CSE211: Introduction to Embedded Systems

Fall 2024

Project Report

Team Members:

Mazen Saeed Mohamed	21P0125
Youssef Habil Abou Elkhier	21P0187
Sara Mohamed Ashour	21P0337
Mostafa Hassan Mohamed	21p0349
Adel Mohamed Adel	21p0113

TABLE OF CONTENTS

1. Project Description

You are required to build a small prototype for a **desktop-based home appliances control** system that satisfies the following requirements:

1. **Develop a Desktop Application to:**
    a. Control a 220 Volt lamp on/off (**You Shouldn't delete the manual switch**).
    b. Control a plug on/off: If it's off, no one can use the plug.
    c. Display the door status. The status is indicated through a magnetic switch.
    d. Display room temperature.
    e. Trigger an alarm if the temperature goes beyond a set value (**Software Alarm + Physical Alarm**).
    f. Save and show the times of opening and closing the door.
2. **Hardware Setup:**
    - Your components must be mounted on a suitable surface, and all wires should be hidden.
3. **Microcontroller Board:**
    - You should use the **TM4C123GH6PM** board to implement this project.
4. **Desktop Application Features:**
    - The desktop application should have a user-friendly interface to monitor and control the system.
    - Real-time updates for all connected devices.
    - Historical logs for door activity and temperature alarms.
    - Adjustable temperature thresholds for triggering alarms.

2. The Contribution of each member of the Group

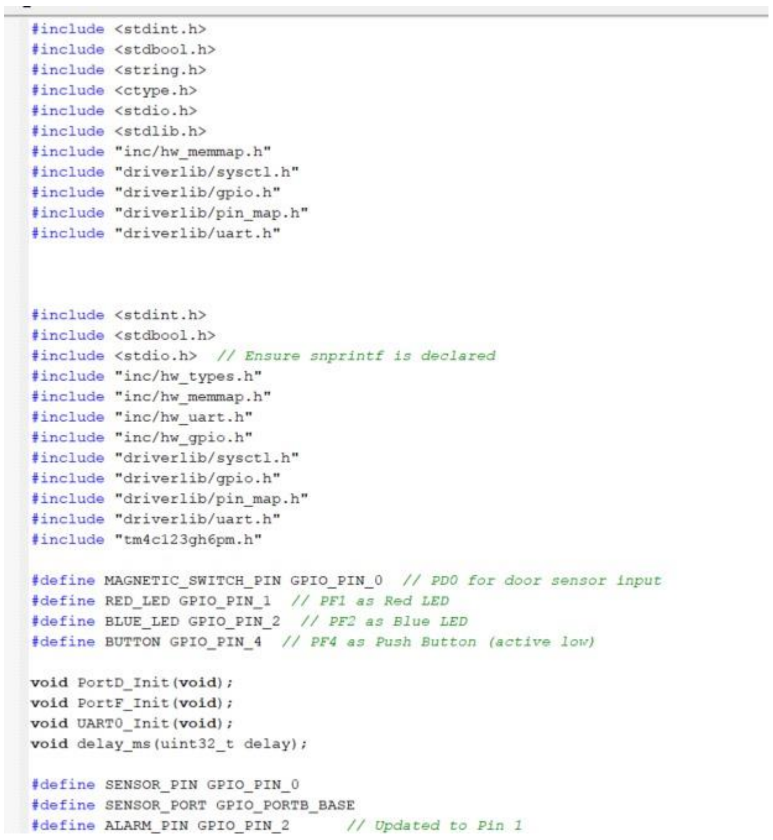
Member Name	**Tasks Completed**
Mazen Saeed	Temperature Sensor
Youssef Habil	Lamp/Plug/Relay
Sara Mohamed	Limit Switch
Mostafa Hassan	Door Sensor
Adel Mohamed	Buzzer

3. Features Overview

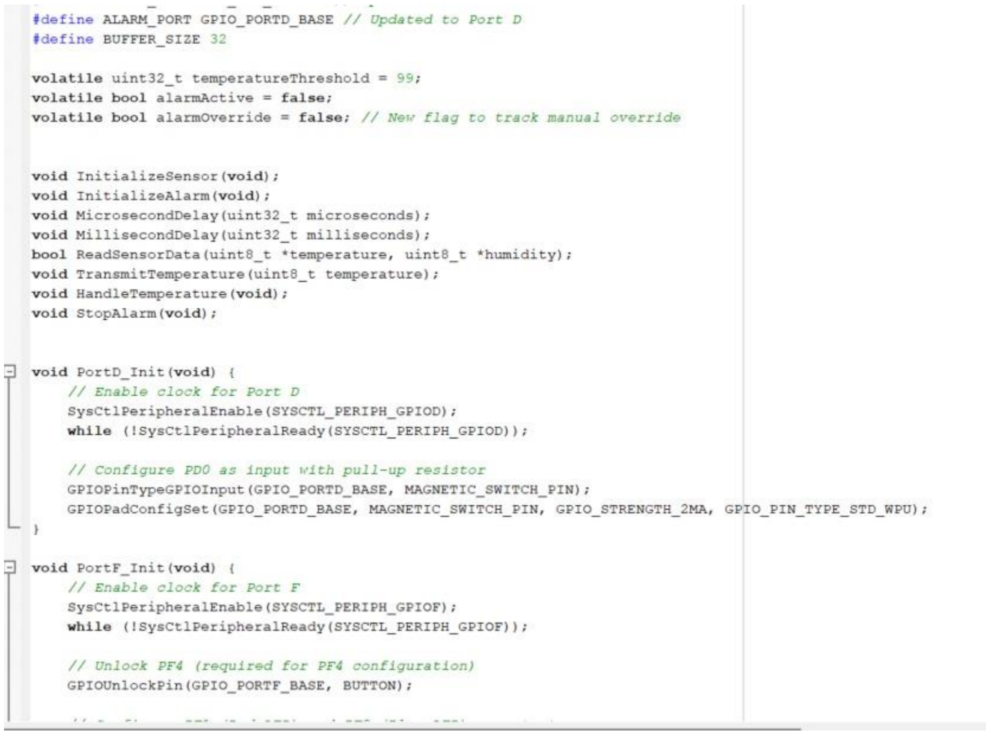
1. **Temperature and Alarm Management**:
    - Reads temperature from a sensor.
    - Activates an alarm if the temperature exceeds the threshold.
    - Deactivates the alarm based on UART command or low temperature.
2. **Door Monitoring**:
    - Reads the state of a magnetic switch (open/closed).
    - Sends door state over UART.
3. **LED Control**:
    - Toggles Red/Blue LEDs via UART or a button press.
4. **UART Communication**:
    - Sends/receives commands to control the system and query states.

4. C Code Description

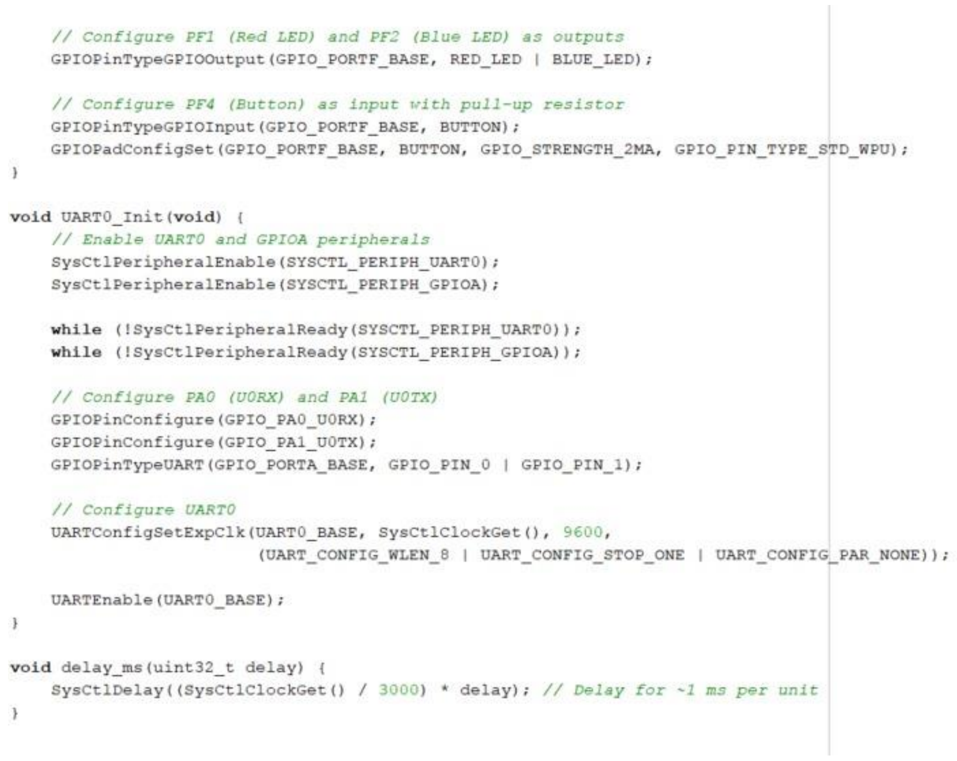
Here is our C code:



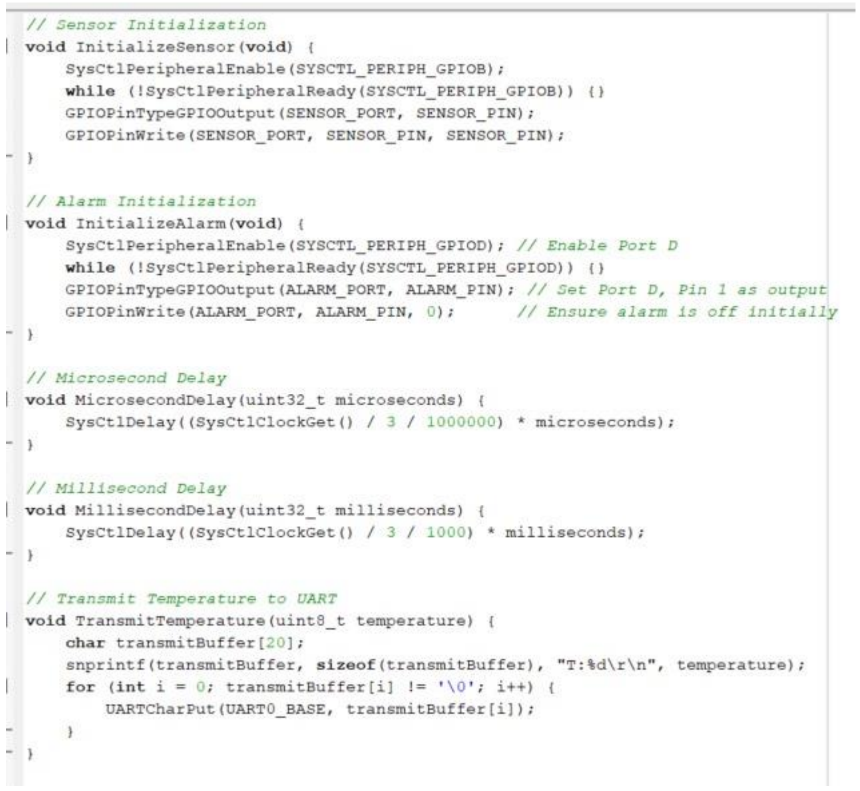
```c
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"



#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>  // Ensure snprintf is declared
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "tm4c123gh6pm.h"

#define MAGNETIC_SWITCH_PIN GPIO_PIN_0  // PD0 for door sensor input
#define RED_LED GPIO_PIN_1  // PF1 as Red LED
#define BLUE_LED GPIO_PIN_2  // PF2 as Blue LED
#define BUTTON GPIO_PIN_4  // PF4 as Push Button (active low)

void PortD_Init(void);
void PortF_Init(void);
void UART0_Init(void);
void delay_ms(uint32_t delay);

#define SENSOR_PIN GPIO_PIN_0
#define SENSOR_PORT GPIO_PORTB_BASE
#define ALARM_PIN GPIO_PIN_2      // Updated to Pin 1
```



