A
Project Report
on

"Lab Automation System"

Submitted By

Mr. Aryan Koul	2020BTEEN00001
Mr. Sanmay Girish Kulkarni	2020BTEEN00008
Mr. Saurabh Nitin Dhande	2020BTEEN00039
Mr. Ranjit Abaji Dhaigude	2020BTEEN00059
Mr. Gaurav Shitalkumar Patil	20202BTEIT00076

Under the guidance of

Prof. M. R. Khare

Department of Electronics Engineering

WALCHAND COLLEGE OF ENGINEERING, SANGLI

(Government-Aided Autonomous Institute)

2023-2024

(This page is intentionally kept blank**)**

LIST OF FIGURES

ABSTRACT

Our project, "**Lab Automation System**" is a real-time operating system (RTOS) project is designed to implement a comprehensive home automation and security system. The system comprises six primary tasks:

1. Fire Security System
2. Gas Detection System
3. Light Automation using PIR Sensor.
4. Fan Automation using PIR Sensor
5. Light Automation using Switch.
6. Fan Automation using Switch

The Overall Intention behind making this project is to automate the Laboratories and provide safety measures to the students and faculty members working over there.

ACKNOWLEDGEMENT

This project has direct reference and inspiration from the project titled "**Smart Lab Security with Energy Conservation** ". Hence our team would like to express our sincere gratitude to all those who have been a part of the "Smart Lab Security with Energy Conservation" project.

Special thanks to our dedicated Team Members who invested their time and expertise while successful completion of this project.

We also extend our appreciation to our mentor Mrs. Meghana Khare mam for her guidance and constant support throughout the completion of this project.

1. Introduction

The project centers on the implementation of a Real-Time Operating System (RTOS) in our college lab, with a focus on task prioritization. The main goal is to effectively manage tasks based on their priority levels. These tasks include critical functions like gas leakage and fire alarm detection, alongside operations like light and fan automation.

This project serves as a practical demonstration of how an RTOS, complemented by semaphores. By simulating situations where timely responses are crucial, we aim to showcase the practical utility of an RTOS. In the ensuing sections of this report, we will go deeper into the specifics of our RTOS implementation into the lab sector mainly, the tasks at hand, and the interaction with semaphores in ensuring the execution of tasks.

Additionally, this project offers an opportunity to explore the practicality in real-time operating environments by implementing the tasks in the college lab according to the priorities set in the code. The project aims to demonstrate the practical applicability of RTOS and semaphores in real-world scenarios.

---

2. Reason behind Taking this Project

We took on this project as a crucial part of our mega project named **Smart Lab with Energy Conservation** project. This larger project involves actually applying these tasks in a real lab setting. The idea of taking this project is to understanding the tasks implemented in the project with the help of the Real time operating systems. Another thing is to focus on real-time operating systems and task priorities to understand how to make these functions work seamlessly.

We want to make sure that when we implement critical tasks like detecting gas leaks or fires, or controlling lights and fans, we do it in a way that's efficient and safe. This project serves as the foundation for our broader goal of creating a smart, energy-efficient lab environment.

---

3. Methodology

The Development of the Real-Time Operating System (RTOS) based **Lab Automation and Security System** involves a structured approach that encompasses several key steps:

- **Requirement Analysis:**
  In this initial phase, the project team thoroughly analyzed the requirements and objectives of the system. This involved understanding the need for fire and gas detection, occupancy-based automation for lighting and fan control, and manual control via switches.
- **Hardware Selection:**
  The selection of appropriate hardware components is crucial. PIR sensors were chosen for occupancy detection, while gas sensors and fire detectors were integrated into the system. Additionally, microcontrollers and communication modules were selected for their compatibility with the RTOS.

- **RTOS Selection:**
  We have selected uC/OS-II RTOS that best suited our project's requirements. The chosen RTOS provides the necessary support for multitasking, scheduling, and real-time responsiveness.

- **Task Design and Implementation:**
  Each of the six tasks was designed as individual modules. The Fire Security System and Gas Detection System were implemented to monitor and respond to emergencies. The Light and Fan Automation tasks, both PIR-based and switch-controlled, were programmed for optimal energy management.

- **Documentation:**
  Detailed documentation was maintained throughout the project's development. This includes hardware specifications, software architecture, and a user manual for the system's operation and maintenance.

---

4. Priority Table of the Project:

Task Number	Name of the Task	Priority Number
Task 1	Fire Security System	1
Task 2	Gas Detection System	2
Task 3	Light Automation using PIR Sensor	3
Task 4	Fan Automation using PIR Sensor	4
Task 5	Light Automation using Switch	5
Task 6	Fan Automation using Switch	6

Figure 4.1: Priority Table

The Above Diagram describes the sequence of priorities given to each task. Task 1 and Task 2 are given the **Highest Priority** and are using **UART0** as shared resource. The task is done using the concept of **Semaphore**. Task 3 to Task 6 are using the concept of **Virtual Parallelism**, allowing CPU to simultaneously execute multiple tasks and switch in between tasks.

- **Concept of Context Switching and Virtual Parallelism:**
  Refers to the process of saving and restoring the state of CPU so that different tasks or processes can be executed. In a multitasking environment, multiple tasks or processes are sharing a single CPU, and the CPU rapidly switches between them, giving the illusion that they are executing simultaneously. Context Switching allows each and every task to run for a brief period of time, known as Time Slice or Quantum, before moving to the next task. **Virtual Parallelism** is the illusion of Parallelism created in software, allowing tasks to be executed concurrently even on the single core system.

- **Concept of Semaphore:**

Critical Section of Code:
Specific Section of the code/set of instructions that accesses the shared resources. In this, task modifies the shared data. If two or more tasks access the critical section simultaneously, and atleast one of them modifies the shared data, it can lead to data inconsistency and program errors.

Shared Resources:
Refers to data/hardware devices that are accessed or used by multiple threads/tasks in concurrent systems. When multiple tasks attempts to access and modify shared resources simultaneously, it can lead to corruption and race conditions as such.

To ensure integrity of shared resources, only one task should be allowed to execute the critical section of the code. Hence, Synchronization tools such as **Semaphore**, **Mutex** and **Locks** are used, to ensure that only one task can enter the critical section, preventing concurrent access and maintaining the data consistency.

Semaphores are used to:
1) Protect Shared Resources:
Used to protect critical section of the code, ensuring only 1 task can access the shared resources at a time.
2) Task Synchronization:
Are used to synchronize the execution of multiple tasks. Are used to signal the completion of specific task, so that other task can access the critical section or shared resources.

---

5. Block Diagram:

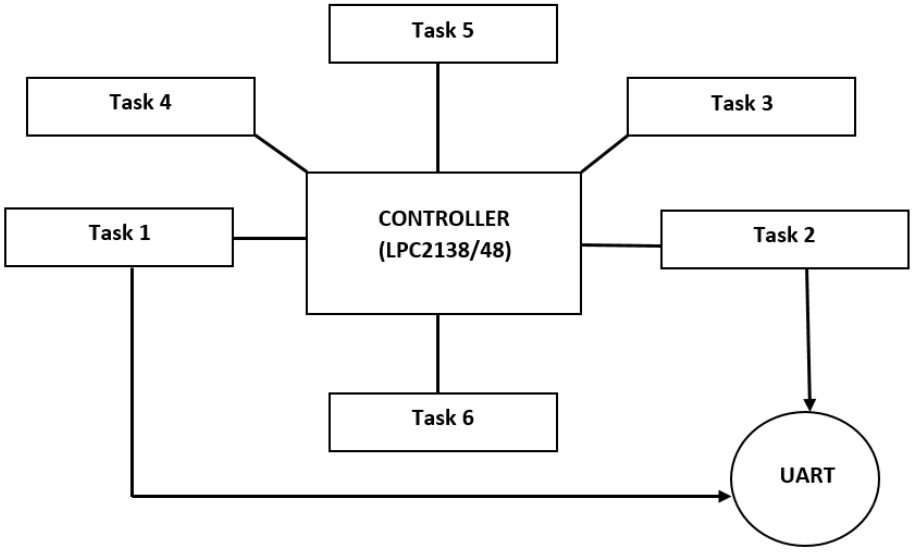

Figure 5.1: Block Diagram of the Project

The Development of the Real-Time Operating System (RTOS) based **Lab Automation and Security System** involved following tasks:

- **Task 1 (Fire Security System):**
  This task monitors for fire-related events and triggers alarms and alerts in case of fire hazards.
- **Task 2 (Gas Detection System):**
  It detects gas leaks and ensures the safety of residents by providing timely warnings.
- **Task 3 (Light Automation using PIR Sensor):**
  This task utilizes Passive Infrared (PIR) Sensors to control lighting based on occupancy, enhancing energy efficiency.
- **Task 4 (Fan Automation using PIR Sensor):**
  Similar to the previous task, it automates fan operations based on room occupancy to optimize comfort and energy consumption.
- **Task 5 (Light Automation using Switch):**
  Allows manual control of lighting, giving users the flexibility to adjust lighting settings as needed.
- **Task 6 (Fan Automation using Switch):**
  Provides manual control over fan operations, allowing users to regulate room ventilation.

