Alisa Kliushina & Charles Marsh
Peter Han
March 19, 2025
CPE 316-01

CPE-316 Final Project Report

Project Description:

The project itself is a Simon Says game, inspired by the game on humanbenchmark.com [1]. There are 12 LED lights assembled in a 3x4 matrix order to resemble the layout of the keypad. The LEDs turn on and off to show a pattern with an increasing number of blinks and increased speed of blinks with every next level. The goal of the game is to repeat the pattern using the corresponding keys on the keypad. Every time the pattern is repeated correctly, the game moves on to a more challenging level, however if the user fails to repeat the pattern properly, the game will start back at level 1. The high score is updated every time the previous high score gets beat and the result becomes saved in the volatile memory and can be viewed on the LCD display. The LCD also plays a key part in guiding the user through the game and displays valuable information including the level number and the high score.

Project Implementation:

A: Bill of Material:

Item	Quantity	Price ($)	Cost to us
Nucleo STM32 Microcontroller	1	$11.04	$0 – Rented from the CPE departament
4x3 Keypad	1	$3.95	Included in the $10 lab kit
MCP23017 I2C I/O Expander	1	$1.69	Included in the $10 lab kit
LED	12	$0.15 (x12)	Included in the $10 lab kit
150 ohm resistor	12(for LED matrix) + …	$0.10 (x12)	Included in the $10 lab kit + own
1k ohm resistor		$0.10	Included in the $10 lab kit
LCD display	1	$8.95	Included in the $10 lab kit

Jumper Cables		$6	$6 box of cables from amazon
USB-A to USB-mini	1	$6.59	$0 – Rented from the CPE departament

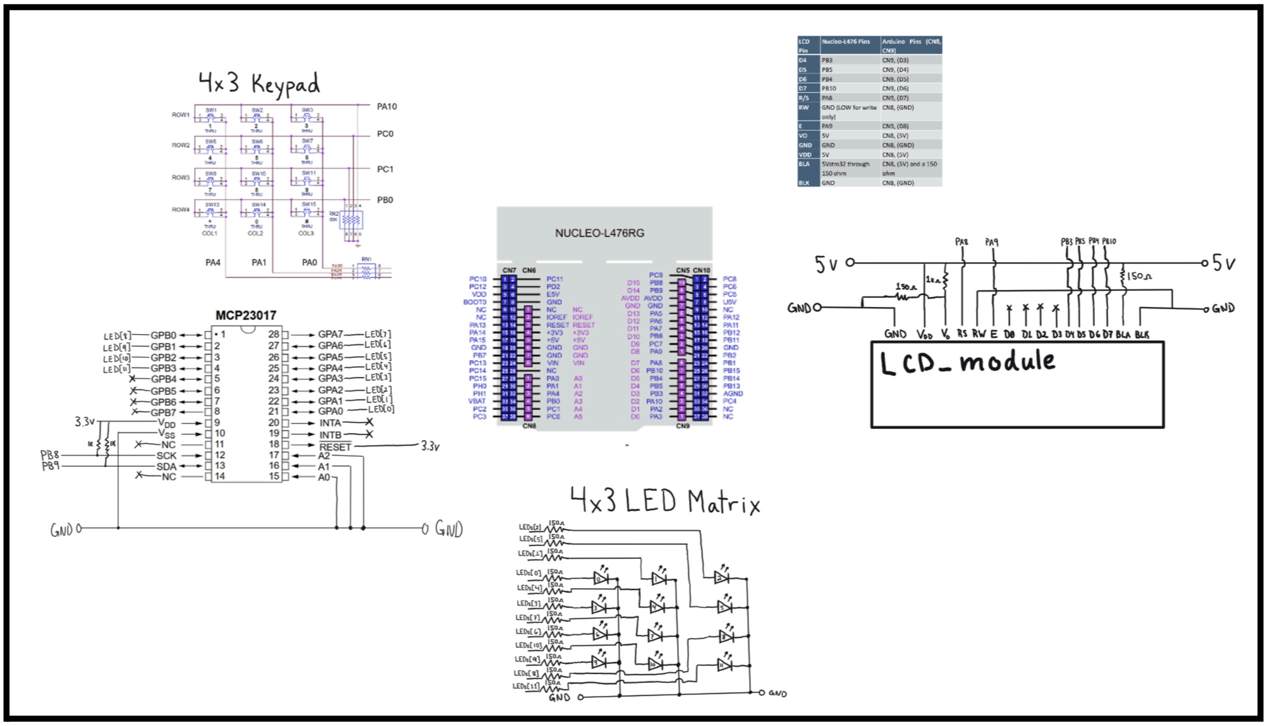

Figure 1: Hardware Schematic

B: Software Design Pseudocode / Flowchart / Diagram:

Below is an FSM showcasing the game logic loop. The theory of the game is fairly straightforward, if the user gets the right answer, they go onto the next level, if not, the game ends.

