uOttawa

Winter 2024

Lab 4: Alarm System with Motion Detection

SEG4145 - Real Time Systems

Lab Section: A06

Due Date: March 30th, 2024

Group 1

Alexis Verana (300116080)

Gavin Gao (300190846)

Alexander Choukeir (300121777)

Hened Saade (300111592)

Jayden Lachhman (8791694)

University of Ottawa

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The goal of this lab is to build upon the Lab 3 alarm system by adding a PIR motion detector and a buzzer. This lab will include the Nucleo-F446RE development in conjunction with the STM32CubeIDE, as seen in previous labs. This lab report will follow the methodology of adding a motion detector to the current alarm system.

2. Objectives

The objective of this lab includes:

- Discerning the additional components needed for the hardware portion
- Dissecting how the PIR motion functions in conjunction with the buzzer
- Programming an additional task to the current code to detect whether there is a person in the vicinity of the alarm system

3. Problem Analysis

In this lab, the team is tasked with connecting a PIR Motion Detector and a buzzer to the alarm system composed in the previous lab using a 4x4 Keypad, an OLED LCD Display Module, and Nucelo-F446RE Development Board. The alarm system has a series of functional expectations that simulate a traditional alarm system, consisting of everything from the previous lab, but now the additional functionalities of interfacing a PIR Motion Detector which detects nearby movement and a buzzer that behaves as a distress signal when a potential threat is acknowledged.

In order to simulate the completed alarm system, a main task consisting of the sequence of operations that should logically follow if motion is detected needs to be outlined. The sequence of behavioural subtasks determined to achieve this are: understanding how to integrate the PIR Motion Detector and the buzzer with the 4x4 Keypad, OLED LCD Display Module, LEDs, and resistors into the circuit such that all components can communicate and send data to the program through the development board, programming a 60 second delay into the alarm system following its arming to allow the user to exit the premises, also programming a 60 second grace period to allow the user to enter the passcode and arm the system before the buzzer can start ringing if motion is detected, and updating the buzzer to synchronise with the LEDs to establish the state of the alarm system (e.g., if the alarm system is deactivated, the OLED Display Module should then read “NOT ARMED”, the green LED should be illuminated, and the buzzer should not sound if motion is detected). These subtasks reflect the behaviours of the simulated alarm and the conditions that must be met to achieve the overall objective of this lab.

The physical design of the board is a foundational task, and not one of the two main tasks mentioned. This design would be achieved by a circuit that both connects the PIR Motion Detector directly to the development board and the buzzer to the circuit board which is then wired to the development board.

Lead resistors are integrated with the buzzer to dampen the current drawn by it, and as described in the lab manual, the PIR Motion Detector is connected to the development board using 3-pins. The rest of the alarm system is identical to the circuit designed in Lab 3.

The main programming task follows from physically designing the circuit where the alarm system must capture the data input from the PIR Motion Detector and pass it to the buzzer while acknowledging the state of the alarm system that it must be coordinated with. Therefore, a *detectMotionTask*, should be used which uses an integer variable, say *countdown*, to represent the 60 second grace period that the user would have after arming the alarm system and displays the remaining seconds in real-time as it counts down on the OLED Display Module. Once this countdown reaches 0, motion detection would be enabled by the readpin given an output signal from the PIR sensor and the alarm system would be considered active and awaiting a threat. Then, and a second timer, say *countdown2,* will begin counting down immediately upon motion detection to give potential threats the chance to disarm the system should it be the user, but if the second timer reaches 0 and if there's still motion detected, the buzzer will sound by toggling the GPIO_PIN 9 to on.

4. Solution Design

The proposed approach/plan to solve the PIR motion detector and buzzer addition problem is to utilize the ReadPin, SSD1306, WritePin, and osDelay functions to achieve the desirable behavior. The plan will also make use of several if statements to achieve the correct system functionality. The main problem is to add additional functionality to the alarm system done in lab 3. The problem involves detecting motion and triggering the buzzer at the right moment in time. In the current system, we need to add the ability for the alarm system to detect motion and trigger the buzzer. We also plan to add the countdown from 59 to 0 seconds. With the use of if statements and functions such as ReadPin, SSD1306, WritePin, and osDelay, we are able to detect motion from the PIR motion detector, turn on/off the buzzer at the right time and display the countdown on the LCD.

The plan to solve the problem involves the usage of several functions to detect motion from the PIR motion detector, turn on/off the buzzer, and display the countdown on the LCD. Firstly, we will be using various pin-related functions. We will be using ReadPin and WritePin. The ReadPin function allows you to read the input state of the specified GPIO pin. The WritePin function allows you to change the output state of the specified GPIO pin.

There are also other various functions that will be used. We will be using several SSD1306 functions. We will be using the GotoXY, Puts, and UpdateScreen functions. The GotoXY function goes to a specific X and Y coordinate on the LCD screen. The Puts function puts a specified message on the LCD screen. Finally, the UpdateScreen function updates the screen. Additionally, the osDelay function will be utilized. The osDelay function pauses the program for a specified amount of time.

We plan to use the code done for lab 3 and modify it accordingly to be able to detect motion, trigger the buzzer, and have a countdown from 59 to 0. We plan to add 2 more pins and another task to our code from lab 3. We will have 1 input pin for the PIR motion detector and 1 output pin for the buzzer. Since we are adding a motion detector and a buzzer, we will be using an additional function to handle motion detection and buzzer triggering. The two first tasks (SetPasswordTask and EnterPasswordTask) will remain unchanged in terms of their responsibility. We plan to make some small additions to these two first tasks so that the countdown and the buzzer can be incorporated into the alarm system (as seen in *Figure 4.1 and Figure 4.2*).

- The function DetectMotionTask() will take no input parameters. This function is responsible for detecting motion, starting the countdown, and triggering the buzzer. It will determine when the buzzer should be turned on (as seen in *Figure 4.3*).