Note that Task 1 and Task 2 will be considered as **Task B (Semaphore Task)** while Task 3 to Task 6 will be referred to **Task A (Virtual Parallelism Task).**

---

6. Flow Diagram:

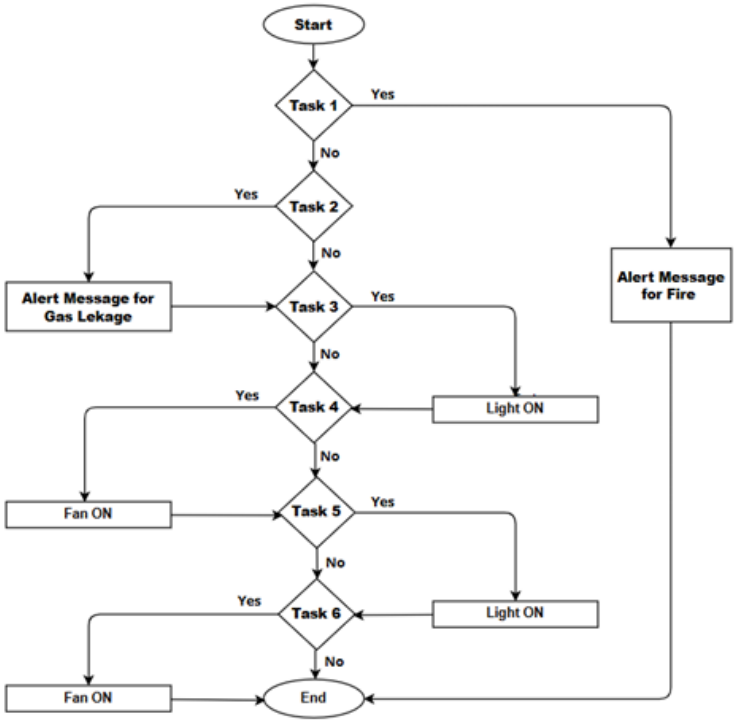

Figure 6.1: Flow Diagram of the Project

The Flow Diagram depicts the execution of 6 Tasks based on:

- When the projects start it first checks for output of fire sensor (Task 1). If fire is sensed. the message "**Fire Detected"** will be displayed and the control will go to end.
- If NO, the control will be given to Task 2.
- The Output of Gas Sensor is checked, if HIGH, the Message "**Gas Detected"** will be displayed and control will be given to Task 3.
- If No, the control will be given to Task 3.
- The Output of PIR (PIR 1) is sensed, if Motion Detected, then the respective bulb will glow and the control will be given to Task 4.
- If the motion is not detected, the respective bulb will be turned OFF and the control is given to Task 4.
- The Output of PIR (PIR 2) is sensed, if Motion Detected, then the respective fan will be turned ON and the control will be given to Task 5.
- If the motion is not detected, the respective fan will be turned OFF and the control is given to Task 5.
- The Switch (SW1) logic condition will be checked. If switch is pressed, then respective Bulb will be turned ON and control will be given to Task 6.
- If Switch is released, then respective Bulb will be turned OFF and control will be given to Task 6.
- The Switch (SW2) logic condition will be checked. If switch is pressed, then respective Fan will be turned ON and control will be given to Task 1.
- If Switch is released, then respective Fan will be turned OFF and control will be given to Task 1.

---

7. Pin Configuration Used:

SrNo.	Task Number	Input Pin	Output Pin
1.	Task 1	P0.15	P0.22
2.	Task 2	P0.16	P0.28
3.	Task 3	P0.12	P0.6
4.	Task 4	P0.13	P0.7
5.	Task 5	P0.10	P0.4
6.	Task 6	P0.11	P0.5

Figure 7.1: Pin Configuration

---

8. Keil Code:

```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>

// Define Stack Length
#define TaskStkLength 64

// Define Respective Task Stack
OS_STK Fire_System_Stk [TaskStkLength];
OS_STK Gas_System_Stk [TaskStkLength];
OS_STK Light_Sense_Stk [TaskStkLength];
OS_STK Fan_Sense_Stk [TaskStkLength];
OS_STK Light_Switch_Stk [TaskStkLength];
OS_STK Fan_Switch_Stk [TaskStkLength];

// Declaration of Tasks:
void Fire_System ();
void Gas_System ();
void Light_Sense ();
void Fan_Sense ();
void Light_Switch ();
void Fan_Switch ();

// Creation of Semaphore Event
OS_EVENT *MySema;
unsigned char err;

// main() function
int main (void)
{
    LED_init();

    UART0_Init();
    TargetInit();
    OSInit ();

    MySema = OSSemCreate(1);

    OSTaskCreate (Fire_System,(void *)0, &Fire_System_Stk [TaskStkLength - 1], 1);
    OSTaskCreate (Gas_System,(void *)0, &Gas_System_Stk [TaskStkLength - 1], 2);
    OSTaskCreate (Light_Sense,(void *)0, &Light_Sense_Stk [TaskStkLength - 1], 3);
    OSTaskCreate (Fan_Sense,(void *)0, &Fan_Sense_Stk [TaskStkLength - 1], 4);
```

```
    OSTaskCreate (Light_Switch,(void *)0, &Light_Switch_Stk[TaskStkLength - 1], 5);
    OSTaskCreate (Fan_Switch,(void *)0, &Fan_Switch_Stk [TaskStkLength - 1], 6);

    OSStart();

    return 0;

}

// Fire_System Task (Task 1)
void Fire_System()
{
    while(1)
    {
        while(IO0PIN & (1<<15))
        {
            OSSemPend(MySema,1,&err);
            UART0_SendData("Fire Detected\n");

            LED_on(4);
            OSTimeDly(4);
            LED_off(4);
            OSTimeDly(4);

            OSSemPost(MySema);
        }

        if (IO0PIN & (1<<16))
        {
            LED_on(5);
            OSTimeDly(4);
            LED_off(5);
            OSTimeDly(4);
        }
    }
}

// Gas_Detection System (Task 2):
void Gas_System()
{
    while(1)
    {
        if(IO0PIN & (1<<16))
        {
            OSSemPend(MySema,1,&err);
            UART0_SendData("Gas Detected\n");
```

```
                LED_on(5);
                OSTimeDly(4);
                LED_off(5);
                OSTimeDly(4);

                OSSemPost(MySema);
        }

    }
}

// Light_Sense Task (Task 3)
void Light_Sense()
{
    while(1)
    {
        if(IO0PIN & (1<<12))
        {
                LED_on(2);
                OSTimeDly(4);
        }
        else
        {
                LED_off(2);
                OSTimeDly(4);
        }
    }
}

// Fan_Sense Task (Task 4)
void Fan_Sense()
{
    while(1)
    {
        if(IO0PIN & (1<<13))
        {
                LED_on(3);
                OSTimeDly(4);
        }
        else
        {
                LED_off(3);
                OSTimeDly(4);
        }
    }
```

```
}

// Light_Switch Task (Task 5)
void Light_Switch()
{
    while(1)
    {
        if(IO0PIN & (1<<10))
        {
            LED_on(0);
            OSTimeDly(4);
        }
        else
        {
            LED_off(0);
            OSTimeDly(4);
        }
    }
}

// Fan_Switch Task (Task 6)
void Fan_Switch()
{
    while(1)
    {
        if(IO0PIN & (1<<11))
        {
            LED_on(1);
            OSTimeDly(4);
        }
        else
        {
            LED_off(1);
            OSTimeDly(4);
        }
    }
}
```

