电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC2004/UESTCHN2004

Embedded Processor

Technical Report：

Smart Locker Security and Environmental Monitoring System

Student's Name	Li Zhengli
Student's UESTC ID#	2023190901029
Student's UoG ID#	2960201

Contents

List of Figures:

Abstract

This project implements a low-power smart locker system based on the NUCLEO-L432KC and mbed OS. The system uses a TMP102 temperature sensor and an ADXL345 accelerometer to monitor environmental conditions and detect tampering. A binary password input mechanism is implemented using two buttons with interrupt-based handling. System status is indicated by LEDs and a buzzer. RTOS threads manage password input and sensor monitoring concurrently, ensuring responsive and energy-efficient operation. The system provides a practical solution for embedded access control in secure environments.

1. Introduction

In recent years, Internet of Things (IoT) technology has advanced rapidly and become increasingly integrated into various aspects of modern life [1]. Smart storage systems constitute a significant application area within the broader framework of IoT [2]. With the increasing demand for intelligent and secure storage solutions, smart lockers have become essential in scenarios such as express parcel delivery, campus storage systems, and shared facilities [2, 3]. However, many commercial systems are either costly, power-intensive, or lack flexibility for customization [3, 4]. Therefore, the development of next-generation low-power smart locker systems is of great significance in enhancing convenience and security in everyday life [3, 4].

This project aims to design and implement a low-power smart locker security and environmental monitoring system based on the NUCLEO-L432KC microcontroller and mbed OS. The system features binary password input using two buttons, environmental sensing via a TMP102 temperature sensor, and tamper detection using an ADXL345 accelerometer. Real-time feedback is provided through LEDs and a buzzer, while RTOS-based multithreading ensures responsive and efficient operation.

The project focuses on secure local access control and physical state

monitoring in standalone embedded systems, which prioritizes modularity, responsiveness, and low power consumption, making it suitable for embedded access control applications in constrained environments such as lockers, cabinets, or remote storage units. Networking and remote access functions are outside the current scope but may be considered in future work. The resulting system demonstrates a modular and energy-efficient smart locker suitable for educational, prototyping, and practical deployment scenarios.

2. Methodology

2.1 Hardware and software required

2.1.1 Hardware (Electronic components)

- One NUCLEO L432KC microcontroller and USB programming cable
- A breadboard
- A TMP102 temperature sensor
- An ADXL345 accelerometer
- An active buzzer
- Five LEDs (two green, one red, one blue, one orange)
- Three push buttons
- Several Dupont wires
- A digital multimeter

2.1.2 Software (Programming environment)

- Mbed Studio
- Keil Studio Cloud

2.2 Circuit design and hardware integration

The smart locker system consists of five distinct modules, including temperature sensing module, tamper detection module, password input module,

alert module and power efficiency (system wake-up) module. The temperature monitoring module and the tamper detection module are responsible for acquiring internal temperature and motion state data within the smart storage cabinet. Upon detection of abnormal temperature or acceleration, an alert is triggered via a buzzer connected to the microcontroller, accompanied by the flashing of an associated LED indicator. The system is equipped with a password input module that utilizes two buttons to implement binary password entry for unlocking operations. In the absence of significant environmental changes over an extended duration, the system automatically enters a low-power sleep mode, from which it can be awakened through a button press. **Figure.1** shows the hardware architecture.