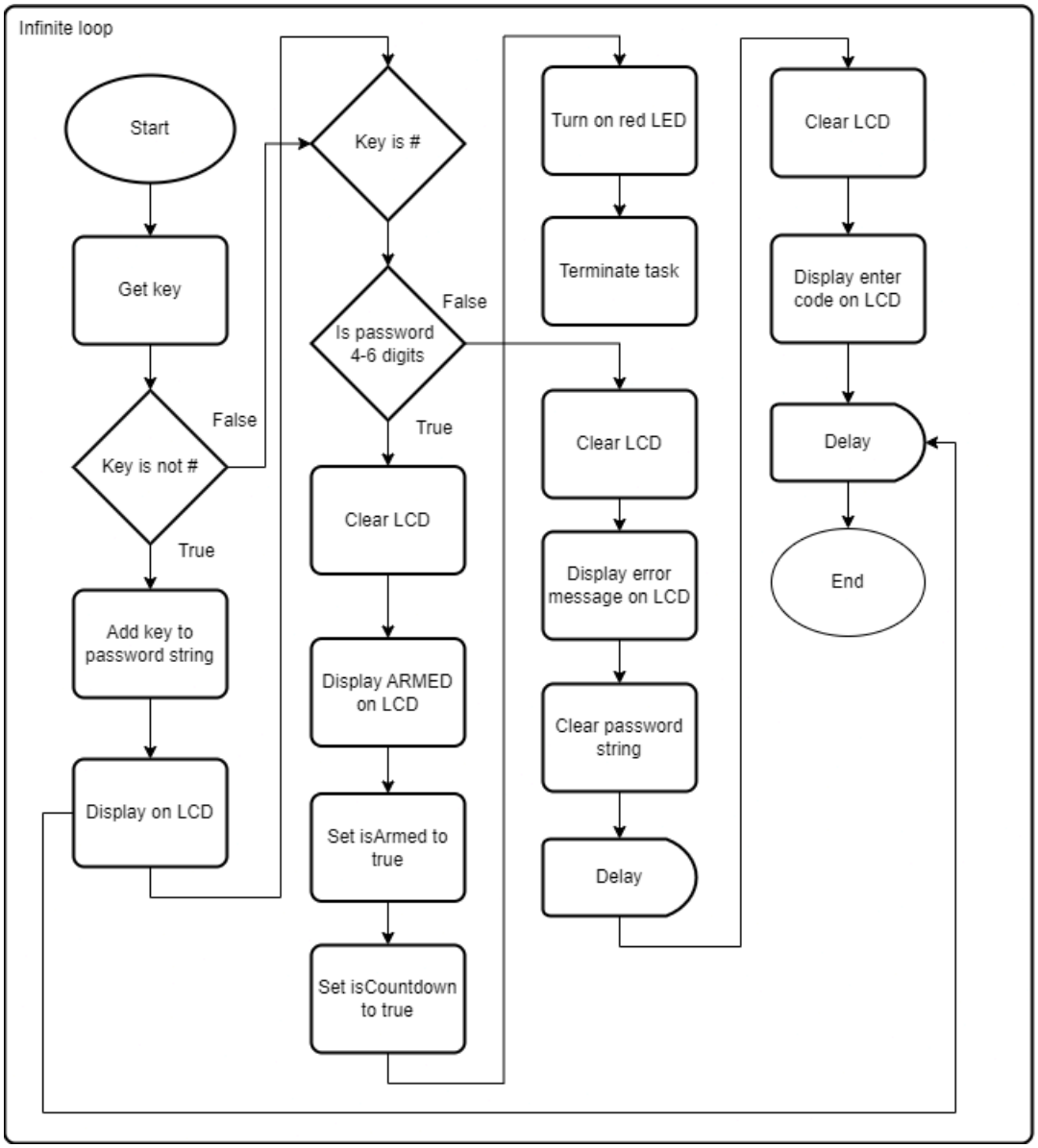

Figure 4.1: *Updated flowchart of SetPasswordTask*

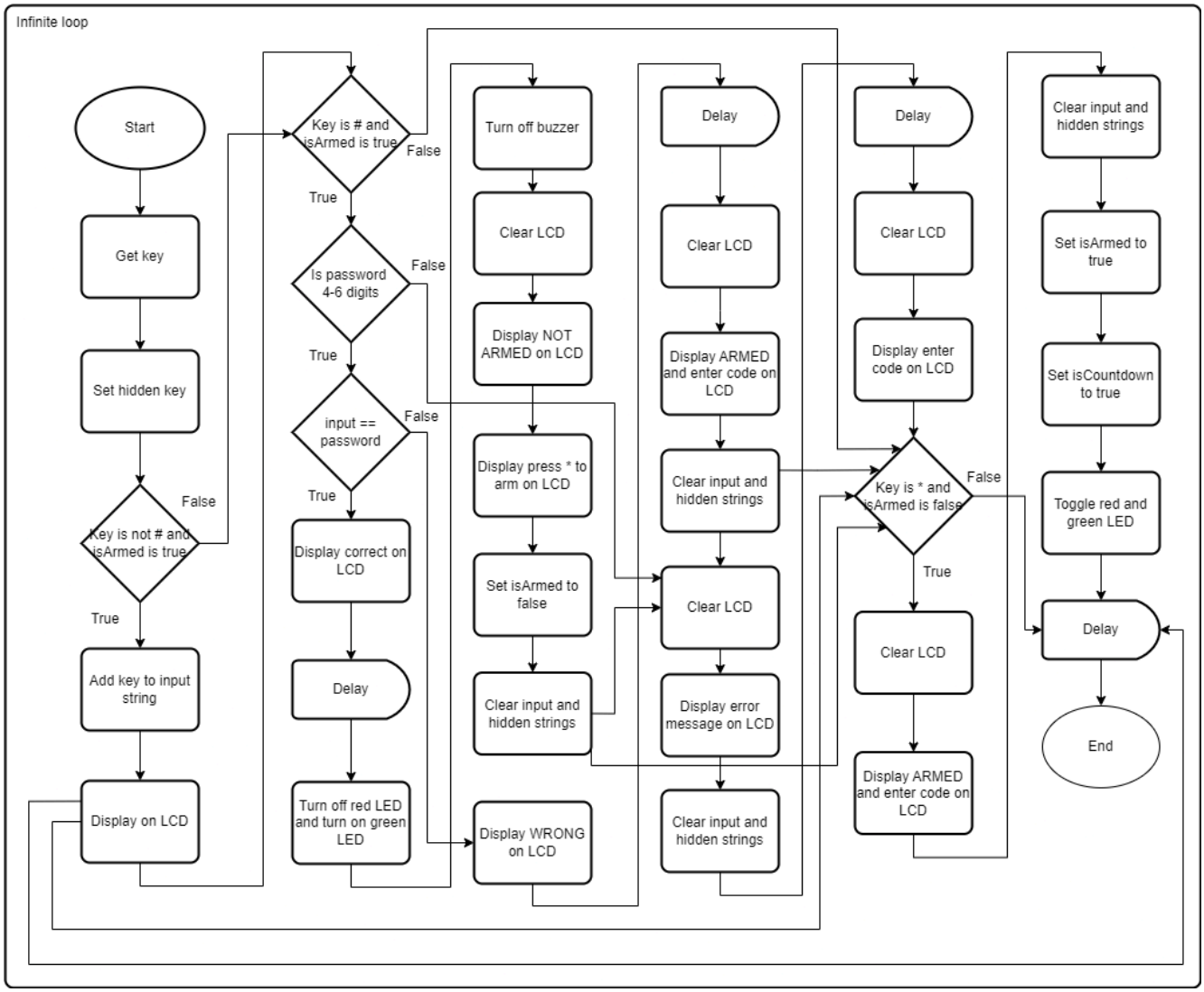

Figure 4.2: *Updated flowchart of EnterPasswordTask*

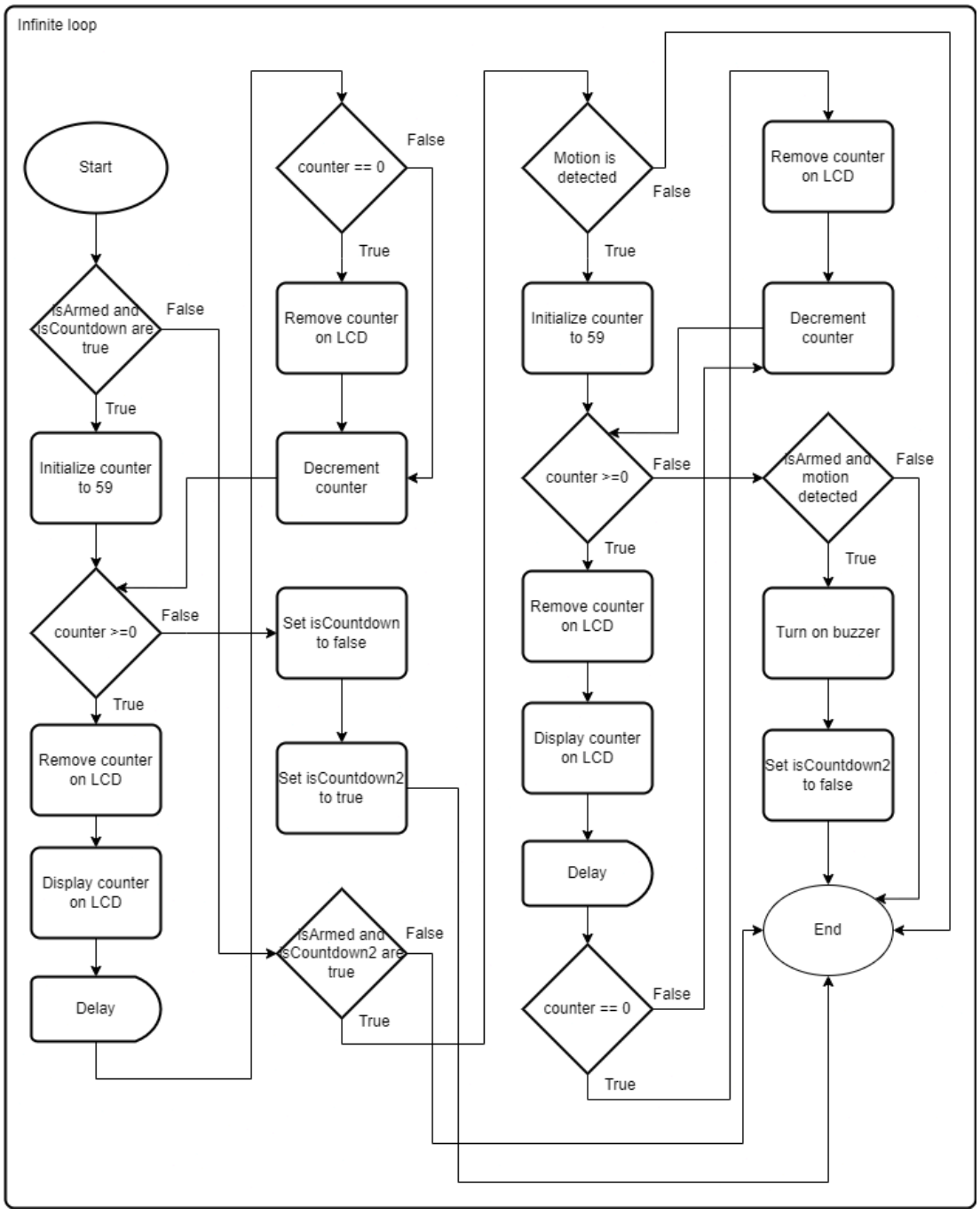

Figure 4.3: *Flowchart of DetectMotionTask*

The DetectMotionTask will be set with normal priority. In the current system, after the password has been set, the enter password task will be executed. Since the DetectMotionTask has the same priority as the EnterPasswordTask, the DetectMotionTask will also be running at the same time. So, the user will able to enter the password and the system will also be detecting motion. Within the DetectMotionTask, the system will trigger a countdown from 59 to 0 once the alarm is triggered and will display it on the LCD. This is to allow the user to leave the premises. Once the PIR motion detector detects motion, it will start another countdown from 59 to 0 and will display it on the LCD.