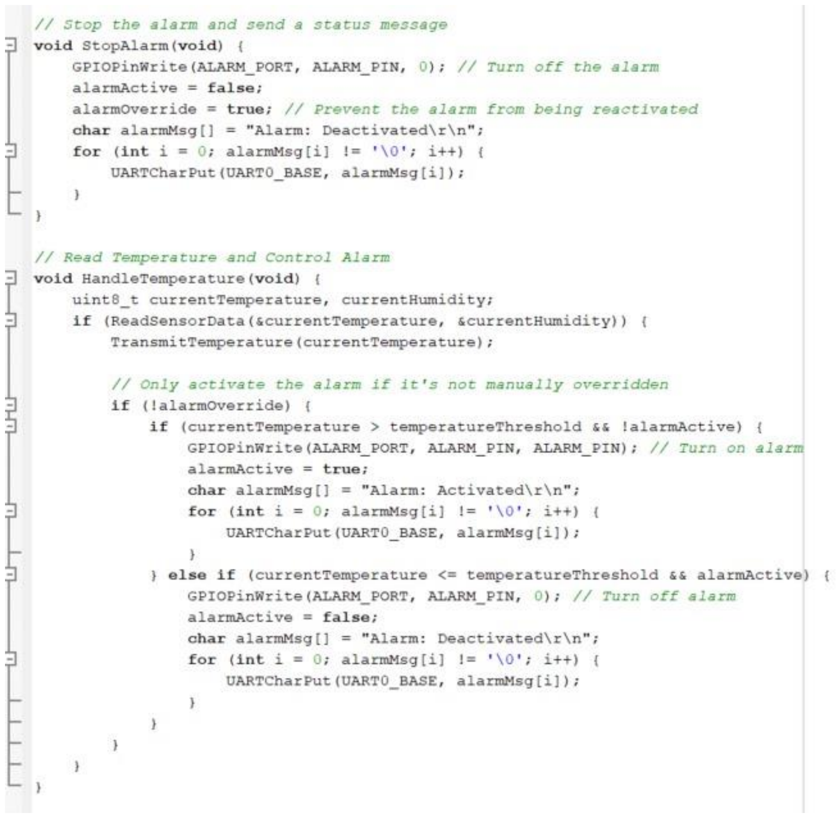
```c
#define ALARM_PORT GPIO_PORTD_BASE // Updated to Port D
#define BUFFER_SIZE 32

volatile uint32_t temperatureThreshold = 99;
volatile bool alarmActive = false;
volatile bool alarmOverride = false; // New flag to track manual override


void InitializeSensor(void);
void InitializeAlarm(void);
void MicrosecondDelay(uint32_t microseconds);
void MillisecondDelay(uint32_t milliseconds);
bool ReadSensorData(uint8_t *temperature, uint8_t *humidity);
void TransmitTemperature(uint8_t temperature);
void HandleTemperature(void);
void StopAlarm(void);


void PortD_Init(void) {
    // Enable clock for Port D
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD));

    // Configure PD0 as input with pull-up resistor
    GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN);
    GPIOPadConfigSet(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}

void PortF_Init(void) {
    // Enable clock for Port F
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));

    // Unlock PF4 (required for PF4 configuration)
    GPIOUnlockPin(GPIO_PORTF_BASE, BUTTON);
```



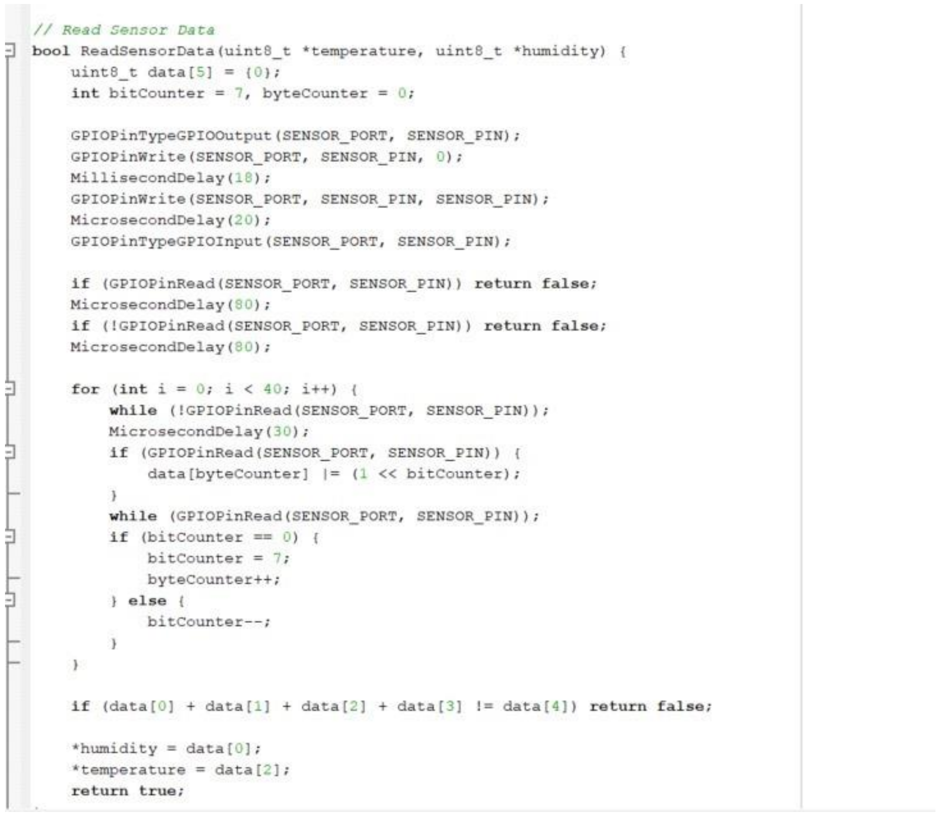
```c
    // Configure PF1 (Red LED) and PF2 (Blue LED) as outputs
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);

    // Configure PF4 (Button) as input with pull-up resistor
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, BUTTON, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}

void UART0_Init(void) {
    // Enable UART0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0));
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));

    // Configure PA0 (U0RX) and PA1 (U0TX)
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Configure UART0
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 9600,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    UARTEnable(UART0_BASE);
}

void delay_ms(uint32_t delay) {
    SysCtlDelay((SysCtlClockGet() / 3000) * delay); // Delay for ~1 ms per unit
}
```



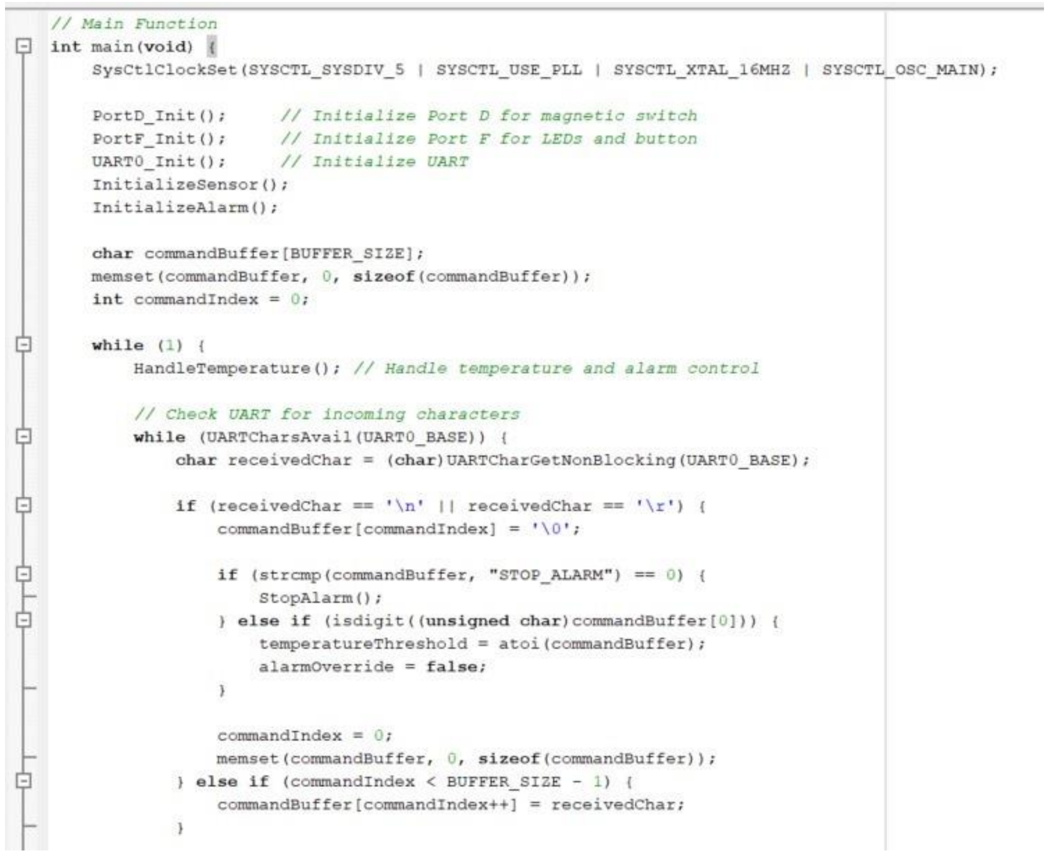
```c
// Sensor Initialization
void InitializeSensor(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB)) {}
    GPIOPinTypeGPIOOutput(SENSOR_PORT, SENSOR_PIN);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, SENSOR_PIN);
}

// Alarm Initialization
void InitializeAlarm(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Enable Port D
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD)) {}
    GPIOPinTypeGPIOOutput(ALARM_PORT, ALARM_PIN); // Set Port D, Pin 1 as output
    GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0);       // Ensure alarm is off initially
}

// Microsecond Delay
void MicrosecondDelay(uint32_t microseconds) {
    SysCtlDelay((SysCtlClockGet() / 3 / 1000000) * microseconds);
}

// Millisecond Delay
void MillisecondDelay(uint32_t milliseconds) {
    SysCtlDelay((SysCtlClockGet() / 3 / 1000) * milliseconds);
}

// Transmit Temperature to UART
void TransmitTemperature(uint8_t temperature) {
    char transmitBuffer[20];
    snprintf(transmitBuffer, sizeof(transmitBuffer), "T:%d\r\n", temperature);
    for (int i = 0; transmitBuffer[i] != '\0'; i++) {
        UARTCharPut(UART0_BASE, transmitBuffer[i]);
    }
}
```