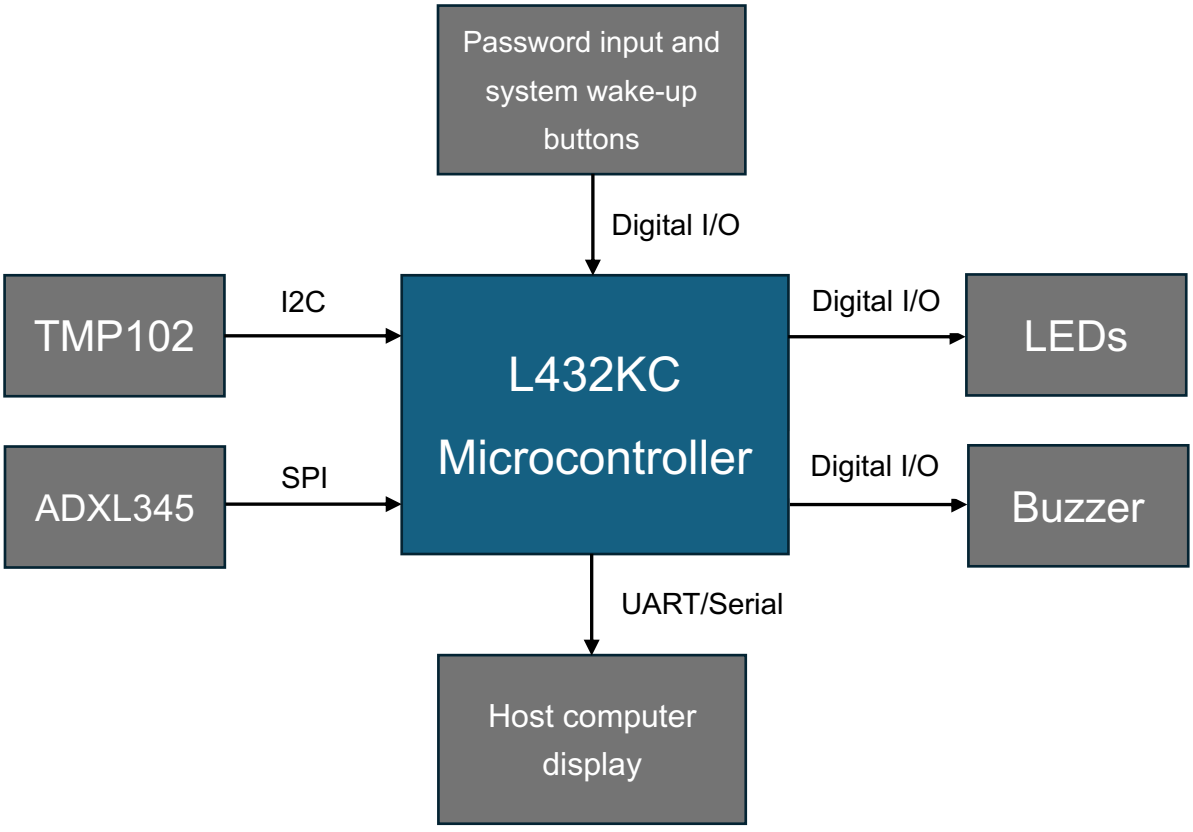

Figure.1: Hardware Architecture

2.2.1 Tamper detection module

The tamper detection module uses the ADXL345 three-axis accelerometer. **Figure.2** illustrates the circuit connections of the ADXL345. When the locker is subject to noticeable tampering, it typically generates a significant acceleration, usually between $1g$ and $2g$ [5]. In cases of violent tampering or physical attacks, the acceleration can reach $3g$ to $5g$ [5]. Based on actual testing, we set the abnormal acceleration threshold at $\sqrt{3}g$ ($\mathbf{1.0}\boldsymbol{g}$ **at each axis)** by comparing the measured acceleration values under different shaking intensities on a breadboard, ensuring adequate protection for fragile and valuable items such as those made of glass. This module communicates with the main control board via the Serial Peripheral Interface (SPI) protocol.

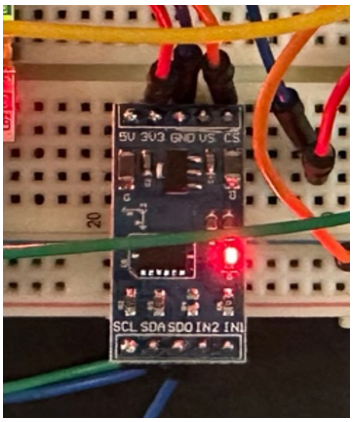

Figure.2: The circuit connections of the ADXL345

SPI is a widely used communication protocol between microcontrollers and various peripheral devices, supporting high-speed data transfer. The SPI interface includes four main signal lines: SCLK (clock line), MOSI (Master Out, Slave In), MISO (Master In, Slave Out), and CS (chip select). Through these connections, bidirectional data transfer is enabled between the L432KC and the ADXL345. The main controller sends a clock signal to the ADXL345 to synchronize data exchange, with actual data transmitted through the MOSI and MISO lines, while the CS line is used to initiate communication with the ADXL345. By precisely controlling these signals, the system accurately acquires acceleration data from the sensor.

The ADXL345 is capable of measuring acceleration along the X, Y, and Z axis. To calculate the total acceleration, the following formula is below, which is

optional however:

$$a_{total} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

However, to simplify the post-sampling decision logic and avoid unnecessary precision loss, the microcontroller directly compares the acceleration values along the three axes with their respective thresholds, instead of using the resultant acceleration magnitude, thereby eliminating the need for square root operations.

2.2.2 Temperature monitoring module

The temperature sensing module uses the TMP102 temperature sensor, and its circuit connection is shown in **Figure.3**. This module communicates with the main controller via the I²C interface. I²C is a commonly used serial communication protocol that allows multiple devices to be connected using only two signal lines: a data line (SDA) and a clock line (SCL). During system initialization, the L432KC configures the necessary I²C parameters, such as the clock speed and the address of the slave device. When the system needs to acquire temperature data, the L432KC sends a request to the TMP102. Upon receiving the request, the TMP102 measures the current temperature, converts the analog signal into a digital value, and sends the result back to the L432KC via the I²C bus. After receiving the data, the L432KC retains four significant digits and makes corresponding decisions based on the temperature.

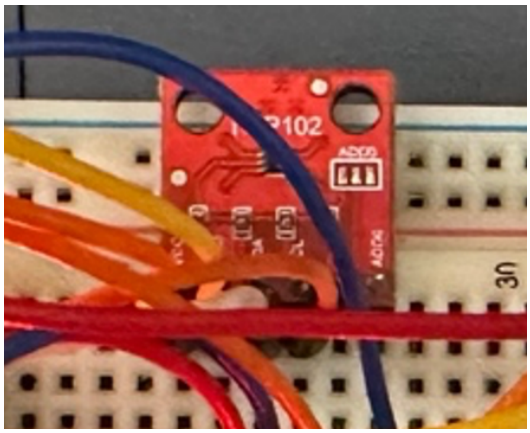

Figure.3: TMP102 and circuit connection

The temperature thresholds for this intelligent storage cabinet are set as follows: the high-temperature limit is **30°C**, and the low-temperature limit is **10°C**. Considering the types of stored items, for food, temperatures above 30°C can promote bacterial growth, leading to spoilage, while temperatures below 10°C may adversely affect its taste [6]. For other types of items, both excessively high and low temperatures may cause various impacts. Therefore, the normal temperature range of 10°C to 30°C is generally regarded as an appropriate storage condition [6].

2.2.3 Password input module

The password input module consists of two buttons, each representing the input of '1' and '0' respectively. Password entry is achieved by combining Digital I/O with the microcontroller's internal pull-up resistors. In the code (main.cpp, see Appendix for details), a five-bit binary password '10110' is predefined. Upon correct password entry, the orange LED on the breadboard will turn off, and the green LED will light up, indicating successful unlocking. However, even in the unlocked state, the two sensors continue to operate normally, ensuring that the system promptly responds to abnormal temperature or external impacts. If an incorrect digit is entered during password input, the input will reset, the buzzer will sound an alarm, and the blue LED will flash, requiring the operator to re-enter the password.