This will give the user time to enter the password before triggering the buzzer. After this countdown, if motion is still being detected, the buzzer will be turned on. This buzzer will stay on until the correct password is entered. After the correct password is entered, the buzzer will be turned off. This detect motion task is also running in an infinite loop, so it is constantly checking whether motion is being detected.

5. Component Usage

Component List	Purpose	Necessity
PIR Motion Detector	The PIR Motion Detector reads differentials of infrared radiation emitted nearby to register the presence of movement. It is interfaced directly into the development board from which positive differential changes between both halves of the sensor constitute the presence of movement, which the alarm system uses as a defence mechanism.	To complete the practical design of an alarm system, a PIR Motion Detector is used to register nearby movement which represents the presence of a potential threat. In a conventional alarm system, potential threats need to be discouraged. But in order to signal distress, those potential threats need to be acknowledged, and constituted by a signal differential using the sensor of the PIR Motion Detector, this can be achieved.
Buzzer	The buzzer is interfaced with the alarm system through physically wired connections and a resistor in the circuit board which connects to the development board. The buzzer complements and completes the alarm system defensively to allow the user to set the alarm and exit the premises safely, and also acts as another layer of intimidation.	The buzzer is needed to alert the user of certain actions in response to the state of the alarm system. Once the PIR Motion Detector discovers movement, the presence of this threat must be broadcasted, and the ringing of the buzzer achieves that. Without the buzzer, there would be no auditory stimulus to discourage potential threats in close proximity to the alarm system.
4x4 Keypad	Serves as the input interface and allows the user to enter alphanumeric characters. This component connects to the Nucleo-F446RE using an 8-pin connector, where 4 pins in the connector represent the columns	The keypad is needed to allow the user to communicate with the program in the context of the alarm system. Using alphanumeric sequences the user must enter digits from the keypad to comply with the expectations

	of the conceptual matrix representation of the keypad, and 4 pins represent the rows respectively. Each cell in this matrix corresponds to a button on the keypad and using this, the user can enter inputs.	of the program, which are captured by the prompts and state indicated on the Display Module.
OLED LCD Display Module SSD1306 IIC 128x32 LCD OLED Display 3.3V ~ 5V 4-pin from Teyleten Robot 0.91 inch (x1)	The OLED LCD Display Module is a four-pin screen that can display text/values depending on how it is programmed. In this lab, the module must display information to the user indicating the state of the alarm system. The states include: setting a passcode, entering a passcode, the validity of the passcode (“CORRECT” or “WRONG”), and whether the system is (“ARMED” or “NOT ARMED”). For each of those states, a corresponding prompt to the user is displayed.	The OLED LCD Display Module acts as a form of visual verification that the circuit and code have been interfaced properly. The electrical circuit design must integrate the 4x4 keypad with the code, such that once the program takes input from the keypad, the OLED LCD Display Module should display how that input has been received and what state the system must transition to in the alarm system context.
Nucleo-F446RE (x1)	Programmable development board which interfaces with the STMCube32IDE. Using its ports, the 4x4 Keypad, OLED LCD Display Module, and circuit board can be interfaced using the aforementioned IDE. The board provides connectivity to the other components using power, analog, input, and ground pins.	To accomplish the assigned tasks, the 4x4 Keypad, OLED LCD Display Module, and circuit board must be powered and connected to this development board. After connecting the computer running the IDE, this enables controllable functioning of these components connected via the breadboard circuit.
Breadboard Circuit (x1)	Allows us to create a simple electronic circuit using low-power components. In this lab, it is used to connect both the OLED LCD panel and LEDs to the development board such that communication between the components is established.	The OLED LCD panel and the LEDs need to be connected to the breadboard because that enables their connectivity. The breadboard effectively serves as the foundation for which electricity passes through and connects together all of the components that are plugged into it.
LEDs (x2)	There are two LEDs in this circuit	The LEDs are indicators of the

	design: one (x1) green LED, and one (x1) red LED. Each LED is connected to the circuit board to serve as a visual indicator of the state of the alarm system. The red LED should be illuminated when the state of the alarm system is "ARMED", and the GREEN LED should be illuminated when the alarm system is "NOT ARMED".	state of the alarm system, thus they are important to demonstrate that the code, OLED LCD Panel and the 4x4 Keypad are interfaced properly. These LEDs provide more confidence that all components are synchronised with the state of the machine.
Resistors (x3)	The resistors are used to control the amount of current that passes through the LEDs.	Resistors are implemented to ensure that the LEDs and the Buzzer do not draw an overwhelming amount of current from the circuit which may cause them to be destroyed.
Wires (x9)	Wires are used to send electrical signals from one component to another.	Each of the electrical components need to be able to communicate with one another to achieve the goal of this lab, and wires transmit the signals that enable this communication. The OLED LCD Display Module, LEDs, 4x4 Keypad, Buzzer, and PIR Motion Detector can communicate with one another because each of them is centrally connected to pins in the development board and powered by the circuit board.

Table 5.1. *Component list and descriptions*

6. Algorithmic Approach

```
extern char key;
char hold[7];
char password[7];
char hidden[7];
int isArmed = 0;
int isCountdown = 0;
int isCountdown2 = 0;
/* USER CODE END PV */
```

Figure 6.1: *Code snippet of variables*

Setting up variables char key for single user input digit, char hold array for the entire user input digits holder; char password array for a pre-set password, used to compare with user input late; char hidden array for masking digits password showing on LCD; int isArmed as a flag to indicate system status in ARMED or UNARMED. Int isCountdown and isCountdown2 flags for timer status.