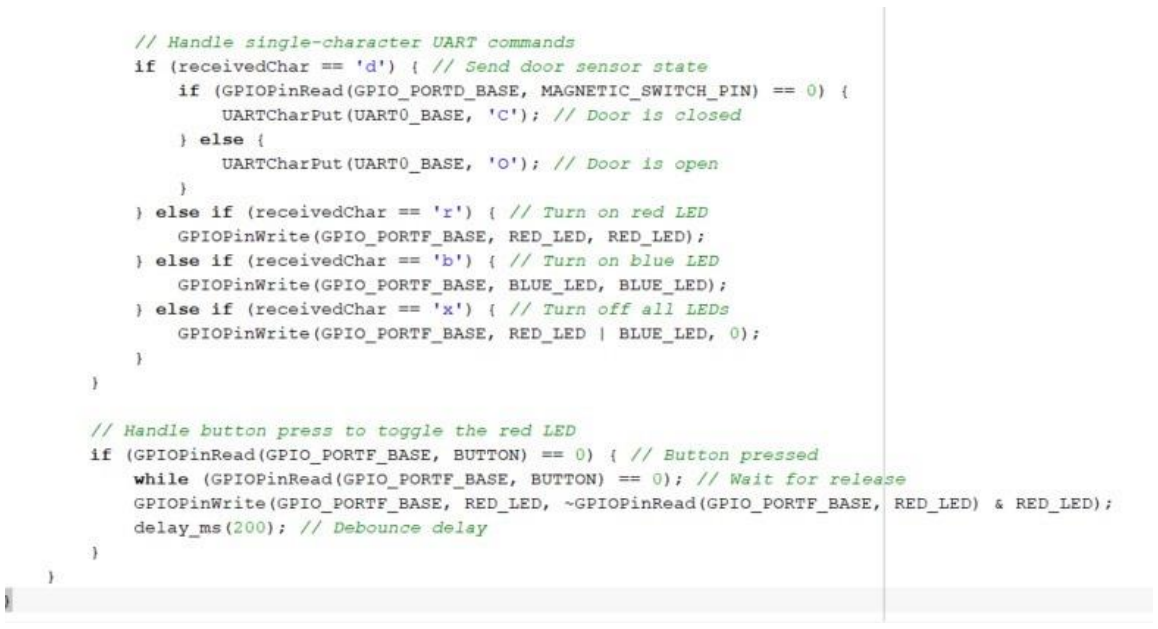
```c
// Stop the alarm and send a status message
void StopAlarm(void) {
    GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0); // Turn off the alarm
    alarmActive = false;
    alarmOverride = true; // Prevent the alarm from being reactivated
    char alarmMsg[] = "Alarm: Deactivated\r\n";
    for (int i = 0; alarmMsg[i] != '\0'; i++) {
        UARTCharPut(UART0_BASE, alarmMsg[i]);
    }
}

// Read Temperature and Control Alarm
void HandleTemperature(void) {
    uint8_t currentTemperature, currentHumidity;
    if (ReadSensorData(&currentTemperature, &currentHumidity)) {
        TransmitTemperature(currentTemperature);

        // Only activate the alarm if it's not manually overridden
        if (!alarmOverride) {
            if (currentTemperature > temperatureThreshold && !alarmActive) {
                GPIOPinWrite(ALARM_PORT, ALARM_PIN, ALARM_PIN); // Turn on alarm
                alarmActive = true;
                char alarmMsg[] = "Alarm: Activated\r\n";
                for (int i = 0; alarmMsg[i] != '\0'; i++) {
                    UARTCharPut(UART0_BASE, alarmMsg[i]);
                }
            } else if (currentTemperature <= temperatureThreshold && alarmActive) {
                GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0); // Turn off alarm
                alarmActive = false;
                char alarmMsg[] = "Alarm: Deactivated\r\n";
                for (int i = 0; alarmMsg[i] != '\0'; i++) {
                    UARTCharPut(UART0_BASE, alarmMsg[i]);
                }
            }
        }
    }
}
```



```c
// Read Sensor Data
bool ReadSensorData(uint8_t *temperature, uint8_t *humidity) {
    uint8_t data[5] = {0};
    int bitCounter = 7, byteCounter = 0;

    GPIOPinTypeGPIOOutput(SENSOR_PORT, SENSOR_PIN);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, 0);
    MillisecondDelay(18);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, SENSOR_PIN);
    MicrosecondDelay(20);
    GPIOPinTypeGPIOInput(SENSOR_PORT, SENSOR_PIN);

    if (GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) return false;
    MicrosecondDelay(80);
    if (!GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) return false;
    MicrosecondDelay(80);

    for (int i = 0; i < 40; i++) {
        while (!GPIOPinRead(SENSOR_PORT, SENSOR_PIN));
        MicrosecondDelay(30);
        if (GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) {
            data[byteCounter] |= (1 << bitCounter);
        }
        while (GPIOPinRead(SENSOR_PORT, SENSOR_PIN));
        if (bitCounter == 0) {
            bitCounter = 7;
            byteCounter++;
        } else {
            bitCounter--;
        }
    }

    if (data[0] + data[1] + data[2] + data[3] != data[4]) return false;

    *humidity = data[0];
    *temperature = data[2];
    return true;
```



```c
// Main Function
int main(void) {
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);

    PortD_Init();     // Initialize Port D for magnetic switch
    PortF_Init();     // Initialize Port F for LEDs and button
    UART0_Init();     // Initialize UART
    InitializeSensor();
    InitializeAlarm();

    char commandBuffer[BUFFER_SIZE];
    memset(commandBuffer, 0, sizeof(commandBuffer));
    int commandIndex = 0;

    while (1) {
        HandleTemperature(); // Handle temperature and alarm control

        // Check UART for incoming characters
        while (UARTCharsAvail(UART0_BASE)) {
            char receivedChar = (char)UARTCharGetNonBlocking(UART0_BASE);

            if (receivedChar == '\n' || receivedChar == '\r') {
                commandBuffer[commandIndex] = '\0';

                if (strcmp(commandBuffer, "STOP_ALARM") == 0) {
                    StopAlarm();
                } else if (isdigit((unsigned char)commandBuffer[0])) {
                    temperatureThreshold = atoi(commandBuffer);
                    alarmOverride = false;
                }

                commandIndex = 0;
                memset(commandBuffer, 0, sizeof(commandBuffer));
            } else if (commandIndex < BUFFER_SIZE - 1) {
                commandBuffer[commandIndex++] = receivedChar;
            }
```



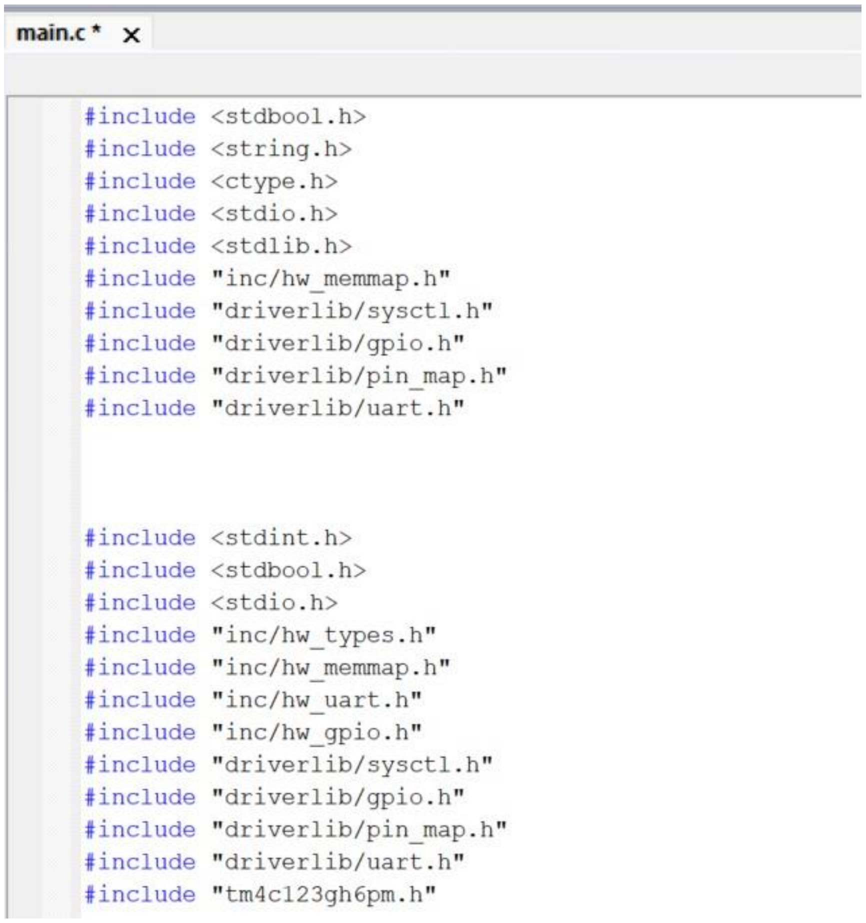
```c
        // Handle single-character UART commands
        if (receivedChar == 'd') { // Send door sensor state
            if (GPIOPinRead(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN) == 0) {
                UARTCharPut(UART0_BASE, 'C'); // Door is closed
            } else {
                UARTCharPut(UART0_BASE, 'O'); // Door is open
            }
        } else if (receivedChar == 'r') { // Turn on red LED
            GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, RED_LED);
        } else if (receivedChar == 'b') { // Turn on blue LED
            GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, BLUE_LED);
        } else if (receivedChar == 'x') { // Turn off all LEDs
            GPIOPinWrite(GPIO_PORTF_BASE, RED_LED | BLUE_LED, 0);
        }
    }

    // Handle button press to toggle the red LED
    if (GPIOPinRead(GPIO_PORTF_BASE, BUTTON) == 0) { // Button pressed
        while (GPIOPinRead(GPIO_PORTF_BASE, BUTTON) == 0); // Wait for release
        GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, ~GPIOPinRead(GPIO_PORTF_BASE, RED_LED) & RED_LED);
        delay_ms(200); // Debounce delay
    }
  }
}
```

3.1 Includes and Definitions

Header Files:

The code uses several header files:

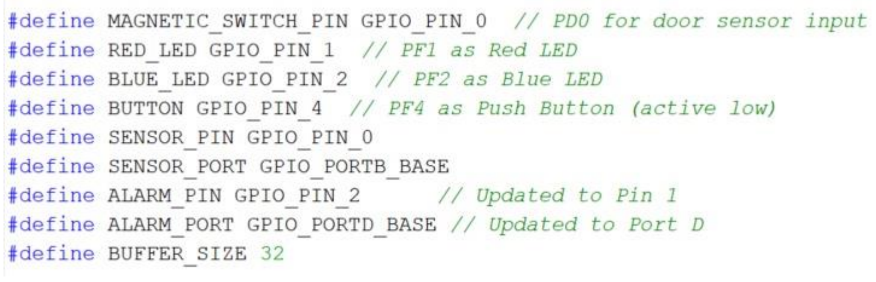
- stdint.h and stdbool.h: Standard libraries for integer types and boolean values.
- string.h, ctype.h, stdio.h, stdlib.h: Libraries for string manipulation, character functions, and I/O operations.
- hw_memmap.h, driverlib/*: Hardware-specific headers for the TM4C123GH6PM microcontroller.
- tm4c123gh6pm.h: Defines hardware registers and addresses specific to the TM4C123GH6PM microcontroller.



```
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"



#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "tm4c123gh6pm.h"
```

