Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG4166/CSI4141/SEG4155 Real-Time System Design Winter 2024

Lab 3: Project's Alarm System

Submission Date: March 18th, 2024

Group Number: 5

Lab section: A01 (Tuesday PM)

Team Member List:

- Moise Baleke – 300207962
- Zinah Al-Saadi – 8867078
- Decaho Gbegbe – 300094197

Table of Contents

Table of Figures

Introduction

Lab 3 represents the first module of the alarm system project. Students will proceed with using the nucleo-F446RE development board as the main tool for the project integration. Lab 3 requires students to interface a 4X4 keypad from the nucleo-F446RE board which will act as the main component in which the user is able to enter a four- or six-digit alphanumeric code to firstly configure the system. Additionally, an LCD display, more specifically the SSD1306 OLED LCD display module, will be connected to the nucleo-F44RE board with the main function of letting the user know whether the system is alarmed successfully or whether it is not armed. LED lights will also be integrated in this lab with red representing an armed system and green representing an unarmed system. Lab 3 allows students to get a deeper understanding of how to configure a matrix keypad correctly.

Objectives

The main objectives of this lab are to:

1. Implement an alarm system using nucleo-F446RE development board.
2. Configure a 4x4 Matrix Membrane Keypad using rows and columns to give different patterns alphanumerically.
3. Connecting the SSD1306 OLED LCD display to properly display whether the system has been armed successfully or not.
4. Visually representing whether the system is armed or not using LED lights.

Problem Analysis

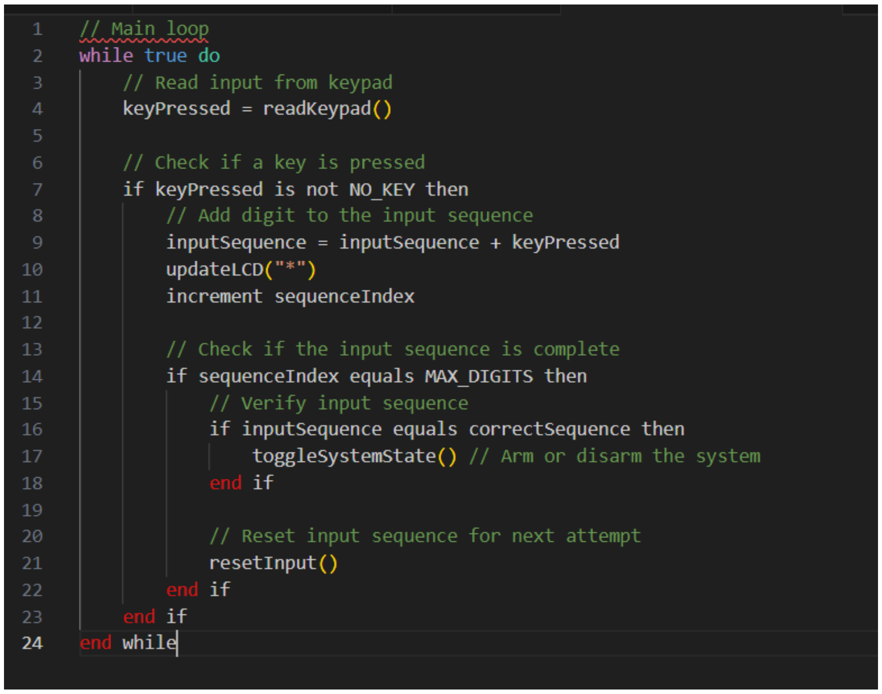
The first step in this lab is to recognize that the system ultimately has three different states that it could be in. The first state is the configuration state. This is the state in which the user initiates the alarm system by entering a specific four- or six-digit code. After state one is executed successfully, the system cannot enter one of the two remaining states; ARMED or NOT ARMED. State two being armed can only be entered if the user enters the security code correctly that was inputted in the configuration state. If the user enters the wrong code, the system will enter state three which is NOT ARMED. It is important to note that when developing the algorithm, it should also allow the user to re-enter state one, the configuration state, in case the user decided to change the security code at any point and should not be limited to just one code only. Students are required to use tasks and not a super loop when creating the algorithm for the alarm system.