As seen in Code A.1.1 and Code A.1.2, we create tasks and initialize input & output configuration

The task seen in Code A.1.3 and Code A.1.4 is responsible for initially setting the system password in the background, after getting inputs from the user, deciding if it is a valid input from 4-6 digits, if invalid showing an error message on LCD and clearing all inputs, asking to input again; if valid, the system goes into ARMED status and turn on red LED, then set isArmed flag and isCoutdown flag to 1.

The task seen in Code A.1.5, Code A.1.6, and Code A.1.7 is responsible for comparing user input passwords and pre-set system passwords, as well as showing hidden passwords on LCD. Each time the user inputs a single digit, the hidden password array has a star digit added and displays it on LCD.

When the # key is pressed and the system status in ARMED, it goes into validity check of passwords; if the input is invalid, or does not match the correct password, it shows an error message for 2s delay and clears all input password arrays; if valid, turns off red LED and Buzzer, turns on Green LED, display a correct message on LCD for 2s delay, the system status is now in UNARMED by setting isArmed flag to 0. At this point, the user can press the star key to return to the Armed status.

The task seen in Code A.1.8 and Code A.1.9 is responsible for checking any motion occurrence and setting up a countdown timer to trigger the buzzer. If the system is in ARMED status and the first timer flag isCountdown = 1, the countdown will begin; set a local variable counter = 59 and a while loop to countdown to 0, showing every time the countdown happens on LCD.

When the counter reaches 0, the counter will stop showing on the LCD, set the first timer isCountdown flag = 0, and the second timer flag isCountdown2 =1. When the system is in Armed status and the second timer flag = 1, it goes into the motion detection phase. If it detects motion by Readpin on an output signal from the PIR sensor, the second timer will begin the countdown, set a local variable counter 2= 59 and a while loop to countdown to 0; if counter2 reaches 0, the counter will stop showing on LCD, and at this point, if there's still motion detected, the buzzer will be sounding by setting on GPIO_Pin9 on.

7. Implementation Evaluation

In evaluating the performance and effectiveness of our implemented solution for the alarm system with the addition of a PIR motion detector and buzzer, several key aspects were considered.

Firstly, in real-world scenarios, the system demonstrated reliable functionality. The integration of the PIR motion detector and buzzer expanded the existing alarm system's capabilities to detect movement and sound an alarm accordingly. During testing, the system effectively responded to simulated intrusion scenarios, triggering the alarm upon detecting motion and providing visual and auditory feedback to the user.

Regarding the coding techniques used, the implementation of leveraged task-based programming, ensuring efficient multitasking and responsive behaviour of the system. By employing tasks for distinct functionalities such as password management, motion detection, and alarm triggering, the code maintained a modular and organised structure. (include appendix examples) This approach enhances readability and maintainability, facilitating easier debugging if necessary.

Moreover, the implementation adhered to coding best practices, such as clear variable naming, comments for clarification, and consistency throughout the code. These practices contributed to the overall readability of our code, which enabled easier comprehension for team members and other members involved.

During testing, the system largely met our expectations, effectively fulfilling the outlined use cases and requirements specified in the lab description. For instance, the countdown functionality provided ample time for the user to arm or disarm the system, enhancing usability and user experience. Additionally, the integration of the PIR motion detector seamlessly complemented the existing alarm system, enhancing its security capabilities.

However, initial hardware integration challenges, particularly with the PIR motion detector, required additional troubleshooting and iteration to achieve desired functionality. Additionally, there was a severe time crunch at play. Due to the time constraint, our initial implementation did not function as desired, which required further troubleshooting to achieve the desired results.

Overall, the implementation of the PIR motion detector and buzzer successfully enhanced the functionality of the alarm system, demonstrating effective integration of hardware components and task-based software design. By critically evaluating the execution of our code and addressing any identified issues, we have created an effective alarm system.

8. Results and Validation

An additional task was added to the existing code to implement the PIR motion detector and buzzer. The code first executes *EnterPasswordTask* this is where the user can set a password between 4 to characters (as seen in FIgure 8.1).

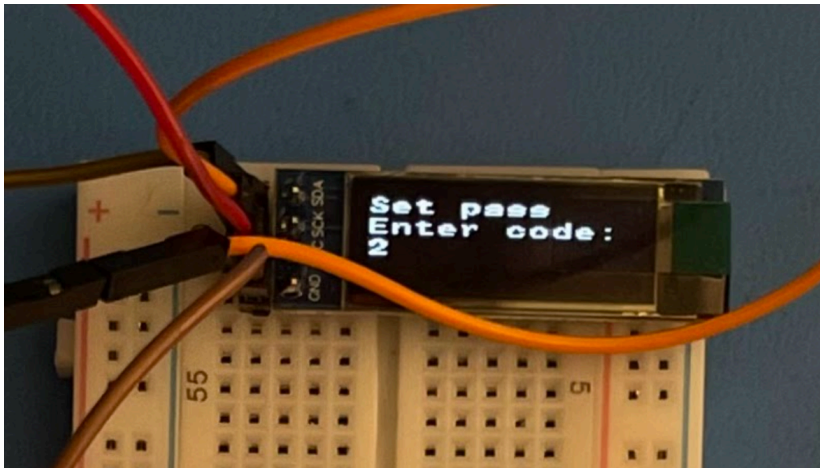

***Figure 8.1**: System prompts user to set a password*

Once the password has been set the alarm system is automatically ARMED, the Red LED is lit, and the countdown for the user to leave begins (as seen in Figure 8.2). During the countdown, the buzzer does not ring as the system allows the user to leave without setting off the alarm system.