2.2.4 Alarm module

The alarm module consists of two LED indicators and an active buzzer, all of which communicate via Digital I/O. When the system detects that the temperature or acceleration exceeds the preset thresholds—indicating either an abnormal environmental temperature (above 30°C or below 10°C) or a physical tampering of the storage cabinet (acceleration reaching $1.0g$ in any direction)—the red LED will begin to flash. If an incorrect password is entered, the blue LED will light up. In either case, the buzzer will emit an alarm sound as a warning. Additionally, the collected data is transmitted to the L432KC microcontroller and then forwarded to a computer, enabling real-time data display for user analysis.

2.2.5 Power efficiency module (System wake-up module)

The low-power management module is designed to minimize energy consumption during periods of environmental stability, which is critical for embedded systems deployed in constrained or unattended scenarios. The system implements a hibernation mode through an interrupt algorithm in software. This module intelligently manages the sleep and wake cycles of the TMP102 temperature sensor and the ADXL345 accelerometer.

During normal operation, the system periodically samples temperature and acceleration data. If all sensor readings remain within predefined “stable” thresholds (e.g., temperature between 10°C and 30°C, and acceleration close to 1g along the Z-axis with minimal X and Y variations), a timer is started. If the system maintains stable conditions for a continuous 30 seconds, the sensors are transitioned into low-power sleep mode via dedicated SPI/I2C register configurations. This reduces their active current draw significantly.

While in sleep mode, the system monitors the external wake-up pin (configured as an interrupt on D9). When triggered (e.g., due to vibration or button interaction), the sensors are reactivated, and the monitoring thread resumes full operation. This ensures that the system remains responsive to meaningful events while conserving energy during idle periods. The microcontroller continues to operate in a low-frequency polling loop using sleep function to further reduce CPU activity.

2.2.6 Circuit integration

In this project, circuit assembly is a crucial step for realizing system functionality. All hardware components, including the NUCLEO L432KC development board, TMP102 temperature sensor, ADXL345 accelerometer, LEDs, buzzer, and physical switch, need to be correctly connected and configured. The TMP102 is connected through the I2C interface, with SDA and SCL linked to the corresponding pins on the development board, along with power (VCC) and ground (GND) connections. The ADXL345 is connected via the SPI interface, including connections for SCLK, MOSI, MISO, and CS pins. LEDs and the buzzer are connected to GPIO pins for alarm signals. The three buttons are

also connected to GPIO pins. **Figure.4** illustrates the actual hardware wiring of the circuit. The detailed connection scheme and corresponding interfaces are presented in the **appendix**.

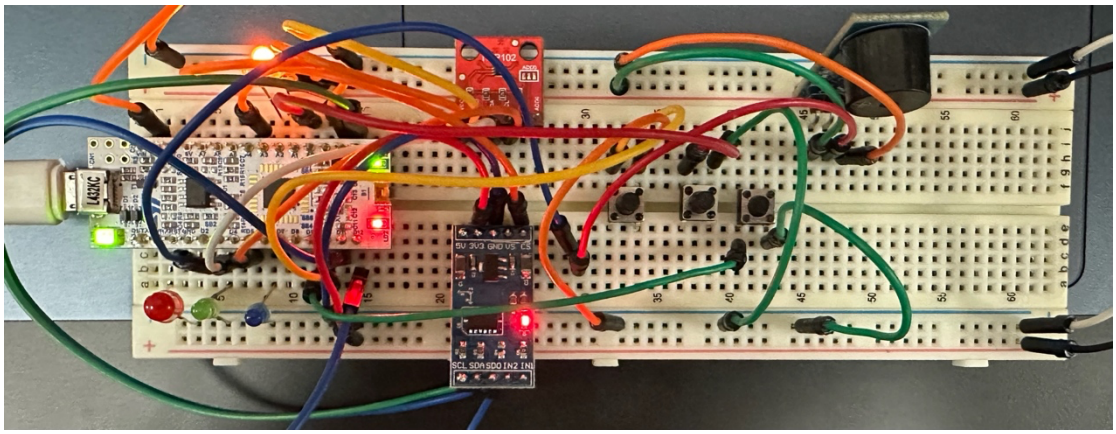

Figure.4 The actual hardware wiring of the circuit

2.3 Software development and programming algorithm

The firmware for the NUCLEO-L432KC development board was developed using the mbed Studio compiler, providing a stable platform for code writing and testing. The software was written in C++ and incorporated dedicated libraries for the TMP102 and ADXL345 sensors to simplify the development process. The system implements several key features, including real-time temperature measurement, tamper detection algorithms, and alarm mechanisms. Additionally, low-power functionality (sleep mode) was achieved through proper configuration of the sensors in the code. Moreover, a lightweight user interface was developed to present real-time sensor data and the current operational state of the system. Specially, this system employs the rtos.h interface for managing multiple threads and performing task scheduling in a real-time operating environment. The detailed source code(main.cpp) and corresponding header files (ADXL345.h/.cpp and TMP102.h/.cpp) are provided in the **appendix**. The algorithmic logic is depicted in **Figure.5**.