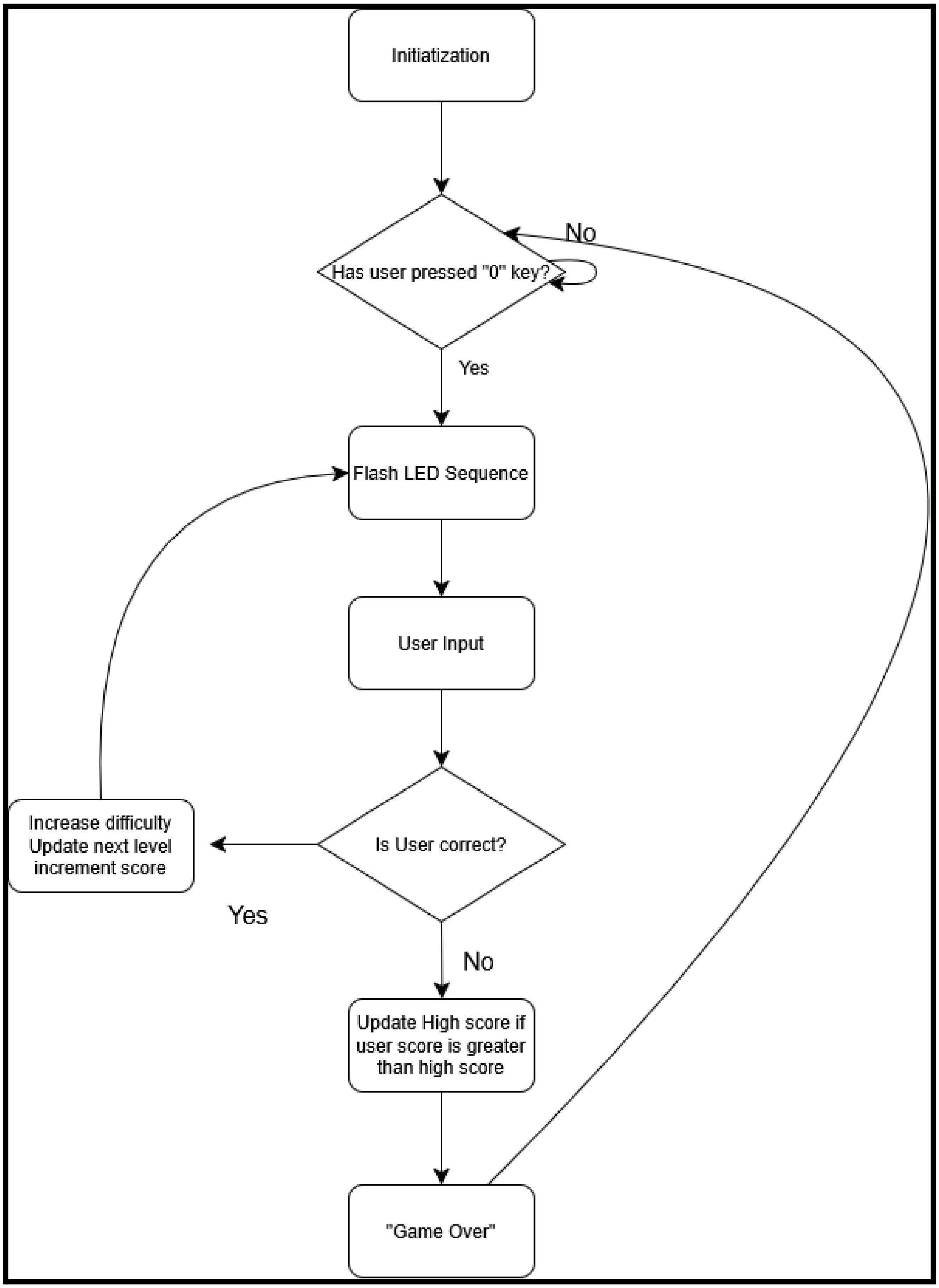

Figure 2: Flowchart of full software structure created with diagrams.net [2]