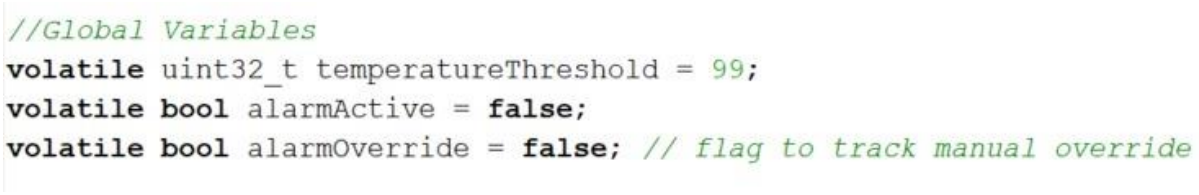
Pin Definitions:

- MAGNETIC_SWITCH_PIN (PD0): Door sensor input.
- RED_LED (PF1): Red LED output.
- BLUE_LED (PF2): Blue LED output.
- BUTTON (PF4): Push button input (active low).
- SENSOR_PIN (PB0): Pin used for temperature sensor communication.
- ALARM_PIN (PD2): Alarm control output.



```
#define MAGNETIC_SWITCH_PIN GPIO_PIN_0  // PD0 for door sensor input
#define RED_LED GPIO_PIN_1  // PF1 as Red LED
#define BLUE_LED GPIO_PIN_2  // PF2 as Blue LED
#define BUTTON GPIO_PIN_4  // PF4 as Push Button (active low)
#define SENSOR_PIN GPIO_PIN_0
#define SENSOR_PORT GPIO_PORTB_BASE
#define ALARM_PIN GPIO_PIN_2      // Updated to Pin 1
#define ALARM_PORT GPIO_PORTD_BASE // Updated to Port D
#define BUFFER_SIZE 32
```

3.2 Global Variables



```
//Global Variables
volatile uint32_t temperatureThreshold = 99;
volatile bool alarmActive = false;
volatile bool alarmOverride = false; // flag to track manual override
```

- **temperatureThreshold:** The maximum temperature before the alarm triggers. Defaults to 99.
- **alarmActive:** Indicates if the alarm is currently active.
- **alarmOverride:** Tracks whether the alarm has been manually disabled to prevent reactivation.

3.3 Peripheral Initialization Functions

1 PortD_Init()

1. Enables the clock for Port D.
2. Configures PD0 (door sensor input) with a pull-up resistor, making it active-low.

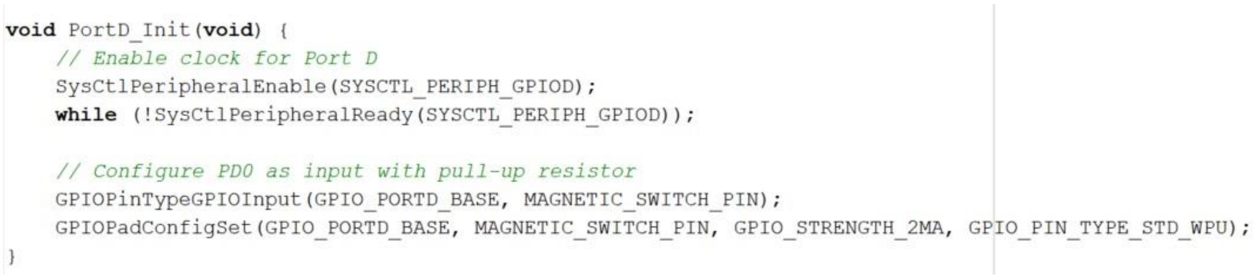
2 PortF_Init()

1. Enables the clock for Port F.
2. Unlocks PF4 (required for buttons on Port F).

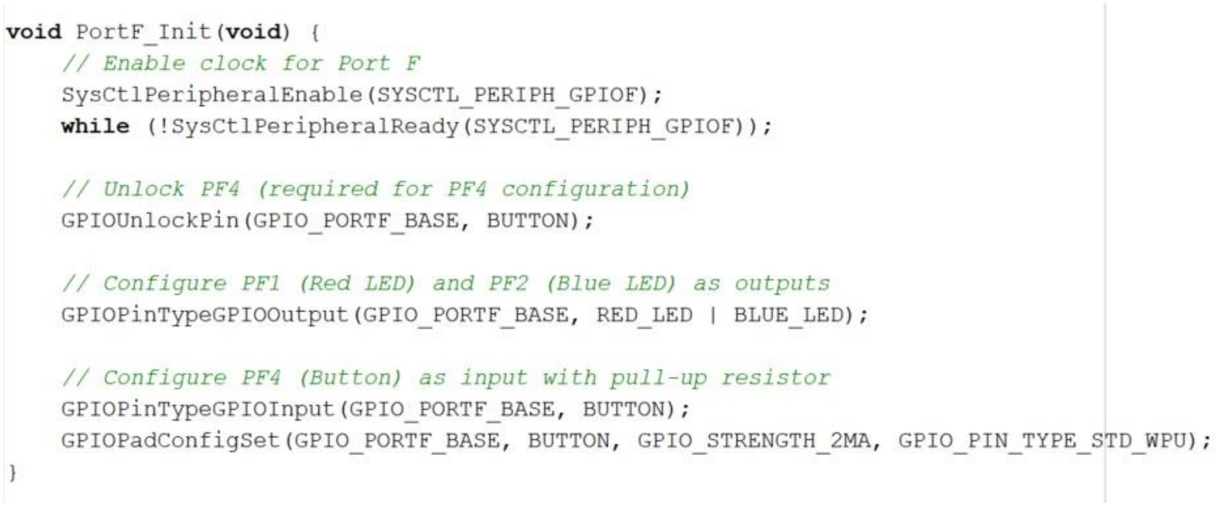
3. Configures:
    - PF1 (Red LED) and PF2 (Blue LED) as outputs.
    - PF4 (button) as an input with a pull-up resistor.

3 UART0_Init()

1. Enables UART0 and Port A.
2. Configures PA0 (RX) and PA1 (TX) for UART functionality.
3. Initializes UART0 with:
    - Baud rate: 9600.
    - 8 data bits, 1 stop bit, no parity.
4. Enables UART communication.



```c
void PortD_Init(void) {
    // Enable clock for Port D
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD));

    // Configure PD0 as input with pull-up resistor
    GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN);
    GPIOPadConfigSet(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}
```



```c
void PortF_Init(void) {
    // Enable clock for Port F
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));

    // Unlock PF4 (required for PF4 configuration)
    GPIOUnlockPin(GPIO_PORTF_BASE, BUTTON);

    // Configure PF1 (Red LED) and PF2 (Blue LED) as outputs
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);

    // Configure PF4 (Button) as input with pull-up resistor
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, BUTTON, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}
```

```c
void UART0_Init(void) {
    // Enable UART0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0));
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));

    // Configure PA0 (U0RX) and PA1 (U0TX)
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Configure UART0
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 9600,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    UARTEnable(UART0_BASE);
}

void delay_ms(uint32_t delay) {
    SysCtlDelay((SysCtlClockGet() / 3000) * delay); // Delay for ~1 ms per unit
}
```

3.4 Helper Functions

```c
// Sensor Initialization
void InitializeSensor(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB)) {}
    GPIOPinTypeGPIOOutput(SENSOR_PORT, SENSOR_PIN);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, SENSOR_PIN);
}

// Alarm Initialization
void InitializeAlarm(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Enable Port D
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD)) {}
    GPIOPinTypeGPIOOutput(ALARM_PORT, ALARM_PIN); // Set Port D, Pin 1 as output
    GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0);       // Ensure alarm is off initially
}

// Microsecond Delay
void MicrosecondDelay(uint32_t microseconds) {
    SysCtlDelay((SysCtlClockGet() / 3 / 1000000) * microseconds);
}

// Millisecond Delay
void MillisecondDelay(uint32_t milliseconds) {
    SysCtlDelay((SysCtlClockGet() / 3 / 1000) * milliseconds);
}
```

InitializeSensor()

1. **Enables the clock for Port B.**
2. **Configures PB0 as an output and sets it high initially.**

InitializeAlarm()

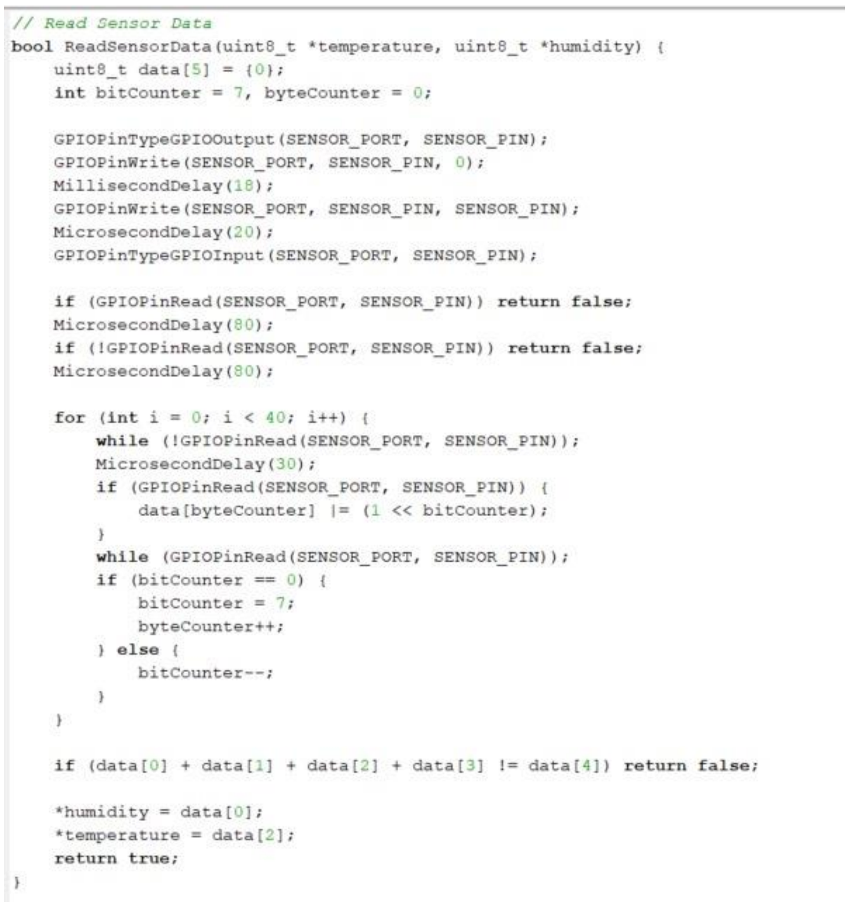
1. **Enables the clock for Port D.**
2. **Configures PD2 as an output and ensures the alarm is off initially.**

MicrosecondDelay() and MillisecondDelay():

- **Create precise delays by calculating cycles based on the system clock.**

3.5 Sensor Communication

ReadSensorData():