Solution Design

The solution implemented for the system that reads input from a keypad, checks the input against a predefined code, updates an LCD display, and sets a system state based on the input follows these steps:

1. **Initialization**:
    - The system initializes all necessary hardware components, including the microcontroller unit (MCU), General-Purpose Input/Output (GPIO) pins, Inter-Integrated Circuit (I2C) for the LCD, and UART for debugging if needed.
    - The GPIO pins are configured for the keypad rows and columns to be able to read the inputs.
    - The I2C communication is set up for interfacing with the OLED LCD display to send display commands and data.
2. **Input Scanning (Keypad Scanning)**:
    - The system scans the keypad for input by setting each row to LOW in turn and checking the columns for a LOW signal, indicating a button press.
    - Upon detecting a button press, the system maps the row and column to the corresponding key value and appends this value to the user's input code.
3. **Displaying Input (LCD Updating)**:
    - As the user enters the code, the LCD displays an asterisk (*) to mask the input code for security purposes.
    - This continues until the user finishes the input sequence, either by pressing a specific "Enter" key or by entering the required number of digits (6 in this case).
4. **Code Verification**:
    - Once the full input code is received, the system checks if it matches the predefined correct code.
    - If the input code matches, the system toggles its state to "armed" or "disarmed".
5. **System State Update (LCD and LEDs)**:
    - The system state is reflected on the LCD, displaying "ARMED" or "NOT ARMED".
    - Additionally, LEDs can be used to indicate the system's state, with a red LED for armed and a green LED for disarmed.
6. **Reset User Input**:
    - After processing the input, the system clears the user's input code in preparation for the next input sequence.

The following pseudo code shows how all these steps could be implemented together to solve the problem.



```
// Main loop
while true do
    // Read input from keypad
    keyPressed = readKeypad()

    // Check if a key is pressed
    if keyPressed is not NO_KEY then
        // Add digit to the input sequence
        inputSequence = inputSequence + keyPressed
        updateLCD("*")
        increment sequenceIndex

        // Check if the input sequence is complete
        if sequenceIndex equals MAX_DIGITS then
            // Verify input sequence
            if inputSequence equals correctSequence then
                toggleSystemState() // Arm or disarm the system
            end if

            // Reset input sequence for next attempt
            resetInput()
        end if
    end if
end while
```

Component Usage

The alarm system project incorporates a set of well-chosen components that interwork to provide a secure and user-friendly interface.

- **Nucleo-F446RE Development Board**: This board is built around the STM32 microcontroller which orchestrates the entire system. The microcontroller's robust processing capabilities are essential for monitoring inputs from the keypad and controlling the output to the LCD and LEDs. Its multitude of GPIO pins are used to interface with the keypad and LEDs, while the I2C channels facilitate communication with the LCD display.
- **4x4 Keypad**: The keypad is a critical input device in our system, featuring 16 keys arranged into 4 rows and 4 columns. It is used for setting and entering a secure pin, which is a crucial aspect of the alarm system, allowing users to activate or deactivate the alarm state. Its design ensures a tactile interface for pin entry, which is more secure and reliable than touch interfaces in the context of security systems.
- **LCD Display**: The display provides immediate visual feedback to the user, echoing pin entry with asterisks to protect the pin's secrecy. It also displays the system's status, providing a clear indication of whether the alarm is armed or not. The choice of an LCD is deliberate, given its low power consumption and high visibility, which is crucial in security systems where clarity and power efficiency are key.
- **LED Lights**: As supplemental indicators, the LED lights offer a quick and intuitive status update. A red LED signifies an armed system, warning the user and potential intruders of

the active security measures. Conversely, a green LED denotes the system's inactive state, indicating safety for entry. These color codes align with universal standards for stop (red) and go (green), ensuring the system's status is understood at a glance.