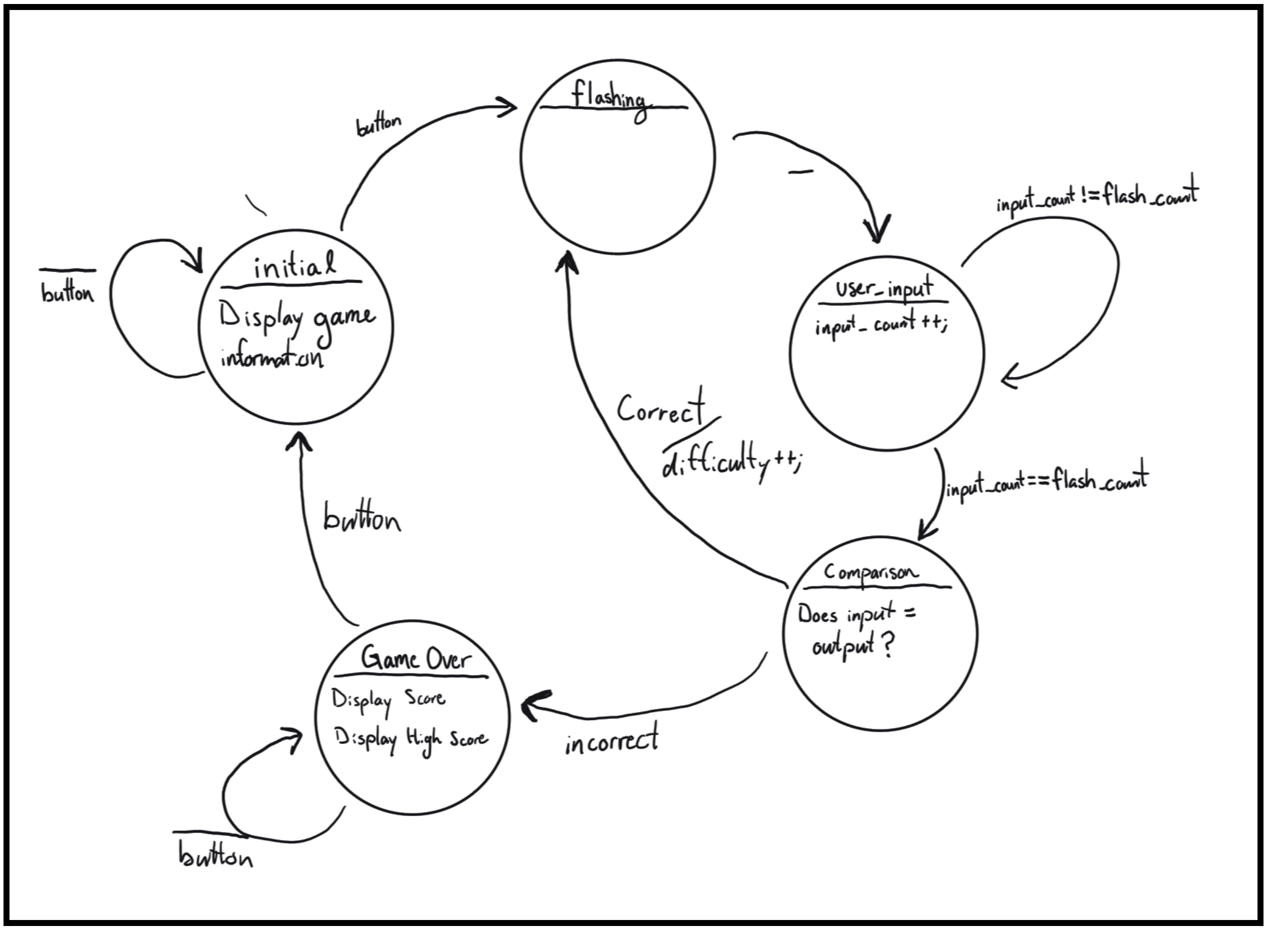

Figure 3: FSM diagram of Game Logic

Testing and Demonstration:

Hardware:

We began our testing by wiring all of the hardware components together according to the diagram. We then transitioned into connecting all of them to software through the ioc file. We have then written sample code for each component separately to ensure that basic functionality for them could compile and run. The sample code was then broken down to a few simple steps. First, we tested that all 12 LEDs can turn on and off one-by-one without a special trigger through the I2C. Secondly, we tested that LCD displayed the basic phrases before moving on to complex tasks and timed displays. Thirdly, we ensured that all of the keys on the keypad could be pressed and lit up a corresponding LED when pressed. After initial testing of the hardware, we were able to transition into software testing and later advanced hardware testing with the game's full functionality.

Software:

Initially, we tested the software in small blocks that connected to different parts of hardware. We firstly ensured that our code prints phrases to the LCD correctly and in a timely manner. After that, we tested that our implementation of I2C could easily communicate with LEDs. It was the most challenging part of the code for us as during testing we had to keep track of which LEDs were connected to the GPIOA and GPIOB ports. Following that, we have revised and updated our code for keypad usage and tested it with hardware, ensuring that it is functional. Debugging the keypad code entailed double checking that all of the cases for the buttons were accurate and compliant with the rest of the code. There were a lot of variables and moving parts to keep track of during the keypad debugging, including the Keypadscan() function. We were able to complete testing easier by moving redundant parts of keypad code into the helper function and only test the new to the project functionality. After initial testing, we were able to revise our code and make it more complex. Every time we added a new piece of complete compiling code, we made sure to test it on the working hardware to not move too far ahead with non-functional code, risking having to backtrack our steps. Finally, we tested our complete code by allowing our friends to play the game without our guidance, ensuring that the code is self explanatory and easy to follow.

Video Demonstration:

A video demonstration of project can be seen at: https://youtu.be/FXmU54PPUjQ [3]

Collaboration and Teamwork:

Through our collaboration we were able to complete the project in a timely manner with outstanding results. The work was split evenly between the team members and completed separately on our own time, as well as during team meetings that took place in the evenings on school days. A lot of brainstorming for implementation was done during our evening meetings and we were able to successfully combine our ideas and write the code together. On top of that our tasks included:

Charles:

- Hardware schematic creation
- FSM diagram creation
- Pseudo code creation
- Handling the LCD logic
- Implementing game levels
- Implementation of game logic
- keypad input for different levels
- Debugging the hardware

- Debugging the software

Alisa:

- Wiring together the hardware
- Setting up the ioc file
- Handling the keypad setup logic
- Implementing blinking patterns for key press
- Setting up the level array
- Debugging the hardware
- Debugging the software

Troubleshooting and Debugging:

The development process involved several hardware and software challenges. Early in the project, we discovered that the I2C module was missing two critical pull-up resistors, which caused intermittent communication failures between the microcontroller and the MCP23017 LED driver. This issue manifested as erratic LED behavior, including flickering or complete unresponsiveness. After hours of debugging the software (initially suspecting protocol timing issues), we finally re-examined the hardware and identified the missing resistors. Adding 4.7kΩ pull-up resistors to the SDA and SCL lines stabilized the I2C bus.