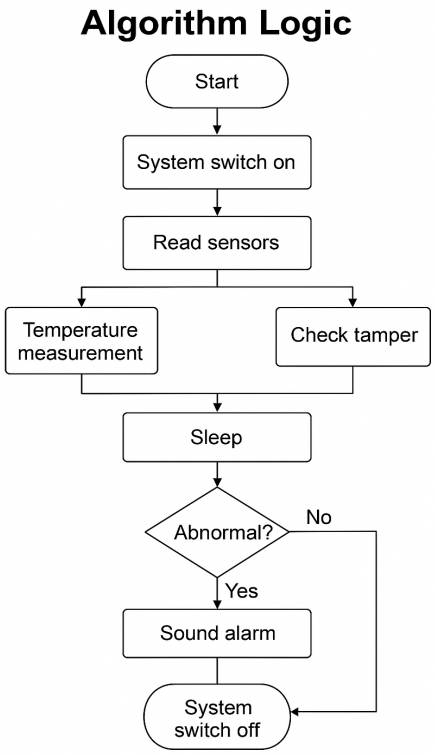

Figure.5: The system's algorithm logic

2.4 Function Verification

2.4.1 Assumptions and Justifications

System Application Environment:

• Assumption: It is assumed that the smart locker system is mainly used in office areas or public environments for storing valuable or fragile items. The surrounding environment is generally stable with minimal external disturbance.

• Justification: In such scenarios, the locker must be sensitive to unusual external events such as tampering or environmental anomalies, in order to ensure the safety of stored items.

Tamper Detection Threshold:

• Assumption: The system sets the acceleration threshold of the ADXL345 accelerometer in the X/Y directions to ±1.0g, which is used to detect potential tampering or unauthorized shaking.

• Justification: Based on practical testing with breadboard shaking at varying intensities, minor vibration does not exceed this threshold, while noticeable disturbances do. This value minimizes false positives while ensuring effective tamper detection.

Temperature Alarm Threshold:

• Assumption: The system defines abnormal temperature ranges as below 10.0°C or above 30.0°C. An alarm is triggered once the environment temperature exceeds these limits.

• Justification: This range accommodates typical indoor temperature fluctuations. Deviations may indicate abnormal operation, fire hazards, or external environmental threats.

Low-Power Entry Conditions:

• Assumption: If the temperature and acceleration readings remain within the defined thresholds continuously for 30 seconds, the system will enter low-power mode by putting the sensors to sleep.

• Justification: This design avoids reacting to short-term stability, while ensuring minimal power consumption during extended periods of inactivity.

2.4.2 Verification

To verify the functional completeness and response accuracy of the system in real scenarios, a series of unit tests and integrated system tests were designed for the main functional modules.

Temperature Monitoring Module Testing:

The TMP102 sensor was tested by observing the real-time temperature readings while manually altering the surrounding temperature - e.g., by moving a hand closer or further away. The printed values on the serial interface were verified against environmental changes, and alarms (LED + buzzer) were checked for correct response when thresholds were exceeded.

Tamper Detection Testing:

With the device placed in a stable state, the output values of the ADXL345

accelerometer were monitored. The system was then subjected to minor and strong vibrations. If the acceleration along the X or Y axis exceeded ±1.0g, the alarm was expected to trigger, which was confirmed by LED and buzzer activation.

Low-Power Mode Testing:

In a static and thermally stable environment, the system was observed for 30 seconds. Upon meeting the conditions, a message “Stable 10s. Entering sleep.” was printed, confirming sensor sleep entry. Wake-up was later triggered via an external interrupt on the designated pin, verifying successful recovery from sleep.

In addition, hardware-level verification was also conducted. Taking the TMP102 as an example, a multimeter was used to directly measure the current between the SDA and GND pins before and after entering sleep mode to verify whether the sensor had truly transitioned into a low-power state. According to the datasheet, the TMP102 typically draws more than $10\,\mu A$ during normal operation and less than $5\,\mu A$ in sleep mode. **Figure.6** and **Figure.7** show the measured current values before $(13.5\,\mu A)$ and after $(4.1\,\mu A)$ entering sleep mode, respectively. The results confirm that the sensor meets the expected low-power requirements.

Figure.6: Current when awake

Figure.7: Current when sleep

Password Unlock Functionality Testing:

The correct binary sequence “10110” was entered via button inputs. When entered correctly, the system displayed “Password CORRECT! Locker UNLOCKED.” on the serial monitor and turned on the green LED. In the case of incorrect input, the blue LED lit up and the buzzer alarm was triggered. Multiple tests ensured consistent and stable behavior across scenarios.

System Integration Testing:

With all modules active, various conditions were simulated—including changes in temperature, acceleration, and button inputs. The system was verified to respond only when logical conditions were met. For instance, abnormal temperature or movement should not trigger alarms when the locker is unlocked. The wake-up mechanism from sleep and the independence between password input and sensor states were also confirmed.

3. Results and Discussion

3.1 Results

3.1.1 Temperature Monitoring Performance

The TMP102 sensor successfully measured ambient temperature and responded sensitively to changes. When a heat source (e.g., a human hand) was brought close to the sensor, the real-time temperature readings increased accordingly. When the temperature exceeded the upper threshold (30.0 °C) or dropped below the lower limit (10.0 °C), the system promptly triggered the buzzer and illuminated the temperature warning LED. The sensor’s accuracy was verified through comparison with known environmental references, showing deviations within ±0.5 °C, which meets application requirements. The acceleration data is displayed on the terminal, and if the acceleration value exceeds the threshold, a corresponding alert will also be shown (**Figure.8**).

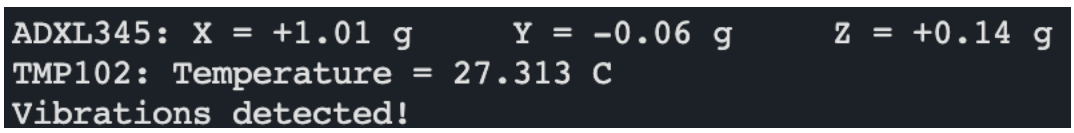

Figure.8: Acceleration information

3.1.2 Tamper Detection Accuracy

Using the ADXL345 accelerometer, the system effectively identified physical tampering. During the test, accelerations along the X and Y axes were measured under various shaking conditions. A threshold of ±1.0g was set based on empirical evaluation. When the locker was disturbed or shaken beyond this threshold, the system activated the alarm (buzzer and LED). Mild vibrations did not result in false positives, demonstrating good robustness of the detection logic. The temperature data is displayed on the terminal, and if the temperature value exceeds the threshold, a corresponding alert will also be shown (**Figure.9**).