---

9. Proteus Simulation Results:

- **LPC2138/48 Flame Sensor Interfacing (Task 1):**

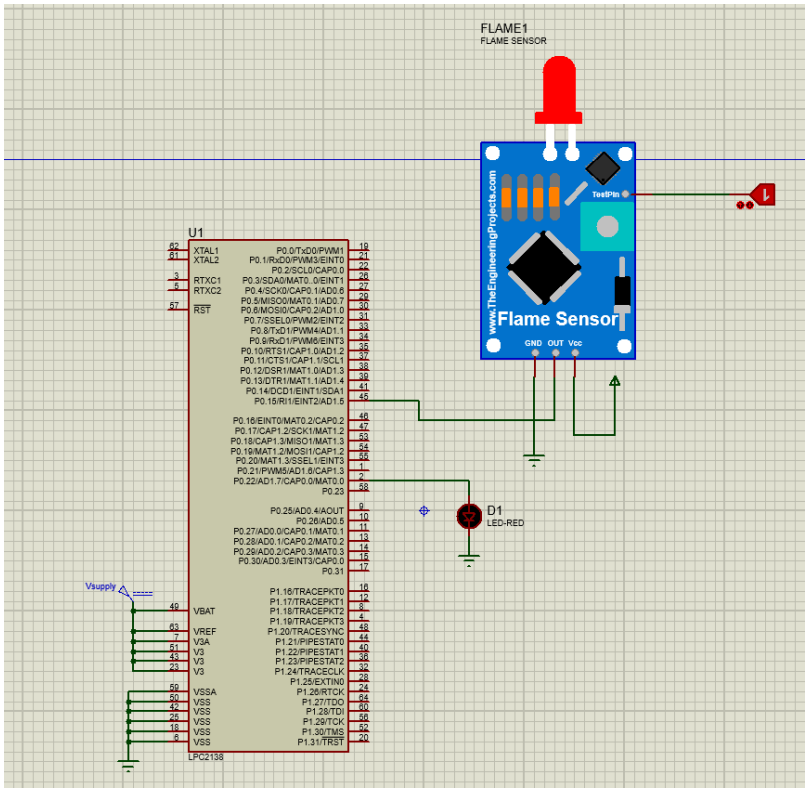

Figure 9.1: LPC2138/48 Flame Sensor Circuit Diagram

Components Used:
LPC2138/48, LED, Logic State, Flame Sensor.

Sr No.	Logic State	Result
1.	HIGH	LED Blinks with specific Delay, indicating presence of Fire.
2.	LOW	LED doesn't blink, indicating absence of Fire.

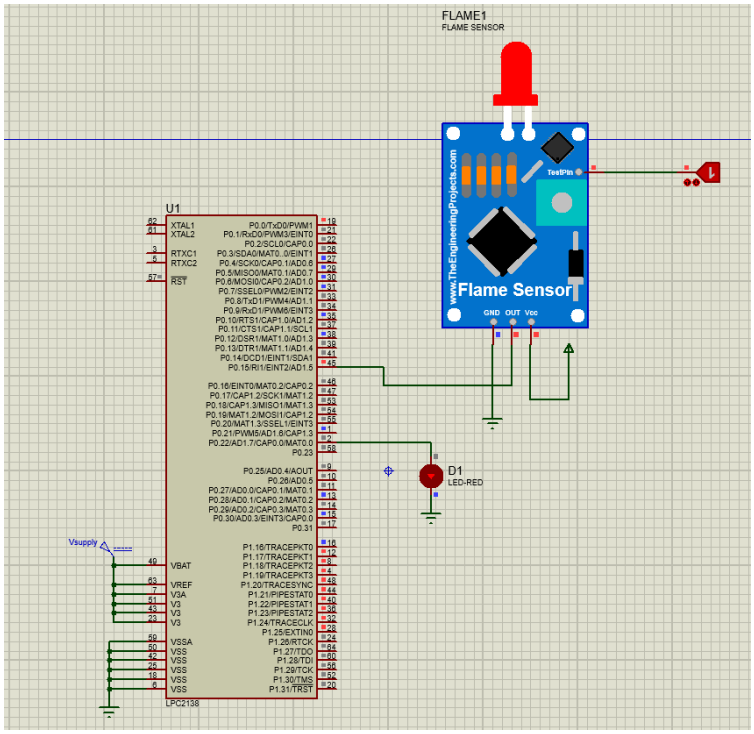

Figure 9.2: Fire Detected

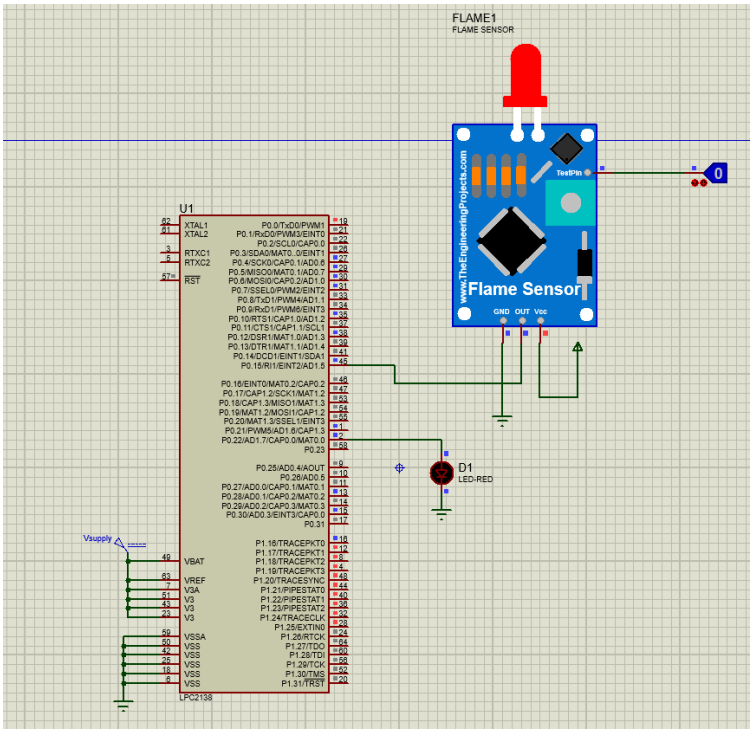

Figure 9.3: Fire Not Detected

- **LPC2138/48 Gas Sensor Interfacing (Task 2):**

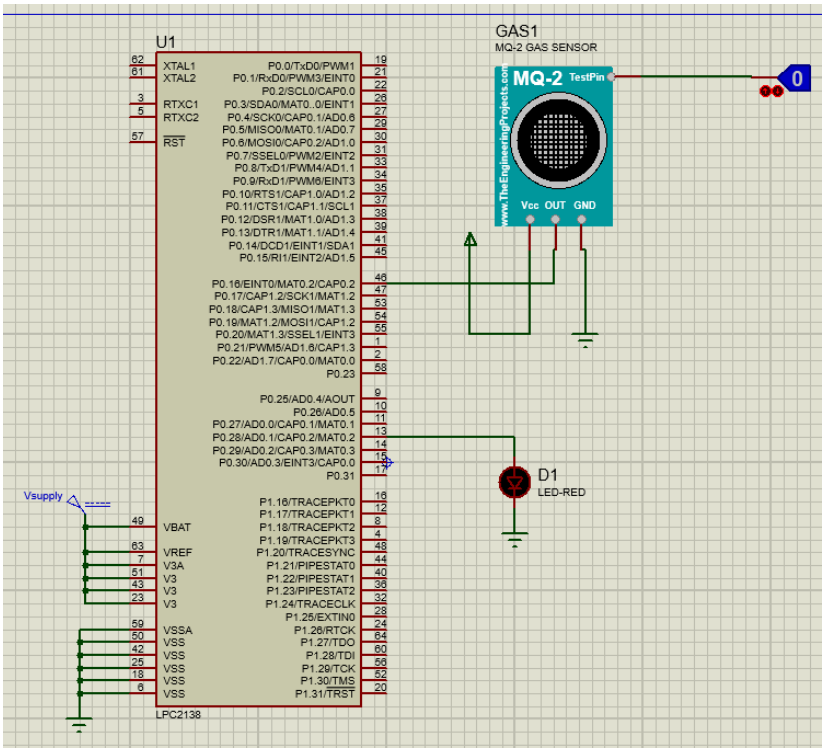

Figure 9.4: LPC2138/48 Gas Sensor Circuit Diagram

Components Used:
LPC2138/48, LED, Logic State, MQ2 Gas Sensor.

Sr No.	Logic State	Result
1.	HIGH	LED blinks with specific Delay indicating presence of Gas.
2.	LOW	LED doesn't blink indicating absence of Gas.