Another significant challenge arose when attempting to flash multiple LEDs in rapid succession. Initially, the LEDs appeared dim or barely visible, even though the code seemed logically sound. We traced this problem to the I2C write speed: the microcontroller was sending commands faster than the MCP23017 could process them, resulting in overlapping instructions. By inserting short delays between I2C write operations and optimizing the code to reduce redundant calls (e.g., turning off all LEDs in a single command instead of iterating through each one), we achieved brighter, crisper LED flashes. Tools like a multimeter and a logic analyzer were indispensable for diagnosing these issues—the multimeter confirmed voltage drops caused by the missing resistors, while the logic analyzer revealed timing conflicts in the I2C protocol.

Lessons Learned:

This project taught us the importance of modular development and systematic problem-solving. Initially, we attempted to code the entire game in one go, resulting in convoluted software with lots of bugs. After struggling with these bugs, we restarted the project and just focused on one aspect at a time. An example of this is when we first focused solely on getting the LEDs to respond reliably via the I2C expander before integrating the keypad or LCD. This modular workflow allowed us to validate each component independently, making it easier to pinpoint issues later on while developing the actual game.

We also learned that hardware failures can hide themselves as software bugs. When setting up the I2C module in software, we couldn't get any of the LEDs to flash and thought it was a software issue with the configuration or something of that nature. Turns out that we were missing two resistors, connecting the I2C connection pins to the 3.3V line, which is why no data was being sent. To mitigate this, we found that maintaining detailed documentation (e.g., recording pin assignments, I2C addresses, and function protocols) through hardware diagrams probably saved lots of time when coming back to work on the project after not touching it for a few days. These lessons will guide our approach to future embedded systems projects.

Future Possibility:

The project holds potential for expansion with additional features and refinements. To make more gameplay variety, new minigames such as a reaction time test or visual memory challenge could be added, making the project a multi-game device. We could improve the packaging of the project, as right now it is all just open breadboards with wires going everywhere, and is a little bit finicky. We could create a 3D-printed casing, which would not only protect the components but also give the device a polished, commercial-grade appearance. Replacing the current keypad with tactile mechanical switches would improve durability and user feedback, also as the keypad can be unreliable. Another addition could be adding a speaker that could introduce sound effects or musical cues to indicate when the user got the right answer or wrong answer, and add music that gets more intense the higher level you get to.

Another thing that would've been added given a larger budget is a larger LED matrix. This would enable more complex patterns and more difficulty, and could also create more possibilities for different things we could do with it. We could use the LEDs and buttons with a speaker as a drum machine pad, that could plug into your laptop where you can create beats, or just other games as well. These upgrades in hardware would elevate the project from a simple memory game to a versatile platform for entertainment, education, and gaming, and could be a fun and affordable way to pass the time.

Summary and Conclusion:

After completion of this project, we have found a new found appreciation for embedded systems in everyday life. The amount of effort to make an embedded systems project work well and seamlessly makes us appreciate even the code that makes a microwave work. When going to the arcade as a child and playing the games, I never thought too deeply about how the game was made, on a hardware or software level. But now, the next time I step into an arcade, I will be thinking about the amount of design that went into the hardware, and the amount of programming it took to make the game function the way it does.

All in all, the project implementation was a great success for our team. We were able to apply the things we have learned in class, such as the use of I2C, keypad and LCD, and learn new things along the way. It was interesting to take all of the independent topics and compile them into a large item that is applicable in the real world. While we have faced some challenges along the way, we were able to overcome them in a timely manner through teamwork and collaboration.

Reference & Bibliography:

[1] “Human Benchmark,” *humanbenchmark.com*. https://humanbenchmark.com/tests/sequence

[2] draw.io, “Flowchart Maker & Online Diagram Software,” *app.diagrams.net*. https://app.diagrams.net/

[3] Video Demonstration, https://youtu.be/FXmU54PPUj

main.c:

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

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

*********************************************************************************
 * @attention
 *
 * Copyright (c) 2025 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.
 *

*********************************************************************************
 */
#include "main.h"
#include "LCD.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define Number_of_Keys 12
#define Number_of_Cols 3
#define PA0 0x0001
#define PA1 0x0002
#define PA4 0x0010
#define PB0 0x0001
#define PC1 0x0002
#define PC0 0x0001
```