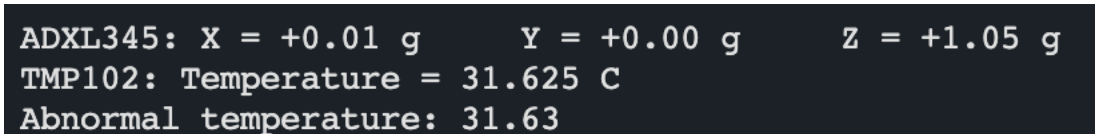

Figure9: Temperature information

3.1.3 Password Authentication and State Control

The system's password input mechanism, using two physical buttons representing binary digits (1 and 0), accurately recognized the preset 5-bit password "10110". Upon correct input, the system unlocked the cabinet, turned on the green LED, and suppressed false alarms during the unlocked state. Incorrect inputs resulted in immediate buzzer alerts and red LED activation. After three consecutive failures, a temporary lockout period was enforced. The interface displayed the progress of input and current system status via the serial monitor, confirming the interactive logic functioned correctly.

3.1.4 Low-Power Mode Effectiveness

The system entered low-power mode when both temperature and acceleration values remained stable for over 30 seconds. In this state, the TMP102 and ADXL345 were switched to sleep via I²C and SPI commands respectively, reducing their current consumption. Multimeter measurements confirmed that TMP102 current dropped from 13.5 μA (active) to 4.1 μA (sleep), aligning with datasheet specifications. Upon external interrupts (e.g., tapping the cabinet or

pressing a button), the system successfully woke up, restored sensor activity, and resumed monitoring.

3.1.5 Integrated System Response

An end-to-end test was conducted to validate the integrated behavior. Under various environmental changes (e.g., temperature fluctuations, physical shocks, incorrect password inputs), the system consistently responded according to predefined logic. When unlocked, tamper and temperature alarms were suppressed. When relocked, sensors re-enabled security monitoring. The system maintained accurate real-time feedback through LEDs, the buzzer, and serial output, confirming that all components functioned cohesively and reliably.

3.2 Discussion

The low-power smart storage cabinet system demonstrated reliable environmental monitoring and access control. The TMP102 sensor accurately tracked temperature variations, while the ADXL345 accelerometer detected physical disturbances indicative of tampering. A binary password input module ensured secure access, allowing only authorized users to unlock the cabinet. To reduce energy consumption, the system incorporates a sleep mode that activates when both temperature and acceleration remain unchanged over a prolonged period. The system automatically wakes upon detecting significant changes, ensuring responsiveness while conserving power. The integrated alarm module, composed of LEDs and a buzzer, provides clear visual and auditory alerts in response to abnormal conditions or incorrect password attempts. Overall, the system combines accurate sensing, intelligent power management, and secure user interaction, offering a comprehensive solution for protected and energy-efficient storage.

4. Conclusions and Future Work

4.1 Conclusions

This project successfully designed and implemented a low-power smart locker system that integrates temperature monitoring, tamper detection,

password-based access control, and low-power operation. The system was developed using the NUCLEO-L432KC microcontroller and mbed OS, with the TMP102 and ADXL345 sensors playing critical roles in environmental monitoring and physical disturbance detection, respectively. The password input mechanism, using binary button input, ensured secure access control while maintaining simplicity in embedded implementation. Moreover, the system implements a hibernation mode through an interrupt algorithm in software.

Functionality tests confirmed that the system responds reliably to changes in temperature and motion, issuing timely visual and auditory alerts via LEDs and a buzzer. The system also effectively entered and exited sleep mode based on environmental stability, demonstrating power efficiency suitable for long-term operation in unattended settings.

Overall, the system achieves a practical balance between security, responsiveness, and energy efficiency. It is suitable for applications in shared or remote storage environments, and its modular design offers a solid foundation for further extensions, such as wireless connectivity, remote access, or cloud-based data logging.

4.2 Future Work

In future iterations, the smart locker system could be enhanced through the integration of wireless communication (e.g., Wi-Fi or BLE) for remote access, cloud-based data logging for improved traceability, and biometric authentication to strengthen security. The incorporation of additional sensors and lightweight AI algorithms may further improve event detection accuracy. Moreover, transitioning to a custom PCB design would support hardware miniaturization and deployment readiness, while exploring energy harvesting solutions such as solar power could enable long-term autonomous operation in off-grid environments.

Reference

[1] A. Rymaszewska, P. Helo and A. Gunasekaran, "IoT powered servitization of manufacturing – an exploratory case study," International Journal of Production Economics, vol. 192, pp. 92-105, 2017.

[2] T. Zhang, Y. Li, J. Li, and X. Fan, “Design of an Android-Based Intelligent Storage Cabinet Control Platform,” Mechanical Design and Manufacturing, no. 6, pp. 28–31, May 2013. DOI: 10.3969/j.issn.1001-2257.2013.03.008

[3] K. F. Yuen et al, "The determinants of customers’ intention to use smart lockers for last-mile deliveries," Journal of Retailing and Consumer Services, vol. 49, pp. 316-326, 2019.

[4] E. K. H. Leung, Z. Ouyang and G. Q. Huang, "Community logistics: a dynamic strategy for facilitating immediate parcel delivery to smart lockers," International Journal of Production Research, vol. 61, (9), pp. 2937-2962, 2023.