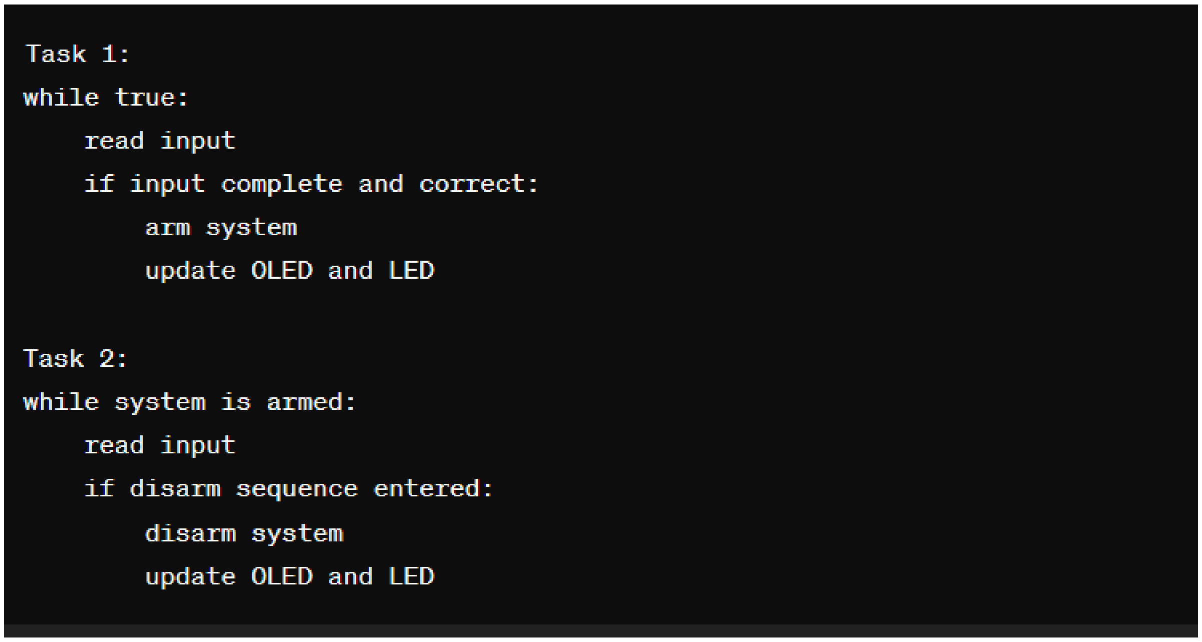
Each component in the system is selected not only for its individual functionality but also for how it complements the system as a whole. The Nucleo-F446RE's capacity for real-time processing and communication ensures the system's responsiveness. The keypad's tangible feedback and familiar layout cater to user comfort and security. The LCD display's clear visuals reinforce user interaction, and the LED indicators provide a persistent, low-energy status signal. Together, these components form a cohesive and efficient alarm system.

Algorithmic approach

In the implemented solution, Task 1 manages user interaction, where it listens for input from the keypad and processes it. Once the user completes the input sequence, if the entered code matches the preset password, the task transitions the system to an "armed" state. This state is visually relayed through an OLED display, which shows "ARMED" and a physical LED that is lit to indicate activation.

Task 2 is a monitoring loop that takes over when the system is armed. It waits for a deactivation sequence and, upon validation of the correct input, disarms the system. The OLED display and LED states are updated to reflect the "disarmed" status.

Both tasks are critical for the system's operation. Task 1 is essential for setting the system's operational state based on user input. Task 2 maintains the system's integrity by ensuring it stays in the armed state until the correct disarm sequence is entered.



```
Task 1:
while true:
    read input
    if input complete and correct:
        arm system
        update OLED and LED

Task 2:
while system is armed:
    read input
    if disarm sequence entered:
        disarm system
        update OLED and LED
```

Implementation Evaluation

The alarm system code, detailed in the Appendix (Code A.1), was designed with the aim of ensuring ease of understanding and the potential for future code changes. The code is organized into separate functions that handle specific tasks, such as setting up the system, reading from the keypad, displaying on the LCD, and changing the system's armed status. This structure enhances the code's clarity and allows for easier updates or troubleshooting.

Although we couldn't test the whole code on the actual Nucleo-F446RE board, it's been checked for syntax errors and the logic has been thoroughly reviewed. Based on this, we expect that when the code is run on the board, the LEDs would light up correctly to show the system's status, and the LCD would properly display the masked pin entries.

In theory, the code should work as intended because it relies on established methods for interacting with the board's components. However, it's important to note that actual performance can only be confirmed through physical testing on the board.