```
#define PA10 0x0400
#define KeyDetect 0x0001
#define KeyLow2High 0x0002
typedef struct
{
    unsigned short sKeyRead;
    unsigned short sKeyReadTempPos;
    unsigned short sKeySend;
    unsigned short sKeyCol;
    char KeyLetter;
    unsigned short sKeyCommand;
} Key_Contorl_struct_t;
typedef enum KeyName     //setting up the 12
different commands we can receive from keys
{
    ONE_command,
    FOUR_command,
    SEVEN_command,
    STAR_command,
    TWO_command,
    FIVE_command,
    EIGHT_command,
    ZERO_command,
    THREE_command,
    SIX_command,
    NINE_command,
    POUND_command
} KeyName;
/******** Structure *******/
Key_Contorl_struct_t sKeyControl[Number_of_Keys]
//establishing which key is which based on the input
and output pins
={
    {PA10,0x8,PA4,0,'1',ONE_command}, // PA10 (read),
PA4 (send)
    {PC0,0x4,PA4,0,'4',FOUR_command}, // PC0 (read),
PA4 (send)
    {PC1,0x2,PA4,0,'7',SEVEN_command}, // PC1 (read),
PA4 (send)
    {PB0,0x1,PA4,0,'*',STAR_command}, // PB0 (read),
```

```
PA4 (send)
    {PA10,0x8,PA1,1,'2',TWO_command}, // PA10 (read),
PA1 (send)
    {PC0,0x4,PA1,1,'5',FIVE_command}, // PC0 (read),
PA1 (send)
    {PC1,0x2,PA1,1,'8',EIGHT_command}, // PC1 (read),
PA1 (send)
    {PB0,0x1,PA1,1,'0',ZERO_command}, // PB0 (read),
PA1 (send)
    {PA10,0x8,PA0,2,'3',THREE_command}, // PA10
(read), PA0 (send)
    {PC0,0x4,PA0,2,'6',SIX_command}, // PC0 (read),
PA0 (send)
    {PC1,0x2,PA0,2,'9',NINE_command}, // PC1 (read),
PA0 (send)
    {PB0,0x1,PA0,2,'#',POUND_command} // PB0 (read),
PA0 (send)
};
unsigned short sKeyStatus;
unsigned short sKeyCurrentCol[Number_of_Cols];
unsigned short sKeyDebouncedCol[Number_of_Cols];
unsigned short sKeyIssued;
unsigned short sKeyPreviousCol[Number_of_Cols];
unsigned short sKeyLow2HighCol[Number_of_Cols];
void Keypadscan(void);
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim5;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM5_Init(void);
static void MX_I2C1_Init(void);
#define I2C_ADDRESS 0x20 // Default MCP23017 address
(A2-A0 grounded)
#define MCP_IODIRA 0x00  // I/O Direction A register
#define MCP_IODIRB 0x01  // I/O Direction B register
#define MCP_GPIOA 0x12   // GPIO Port A
```

```c
#define MCP_GPIOB 0x13   // GPIO Port B
void I2C_Write(uint8_t reg, uint8_t data) {
   uint8_t buffer[2] = {reg, data};
   HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDRESS << 1,
buffer, 2, 100);
}
void LED_Toggle(int num) {
   if (num < 1 || num > 12) {
       return; // Invalid input, do nothing
   }
   uint8_t led_index = num - 1; // Convert to 0-based
index (0-11)
   // Turn off all LEDs first
   I2C_Write(MCP_GPIOA, 0x00);
   I2C_Write(MCP_GPIOB, 0x00);
   // Activate the specified LED
   if (led_index < 8) {
       I2C_Write(MCP_GPIOA, 1 << led_index);
   } else {
       I2C_Write(MCP_GPIOB, 1 << (led_index - 8));
   }
}
int game_pattern[] = {4, 2, 1, 6, 9, 10, 12, 6, 1, 3,
8, 7, 3, 11, 5, 5, 8};
int main(void)
{
   int highscore = 0; //start high score at zero
   int user_input[17]; // Fixed size array to store
user inputs
   int flash_num = 3;
   int level_timer = 800;
   int timer = 0;
   bool game_start = false; //false not started, true
game starts
   int flash_pattern_circle[10] = {1, 2, 3, 6, 9, 12,
11, 10, 7, 4};
   HAL_Init();
   SystemClock_Config();
   MX_GPIO_Init();
   MX_SPI1_Init();
```

```c
    MX_TIM2_Init();
    MX_TIM5_Init();
    MX_I2C1_Init();
    I2C_Write(MCP_IODIRA, 0x00);   // Set all Port A as
outputs
    I2C_Write(MCP_IODIRB, 0x00);   // Set all Port B as
outputs
    HAL_Delay(100);
    LcdInit();
    LcdClear();
    LcdPutS("Welcome to Simon");
    LcdGoto(1, 0);
    LcdPutS("Says Minigame!");
    for (int i = 5; i > 0; i--) // flash the circle 5
times as "loading"
        {
            for (int j = 0; j < 10; j++) {
                LED_Toggle(flash_pattern_circle[j]);
                HAL_Delay(100);
            }
        }
    // Display high score
    LcdClear();
    LcdPutS("High score:");
    LcdGoto(1, 0);
    char highscore_str[16];
    sprintf(highscore_str, "%d", highscore);
    LcdPutS(highscore_str);
    for (int i = 5; i > 0; i--) // flash the circle 5
times as "loading"
    {
        for (int j = 0; j < 10; j++) {
            LED_Toggle(flash_pattern_circle[j]);
            HAL_Delay(100);
        }
    }
    I2C_Write(MCP_GPIOA, 0x00);
    I2C_Write(MCP_GPIOB, 0x00);
    while (1) {
        game_start = false;
```