[5] A. Arno, A. Toyoda, and K. Sasase, “Accelerometer assisted authentication scheme for smart bicycle lock,” in Proc. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, Dec. 2015, pp. 160–165. doi: 10.1109/WF-IoT.2015.7389108

[6] X. Sun, E. A. Baldwin, J. Manthey, C. Dorado, T. Rivera, and J. Bai, "Effect of Preprocessing Storage Temperature and Time on the Physicochemical Properties of Winter Melon Juice," *Journal of Food Quality*, May 2, 2022. DOI: 10.1155/2022/3237639.

Appendix

Below are codes of main.cpp, TMP102.h/.cpp, ADXL345.h/.cpp:

1. main.cpp

```
#include "mbed.h"
#include "rtos.h"
#include "ADXL345.h"
#include "TMP102.h"
#include <cmath>
#include <cstring>
using namespace std;

// --- I2C and SPI configuration ---
#define I2CSDA D4
#define I2CSCL D5
#define TMP102_TEMP_REG 0x00
#define TMP102_CONFIG_REG 0x01

#define SPI1_MOSI D11
#define SPI1_MISO D12
#define SPI1_SCLK D13
#define SPI1_CS A3

// --- Control pin definitions ---
#define BUZZER_PIN D2
#define LOCKED_LED_PIN A1
#define UNLOCKED_LED_PIN A2
#define ALERT_LED_PIN D0
#define SWITCH_ONE_PIN D10
#define SWITCH_ZERO_PIN A0
#define PASSWORD_CORRECT_LED_PIN D3
#define PASSWORD_INCORRECT_LED_PIN D6
#define WAKE_INTERRUPT_PIN D9

#define DEBOUNCE_TIME_MS 200

const int PASSWORD_LENGTH = 5;
const int PASSWORD_SEQUENCE[PASSWORD_LENGTH] = {1,0,1,1,0}; // The passcode

const int TMP102_ADDRESS = 0x90;
const float TEMPERATURE_THRESHOLD_HIGH = 30.0;
const float TEMPERATURE_THRESHOLD_LOW = 10.0;
```

```
const float ADXL345_TAMPER_THRESHOLD_G = 0.1;

bool stable_timer_running = false;
volatile bool wake_requested = false;

#define ADXL_SLEEP_THRESHOLD_XY 1.0f
#define ADXL_SLEEP_THRESHOLD_Z_LOW 0.8f
#define ADXL_SLEEP_THRESHOLD_Z_HIGH 1.2f
#define ENV_SLEEP_CHECK_DURATION 30s

// --- Hardware interface objects ---
I2C tmp102_i2c(I2CSDA, I2CSCL);
SPI adxl345_spi(SPI1_MOSI, SPI1_MISO, SPI1_SCLK);
DigitalOut adxl345_cs(SPI1_CS);
DigitalOut buzzer(BUZZER_PIN);
DigitalOut locked_led(LOCKED_LED_PIN);
DigitalOut unlocked_led(UNLOCKED_LED_PIN);
DigitalOut alert_led(ALERT_LED_PIN);
InterruptIn switch_one(SWITCH_ONE_PIN);
InterruptIn switch_zero(SWITCH_ZERO_PIN);
InterruptIn wake_interrupt(WAKE_INTERRUPT_PIN);
DigitalOut password_correct_led(PASSWORD_CORRECT_LED_PIN);
DigitalOut password_incorrect_led(PASSWORD_INCORRECT_LED_PIN);

// --- System state variables ---
volatile bool system_locked = true;
volatile int incorrect_password_attempts = 0;
volatile int current_password_input_index = 0;
volatile bool switch_one_pressed = false;
volatile bool switch_zero_pressed = false;
volatile bool sensors_sleeping = false;

// --- Events and timers ---
EventFlags password_event_flags;
#define PASSWORD_INPUT_READY 0x01
Timer debounce_timer_one;
Timer debounce_timer_zero;
Timer stable_condition_timer;

// --- Sensor data buffers ---
char adxl_buffer[6];
int16_t adxl_raw_data[3];
float accel_x, accel_y, accel_z;
char tmp102_temp_reg_data[2];
```

```
float current_temperature_c;

// --- Interrupt handlers: switch debouncing ---
void on_switch_one_press() {
    if (debounce_timer_one.elapsed_time().count() > DEBOUNCE_TIME_MS * 1000)
{
        switch_one_pressed = true;
        password_event_flags.set(PASSWORD_INPUT_READY);
        debounce_timer_one.reset();
    }
}

void on_switch_zero_press() {
    if (debounce_timer_zero.elapsed_time().count() > DEBOUNCE_TIME_MS * 1000)
{
        switch_zero_pressed = true;
        password_event_flags.set(PASSWORD_INPUT_READY);
        debounce_timer_zero.reset();
    }
}

// --- Control LEDs and buzzer ---
void set_system_main_feedback(bool locked, bool alert) {
    locked_led = locked;
    unlocked_led = !locked;
    alert_led = alert;
    if (alert) buzzer = 1;
    else if (!password_incorrect_led) buzzer = 0;
}

// --- Tamper detection ---
void handle_tamper_detection() {
    if (fabs(accel_y) > ADXL345_TAMPER_THRESHOLD_G || fabs(accel_x) >
ADXL345_TAMPER_THRESHOLD_G) {
        printf("Vibrations detected!\n");
        set_system_main_feedback(system_locked, true);
        ThisThread::sleep_for(1s);
        set_system_main_feedback(system_locked, false);
    }
}

// --- Environmental temperature anomaly detection ---
void handle_environmental_monitoring() {
```

```
    if (current_temperature_c > TEMPERATURE_THRESHOLD_HIGH ||
current_temperature_c < TEMPERATURE_THRESHOLD_LOW) {
        printf("Abnormal temperature: %.2f\n", current_temperature_c);
        set_system_main_feedback(system_locked, true);
        ThisThread::sleep_for(1s);
        set_system_main_feedback(system_locked, false);
    }
}

// --- Wake up ---
void wake_from_interrupt() {
    wake_requested = true;
}
// --- Sleep judge ---
bool conditions_within_sleep_range() {
    return (current_temperature_c >= TEMPERATURE_THRESHOLD_LOW &&
            current_temperature_c <= TEMPERATURE_THRESHOLD_HIGH &&
            fabs(accel_x) <= ADXL_SLEEP_THRESHOLD_XY &&
            fabs(accel_y) <= ADXL_SLEEP_THRESHOLD_XY &&
            accel_z >= ADXL_SLEEP_THRESHOLD_Z_LOW && accel_z <=
ADXL_SLEEP_THRESHOLD_Z_HIGH);
}

void set_adxl345_sleep_mode(bool sleep) {
    adxl345_cs = 0;
    adxl345_spi.write(0x2D);
    adxl345_spi.write(sleep ? 0x04 : 0x08);
    adxl345_cs = 1;
}

void set_tmp102_sleep_mode(bool sleep) {
    char config_data[3] = {TMP102_CONFIG_REG, (char)(sleep ? 0xE0 : 0x60),
0xA0};
    tmp102_i2c.write(TMP102_ADDRESS, (char*)config_data, 3);
}


// password input
void password_input_thread() {
    while (true) {
        // Wait for a button press (either switch_one or switch_zero)
        password_event_flags.wait_any(PASSWORD_INPUT_READY);

        // Clear button flags immediately after the event is received
        // This is crucial for single-shot processing of a button press.
```

```
        bool current_switch_one_state = switch_one_pressed;
        bool current_switch_zero_state = switch_zero_pressed;
        switch_one_pressed = false;
        switch_zero_pressed = false;

        if (system_locked) { // Only process password input if the system is
locked
            int input_digit = -1;
            if (current_switch_one_state) {