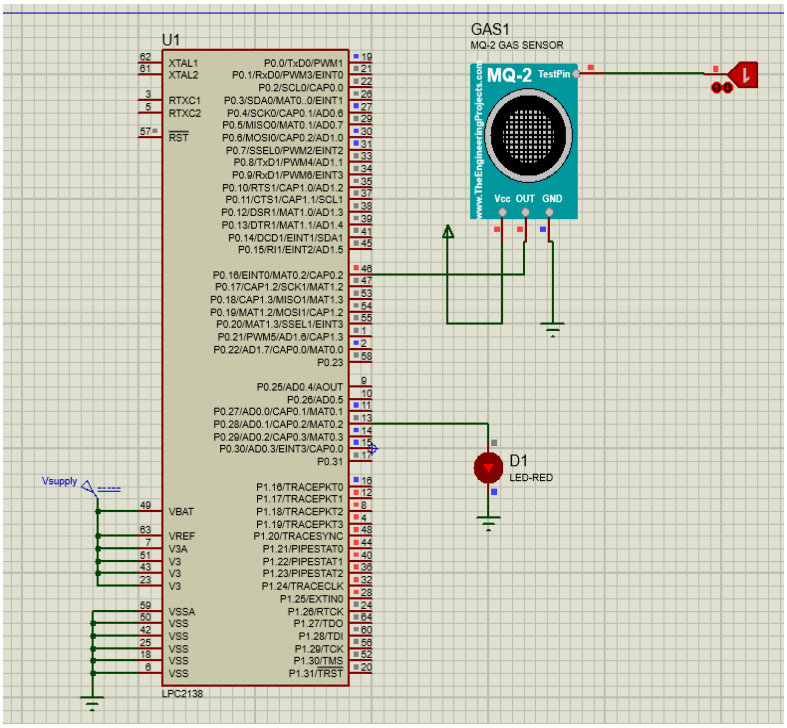

Figure 9.5: Gas Detected

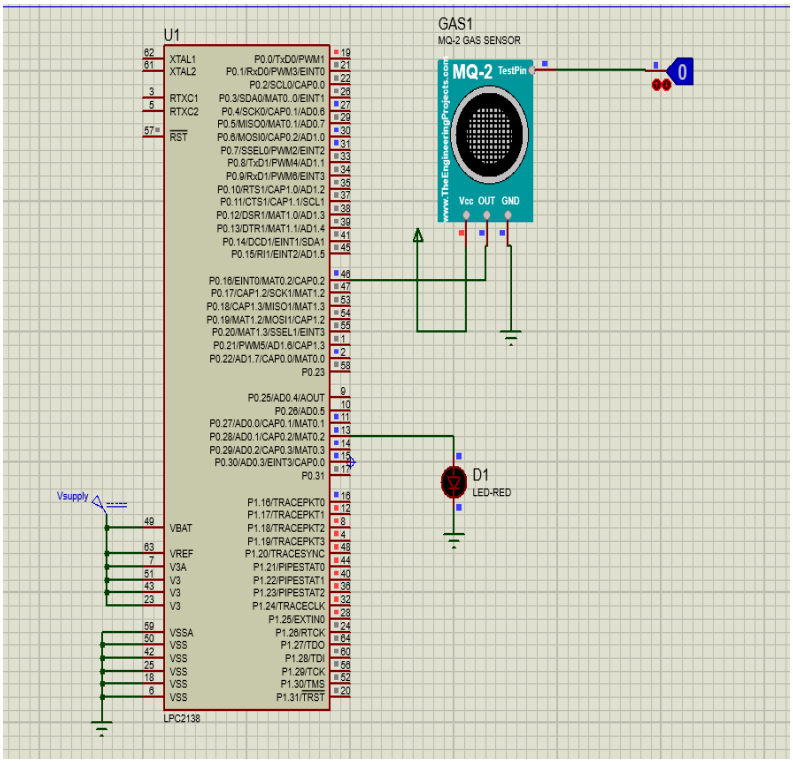

Figure 9.6: Gas Not Detected

- **Light Automation using PIR Sensor (Task 3):**

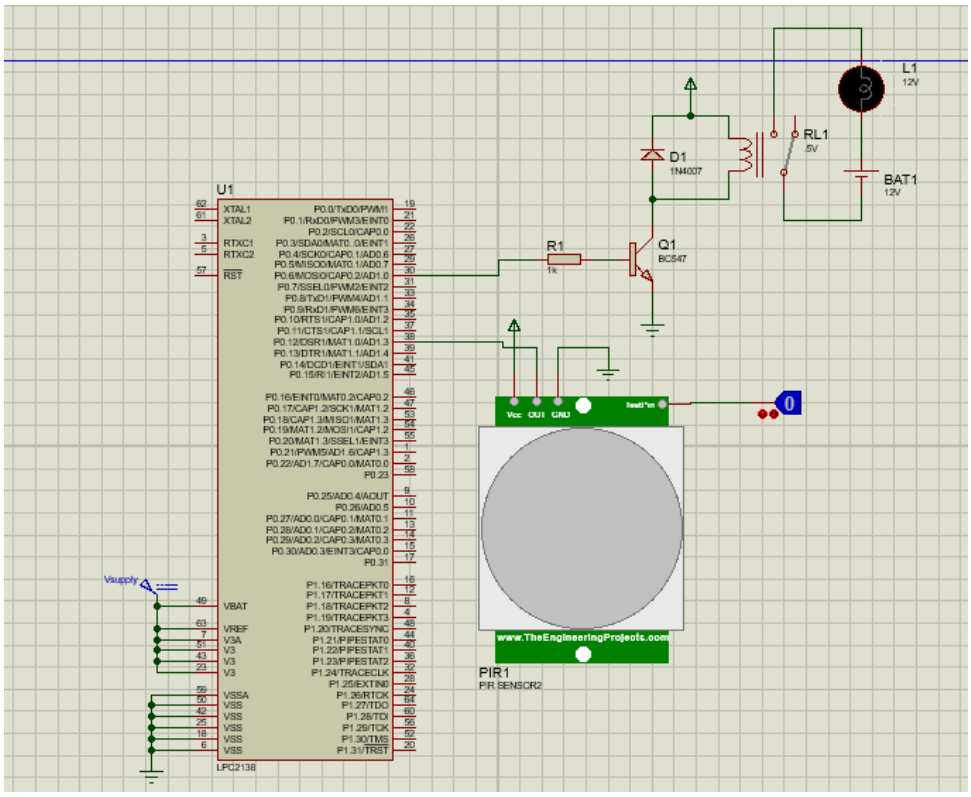

Figure 9.7: Light Automation using PIR Sensor (Circuit Diagram)

Components Used:

LPC2138/48, Resistor, Transistor (BC547), Diode (1N4007), Relay, Bulb, Battery, PIR Sensor, Logic State.

Sr No.	Logic State	Result
1.	HIGH	Motion Detected and hence Bulb starts to Glow.
2.	LOW	No Motion Detected and hence, bulb doesn't glow.