```
        flash_num = 3;
        level_timer = 800;
        while (!game_start) //while waiting for 0 to
be pressed
        {
            timer++;
            if (timer == 10) {
                LcdClear();
                LcdPutS("Press 0 to");
                LcdGoto(1, 0);
                LcdPutS("Start!");
                timer = 0;
            }
            Keypadscan(); //scan keypad to see if 0
was pressed to start
            if ((sKeyStatus & KeyDetect) &&
(sKeyIssued != 0xFFFF))        // check for if a
valid key was detected
            {
                switch (sKeyIssued)     //cases for
different valid keys detected
                {
                    case ZERO_command:    //actions
for when key 0 is detected
                        game_start = true;
                        LcdClear();
                        I2C_Write(MCP_GPIOA, 0x00);
                        I2C_Write(MCP_GPIOB, 0x00);
                        LcdPutS("Ready...");
                        I2C_Write(MCP_GPIOA, (1 << 2)
| (1 << 5) | (1 << 8) | (1 << 11));
                        HAL_Delay(1000);
                        LcdClear();
                        I2C_Write(MCP_GPIOA, 0x00);
                        I2C_Write(MCP_GPIOB, 0x00);
                        LcdPutS("Set...");
                        I2C_Write(MCP_GPIOA, (1 << 1)
| (1 << 4) | (1 << 7) | (1 << 10));
                        HAL_Delay(1000);
                        LcdClear();
```

```
                        I2C_Write(MCP_GPIOA, 0x00);
                        I2C_Write(MCP_GPIOB, 0x00);
                        LcdPutS("Go!");
                        I2C_Write(MCP_GPIOA, (1 << 0)
| (1 << 3) | (1 << 6) | (1 << 9));
                        HAL_Delay(1000);
                        LcdClear();
                        I2C_Write(MCP_GPIOA, 0x00);
                        I2C_Write(MCP_GPIOB, 0x00);
                        LcdPutS("Starting Level 1");
                        HAL_Delay(1000);
                        break;
                    default:
                        break;
                }
            }
        }
        bool game_over = false;
        while (!game_over) {
            //FLASHING PATTERN
            LcdClear();
            LcdPutS("Watch closely!!");
            HAL_Delay(500);
            for (int i = 0; i < flash_num; i++) {
                LED_Toggle(game_pattern[i]);
                HAL_Delay(level_timer);
                HAL_Delay(100); // Short off time
between flashes
            }
            //turn off the last LED
            I2C_Write(MCP_GPIOA, 0x00);
              I2C_Write(MCP_GPIOB, 0x00);
            //LCD display instructions
            LcdClear();
            LcdPutS("Repeat the");
            LcdGoto(1, 0);
            LcdPutS("Pattern!");
            // Collect user inputs
            int user_num = 0;
            while (user_num < flash_num) {
```

```
                    Keypadscan();
                    if ((sKeyStatus & KeyDetect) &&
(sKeyIssued != 0xFFFF)) {
                        int pressed_led = -1;
                        switch(sKeyIssued) {
                            case ONE_command: pressed_led
= 1; break;
                            case TWO_command: pressed_led
= 2; break;
                            case THREE_command:
pressed_led = 3; break;
                            case FOUR_command: pressed_led
= 4; break;
                            case FIVE_command: pressed_led
= 5; break;
                            case SIX_command: pressed_led
= 6; break;
                            case SEVEN_command:
pressed_led = 7; break;
                            case EIGHT_command:
pressed_led = 8; break;
                            case NINE_command: pressed_led
= 9; break;
                            case ZERO_command: pressed_led
= 11; break;
                            case STAR_command: pressed_led
= 10; break;
                            case POUND_command:
pressed_led = 12; break;
                            default: pressed_led = -1;
break;
                        }
                        if (pressed_led != -1) {
                            user_input[user_num] =
pressed_led;
                            user_num++;
                            // Flash the corresponding LED
                            LED_Toggle(pressed_led);
                            HAL_Delay(200);
                            I2C_Write(MCP_GPIOA, 0x00);
```

```
                              I2C_Write(MCP_GPIOB, 0x00);
                              sKeyStatus &= ~(KeyDetect |
KeyLow2High); // Clear key status
                         }
                    }
               }
               // Compare user input with game_pattern
               bool correct = true;
               for (int i = 0; i < flash_num; i++) {
                    if (user_input[i] != game_pattern[i])
{
                         correct = false;
                         break;
                    }
               }
               if (correct) {
                    // Proceed to next level
                    flash_num++;
                    if (flash_num > 17) {
                         LcdClear();
                         LcdPutS("Good Job!");
                         HAL_Delay(2000);
                         flash_num = 3;
                         level_timer = 800;
                         game_over = true;
                    } else {
                         level_timer -= 15;
                         if (level_timer < 100) level_timer
= 100; // Prevent timer from getting too small
                         LcdClear();
                         char level_msg[16];
                         sprintf(level_msg, "Level %d",
flash_num - 2);
                         LcdPutS(level_msg);
                         HAL_Delay(1000);
                    }
               } else {
                    // Game Over
                    game_over = true;
                    LcdClear();
```