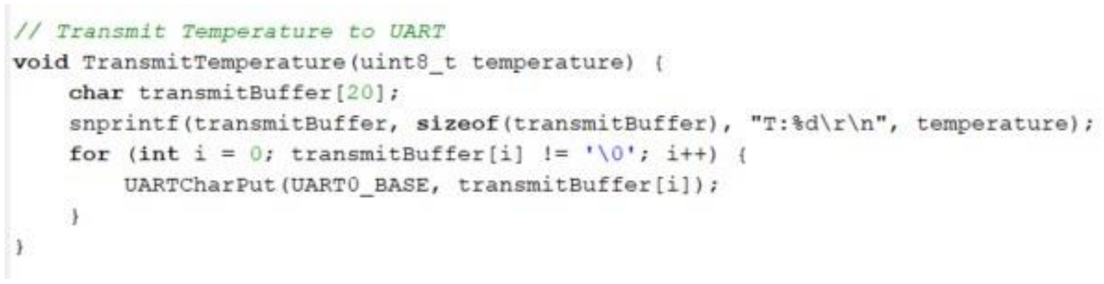
```
// Read Sensor Data
bool ReadSensorData(uint8_t *temperature, uint8_t *humidity) {
    uint8_t data[5] = {0};
    int bitCounter = 7, byteCounter = 0;

    GPIOPinTypeGPIOOutput(SENSOR_PORT, SENSOR_PIN);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, 0);
    MillisecondDelay(18);
    GPIOPinWrite(SENSOR_PORT, SENSOR_PIN, SENSOR_PIN);
    MicrosecondDelay(20);
    GPIOPinTypeGPIOInput(SENSOR_PORT, SENSOR_PIN);

    if (GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) return false;
    MicrosecondDelay(80);
    if (!GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) return false;
    MicrosecondDelay(80);

    for (int i = 0; i < 40; i++) {
        while (!GPIOPinRead(SENSOR_PORT, SENSOR_PIN));
        MicrosecondDelay(30);
        if (GPIOPinRead(SENSOR_PORT, SENSOR_PIN)) {
            data[byteCounter] |= (1 << bitCounter);
        }
        while (GPIOPinRead(SENSOR_PORT, SENSOR_PIN));
        if (bitCounter == 0) {
            bitCounter = 7;
            byteCounter++;
        } else {
            bitCounter--;
        }
    }

    if (data[0] + data[1] + data[2] + data[3] != data[4]) return false;

    *humidity = data[0];
    *temperature = data[2];
    return true;
}
```

This function reads temperature and humidity from a DHT sensor:

1. Sends a start signal:
   - Drives PB0 low for 18 ms to initiate communication.
   - Pulls it high for 20 µs, then switches to input mode.
2. Waits for the sensor's response.
3. Reads 40 bits of data:
   - 8 bits each for humidity integer, humidity decimal, temperature integer, temperature decimal, and checksum.
4. Validates the checksum.
5. Returns true if successful, along with temperature and humidity values.

3.6 UART Communication

TransmitTemperature()

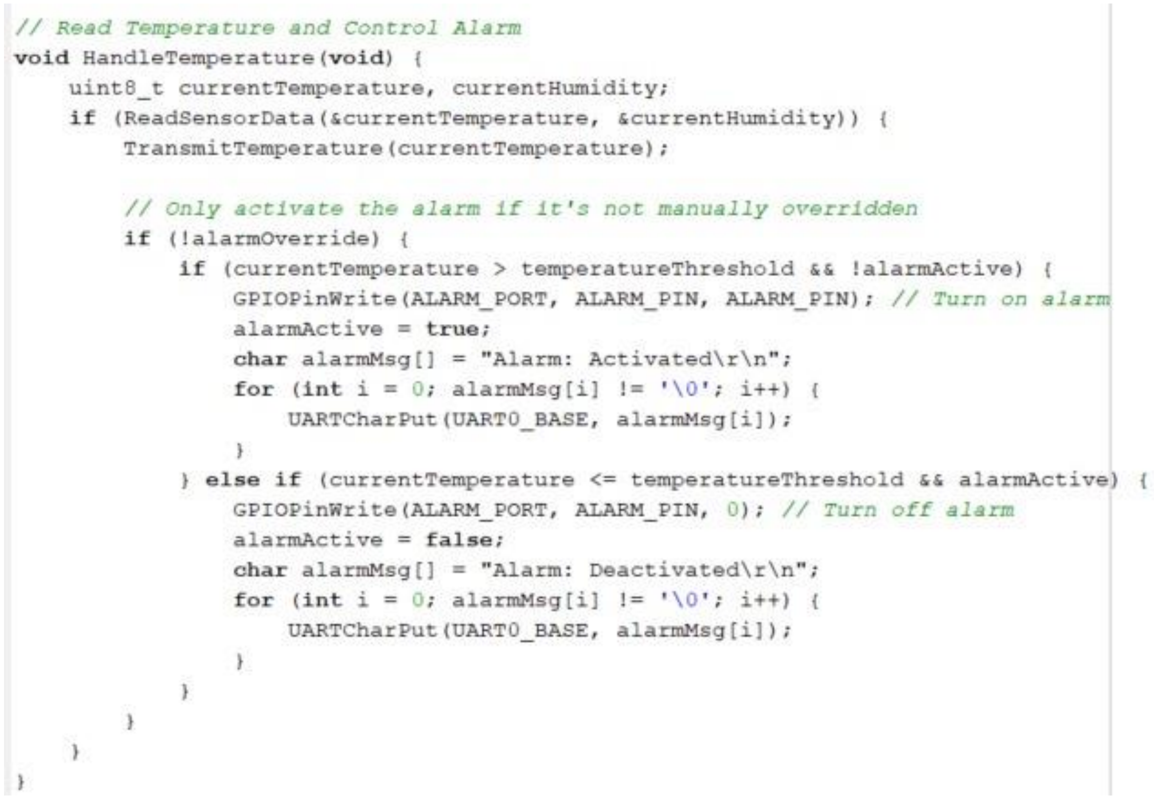


```
// Transmit Temperature to UART
void TransmitTemperature(uint8_t temperature) {
    char transmitBuffer[20];
    snprintf(transmitBuffer, sizeof(transmitBuffer), "T:%d\r\n", temperature);
    for (int i = 0; transmitBuffer[i] != '\0'; i++) {
        UARTCharPut(UART0_BASE, transmitBuffer[i]);
    }
}
```

1. Formats the temperature into a string (e.g., "T:25\r\n").
2. Sends the string byte-by-byte via UART.

3.7 Alarm and Temperature Control

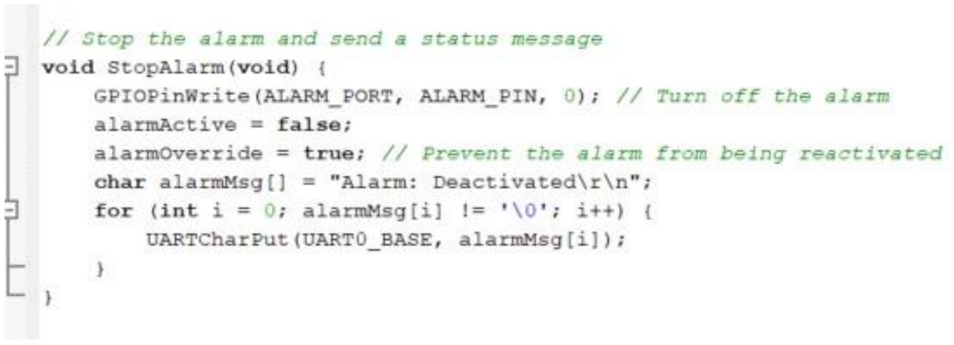
HandleTemperature()



```
// Read Temperature and Control Alarm
void HandleTemperature(void) {
    uint8_t currentTemperature, currentHumidity;
    if (ReadSensorData(&currentTemperature, &currentHumidity)) {
        TransmitTemperature(currentTemperature);

        // Only activate the alarm if it's not manually overridden
        if (!alarmOverride) {
            if (currentTemperature > temperatureThreshold && !alarmActive) {
                GPIOPinWrite(ALARM_PORT, ALARM_PIN, ALARM_PIN); // Turn on alarm
                alarmActive = true;
                char alarmMsg[] = "Alarm: Activated\r\n";
                for (int i = 0; alarmMsg[i] != '\0'; i++) {
                    UARTCharPut(UART0_BASE, alarmMsg[i]);
                }
            } else if (currentTemperature <= temperatureThreshold && alarmActive) {
                GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0); // Turn off alarm
                alarmActive = false;
                char alarmMsg[] = "Alarm: Deactivated\r\n";
                for (int i = 0; alarmMsg[i] != '\0'; i++) {
                    UARTCharPut(UART0_BASE, alarmMsg[i]);
                }
            }
        }
    }
}
```

1. Reads the current temperature and humidity using ReadSensorData().
2. Transmits the temperature via UART.
3. Alarm Logic:
    - Activates the alarm if the temperature exceeds the threshold, provided it hasn't been overridden.
    - Deactivates the alarm if the temperature falls below the threshold.

StopAlarm()



```
// Stop the alarm and send a status message
void StopAlarm(void) {
    GPIOPinWrite(ALARM_PORT, ALARM_PIN, 0); // Turn off the alarm
    alarmActive = false;
    alarmOverride = true; // Prevent the alarm from being reactivated
    char alarmMsg[] = "Alarm: Deactivated\r\n";
    for (int i = 0; alarmMsg[i] != '\0'; i++) {
        UARTCharPut(UART0_BASE, alarmMsg[i]);
    }
}
```

1. Turns off the alarm.
2. Sends an alarm deactivation message via UART.
3. Sets alarmOverride to true, preventing further activation.

