owners to environmental dangers.

Extended Battery Life

Utilize solar charging or ultra-low-power components to increase the battery life and reduce maintenance.

Voice and Sound Recognition

Implement microphones and sound recognition to identify familiar voices or distress sounds and trigger specific actions or alerts.

Cloud Analytics Dashboard

Develop advanced analytics dashboards to visualize historical trends in pet health, location habits, and activity levels.

Multi-Pet Management

Allow tracking and managing multiple pets through a single app interface with individual profiles and data logs.

Emergency Contact Notification

If abnormal readings are detected, send automated notifications to predefined emergency contacts (e.g., vet, neighbors, family).accuracy.

CONCLUSION

The Smart Collar for Pets using IoT offers a modern solution for monitoring the health and safety of pets. It provides real-time insights into pet location, vital signs, and movement behavior, thus enabling early detection of potential health issues. The system's integration with cloud platforms and a mobile interface ensures easy access to data, making it an ideal companion for pet owners who want to ensure their pets are safe, healthy, and well-monitored.

Future enhancements may include:

1. Integration of machine learning for predictive analytics.

2.Improved battery life with energy-efficient components.

3.Enhanced user interface with AI chatbots for pet advice.

4.Addition of environmental sensors (e.g., gas, humidity) for outdoor pets.

By leveraging IoT technology, the smart collar represents a leap forward in pet care and management.