```
                LcdPutS("Game Over Loser");
                int score = flash_num - 3;
                LcdGoto(1, 0);
                char score_msg[16];
                sprintf(score_msg, "Score: %d",
score);
                LcdPutS(score_msg);
                HAL_Delay(2000);
                if (score > highscore) {
                    highscore = score;
                }
                LcdClear();
                LcdPutS("High Score:");
                LcdGoto(1, 0);
                sprintf(score_msg, "%d", highscore);
                LcdPutS(score_msg);
                HAL_Delay(3000);
            }
        }
    }
}
void Keypadscan()
{
   unsigned short sIndex;
   unsigned short Temp;
   // Clear all key records
   for (sIndex=0; sIndex<Number_of_Cols; sIndex++)
   {
       sKeyCurrentCol[sIndex] = 0x00;
   }
   // Read all 3 column
   for (sIndex=0; sIndex<Number_of_Keys; sIndex++)
   {
       GPIOA->ODR &=~(PA4 | PA1 | PA0);
       GPIOA->ODR |= sKeyControl[sIndex].sKeySend;
       HAL_Delay(1); // Adjusted delay for stability
       switch (sKeyControl[sIndex].sKeyCommand)
       {
           case ONE_command:
           case TWO_command:
```

```
            case THREE_command:
                if (GPIOA->IDR &
sKeyControl[sIndex].sKeyRead)

sKeyCurrentCol[sKeyControl[sIndex].sKeyCol]=
sKeyControl[sIndex].sKeyReadTempPos;
               break;
            case FOUR_command:
            case FIVE_command:
            case SIX_command:
            case SEVEN_command:
            case EIGHT_command:
            case NINE_command:
                if (GPIOC->IDR &
sKeyControl[sIndex].sKeyRead)

sKeyCurrentCol[sKeyControl[sIndex].sKeyCol] =
sKeyControl[sIndex].sKeyReadTempPos;
               break;
            case STAR_command:
            case ZERO_command:
            case POUND_command:
                if (GPIOB->IDR &
sKeyControl[sIndex].sKeyRead)

sKeyCurrentCol[sKeyControl[sIndex].sKeyCol] =
sKeyControl[sIndex].sKeyReadTempPos;
               break;
       }
   }
   // Check if a key is steadily read
   for (sIndex=0; sIndex<Number_of_Cols; sIndex++)
   {
       if ((sKeyCurrentCol[sIndex] ==
sKeyDebouncedCol[sIndex]) &&
           (sKeyCurrentCol[sIndex] != 0x0000))
           break;
   }
   if (sIndex <Number_of_Cols)
   {
```

```
        // Check for push on/ push off (Low To High)
        for (sIndex=0; sIndex<Number_of_Cols;
sIndex++)
        {
            Temp = sKeyCurrentCol[sIndex] ^
sKeyPreviousCol[sIndex];
            sKeyLow2HighCol[sIndex] =
(sKeyCurrentCol[sIndex] & Temp);
        }
        // Update Previous records
        for (sIndex=0; sIndex<Number_of_Cols;
sIndex++)
        {
            sKeyPreviousCol[sIndex] =
sKeyCurrentCol[sIndex];
        }
        // Find which key is JUST depressed (Low To
High)
        for (sIndex=0 ; sIndex<Number_of_Keys;
sIndex++)
        {
            if
(sKeyLow2HighCol[sKeyControl[sIndex].sKeyCol] &
                sKeyControl[sIndex].sKeyReadTempPos)
            {
                sKeyIssued =
sKeyControl[sIndex].sKeyCommand;
                sKeyStatus |= (KeyDetect |
KeyLow2High);
                break;
            }
            else
                sKeyIssued = 0xFFFF;
        }
    }
    else
    {
        sKeyStatus &= ~(KeyDetect | KeyLow2High);
        for (sIndex=0; sIndex<Number_of_Cols;
sIndex++)
```

```
            sKeyPreviousCol[sIndex] = 0;
    }
    // Transfer Current reading to debounced reading
    for (sIndex=0; sIndex<Number_of_Cols; sIndex++)
    {
        sKeyDebouncedCol[sIndex] =
sKeyCurrentCol[sIndex];
        sKeyLow2HighCol[sIndex] = 0;
    }
}
/**
 * @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
 */
 if
(HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE
_SCALE1) != HAL_OK)
 {
   Error_Handler();
 }
 /** 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 = 1;
 RCC_OscInitStruct.PLL.PLLN = 10;
```

```c
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
 RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
 RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
 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_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_4) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void)
{
 hi2c1.Instance = I2C1;
 //hi2c1.Init.Timing = 0x10D19CE4;
 hi2c1.Init.Timing = 0x00707CBB;  // 100kHz @ 16MHz
clock
 hi2c1.Init.OwnAddress1 = 0;
 hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
 hi2c1.Init.DualAddressMode =
I2C_DUALADDRESS_DISABLE;
 hi2c1.Init.OwnAddress2 = 0;
```