Results and Validation

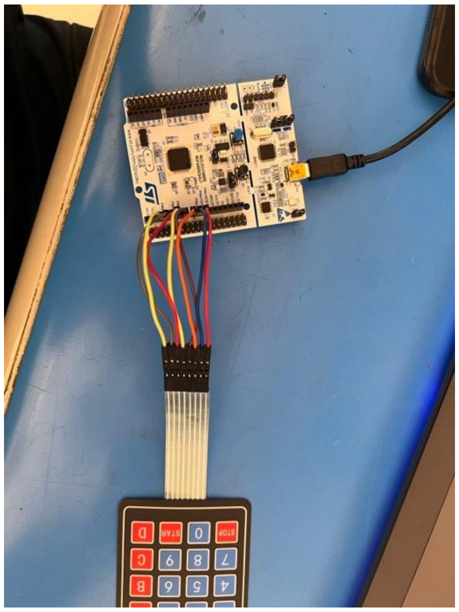


Figure 1: keypad connectivity to nucleo-F446RE

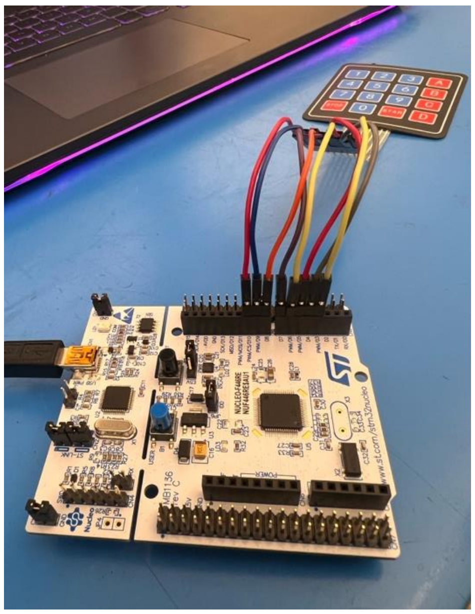


Figure 2: keypad connectivity to nucleo-F446RE with pins

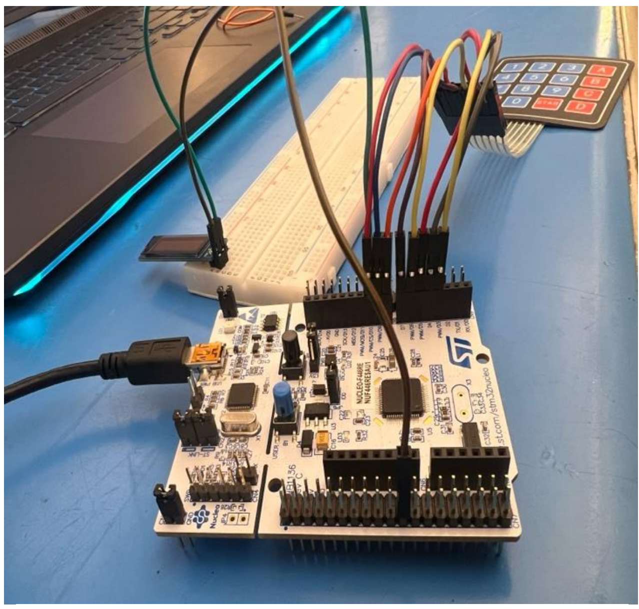


Figure 3: keypad connectivity to nucleo-F446RE with pins and LCD display connectivity

Addressing challenges

In this project, our main challenge was managing development time effectively. The integration of the keypad, LCD, and LEDs into the alarm system required meticulous programming and testing, which proved time-consuming. To overcome this, we prioritized modular testing, ensuring each part worked flawlessly on its own. The key takeaway for future work is to start the development process earlier, allowing for comprehensive testing and integration, which is vital for complex systems involving multiple hardware components. This approach will help mitigate time constraints and enhance project outcomes.

Conclusion

In conclusion, the first module of the alarm system project allowed us to familiarize ourselves with the concept of a 4x4 Matrix Membrane keypad. We were able to understand the functionality of the rows and columns of the keypad and how to integrate it into our system correctly. The use of tasks enabled us to solve our problem proposed by this lab which is representing the different states that the system can enter which are; configuration state, armed state and not armed state. By connecting the LCD display to the nucleo-F446RE board, the alarm system had a visual representation of whether the system was armed successfully or not upon any code entry and whether it matched the code provided in the configuration state.

