Lab

* Main Task 2

1. The alarm is activated when gas is detected, or the over temperature is activated.

**Main Task Code:**

#include "mbed.h"

#include "arm\_book\_lib.h"

int main()

{

AnalogIn smokeSensor(A0);

AnalogIn thermalSensor(A1);

DigitalOut ledAlarm(D5);

const float smokeLimit = 0.4f;

const float heatLimit = 35.0f;

ledAlarm = OFF;

while (true) {

float smokeLevel = smokeSensor.read();

float sensorVoltage = thermalSensor.read() \* 3.3f;

float tempCelsius = sensorVoltage \* 100.0f;

if (smokeLevel > smokeLimit || tempCelsius > heatLimit) {

ledAlarm = ON;

} else {

ledAlarm = OFF;

}

}

}

1. Once triggered, the alarm remains active even if the gas or temperature

levels return to normal.

**Main Task Code:**

#include "mbed.h"

#include "arm\_book\_lib.h"

int main()

{

AnalogIn gasSensorPin(A0); // MQ2 sensor input

AnalogIn tempSensorPin(A1); // LM35 temperature sensor input

DigitalOut alertIndicator(D5); // LED for alert signaling

const float gasCutoff = 0.4f; // Gas trigger threshold

const float tempCutoff = 35.0f; // Temperature trigger threshold (°C)

bool triggeredAlert = OFF; // Stays ON once triggered

while (true) {

float gasRead = gasSensorPin.read();

float voltageRead = tempSensorPin.read() \* 3.3f;

float temperature = voltageRead \* 100.0f;

// Once alarm is triggered, it stays latched ON

if (gasRead > gasCutoff || temperature > tempCutoff) {

triggeredAlert = ON;

}

alertIndicator = triggeredAlert;

}

}

1. If gas and high temperature are detected simultaneously, the system should escalate to an "emergency" mode where the alarm flashes rapidly and can only be deactivated by entering a 4-digit security code.

**Main Task Code:**

#include "mbed.h"

#include "arm\_book\_lib.h"

// Sensor inputs

AnalogIn flammableSensor(A0);

AnalogIn heatSensor(A1);

// Outputs

DigitalOut warningLight(D5);

// Keypad-style buttons

DigitalIn btnA(D2); // Digit 1

DigitalIn btnB(D3); // Digit 2

DigitalIn btnC(D4); // Digit 3

DigitalIn btnD(D6); // Digit 4

// Threshold limits

const float flammableLimit = 0.4f;

const float temperatureLimit = 35.0f;

// System states

bool alertEngaged = OFF;

bool dangerState = OFF;

int inputSequence[4];

int inputPos = 0;

const int passcode[4] = {1, 2, 3, 4};

// Button debounce check

bool buttonPressed(DigitalIn &btn) {

if (btn == 1) {

ThisThread::sleep\_for(200ms);

return btn == 1;

}

return false;

}

int main() {

// Configure button modes

btnA.mode(PullDown);

btnB.mode(PullDown);

btnC.mode(PullDown);

btnD.mode(PullDown);

warningLight = OFF;

while (true) {

float gasDetected = flammableSensor.read();

float voltage = heatSensor.read() \* 3.3f;

float temperature = voltage \* 100.0f;

if ((gasDetected > flammableLimit || temperature > temperatureLimit) && !dangerState) {

alertEngaged = ON;

}

if (gasDetected > flammableLimit && temperature > temperatureLimit) {

dangerState = true;

}

if (dangerState) {

warningLight = !warningLight;

ThisThread::sleep\_for(200ms);

if (buttonPressed(btnA)) inputSequence[inputPos++] = 1;

if (buttonPressed(btnB)) inputSequence[inputPos++] = 2;

if (buttonPressed(btnC)) inputSequence[inputPos++] = 3;

if (buttonPressed(btnD)) inputSequence[inputPos++] = 4;

if (inputPos == 4) {

bool codeMatched = true;

for (int i = 0; i < 4; i++) {

if (inputSequence[i] != passcode[i]) {

codeMatched = false;

break;

}

}

if (codeMatched) {

dangerState = false;

alertEngaged = false;

warningLight = OFF;

ThisThread::sleep\_for(300ms);

}

inputPos = 0;

}

} else {

warningLight = alertEngaged;

ThisThread::sleep\_for(100ms);

}

}

}

1. If five incorrect codes are entered consecutively, the system should lock, blocking any further code entries for 60 seconds.

**Main Task Code:**

#include "mbed.h"

#include "arm\_book\_lib.h"

// Sensor setup

AnalogIn gasMonitor(A0);

AnalogIn tempMonitor(A1);

// LEDs

DigitalOut alertLED(D5);

DigitalOut lockLED(D7);

// Input buttons for code

DigitalIn key1(D2);

DigitalIn key2(D3);

DigitalIn key3(D4);

DigitalIn key4(D6);

// Threshold constants

const float gasTrigger = 0.4f;

const float tempTrigger = 35.0f;

// Flags

bool alertActive = OFF;

bool emergencyAlert = OFF;

bool lockedOut = false;

int codeInput[4];

int codePointer = 0;

const int secretCode[4] = {1, 2, 3, 4};

int failedTries = 0;

Timer lockoutClock;

// Debounce utility

bool detectButton(DigitalIn &btn) {

if (btn == 1) {

ThisThread::sleep\_for(200ms);

return btn == 1;

}

return false;