                input_digit = 1;
            } else if (current_switch_zero_state) {
                input_digit = 0;
            }

            if (input_digit != -1) {
                //printf("Password input: %d\n\r", input_digit);
                if (input_digit ==
PASSWORD_SEQUENCE[current_password_input_index]) {
                    current_password_input_index++;
                    if (current_password_input_index == PASSWORD_LENGTH) {
                        // Correct password entered!
                        printf("\033[2J\033[1;1H");
                        printf("Present state of input password: 10110\nPassword
CORRECT! Locker UNLOCKED.\n\r");
                        system_locked = false; // Unlock the locker
                        incorrect_password_attempts = 0; // Reset incorrect
attempts
                        current_password_input_index = 0; // Reset for next time
                        password_correct_led = 1; // Green LED on
                        password_incorrect_led = 0; // Red LED off
                        buzzer = 0; // Ensure buzzer is off, main system
feedback might turn it on later
                        set_system_main_feedback(system_locked, false); //
Update main system LEDs
                        ThisThread::sleep_for(2s); // Keep green LED on for 2
seconds
                        password_correct_led = 0; // Turn off green LED

                    } else {
                        //printf("Correct digit. Progress: %d/%d\n\r",
current_password_input_index, PASSWORD_LENGTH);

                        printf("\033[2J\033[1;1H");
                        printf("Present state of input password: ");
```

```
                    switch (current_password_input_index){
                        case 1: printf("1****\n");    break;
                        case 2: printf("10***\n");    break;
                        case 3: printf("101**\n");    break;
                        case 4: printf("1011*\n");    break;
                        default: printf("something wrong\n");
                    }

                }
            } else {
                // Incorrect digit
                printf("Password INCORRECT digit at position %d!
Resetting.\n\r", current_password_input_index + 1);
                incorrect_password_attempts++;
                current_password_input_index = 0; // Reset password input
sequence

                password_incorrect_led = 1; // Red LED on
                buzzer = 1; // Buzzer on for incorrect password
                password_correct_led = 0; // Green LED off

                printf("Incorrect attempts: %d\n\r",
incorrect_password_attempts);
                if (incorrect_password_attempts >= 3) { // Example: 3
incorrect attempts for lockout
                    printf("Too many incorrect password attempts! System
temporarily locked out for password input.\n\r");
                    // Implement a longer lockout period if desired, e.g.,
ThisThread::sleep_for(10s);
                    // For now, we just reset attempts.
                    incorrect_password_attempts = 0; // Reset attempts after
lockout
                }
                ThisThread::sleep_for(1s); // Keep red LED and buzzer on for
1 second
                password_incorrect_led = 0; // Turn off red LED
                buzzer = 0; // Turn off buzzer
            }
        }
    } else { // System is UNLOCKED
        if (current_switch_one_state) { // Assuming Switch '1' can also
re-lock if system is unlocked
            printf("Switch '1' pressed while unlocked. Relocking
system...\n\r");
```

```
            system_locked = true;
            set_system_main_feedback(system_locked, false);
        }
        // Clear any flags if not used for re-locking, this part is
already covered by clearing flags above.
    }
  }
}

// --- Enhanced Sensor Monitoring Thread ---

void enhanced_sensor_monitoring_thread() {
  wake_interrupt.fall(&wake_from_interrupt);
  wake_interrupt.mode(PullUp);

  while (true) {
    if (!sensors_sleeping) {
      printf("\033[2J\033[1;1H");
      read_adxl345_data();
      read_tmp102_data();

      if (conditions_within_sleep_range()) {
        if (!stable_timer_running) {
          stable_condition_timer.start();
          stable_timer_running = true;
        } else if (stable_condition_timer.elapsed_time() >=
ENV_SLEEP_CHECK_DURATION) {
          printf("\033[2J\033[1;1H");
          printf("Stable 10s. Entering sleep.\n");
          set_adxl345_sleep_mode(true);
          set_tmp102_sleep_mode(true);
          sensors_sleeping = true;
          stable_condition_timer.stop();
          stable_timer_running = false;
        }
      } else {
        stable_condition_timer.reset();
        stable_timer_running = false;
      }

      handle_tamper_detection();
      handle_environmental_monitoring();

    } else {
```

```
            if (wake_requested) {
                printf("Sensors have been woken up, resuming monitoring.\n");
                set_adxl345_sleep_mode(false);
                set_tmp102_sleep_mode(false);
                sensors_sleeping = false;
                stable_condition_timer.reset();
                stable_timer_running = false;
                wake_requested = false;
            }
        }

        ThisThread::sleep_for(500ms);
    }
}

int main() {
    initialize_adxl345_spi();
    configure_tmp102();

    debounce_timer_one.start();
    debounce_timer_zero.start();

    switch_one.fall(&on_switch_one_press);
    switch_one.mode(PullUp);
    switch_zero.fall(&on_switch_zero_press);
    switch_zero.mode(PullUp);

    set_system_main_feedback(system_locked, false);
    password_correct_led = 0;
    password_incorrect_led = 0;
    buzzer = 0;

    printf("System Init. State: %s\n", system_locked ? "LOCKED" :
"UNLOCKED");

    Thread password_thread;
    password_thread.start(callback(password_input_thread));

    Thread sensor_thread;
    sensor_thread.start(callback(enhanced_sensor_monitoring_thread));

    while (true) {
        ThisThread::sleep_for(1s);
    }
```

```
}
```