Workload Distribution

The table below shows the work undertaken by each team member

Student name	Student number	Working percent
Moise Baleke	300207962	100%
Zinah Al-Saadi	8867078	100%
Decaho Gbegbe	300094197	100%

Appendix

A-1:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
*
```

```
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE
file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

*****************************************************************************
*
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ----------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */
#include "Keypad4X4.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "fonts.h"
#include "ssd1306.h"
#include <stdarg.h>


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
-*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
-*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;
```

```c
/* USER CODE BEGIN PV */
extern char key;
char key_2;
char hold[6];
bool armed = false;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
-*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
-*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */

char* replaceCharsWithAsterisks(const char* input, int size) {
    // Allocate memory for the new array (+1 for the null terminator)
    char* result = (char*)malloc((size + 1) * sizeof(char));
    // Replace each char with an asterisk
    for (int i = 0; i < size; i++) {
        result[i] = '*';
    }

    // Add null terminator at the end
    result[size] = '\0';

    return result;
}
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration-------------------------------------------------------
-*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* System Initialization */
  HAL_Init(); // Initialize the HAL Library
  SystemClock_Config(); // Configure the system clock
  MX_GPIO_Init(); // Initialize all configured GPIO
  MX_USART2_UART_Init(); // Initialize UART2 for serial communication
  MX_I2C1_Init(); // Initialize I2C1 for communication with peripherals like
the OLED screen

  /* Peripheral Initialization */
  SSD1306_Init(); // Initialize the OLED screen
  SSD1306_GotoXY (0,0); // Set cursor at the top-left corner of the OLED
screen
  SSD1306_Puts ("Enter Code:", &Font_11x18, 1); // Prompt user to enter the
security code
  SSD1306_GotoXY (0, 30); // Set cursor position to next line
  SSD1306_UpdateScreen(); // Refresh the OLED screen to display the message
  HAL_Delay (500); // Delay for a short period to allow the display to
stabilize

  /* Loop indefinitely */
  while (1)
  {
    /* Key-press Handling and System Armed State Indication */
    int numKeyPresses = 0; // Counter for the number of keypresses
    char securityCode[7] = {0}; // Buffer to hold the security code,
initialized to empty
    bool isSystemArmed = false; // Flag to indicate if the system is armed

    // Loop to collect key presses and arm/disarm the system
    for(int i = 0; i < 6; i++){
        char keyPress = Get_Key(); // Get the key pressed from the keypad
        numKeyPresses++; // Increment the key press counter

        // Check if '#' is pressed and the security code length is 4 or 6
digits
        if(keyPress == '#' && (strlen(securityCode) == 4 ||
strlen(securityCode) == 6)){
            // Code to arm the system and indicate the state on OLED
            SSD1306_GotoXY (0, 30);
            SSD1306_Puts ("ARMED", &Font_11x18, 1);
            SSD1306_UpdateScreen();
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET); // Turn on
the ARMED indicator LED
            isSystemArmed = true;
            break; // Exit loop if system is armed
        }
        else{
            // Append the pressed key to the security code
            strncat(securityCode, &keyPress, 1);

            // Mask the security code with asterisks on OLED
            SSD1306_GotoXY (0, 30);
            char* asterisksDisplay = replaceCharsWithAsterisks(securityCode,
numKeyPresses);
            SSD1306_Puts (asterisksDisplay, &Font_11x18, 1);
            SSD1306_UpdateScreen();
            free(asterisksDisplay); // Free the dynamically allocated memory
for asterisksDisplay
        }
    }
    // Additional logic to handle system disarm or further actions can be
added here
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 400000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, KC0_Pin|GPIO_PIN_13|GPIO_PIN_14|KC3_Pin
                          |KC1_Pin|KC2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : KC0_Pin PB13 PB14 KC3_Pin
                           KC1_Pin KC2_Pin */
  GPIO_InitStruct.Pin = KC0_Pin|GPIO_PIN_13|GPIO_PIN_14|KC3_Pin
                          |KC1_Pin|KC2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : KR1_Pin */
  GPIO_InitStruct.Pin = KR1_Pin;
```

```c
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(KR1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : KR3_Pin KR2_Pin */
  GPIO_InitStruct.Pin = KR3_Pin|KR2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : KR0_Pin */
  GPIO_InitStruct.Pin = KR0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(KR0_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to
increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

A-2: Demonstration video
https://uottawa-my.sharepoint.com/personal/mbale067_uottawa_ca/Documents/IMG_6260.MOV?csf=1&web=1&e=YuTjHp&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D