}

int main() {

// Button configuration

key1.mode(PullDown);

key2.mode(PullDown);

key3.mode(PullDown);

key4.mode(PullDown);

alertLED = OFF;

lockLED = OFF;

while (true) {

float gasLevel = gasMonitor.read();

float tempReading = tempMonitor.read() \* 3.3f \* 100.0f;

// Standard alert condition

if ((gasLevel > gasTrigger || tempReading > tempTrigger) && !emergencyAlert) {

alertActive = ON;

}

// Emergency mode activation

if (gasLevel > gasTrigger && tempReading > tempTrigger) {

emergencyAlert = true;

}

// LOCKOUT HANDLING

if (lockedOut) {

alertLED = OFF;

lockLED = !lockLED;

ThisThread::sleep\_for(1000ms);

if (lockoutClock.elapsed\_time() >= 60s) {

lockedOut = false;

failedTries = 0;

lockLED = OFF;

lockoutClock.stop();

lockoutClock.reset();

}

continue;

}

// EMERGENCY MODE

if (emergencyAlert) {

alertLED = !alertLED;

ThisThread::sleep\_for(200ms);

// Collect code input

if (detectButton(key1)) codeInput[codePointer++] = 1;

if (detectButton(key2)) codeInput[codePointer++] = 2;

if (detectButton(key3)) codeInput[codePointer++] = 3;

if (detectButton(key4)) codeInput[codePointer++] = 4;

if (codePointer == 4) {

bool codeCorrect = true;

for (int i = 0; i < 4; i++) {

if (codeInput[i] != secretCode[i]) {

codeCorrect = false;

break;

}

}

if (codeCorrect) {

emergencyAlert = false;

alertActive = false;

alertLED = OFF;

failedTries = 0;

ThisThread::sleep\_for(300ms);

} else {

failedTries++;

if (failedTries >= 5) {

lockedOut = true;

lockoutClock.start();

}

}

codePointer = 0;

}

} else {

alertLED = alertActive;

ThisThread::sleep\_for(100ms);

}

}

}

1. While the system is locked, an LED should blink slowly to indicate the lockout status.

**Main Task Code:**

#include "mbed.h"

#include "arm\_book\_lib.h"

// Analog sensors

AnalogIn gasInput(A0);

AnalogIn tempInput(A1);

// Indicators

DigitalOut emergencyLED(D5);

DigitalOut secureLED(D7);

// Code entry buttons

DigitalIn btnOne(D2);

DigitalIn btnTwo(D3);

DigitalIn btnThree(D4);

DigitalIn btnFour(D6);

// Safety limits

const float gasThreshold = 0.4f;

const float tempThreshold = 35.0f;

// PIN and status

const int pinCode[4] = {1, 2, 3, 4};

int userCode[4];

int codeIndex = 0;

bool alarmEngaged = OFF;

bool criticalAlert = OFF;

bool isLocked = false;

bool blinkState = false;

bool lockTimerStarted = false;

int failCounter = 0;

Timer lockTimer;

Timer blinkTimer;

// Button debouncing

bool checkPress(DigitalIn &button) {

if (button == 1) {

ThisThread::sleep\_for(150ms);

return button == 1;

}

return false;

}

int main() {

// Configure buttons with pull-down resistors

btnOne.mode(PullDown);

btnTwo.mode(PullDown);

btnThree.mode(PullDown);

btnFour.mode(PullDown);

emergencyLED = OFF;

secureLED = OFF;

while (true) {

// Lockout handling

if (isLocked) {

if (!lockTimerStarted) {

lockTimer.start();

blinkTimer.start();

lockTimerStarted = true;

}

if (blinkTimer.elapsed\_time() >= 500ms) {

blinkState = !blinkState;

secureLED = blinkState;

blinkTimer.reset();

}

if (lockTimer.elapsed\_time() >= 60s) {

isLocked = false;

failCounter = 0;

secureLED = OFF;

blinkState = false;

lockTimer.stop();

blinkTimer.stop();

lockTimer.reset();

blinkTimer.reset();

lockTimerStarted = false;

}

ThisThread::sleep\_for(50ms);

continue;

}

// Read sensor values

float gasLevel = gasInput.read();

float tempCelsius = tempInput.read() \* 3.3f \* 100.0f;

// General alarm activation

if ((gasLevel > gasThreshold || tempCelsius > tempThreshold) && !criticalAlert) {

alarmEngaged = ON;

}

if (gasLevel > gasThreshold && tempCelsius > tempThreshold) {

criticalAlert = true;

}

// Emergency mode: LED blinks and waits for code

if (criticalAlert) {

emergencyLED = !emergencyLED;

ThisThread::sleep\_for(200ms);

if (checkPress(btnOne)) userCode[codeIndex++] = 1;

if (checkPress(btnTwo)) userCode[codeIndex++] = 2;

if (checkPress(btnThree)) userCode[codeIndex++] = 3;

if (checkPress(btnFour)) userCode[codeIndex++] = 4;

if (codeIndex == 4) {

bool isCorrect = true;

for (int i = 0; i < 4; i++) {

if (userCode[i] != pinCode[i]) {

isCorrect = false;

break;

}

}

if (isCorrect) {

criticalAlert = false;

alarmEngaged = false;

emergencyLED = OFF;

failCounter = 0;

} else {

failCounter++;

if (failCounter >= 5) {

isLocked = true;

}

}

codeIndex = 0;

}

} else {

// Normal blinking state

emergencyLED = alarmEngaged;

ThisThread::sleep\_for(100ms);

}

}

}