2. TMP102.h

```
// TMP102.h
#ifndef TMP102_H
#define TMP102_H

#include "mbed.h"

extern I2C tmp102_i2c;
extern const int TMP102_ADDRESS;
extern char tmp102_temp_reg_data[2];
extern float current_temperature_c;

void configure_tmp102();
void read_tmp102_data();

#endif // TMP102_H
```

3. TMP102.cpp

```
// TMP102.cpp
#include "TMP102.h"

#define TMP102_TEMP_REG 0x00
#define TMP102_CONFIG_REG 0x01

void configure_tmp102() {
    char config_data[3];
    config_data[0] = TMP102_CONFIG_REG;
    config_data[1] = 0x60;
    config_data[2] = 0xA0;
    tmp102_i2c.write(TMP102_ADDRESS, config_data, 3);
}

void read_tmp102_data() {
    char temp_reg_address[1];
    temp_reg_address[0] = TMP102_TEMP_REG;
    tmp102_i2c.write(TMP102_ADDRESS, temp_reg_address, 1);

    tmp102_i2c.read(TMP102_ADDRESS, tmp102_temp_reg_data, 2);
```

```
    unsigned short raw_temp = (tmp102_temp_reg_data[0] << 4) |
(tmp102_temp_reg_data[1] >> 4);
    current_temperature_c = 0.0625 * raw_temp;

    printf("TMP102: Temperature = %.3f C\n\r", current_temperature_c);
}
```

4. ADXL345.h

```
// ADXL345.h
#ifndef ADXL345_H
#define ADXL345_H

#include "mbed.h"

extern SPI adxl345_spi;
extern DigitalOut adxl345_cs;
extern char adxl_buffer[6];
extern int16_t adxl_raw_data[3];
extern float accel_x, accel_y, accel_z;

void initialize_adxl345_spi();
void read_adxl345_data();

#endif // ADXL345_H
```

5. ADXL345.cpp

```
// ADXL345.cpp
#include "ADXL345.h"

void initialize_adxl345_spi() {
    adxl345_cs = 1;
    adxl345_spi.format(8, 3);
    adxl345_spi.frequency(2000000);

    adxl345_cs = 0;
    adxl345_spi.write(0x31);
    adxl345_spi.write(0x0B);
    adxl345_cs = 1;

    adxl345_cs = 0;
    adxl345_spi.write(0x2D);
```

```
    adxl345_spi.write(0x08);
    adxl345_cs = 1;

    ThisThread::sleep_for(100ms);
    adxl345_cs = 0;
    adxl345_spi.write(0x80 | 0x00);
    char device_id = adxl345_spi.write(0x00);
    adxl345_cs = 1;

    printf("DEBUG: ADXL345 Device ID: 0x%X (Expected 0xE5)\n\r", (unsigned
char)device_id);

    if (device_id == 0xE5) {
        printf("DEBUG: ADXL345 Initialisation successful!\n\r");
    } else {
        printf("ERROR: ADXL345 Initialisation FAILED!\n\r");
    }
}

void read_adxl345_data() {
    adxl345_cs = 0;
    adxl345_spi.write(0x80 | 0x40 | 0x32);

    for (int i = 0; i < 6; i++) {
        adxl_buffer[i] = adxl345_spi.write(0x00);
    }
    adxl345_cs = 1;

    adxl_raw_data[0] = (adxl_buffer[1] << 8) | adxl_buffer[0];
    adxl_raw_data[1] = (adxl_buffer[3] << 8) | adxl_buffer[2];
    adxl_raw_data[2] = (adxl_buffer[5] << 8) | adxl_buffer[4];

    accel_x = 0.004 * adxl_raw_data[0];
    accel_y = 0.004 * adxl_raw_data[1];
    accel_z = 0.004 * adxl_raw_data[2];

    printf("ADXL345: X = %+1.2f g\t Y = %+1.2f g\t Z = %+1.2f g\n\r",
accel_x, accel_y, accel_z);
}
```