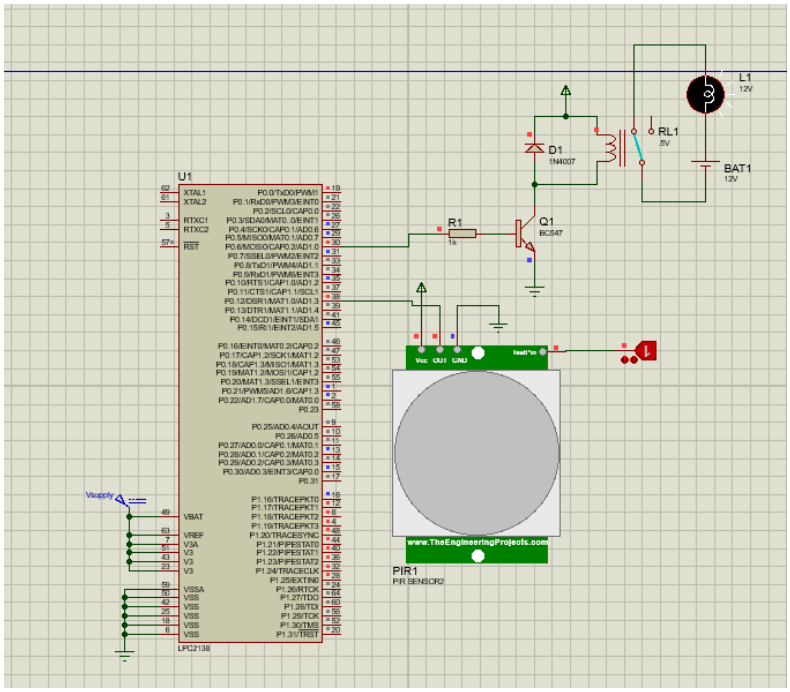

Figure 9.8: Motion Detected

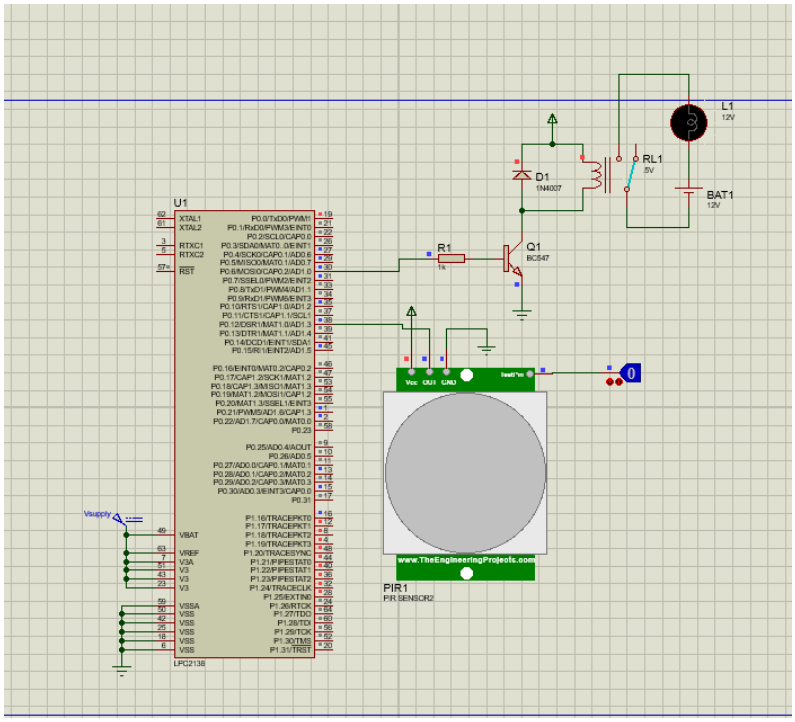

Figure 9.9: No Motion Detected

- **Fan Automation using PIR Sensor (Task 4):**

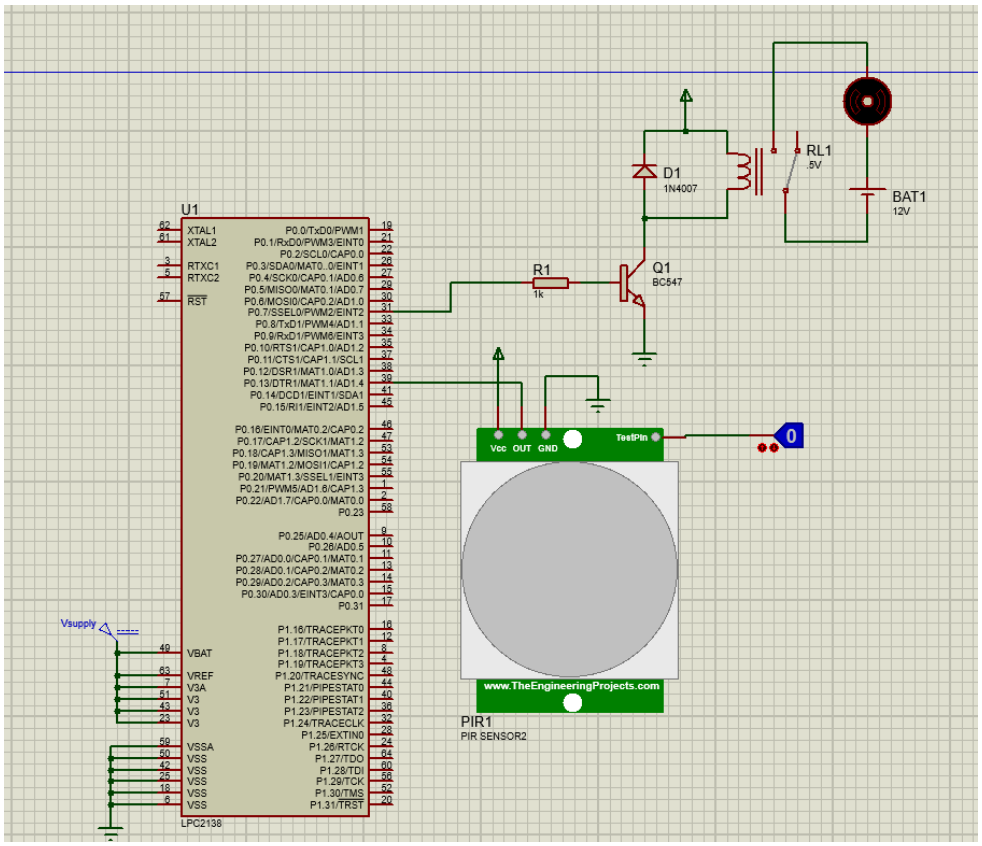

Figure 9.10: Fan Automation using PIR Sensor (Circuit Diagram)

Components Used:

LPC2138/48, Resistor, Transistor (BC547), Diode (1N4007), Relay, DC Motor, Battery. PIR Sensor, Logic State.

Sr No.	Logic State	Result
1.	HIGH	Motion Detected and hence Fan Starts.
2.	LOW	No Motion Detected and hence, Fan Stops.