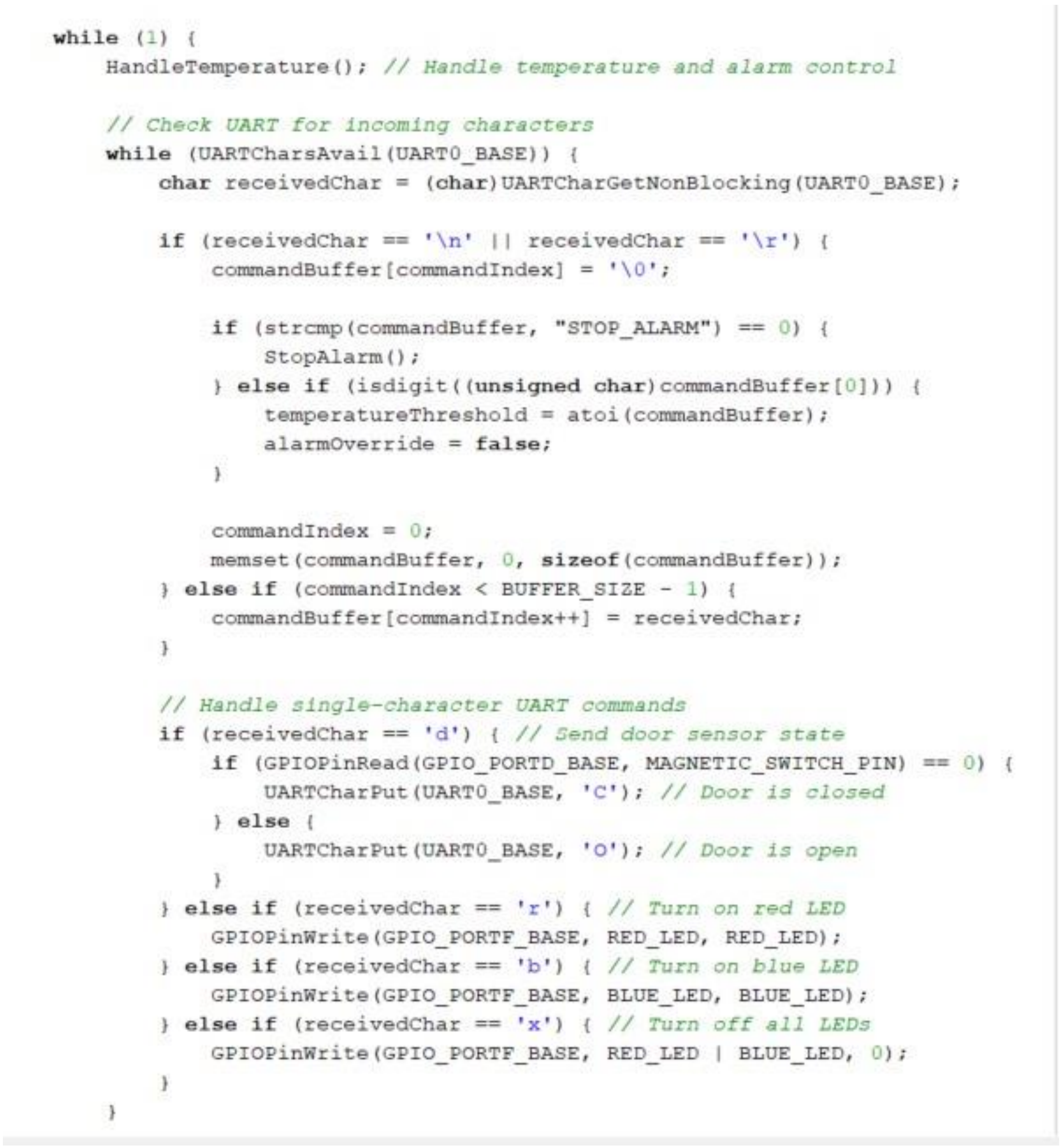
3.8 Main Loop

Initialization

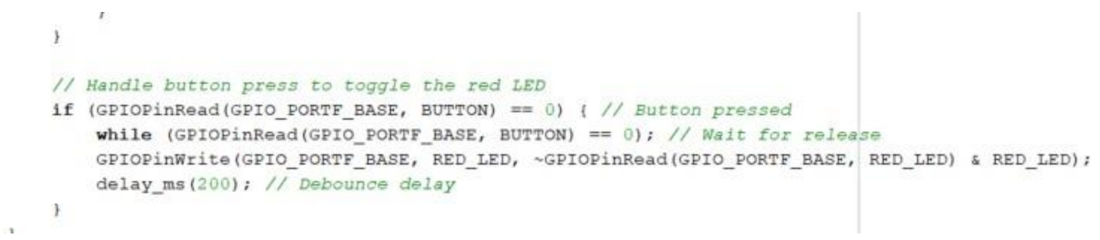
1. Configures system clock for 40 MHz operation.
2. Initializes peripherals:
    - Port D (door sensor), Port F (LEDs, button).
    - UART0, sensor, and alarm.

Infinite Loop

1. Temperature Monitoring:
    - Continuously checks the temperature using HandleTemperature().
2. UART Commands:
    - Handles multi-character commands like:
        - "STOP_ALARM": Disables the alarm and sets the override flag.
        - Numeric commands: Updates the temperatureThreshold.
    - Handles single-character commands:
        - 'd': Sends door sensor status ('C' for closed, 'O' for open).
        - 'r': Turns on the red LED.
        - 'b': Turns on the blue LED.
        - 'x': Turns off all LEDs.
3. Button Handling:
    - Toggles the red LED when the button is pressed.

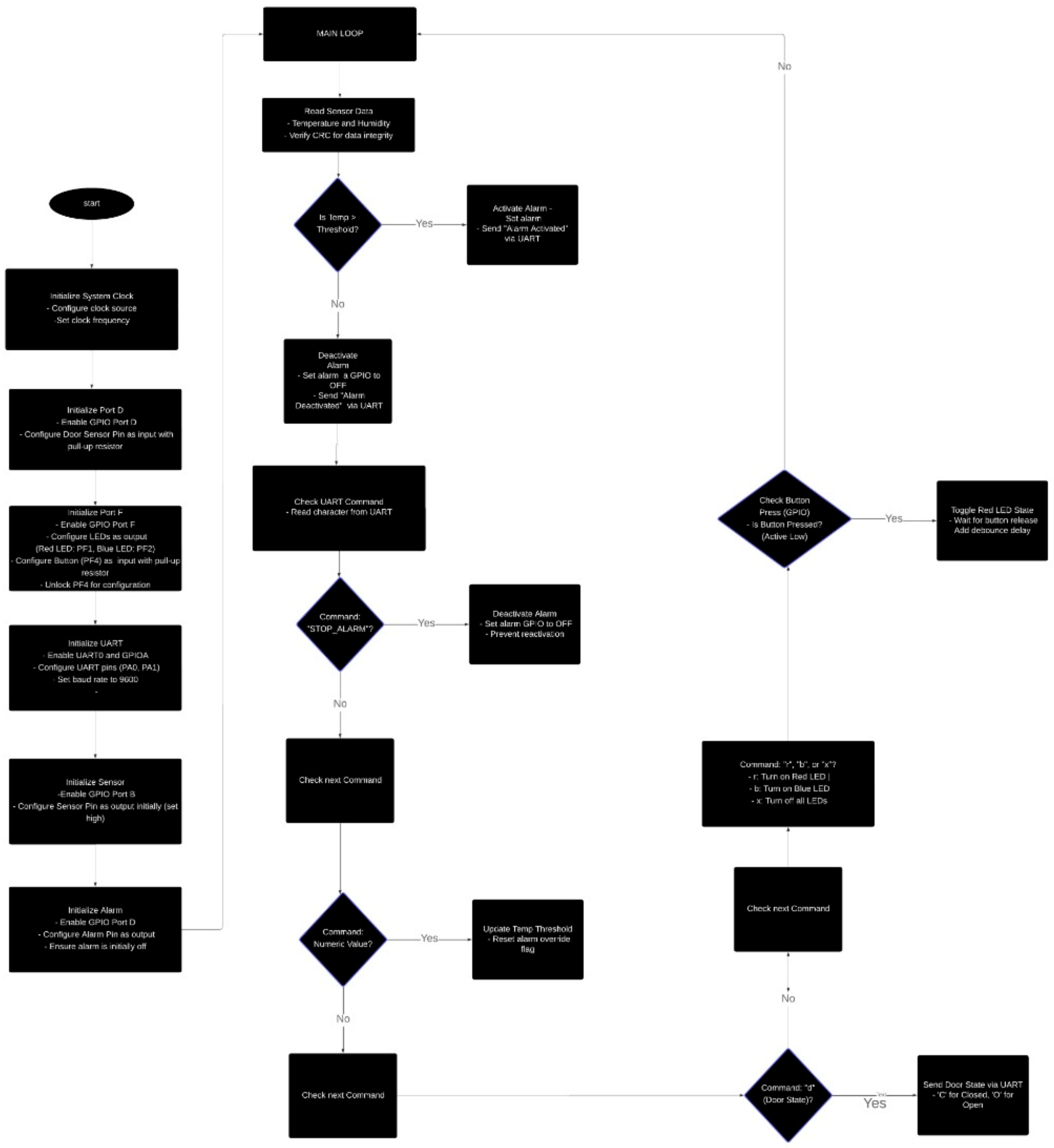


```c
while (1) {
    HandleTemperature(); // Handle temperature and alarm control

    // Check UART for incoming characters
    while (UARTCharsAvail(UART0_BASE)) {
        char receivedChar = (char)UARTCharGetNonBlocking(UART0_BASE);

        if (receivedChar == '\n' || receivedChar == '\r') {
            commandBuffer[commandIndex] = '\0';

            if (strcmp(commandBuffer, "STOP_ALARM") == 0) {
                StopAlarm();
            } else if (isdigit((unsigned char)commandBuffer[0])) {
                temperatureThreshold = atoi(commandBuffer);
                alarmOverride = false;
            }

            commandIndex = 0;
            memset(commandBuffer, 0, sizeof(commandBuffer));
        } else if (commandIndex < BUFFER_SIZE - 1) {
            commandBuffer[commandIndex++] = receivedChar;
        }

        // Handle single-character UART commands
        if (receivedChar == 'd') { // Send door sensor state
            if (GPIOPinRead(GPIO_PORTD_BASE, MAGNETIC_SWITCH_PIN) == 0) {
                UARTCharPut(UART0_BASE, 'C'); // Door is closed
            } else {
                UARTCharPut(UART0_BASE, 'O'); // Door is open
            }
        } else if (receivedChar == 'r') { // Turn on red LED
            GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, RED_LED);
        } else if (receivedChar == 'b') { // Turn on blue LED
            GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, BLUE_LED);
        } else if (receivedChar == 'x') { // Turn off all LEDs
            GPIOPinWrite(GPIO_PORTF_BASE, RED_LED | BLUE_LED, 0);
        }
    }
```



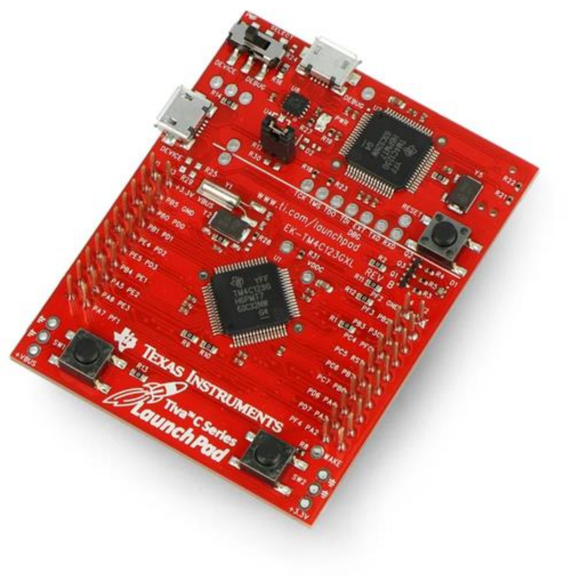
```c
    }

    // Handle button press to toggle the red LED
    if (GPIOPinRead(GPIO_PORTF_BASE, BUTTON) == 0) { // Button pressed
        while (GPIOPinRead(GPIO_PORTF_BASE, BUTTON) == 0); // Wait for release
        GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, ~GPIOPinRead(GPIO_PORTF_BASE, RED_LED) & RED_LED);
        delay_ms(200); // Debounce delay
    }
```