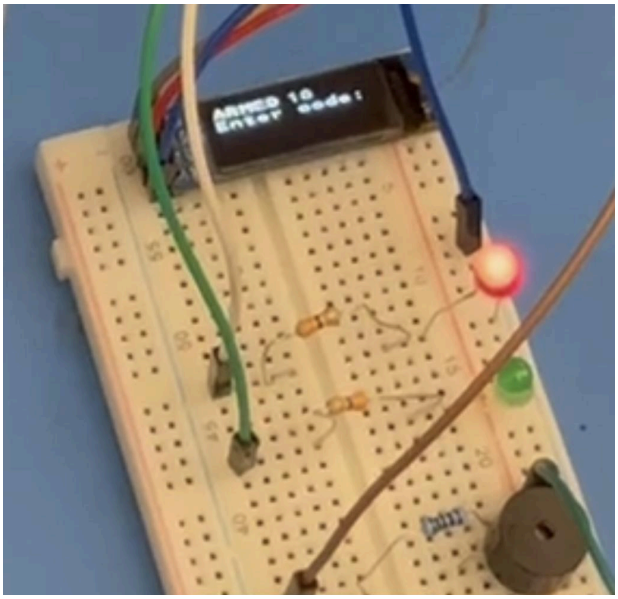

***Figure 8.2**: System is ARMED and countdown begins*

When the user comes back into view of the PIR motion detector, the buzzer alarms and a new countdown begins. During this countdown, the user has 60 seconds to enter in the correct password to disarm the system. As the user is entering the password the buzzer is ringing until the alarm is successfully disarmed. When the system is disarmed with the correct password, the Green LED is lit and the LCD Display shows NOT ARMED (as seen in Figure 8.3).

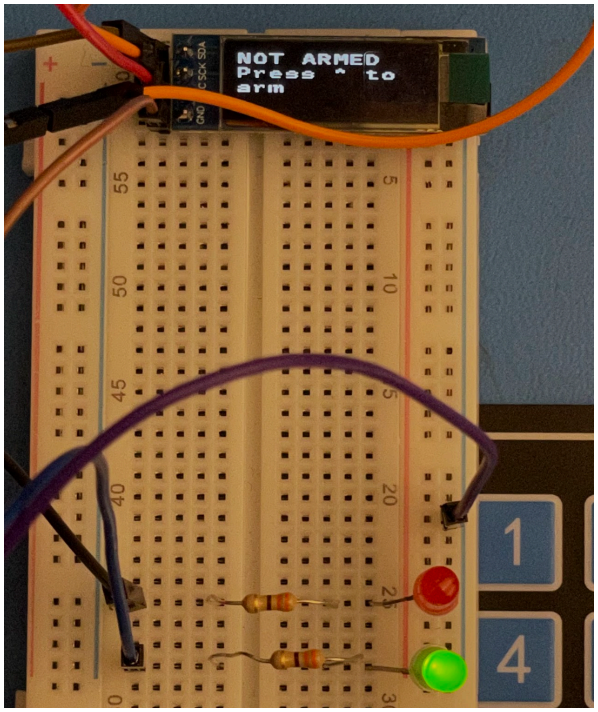

Figure 8.3: *Alarm system is successfully disarmed*

Similar to Lab 3, the alarm system still recognizes when the initial set password does not meet the password requirements and tells the user when the inputted password is incorrect. The *DetectMotionTask* successfully works alongside the previous tasks that were implemented in the previous lab. With this new task, the team was able to easily detect any errors as we separated the different concerns. Overall, the initial goal of implementing a motion detector and buzzer was obtained.

9. Addressing Challenges

During this lab, we encountered some challenges. One of the challenges we encountered was with the hardware components provided during the lab. Some of the hardware components such as the breadboard, PIR motion detector, and LEDs were not working. This caused a lot of back and forth trying to find functional equipment. This significantly hindered our progress and ultimately blocked us from progressing forward with the lab. This in turn resulted in wasted time. To address this challenge, we informed the TA about the components that were not working. The TA then provided us with replacement components. After replacing numerous items, we were able to get functional components. Even though it took a significant amount of time to get working components, the problem was solved in the end. We were able to carry out the lab smoothly.

Another challenge encountered was with the lab rescheduling. During the first lab session, we were informed that the next lab session would take place in April. Later, we received an email that the second lab session had been rescheduled to the next day. This change was communicated a day before

the rescheduled session. Because of this rescheduling, some team members were not able to attend since it conflicted with their schedules. Since some team members were not able to attend and the fact that we were not able to finish the lab during the first session, this greatly affected the ability to finish the lab and demo it properly. Due to this, there was an urgent need to finish the lab with limited time available. There was now a time crunch to finish up the lab. With perseverance and determination, we were able to finish up the lab with the limited time that was available. We were able to complete the lab and demo it in time. Even though we were faced with a significant challenge, the problem was solved in the end.

10. Conclusion

In conclusion, the initial goal of implementing the PIR motion detector and buzzer was achieved. The beginning of the lab required a lot of trial and error with the PIR motion detector. The team first made sure to understand how the PIR motion detector and buzzer would interact. When the team had a greater understanding of these new components, we were able to add them to the current alarm system. As a team, we faced a few challenges with the hardware components because we kept running issues to the materials not working. In turn, this consumed a lot of time by going back and forth with the different hardware components to find functional pieces. Despite this challenge, the team was able to put all the components together to create an alarm system the could detect motion and alert the user with a buzzer.

11. Workload Distribution

The report tasks were assigned and completed by the group members as seen in *Table 11.1*. The completion of all other tasks were equally contributed to by everyone.