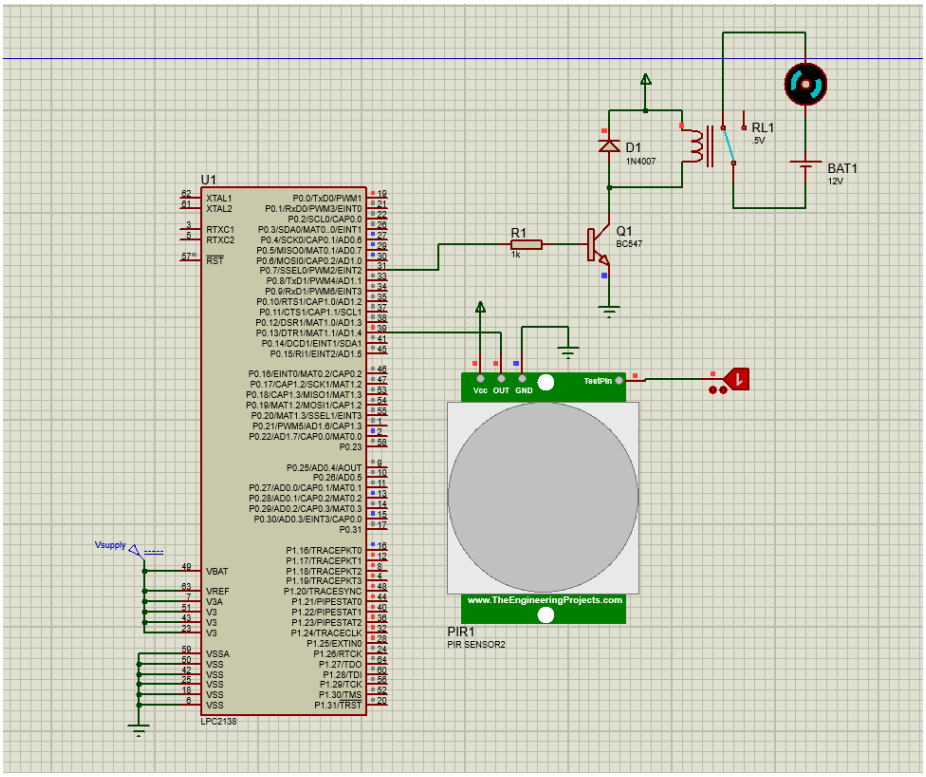

Figure 9.11: Motion Detected

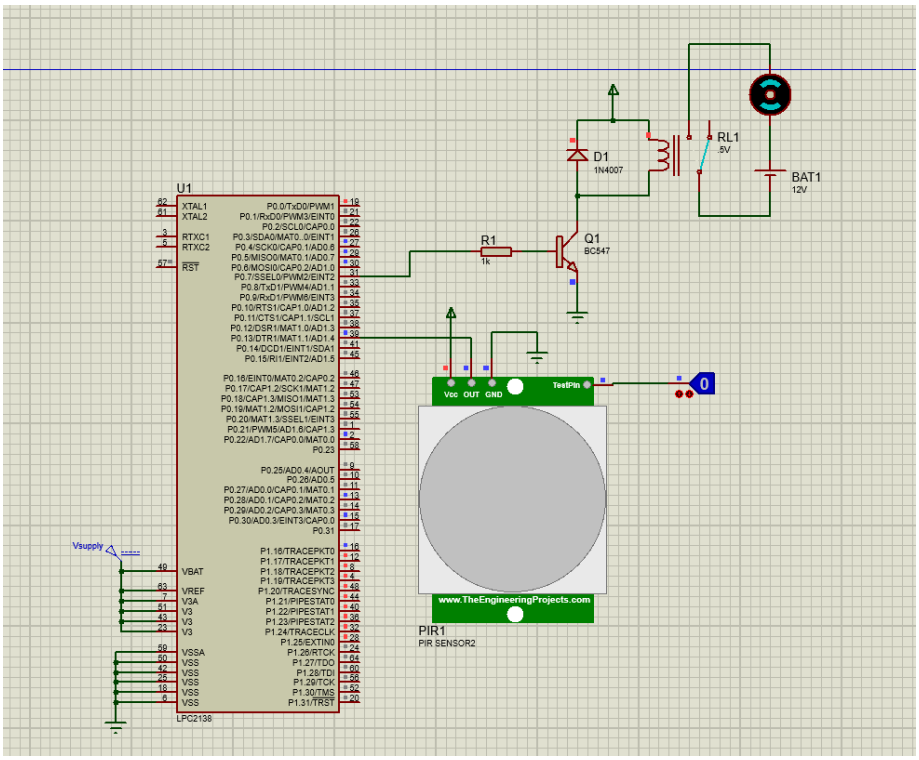

Figure 9.12: No Motion Detected

- **Light Automation using Switch (Task 5):**

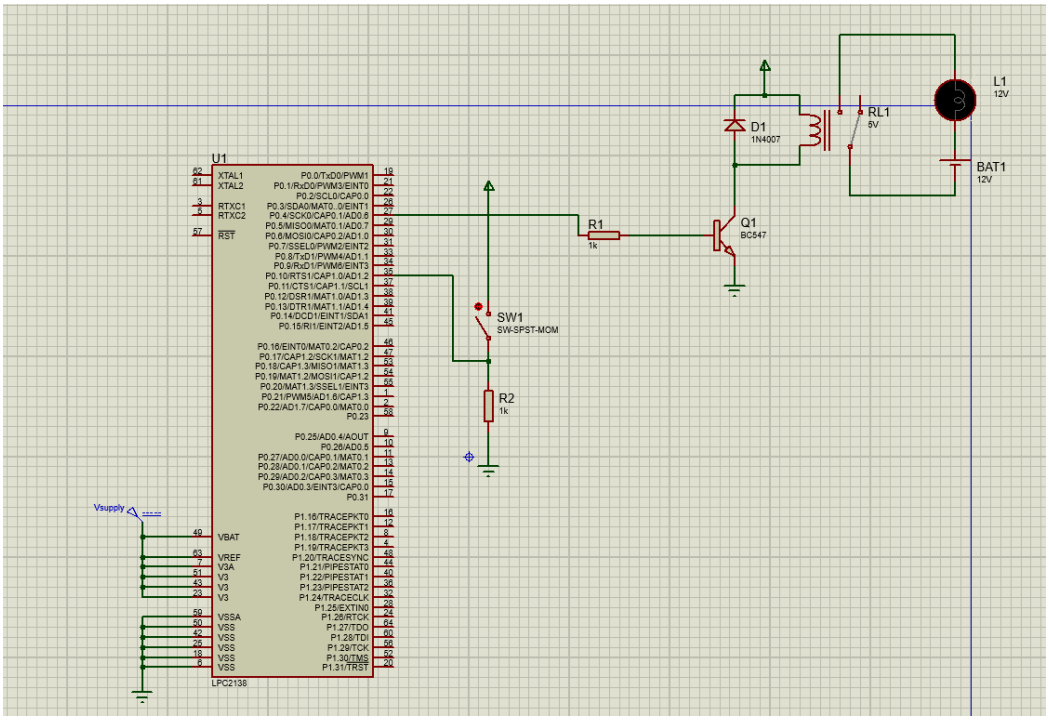

Figure 9.13: Light Automation using Switch (Circuit Diagram)

Components Used:

LPC2138/48, Resistor, Transistor (BC547), Diode (1N4007), Relay, Bulb, Battery, SPST Switch.

Sr No.	Switch Status	Result
1.	Pressed	Bulbs Glows.
2.	Released	Bulb Doesn't Glow.