```
 hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
 hi2c1.Init.GeneralCallMode =
I2C_GENERALCALL_DISABLE;
 hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
 if (HAL_I2C_Init(&hi2c1) != HAL_OK)
 {
   Error_Handler();
 }
 /** Configure Analogue filter
 */
 if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1,
I2C_ANALOGFILTER_ENABLE) != HAL_OK)
 {
   Error_Handler();
 }
 /** Configure Digital filter
 */
 if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) !=
HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN I2C1_Init 2 */
 /* USER CODE END I2C1_Init 2 */
}
/**
 * @brief SPI1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI1_Init(void)
{
 /* USER CODE BEGIN SPI1_Init 0 */
 /* USER CODE END SPI1_Init 0 */
 /* USER CODE BEGIN SPI1_Init 1 */
 /* USER CODE END SPI1_Init 1 */
 /* SPI1 parameter configuration*/
 hspi1.Instance = SPI1;
 hspi1.Init.Mode = SPI_MODE_MASTER;
 hspi1.Init.Direction = SPI_DIRECTION_2LINES;
```

```c
 hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
 hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
 hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
 hspi1.Init.NSS = SPI_NSS_SOFT;
 hspi1.Init.BaudRatePrescaler =
SPI_BAUDRATEPRESCALER_256;
 hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
 hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
 hspi1.Init.CRCCalculation =
SPI_CRCCALCULATION_DISABLE;
 hspi1.Init.CRCPolynomial = 7;
 hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
 hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
 if (HAL_SPI_Init(&hspi1) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN SPI1_Init 2 */
 /* USER CODE END SPI1_Init 2 */
}
/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{
 /* USER CODE BEGIN TIM2_Init 0 */
 /* USER CODE END TIM2_Init 0 */
 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 /* USER CODE BEGIN TIM2_Init 1 */
 /* USER CODE END TIM2_Init 1 */
 htim2.Instance = TIM2;
 htim2.Init.Prescaler = 3999;
 htim2.Init.CounterMode = TIM_COUNTERMODE_DOWN;
 htim2.Init.Period = 19;
 htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV2;
 htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_ENABLE;
```

```
 if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
 {
   Error_Handler();
 }
 sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
 if (HAL_TIM_ConfigClockSource(&htim2,
&sClockSourceConfig) != HAL_OK)
 {
   Error_Handler();
 }
 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
 sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
 if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN TIM2_Init 2 */
 /* USER CODE END TIM2_Init 2 */
}
/**
 * @brief TIM5 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM5_Init(void)
{
 /* USER CODE BEGIN TIM5_Init 0 */
 /* USER CODE END TIM5_Init 0 */
 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 /* USER CODE BEGIN TIM5_Init 1 */
 /* USER CODE END TIM5_Init 1 */
 htim5.Instance = TIM5;
 htim5.Init.Prescaler = 0;
 htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim5.Init.Period = 4294967295;
 htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

```
 htim5.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
 {
   Error_Handler();
 }
 sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
 if (HAL_TIM_ConfigClockSource(&htim5,
&sClockSourceConfig) != HAL_OK)
 {
   Error_Handler();
 }
 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
 sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
 if (HAL_TIMEx_MasterConfigSynchronization(&htim5,
&sMasterConfig) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN TIM5_Init 2 */
 /* USER CODE END TIM5_Init 2 */
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
//static void MX_GPIO_Init(void) {
//    GPIO_InitTypeDef GPIO_InitStruct = {0};
//
//    // Enable clocks
//    __HAL_RCC_GPIOB_CLK_ENABLE();
//
//    // Configure I2C1 pins (PB8: SCL, PB9: SDA)
//    GPIO_InitStruct.Pin = GPIO_PIN_8 | GPIO_PIN_9;
//    GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
// Open-drain
//    GPIO_InitStruct.Pull = GPIO_PULLUP;
```

```c
// Required for I2C
//     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
//     GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
// Correct alternate function
//     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//
//     // Disable all other unnecessary GPIO
configurations
//}
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_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOA,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_8
                          |GPIO_PIN_9,
GPIO_PIN_RESET);
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_10|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                          |GPIO_PIN_6,
GPIO_PIN_RESET);
 /*Configure GPIO pins : PC0 PC1 */
 GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
 /*Configure GPIO pins : PA0 PA1 PA4 PA8
                          PA9 */
 GPIO_InitStruct.Pin =
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_8
                          |GPIO_PIN_9;
 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 pin : PB0 */
 GPIO_InitStruct.Pin = GPIO_PIN_0;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
 /*Configure GPIO pins : PB10 PB3 PB4 PB5
                           PB6 */
 GPIO_InitStruct.Pin =
GPIO_PIN_10|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                         |GPIO_PIN_6;
 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 : PA10 */
 GPIO_InitStruct.Pin = GPIO_PIN_10;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
 HAL_GPIO_Init(GPIOA, &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  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 */
```