Alexander Choukeir	Solution Design, Addressing Challenges, Code, Report Overview
Alexis Verana	Introduction, Objectives, Results and Validation, Code, Conclusion
Gavin Gao	Algorithmic Approach
Hened Saade	Implementation Evaluation, Table of Contents, Table of Figures, Table of Tables, Appendix, Report Overview
Jayden Lachman	Problem Analysis, Component Usage, Workload Distribution, Report Overview

Table 11.1. *Workload Distribution*

Appendix

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
/* USER CODE BEGIN 2 */
/* USER CODE BEGIN 2 */
  SSD1306_Init();
  SSD1306_GotoXY (0,0);
  SSD1306_UpdateScreen();
  SSD1306_Puts ("Set pass", &Font_11x18, 1);
  SSD1306_UpdateScreen();
  SSD1306_GotoXY (0, 20);
  SSD1306_UpdateScreen();
  SSD1306_Puts ("Enter code:", &Font_11x18, 1);
  SSD1306_UpdateScreen();
  HAL_Delay (500);
```

Code A.1.1: *Code snippets of initialization and task creation*

```
/* Create the thread(s) */
/* creation of setPasswordTask */
setPasswordTaskHandle = osThreadNew(SetPasswordTask, NULL, &setPasswordTask_attributes);

/* creation of enterPasswordTa */
enterPasswordTaHandle = osThreadNew(EnterPasswordTask, NULL, &enterPasswordTa_attributes);

/* creation of detectMotionTas */
detectMotionTasHandle = osThreadNew(DetectMotionTask, NULL, &detectMotionTas_attributes);
```

Code A.1.2: *Code snippets of initialization and task creation*

```
void SetPasswordTask(void *argument) // SET PASSWORD TASK
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {

          // Get key
          key = Get_Key();

          // If the key pressed is not #
          if (key != '#') {
                  strncat(password, &key, 1); // Append
                  SSD1306_GotoXY (0, 40);
                  SSD1306_UpdateScreen();
                  SSD1306_Puts (password, &Font_11x18, 1); // Display on LCD
                  SSD1306_UpdateScreen();
          }

          // If the key pressed is #
          if (key == '#') {

                  // If the password is between 4-6 digits
                  if (strlen(password) >= 4 && strlen(password) <= 6) {
                          SSD1306_Clear();
                          SSD1306_UpdateScreen();
                          SSD1306_GotoXY (0, 0);
                          SSD1306_UpdateScreen();
                          SSD1306_Puts ("ARMED", &Font_11x18, 1); // Display ARMED on LCD
                          SSD1306_UpdateScreen();
                          SSD1306_GotoXY (0, 20);
                          SSD1306_UpdateScreen();
                          SSD1306_Puts ("Enter code:", &Font_11x18, 1);
                          SSD1306_UpdateScreen();
                          isArmed = 1; // Set isArmed to true
                          isCountdown = 1; // Set isCountdown to true
                          HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6, GPIO_PIN_SET); // Turn on red LED
                          osThreadTerminate(setPasswordTaskHandle); // Terminate the task
                  } else { // If the password is not between 4-6 digits
```

Code A.1.3: *Code snippets of SetPasswordTask*

```
                } else { // If the password is not between 4-6 digits
                        SSD1306_Clear();
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 0);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("Password is", &Font_11x18, 1); // Display error message on LCD
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 20);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("not 4-6", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 40);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("digits", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                        strcpy(password, ""); // Reset password string
                        osDelay(2000); // Display error message for 2 seconds
                        SSD1306_Clear();
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0,0);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("Set pass", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 20);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("Enter code:", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                }

        }

        HAL_Delay (500);
  }
  /* USER CODE END 5 */
}
```

Code A.1.4: *Code snippets of SetPasswordTask*

```
void EnterPasswordTask(void *argument) // ENTER PASSWORD TASK
{
  /* USER CODE BEGIN EnterPasswordTask */
  /* Infinite loop */
  for(;;)
  {
          // Get key
          key = Get_Key();

          // Char for *
          char hiddenKey = '*';

          // If the key pressed is not # and isArmed is true
          if (key != '#' && isArmed) {
                  strncat(hold, &key, 1); // Append
                  strncat(hidden, &hiddenKey, 1); // Append
                  SSD1306_GotoXY (0, 40);
                  SSD1306_UpdateScreen();
                  SSD1306_Puts (hidden, &Font_11x18, 1); // Display * on LCD
                  SSD1306_UpdateScreen();
          }

          // If the key pressed is # and isArmed is true
          if (key == '#' && isArmed) {

                  // If the input is between 4-6 digits
                  if (strlen(hold) >= 4 && strlen(hold) <= 6) {

                          // If the input matches the password set
                          if (strcmp(hold, password) == 0) {
                                  SSD1306_GotoXY (0, 0);
                                  SSD1306_UpdateScreen();
                                  SSD1306_Puts ("CORRECT", &Font_11x18, 1); // Display correct on LCD
                                  SSD1306_UpdateScreen();
                                  osDelay(2000); // Display correct for 2 seconds
                                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6, GPIO_PIN_RESET); // Turn off red LED
                                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_SET); // Turn on green LED
                                  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9, GPIO_PIN_RESET); // Turn off buzzer
                                  SSD1306_Clear();
                                  SSD1306_UpdateScreen();
                                  SSD1306_GotoXY (0, 0);
                                  SSD1306_UpdateScreen();
                                  SSD1306_Puts ("NOT ARMED", &Font_11x18, 1); // Display NOT ARMED on LCD
                                  SSD1306_UpdateScreen();
                                  SSD1306_GotoXY (0, 20);
                                  SSD1306_UpdateScreen();
                                  SSD1306_Puts ("Press * to", &Font_11x18, 1); // Display press * to arm on LCD
                                  SSD1306_UpdateScreen();
                                  SSD1306_GotoXY (0, 40);
                                  SSD1306_UpdateScreen();
                                  SSD1306_Puts ("arm", &Font_11x18, 1);
                                  SSD1306_UpdateScreen();
                                  isArmed = 0; // Set isArmed to false
                                  strcpy(hold, ""); // Reset input string
                                  strcpy(hidden, ""); // Reset hidden string
                          } else { // If the input does not match the password set
```

Code A.1.5: *Code snippets of EnterPasswordTask*