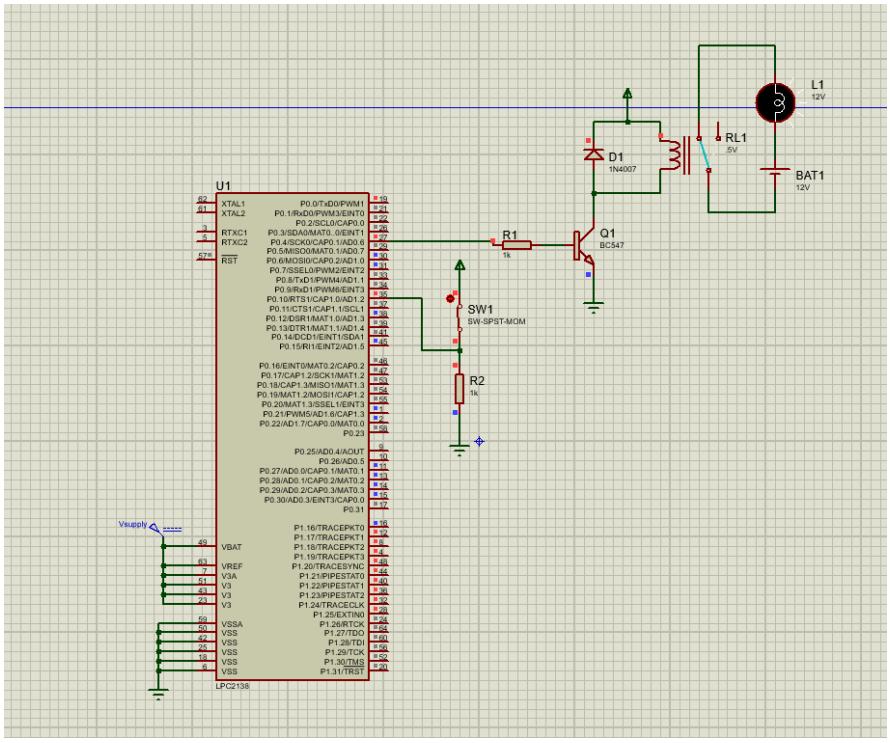

Figure 9.14: Switch Pressed

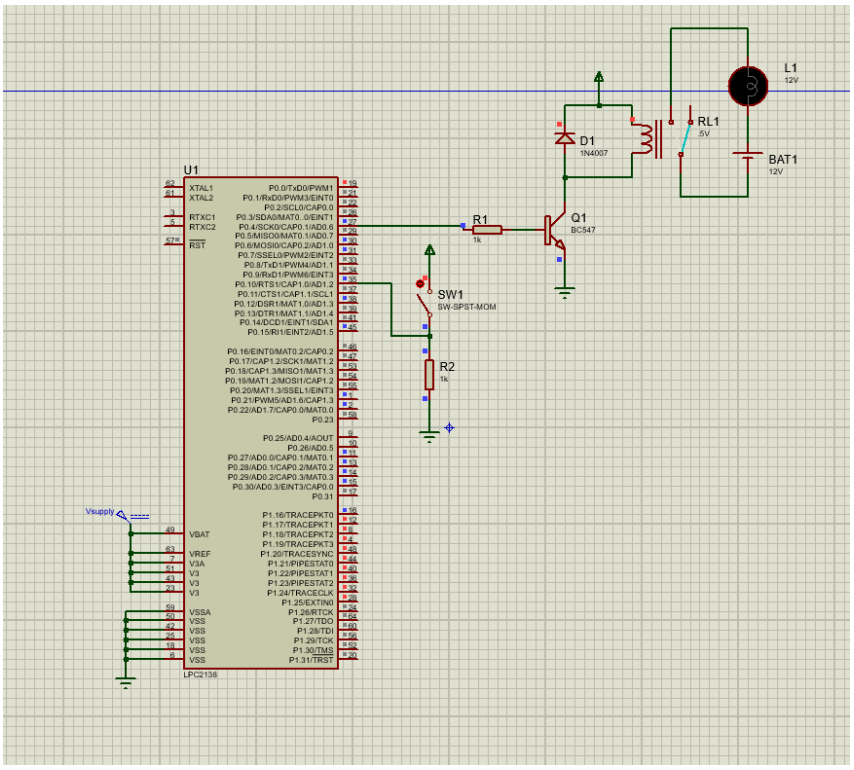

Figure 9.15: Switch Released

- **Fan Automation using Switch (Task 6):**

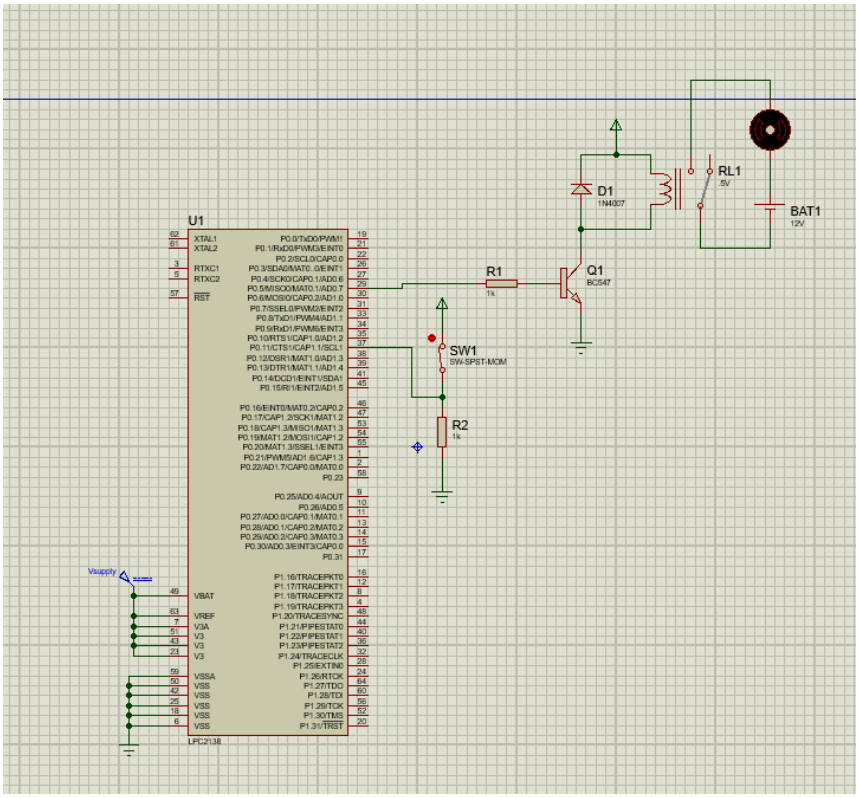

Figure 9.16: Fan Automation using Switch (Circuit Diagram)

Components Used:

LPC2138/48, Resistor, Transistor (BC547), Diode (1N4007), Relay, Fan, Battery, SPST Switch.

Sr No.	Switch Status	Result
1.	Pressed	Fan Starts Glows.
2.	Released	Fan Doesn't Glow.

