**tutorial 2**

**Main task (code)**

Design a program for an embedded system that meets the following requirements.

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

#include "mbed.h"

#include "arm\_book\_lib.h"

int main()

{

AnalogIn mq2Sensor(A0); // Gas sensor (MQ2)

AnalogIn lm35Sensor(A1); // Temperature sensor (LM35)

DigitalOut warningLED(D5); // Alarm indicator LED

warningLED = OFF;

const float mq2Limit = 0.4f; // Threshold for gas level

const float tempLimit = 35.0f; // Threshold for temperature in °C

while (true) {

float gasLevel = mq2Sensor.read(); // Value: 0.0 to 1.0

float voltage = lm35Sensor.read() \* 3.3f; // Sensor output voltage

float temperature = voltage \* 100.0f; // Convert to Celsius

if (gasLevel > mq2Limit || temperature > tempLimit) {

warningLED = ON;

} else {

warningLED = OFF;

}

}

}

1. Once triggered, the alarm remains active even if the gas or temperature levels return to normal.

#include "mbed.h"

#include "arm\_book\_lib.h"

int main()

{

AnalogIn gasInput(A0); // MQ2 sensor input

AnalogIn tempInput(A1); // LM35 temperature input

DigitalOut alertLED(D5); // Warning LED

const float gasLimit = 0.4f; // Threshold for gas

const float tempLimit = 35.0f; // Threshold for temperature

bool triggered = OFF; // Latching alarm state

while (true) {

float gasReading = gasInput.read();

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

float tempReading = voltageOut \* 100.0f;

// Once alarm condition is met, stay on

if (gasReading > gasLimit || tempReading > tempLimit) {

triggered = ON;

}

alertLED = triggered;

}

}

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.

#include "mbed.h"

#include "arm\_book\_lib.h"

// Inputs

AnalogIn gasInput(A0);

AnalogIn temperatureInput(A1);

// Outputs

DigitalOut statusLED(D5);

// Button inputs (represent digits 1–4)

DigitalIn digitBtn1(D2);

DigitalIn digitBtn2(D3);

DigitalIn digitBtn3(D4);

DigitalIn digitBtn4(D6);

// Threshold levels

const float gasLimit = 0.4f;

const float tempLimit = 35.0f;

// Alarm states

bool alarmTriggered = OFF;

bool emergencyActive = OFF;

int inputCode[4];

int inputIndex = 0;

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

// Button debounce

bool checkButton(DigitalIn &btn) {

if (btn == 1) {

ThisThread::sleep\_for(200ms);

return btn == 1;

}

return false;

}

int main() {

// Setup pulldowns

digitBtn1.mode(PullDown);

digitBtn2.mode(PullDown);

digitBtn3.mode(PullDown);

digitBtn4.mode(PullDown);

statusLED = OFF;

while (true) {

float gasLevel = gasInput.read();

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

float temperature = voltage \* 100.0f;

// Trigger alarm if any sensor exceeds limit

if ((gasLevel > gasLimit || temperature > tempLimit) && !emergencyActive) {

alarmTriggered = ON;

}

// Trigger emergency if both conditions are met

if (gasLevel > gasLimit && temperature > tempLimit) {

emergencyActive = true;

}

if (emergencyActive) {

// Flashing LED in emergency

statusLED = !statusLED;

ThisThread::sleep\_for(200ms);

// Code input handling

if (checkButton(digitBtn1)) inputCode[inputIndex++] = 1;

if (checkButton(digitBtn2)) inputCode[inputIndex++] = 2;

if (checkButton(digitBtn3)) inputCode[inputIndex++] = 3;

if (checkButton(digitBtn4)) inputCode[inputIndex++] = 4;

// Verify input

if (inputIndex == 4) {

bool match = true;

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

if (inputCode[i] != unlockCode[i]) {

match = false;

break;

}

}

if (match) {

emergencyActive = false;

alarmTriggered = false;

statusLED = OFF;

ThisThread::sleep\_for(300ms);

}

inputIndex = 0;

}

} else {

// Regular LED control

statusLED = alarmTriggered;

ThisThread::sleep\_for(100ms);

}

}

}

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

#include "mbed.h"

#include "arm\_book\_lib.h"

// Inputs

AnalogIn gasInput(A0);

AnalogIn tempInput(A1);

// Outputs

DigitalOut mainLED(D5); // Main alarm LED

DigitalOut blockLED(D7); // LED for lockout indication

// Buttons representing each digit

DigitalIn digitBtn1(D2); // Button 1

DigitalIn digitBtn2(D3); // Button 2

DigitalIn digitBtn3(D4); // Button 3

DigitalIn digitBtn4(D6); // Button 4

// Threshold constants

const float gasLimit = 0.4f;

const float tempLimit = 35.0f;