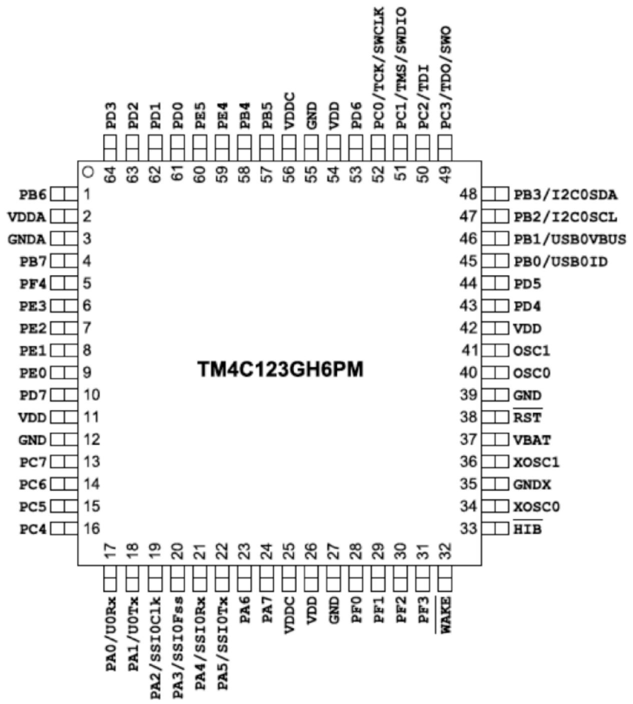
5. Flow Charts of the main Flow of the Program



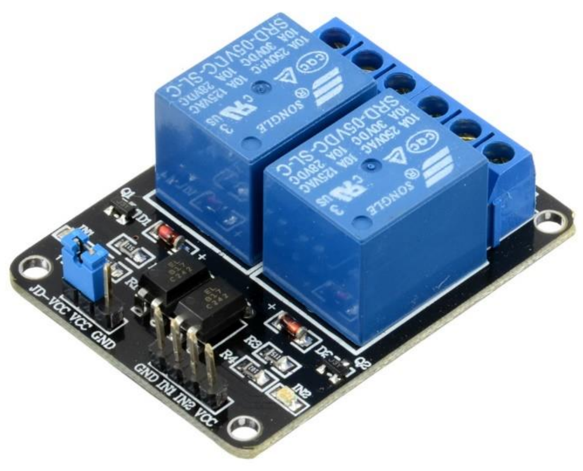
6. List of Components

Tiva™ TM4C123GH6PM Microcontroller:





Relay Module (2 channels):

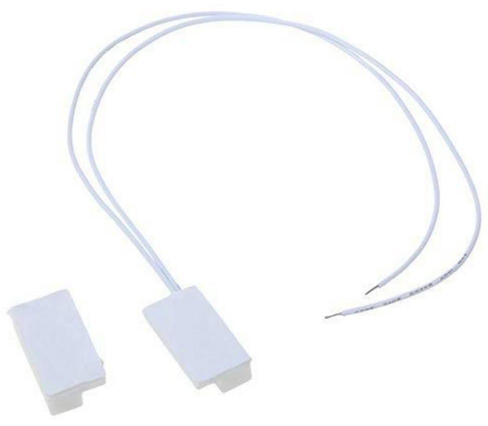


Adaptor 5V/2A:

DC female power plug:

Limit Switch:

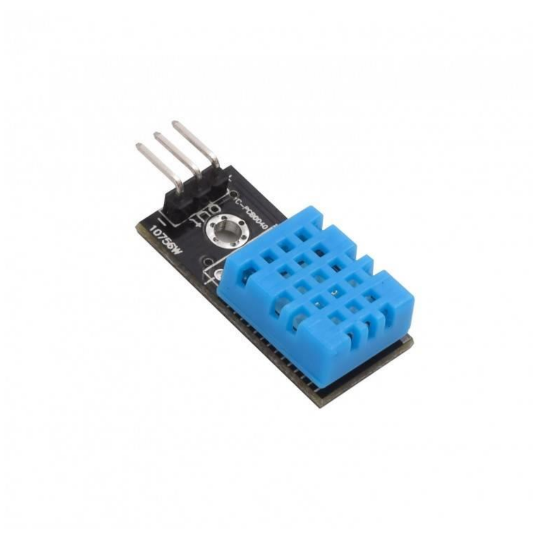
Magnetic Door Switch



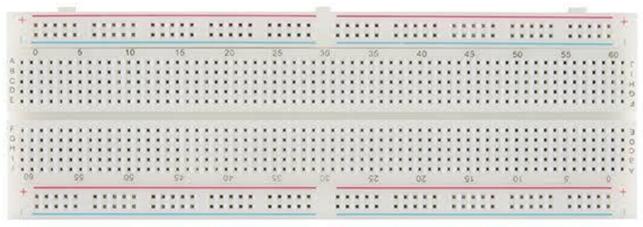
Buzzer:

Male and Female Plugs:

Humidity Temperature Sensor DHT11:



Bread board:



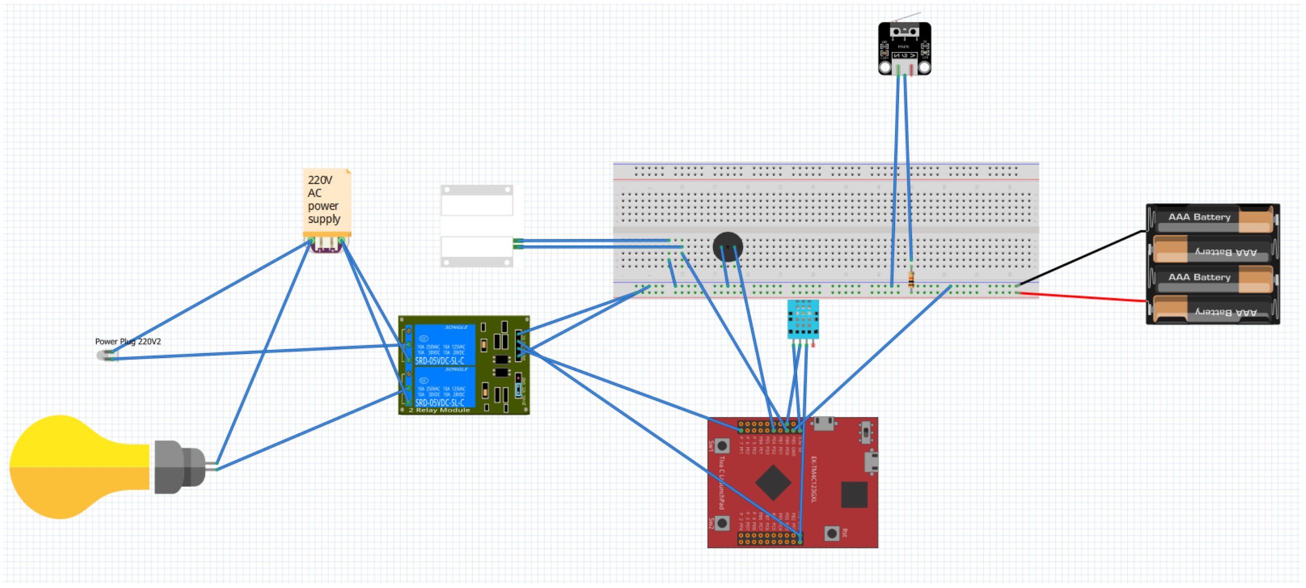
Jumpers:

Lamp:

Lamp Holder:

7. Wiring Diagram

Here is our whole circuit:



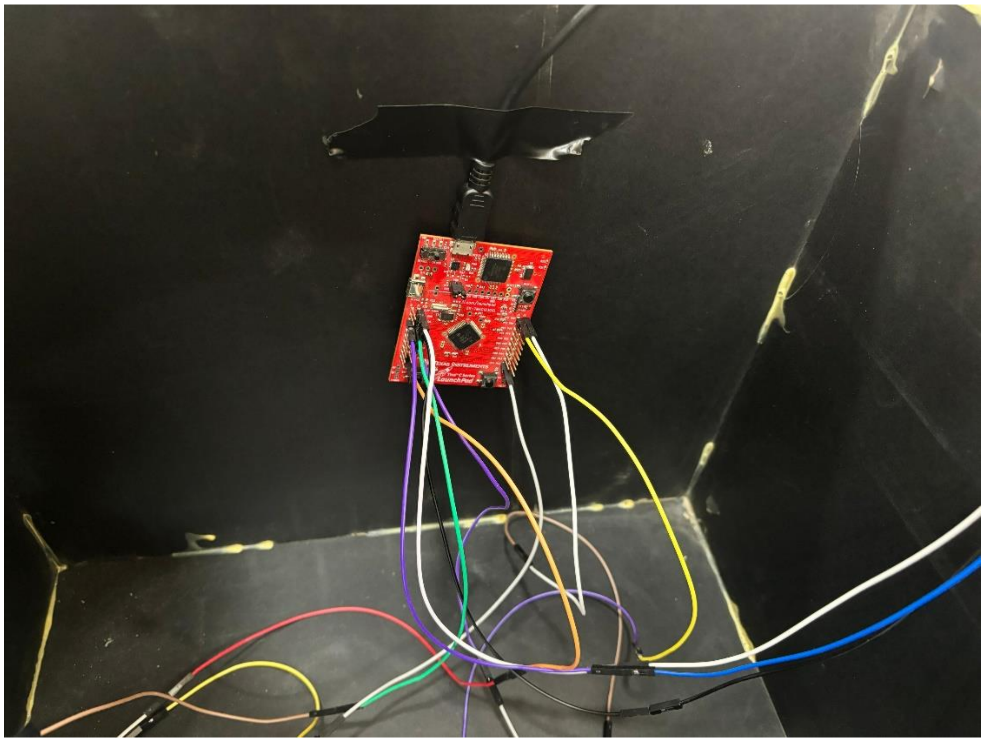
Explanation of the Circuit

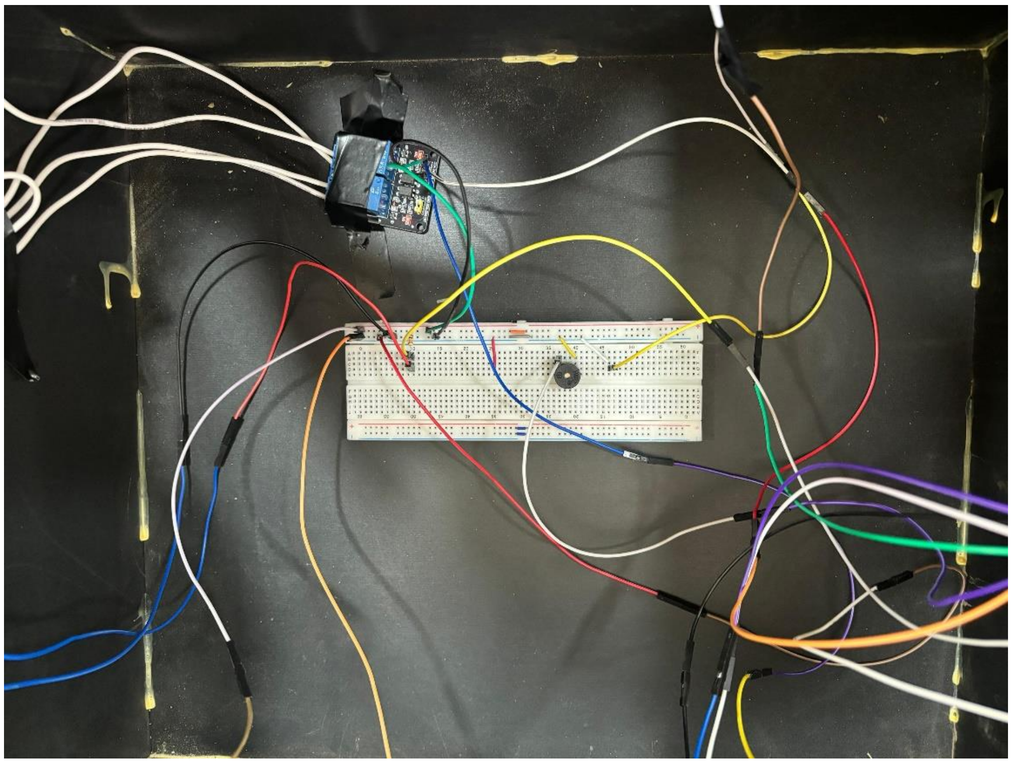
1. **5V/2A Power Adapter:**
   - **Provides a stable 5V DC output with sufficient current (2A) to power all low-voltage components in the circuit, including the microcontroller, sensors, and relay module.**
   - **This replaces the need for the AAA battery pack and simplifies the power supply.**
2. **Relay Module:**