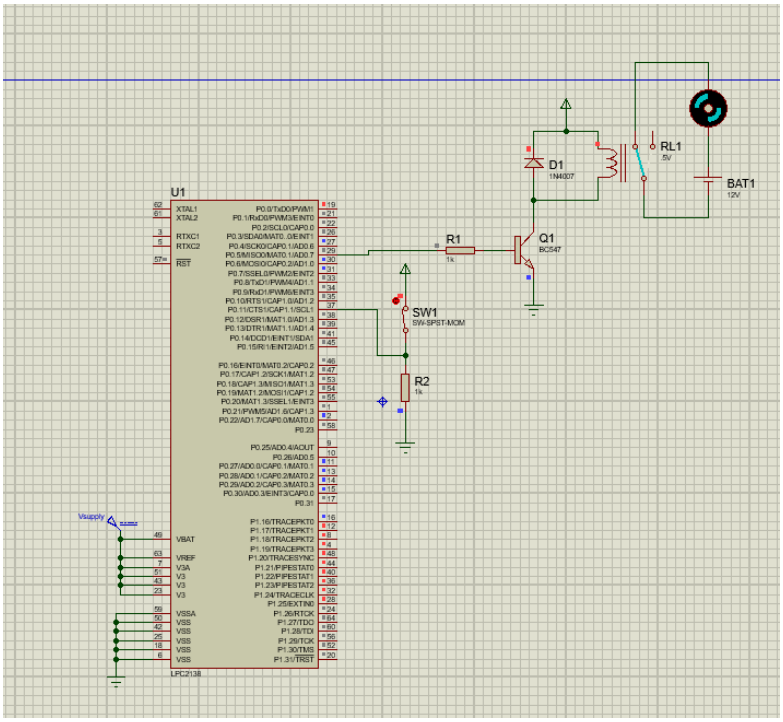

Figure 9.17: Switch Pressed

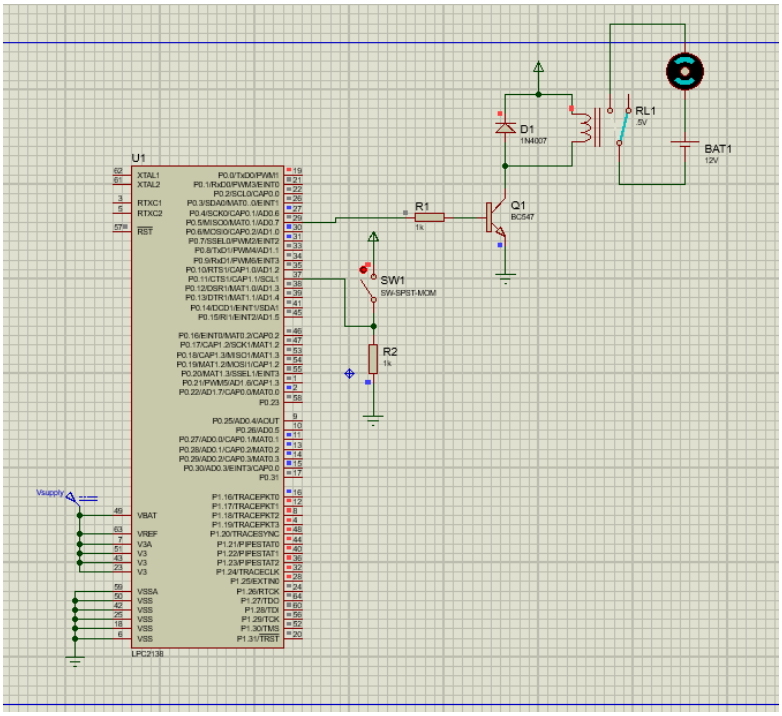

Figure 9.18: Switch Released

- **Task A (Virtual Parallelism):**

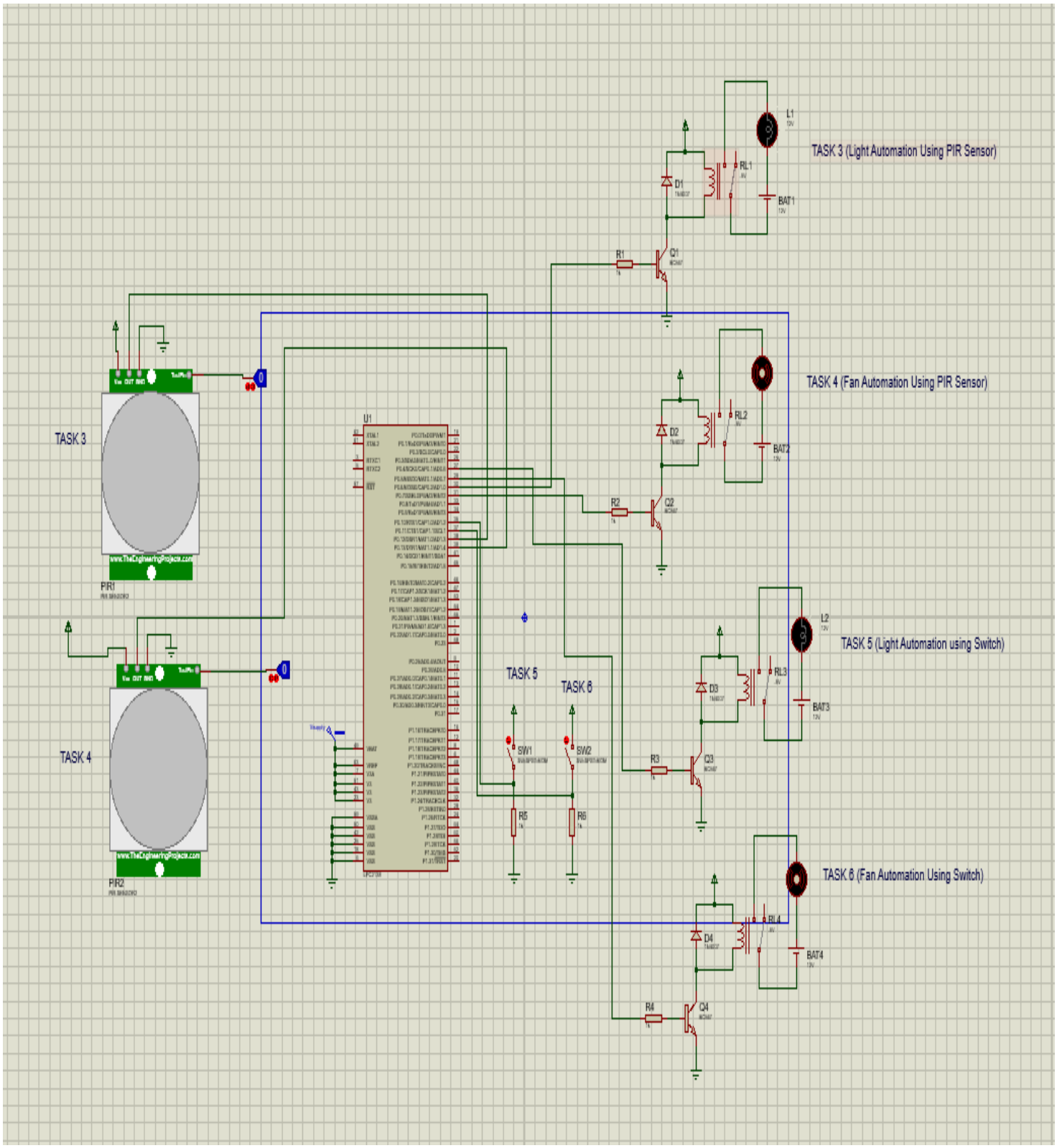

Figure 9.19: Task A (Circuit Diagram)

Sr No.	Task 3 PIR Status	Task 4 PIR Status	Task 5 Switch Status	Task 6 Switch Status	Result
1.	LOW	LOW	Released	Released	No Task Executed.
2.	LOW	LOW	Released	Pressed	Task 6 Executed.
3.	LOW	LOW	Pressed	Released	Task 5 Executed.
4.	LOW	LOW	Pressed	Pressed	Task 5 and Task 6 Executed.
5.	LOW	HIGH	Released	Released	Task 4 Executed.
6.	LOW	HIGH	Released	Pressed	Task 4 and Task 6 Executed.
7.	LOW	HIGH	Pressed	Released	Task 4 and Task 5 Executed.
8.	LOW	HIGH	Pressed	Pressed	Task 4, Task 5 and Task 6 Executed.
9.	HIGH	LOW	Released	Released	Task 3 Executed,
10.	HIGH	LOW	Released	Pressed	Task 3 and Task 6 Executed.
11.	HIGH	LOW	Pressed	Released	Task 3 and Task 5 Executed.
12.	HIGH	LOW	Pressed	Pressed	Task 3, Task 5 and Task 6 Executed.
13.	HIGH	HIGH	Released	Released	Task 3 and Task 4 Executed.
14.	HIGH	HIGH	Released	Pressed	Task 3, Task 4 and Task 6 Executed.
15.	HIGH	HIGH	Pressed	Released	Task 3, Task 4 and Task 5 Executed.
16.	HIGH	HIGH	Pressed	Pressed	All Tasks Executed.

Figure 9.20: Task A Status Diagram

- **Task B (Semaphores):**

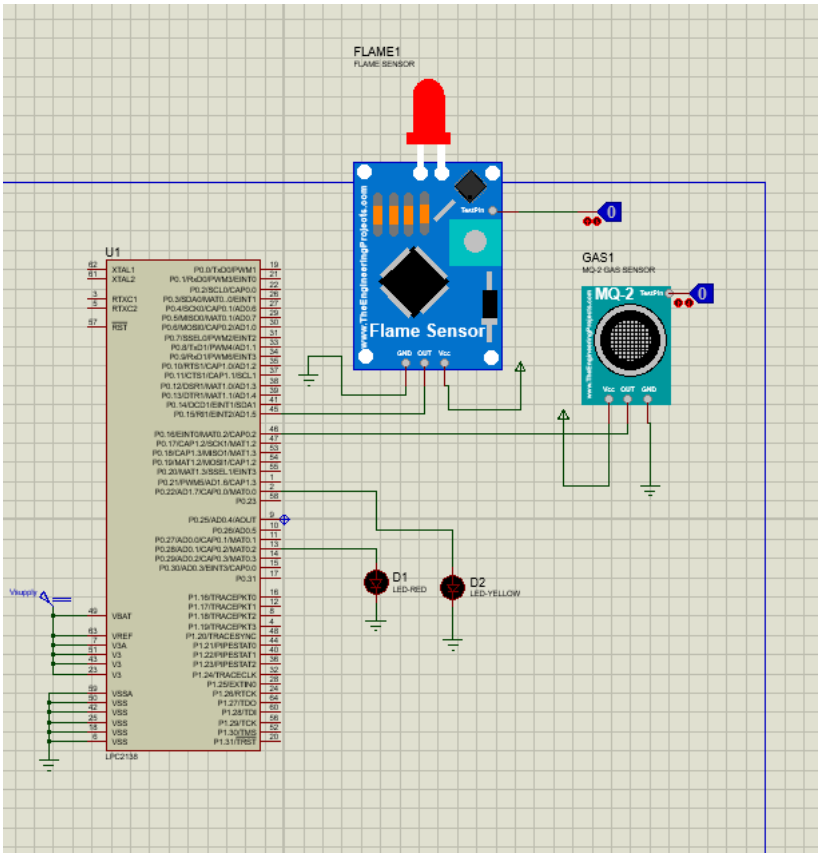

Figure 9.21: Task B (Circuit Diagram)

Sr No.	Task 1 Sensor Status	Task 2 Sensor Status	Result
1.	LOW	LOW	No Message Displayed on UART.
2.	LOW	HIGH	Gas Detected Messaged Displayed on UART and Diode D1 starts to Blink.
3.	HIGH	LOW	Fire Detected Messaged Displayed on UART and Diode D2 starts to Blink.
4.	HIGH	HIGH	First, Fire Detected Message Displayed on UART and Diode D2 starts to Blink, then Gas Detected Message Displayed on UART and Diode D1 starts to Blink. This will be a continuous process.

Figure 9.22: Task B Status Diagram

10. Output

The Expected Output of **Lab Automation and Security System:**

- Enhanced security through real-time fire and gas leak detection, with immediate alerts and alarms.
- Improved energy efficiency through automated lighting and fan control based on occupancy, reducing energy consumption.
- Automated control over lighting and fan operations based on occupancy, ensuring optimal comfort and energy efficiency.

---

11. Result

The result of the implementation of the RTOS in our college lab, integrated with semaphore and tasks priorities, yielded highly successful results. Through careful coding in Keil and Proteus Software, we observed a seamless execution of tasks in strict accordance with their assigned priority levels. High-priority tasks, including gas leakage detection and fire alarm activation, consistently took precedence over lower-priority operations like light and fan automation. This outcome strongly gives the effectiveness of our task prioritization mechanism, implemented through semaphores.

---

12. Conclusion

The result of the implementation of the RTOS in our college lab, integrated with semaphore and tasks priorities, yielded highly successful results. Through careful coding in Keil and Proteus Software, we observed a seamless execution of tasks in strict accordance with their assigned priority levels. High-priority tasks, including gas leakage detection and fire alarm activation, consistently took precedence over lower-priority operations like light and fan automation. This outcome strongly gives the effectiveness of our task prioritization mechanism, implemented through semaphores.

---

13. References

1. *"Home Automation with Arduino: Automate your Home using Open-Source Hardware"*, by Marco Schwartz
2. *"ARM System Developer's Guide: Designing and Optimizing System Software"*, by Andrew Sloss, Dominic Symes and Chris Wright.