// State flags

bool alarmActive = OFF;

bool emergencyTriggered = OFF;

bool blockedState = false;

int inputCode[4];

int inputPos = 0;

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

int failCount = 0;

Timer blockTimer;

// Simple debounce logic

bool debounceButton(DigitalIn &btn) {

if (btn == 1) {

ThisThread::sleep\_for(200ms);

return btn == 1;

}

return false;

}

int main() {

// Pull-down resistors setup

digitBtn1.mode(PullDown);

digitBtn2.mode(PullDown);

digitBtn3.mode(PullDown);

digitBtn4.mode(PullDown);

mainLED = OFF;

blockLED = OFF;

while (true) {

float gasLevel = gasInput.read();

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

float tempC = tempVolt \* 100.0f;

// Trigger alarm if either sensor goes above limit

if ((gasLevel > gasLimit || tempC > tempLimit) && !emergencyTriggered) {

alarmActive = ON;

}

// Trigger emergency if both are too high

if (gasLevel > gasLimit && tempC > tempLimit) {

emergencyTriggered = true;

}

// 🔐 Lockout logic

if (blockedState) {

mainLED = OFF;

blockLED = !blockLED;

ThisThread::sleep\_for(1000ms);

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

blockedState = false;

failCount = 0;

blockLED = OFF;

blockTimer.stop();

blockTimer.reset();

}

continue; // Skip to top

}

// 🚨 Emergency behavior

if (emergencyTriggered) {

mainLED = !mainLED;

ThisThread::sleep\_for(200ms); // Blink fast

// Check for button entries

if (debounceButton(digitBtn1)) inputCode[inputPos++] = 1;

if (debounceButton(digitBtn2)) inputCode[inputPos++] = 2;

if (debounceButton(digitBtn3)) inputCode[inputPos++] = 3;

if (debounceButton(digitBtn4)) inputCode[inputPos++] = 4;

if (inputPos == 4) {

bool codeMatch = true;

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

if (inputCode[i] != accessCode[i]) {

codeMatch = false;

break;

}

}

if (codeMatch) {

emergencyTriggered = false;

alarmActive = false;

mainLED = OFF;

failCount = 0;

ThisThread::sleep\_for(300ms);

} else {

failCount++;

if (failCount >= 5) {

blockedState = true;

blockTimer.start();

}

}

inputPos = 0;

}

} else {

// Normal LED alarm behavior

mainLED = alarmActive;

ThisThread::sleep\_for(100ms);

}

}

}

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

#include "mbed.h"

#include "arm\_book\_lib.h"

// Sensor inputs

AnalogIn gasInput(A0);

AnalogIn tempInput(A1);

// Output indicators

DigitalOut alertLED(D5); // Alarm light

DigitalOut blockIndicator(D7); // Lockout blinking LED

// Button inputs (code digits)

DigitalIn btn1(D2);

DigitalIn btn2(D3);

DigitalIn btn3(D4);

DigitalIn btn4(D6);

// Thresholds

const float gasLimit = 0.4f;

const float tempLimit = 35.0f;

// State tracking

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

int inputBuffer[4];

int entryIndex = 0;

bool alarmOn = OFF;

bool emergencyState = OFF;

bool inLockout = false;

bool blinkState = false;

bool lockoutInit = false;

int attempts = 0;

Timer lockoutTimer;

Timer blinkTimer;

// Button debounce logic

bool checkBtn(DigitalIn &btn) {

if (btn == 1) {

ThisThread::sleep\_for(150ms);

return btn == 1;

}

return false;

}

int main() {

// Configure buttons with pulldown mode

btn1.mode(PullDown);

btn2.mode(PullDown);

btn3.mode(PullDown);

btn4.mode(PullDown);

alertLED = OFF;

blockIndicator = OFF;

while (true) {

// LOCKOUT HANDLING

if (inLockout) {

if (!lockoutInit) {

lockoutTimer.start();

blinkTimer.start();

lockoutInit = true;

}

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

blinkState = !blinkState;

blockIndicator = blinkState;

blinkTimer.reset();

}

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

inLockout = false;

attempts = 0;

blockIndicator = OFF;

blinkState = false;

lockoutInit = false;

lockoutTimer.stop();

lockoutTimer.reset();

blinkTimer.stop();

blinkTimer.reset();

}

ThisThread::sleep\_for(50ms);

continue;

}

// SENSOR MONITORING

float gasReading = gasInput.read();

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

// Trigger alarm if either threshold exceeded

if ((gasReading > gasLimit || temperature > tempLimit) && !emergencyState) {

alarmOn = ON;

}

// Emergency mode if both are triggered

if (gasReading > gasLimit && temperature > tempLimit) {

emergencyState = true;

}

// EMERGENCY RESPONSE

if (emergencyState) {

alertLED = !alertLED;

ThisThread::sleep\_for(200ms);\_