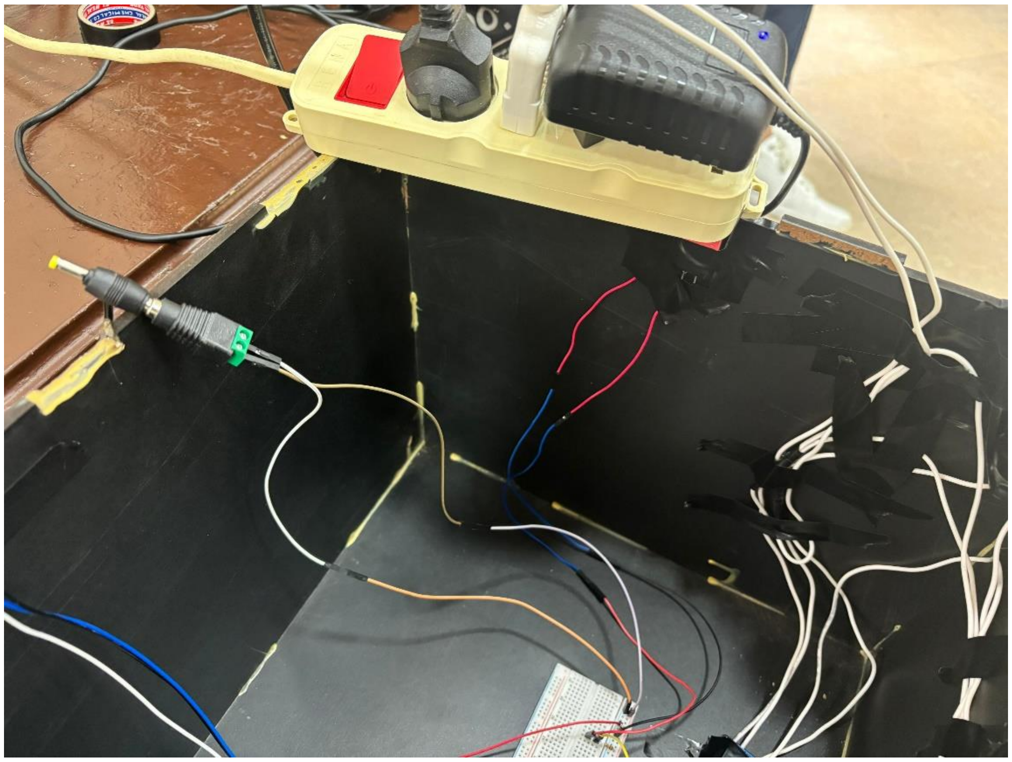
- Still used to control the light bulb. The relay isolates the low-voltage (5V) control circuit from the high-voltage AC circuit.
- Relay input is powered by the 5V adapter and controlled by signals from the microcontroller.
- Relay output terminals manage the AC connection to the lamp.

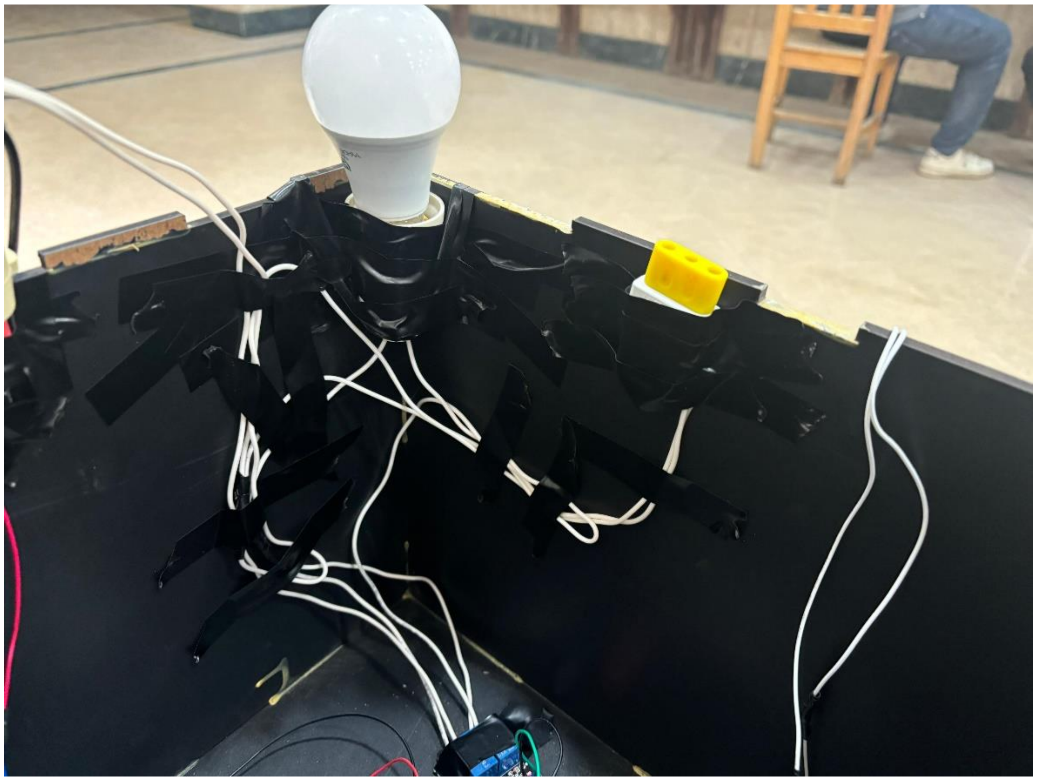
3. **Tiva™ TM4C123GH6PM Microcontroller:**
   - **The central processing unit of the circuit. It:**
     - **Receives 5V power from the adapter.**
     - **Controls the relay module.**
     - **Processes data from sensors.**
4. **DHT11 Temperature and Humidity Sensor:**
   - **Monitors temperature and humidity levels.**
   - **Powered by the 5V adapter and communicates with the microcontroller via its data pin.**
5. **Door Magnetic Switch:**
   - **Detects door motion and signals the microcontroller to trigger the relay when the door is (opened/closed)**
   - **Powered by the 5V adapter.**
6. **Lamp**
   - **Operates on 220V AC.**
   - **Controlled by the relay module.**
   - **Turns on or off based on signals sent from the microcontroller.**
7. **Breadboard**
   - **Used to distribute 5V and GND from the adapter to all components.**

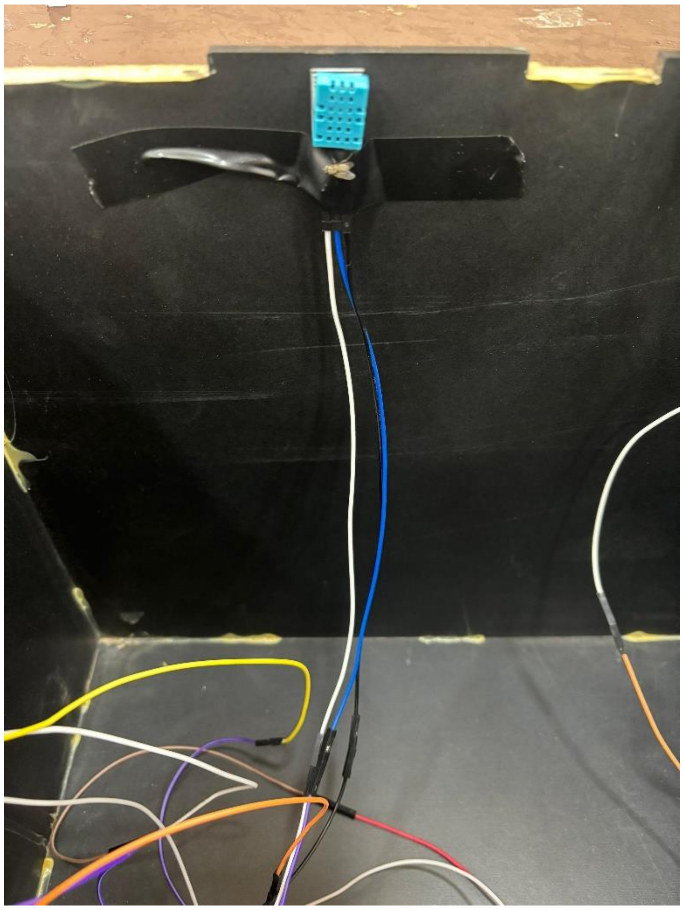
8. Prototype Photos











9. Problems faced and how we managed to solve it

1. Misunderstanding of Relay Connections

Problem:

- We encountered a misunderstanding regarding the relay connection. Specifically, we were unsure whether the relay should be connected to the "normally open" (NO) or "normally closed" (NC) terminal.

Reason:

- This confusion arose due to a lack of clarity in the documentation and an incorrect assumption about the default state of the relay terminals.

Result:

- The relay operated in the opposite manner than intended. For example, turning the switch "on" would result in the output turning "off," and vice versa, causing significant confusion during initial testing.

Solution:

- We carefully revisited the relay's datasheet and re-examined the connections. By correctly identifying and wiring the relay to the "normally open" terminal, the issue was resolved, and the relay functioned as expected.

2. Issues with LM35 Sensor

Problem:

- The LM35 temperature sensor provided random and inaccurate readings. Additionally, the sensor short-circuited during the testing phase.

Reason:

- The random readings were likely caused by electrical noise or incorrect connections. The short circuit occurred due to an accidental voltage spike or miswiring.

Result:

- The temperature readings were unreliable, which rendered the system ineffective for monitoring temperature. The short circuit also damaged the sensor, necessitating a replacement.

Solution:

- We replaced the LM35 sensor with a DHT11 sensor, which offered more stable and accurate temperature readings. This change also improved the overall reliability of the system.

3. Integration with the GUI

Problem:

- When integrating the door sensor and the temperature sensor with the GUI, the status would intermittently show "unknown" instead of the correct values.

Reason:

- The issue was caused by the simultaneous querying of both sensors. The system attempted to send and receive data for both sensors at the same time, which led to communication conflicts and occasional data loss.

Result:

- The GUI displayed incorrect or missing statuses for the sensors, which diminished the usability of the system.

Solution:

- We implemented a delay between the readings of each sensor. By staggering the queries and allowing a brief pause between them, we ensured that the readings for both the door sensor and the temperature sensor were received and displayed correctly.

10. Drive Link

https://drive.google.com/drive/folders/1jkStbpusBQmS27RDFVgIwMNg67N_fRjd