```
                        } else { // If the input does not match the password set
                                SSD1306_GotoXY (0, 0);
                                SSD1306_UpdateScreen();
                                SSD1306_Puts ("WRONG", &Font_11x18, 1); // Display wrong on LCD
                                SSD1306_UpdateScreen();
                                osDelay(2000); // Display error message for 2 seconds
                                SSD1306_Clear();
                                SSD1306_UpdateScreen();
                                SSD1306_GotoXY (0, 0);
                                SSD1306_UpdateScreen();
                                SSD1306_Puts ("ARMED", &Font_11x18, 1); // Display ARMED on LCD
                                SSD1306_UpdateScreen();
                                SSD1306_GotoXY (0, 20);
                                SSD1306_UpdateScreen();
                                SSD1306_Puts ("Enter code:", &Font_11x18, 1);
                                SSD1306_UpdateScreen();
                                strcpy(hold, ""); // Reset input string
                                strcpy(hidden, ""); // Reset hidden string
                        }
                } else { // If the input is not between 4-6 digits
                        SSD1306_Clear();
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 0);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("WRONG Entry", &Font_11x18, 1); // Display error message on LCD
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 20);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("is not", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 40);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("4-6 digits", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                        strcpy(hold, ""); // Reset input string
                        strcpy(hidden, ""); // Reset hidden string
                        osDelay(2000); // Display error message for 2 seconds
                        SSD1306_Clear();
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0,0);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("ARMED", &Font_11x18, 1); // Display ARMED on LCD
                        SSD1306_UpdateScreen();
                        SSD1306_GotoXY (0, 20);
                        SSD1306_UpdateScreen();
                        SSD1306_Puts ("Enter code:", &Font_11x18, 1);
                        SSD1306_UpdateScreen();
                }

        }
```

Code A.1.6: *Code snippets of EnterPasswordTask*

```
        // If key pressed is * and isArmed is false
        if (key == '*' && !isArmed) {
                SSD1306_Clear();
                SSD1306_UpdateScreen();
                SSD1306_GotoXY (0, 0);
                SSD1306_UpdateScreen();
                SSD1306_Puts ("ARMED", &Font_11x18, 1); // Display ARMED on LCD
                SSD1306_UpdateScreen();
                SSD1306_GotoXY (0, 20);
                SSD1306_UpdateScreen();
                SSD1306_Puts ("Enter code:", &Font_11x18, 1);
                SSD1306_UpdateScreen();
                strcpy(hold, ""); // Reset input string
                strcpy(hidden, ""); // Reset hidden string
                isArmed = 1; // Set isArmed to true
                isCountdown = 1; // Set isCountdown to true
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6, GPIO_PIN_SET); // Turn on red LED
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_RESET); // Turn off green LED
        }

        HAL_Delay (500);
  }
  /* USER CODE END EnterPasswordTask */
}
```

Code A.1.7: *Code snippets of EnterPasswordTask*

```
void DetectMotionTask(void *argument) // DETECT MOTION TASK
{
  /* USER CODE BEGIN DetectMotionTask */

  /* Infinite loop */
  for(;;)
  {
          // If isArmed is true and isCountdown is true
          if (isArmed && isCountdown) {
                  int counter = 59; // Initialize counter

                  // Loop until counter is 0
                  while (counter >= 0) {
                          char countdown[3]; // Create countdown string
                          sprintf(countdown, "%d", counter); // Format
                          SSD1306_GotoXY (60, 0);
                          SSD1306_UpdateScreen();
                          SSD1306_Puts ("  ", &Font_11x18, 1); // Clear area where the countdown is located
                          SSD1306_UpdateScreen();
                          SSD1306_GotoXY (60, 0);
                          SSD1306_UpdateScreen();
                          SSD1306_Puts (countdown, &Font_11x18, 1); // Display current counter value
                          SSD1306_UpdateScreen();
                          osDelay(1000); // Delay for 1 second

                          // If counter is 0, clear area where the countdown is located
                          if (counter == 0) {
                                SSD1306_GotoXY (60, 0);
                                SSD1306_UpdateScreen();
                                SSD1306_Puts ("  ", &Font_11x18, 1);
                                SSD1306_UpdateScreen();
                          }
                          counter--; // Decrement counter

                  }

                  isCountdown = 0; // Set isCountdown to false
                  isCountdown2 = 1; // Set isCountdown2 to true

            // If isArmed is true and isCountdown2 is true
          } else if (isArmed && isCountdown2) {

                  // If motion is detected
                  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)) {
```

Code A.1.8: *Code snippets of DetectMotionTask*

```
        // If isArmed is true and isCountdown2 is true
    } else if (isArmed && isCountdown2) {

            // If motion is detected
            if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)) {

                    int counter2 = 59; // Initialize counter

                    // Loop until counter is 0
                    while (counter2 >= 0) {
                            char countdown2[3]; // Create countdown string
                            sprintf(countdown2, "%d", counter2); // Format
                            SSD1306_GotoXY (60, 0);
                            SSD1306_UpdateScreen();
                            SSD1306_Puts ("  ", &Font_11x18, 1); // Clear area where the countdown is located
                            SSD1306_UpdateScreen();
                            SSD1306_GotoXY (60, 0);
                            SSD1306_UpdateScreen();
                            SSD1306_Puts (countdown2, &Font_11x18, 1); // Display current counter value
                            SSD1306_UpdateScreen();
                            osDelay(1000); // Delay for 1 second

                            // If counter is 0, clear area where the countdown is located
                            if (counter2 == 0) {
                                    SSD1306_GotoXY (60, 0);
                                    SSD1306_UpdateScreen();
                                    SSD1306_Puts ("  ", &Font_11x18, 1);
                                    SSD1306_UpdateScreen();
                            }
                            counter2--; // Decrement counter

                    }

                    // If isArmed is true and motion is detected
                    if (isArmed && HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)) {
                            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9, GPIO_PIN_SET); // Turn on buzzer
                            isCountdown2 = 0; // Set isCountdown2 to false

                    }
            }
    }


}
```

Code A.1.9: *Code snippets of DetectMotionTask*

Link to OneDrive with Code:
https://uottawa-my.sharepoint.com/personal/jlach107_uottawa_ca/_layouts/15/guestaccess.aspx?share=EvOjT51mweJLhEH13eLm9OwBOM4pM59jcEBCnQXL2tjTdQ&e=2a0udH