Project: MIMIC

Jocelyn Lee, Lauren DeFelice, and Kayla Kohr

ECE 3130-001: Microcomputer Systems

April 30, 2025

Table of Contents

Introduction

The main goal of this project is to develop an interactive memory game on a NUCLEO board, utilizing both built-in and external LEDs, a keypad and push buttons for user input, an LCD display for feedback, and a buzzer for a more exciting gameplay. The game, named MIMIC, aims to challenge users by presenting a series of LED patterns that they must replicate in sequence that increases the further the user gets.

Project Report Overview

This report explains how “MIMIC”, the memory game, was designed and built from start to finish. It includes our original goals, the planning process, and how the game was developed and tested. The report outlines the project specifications, describes the game’s flow, and breaks down how both the hardware and software were implemented. It also discusses the challenges we faced and how we solved them. Finally, the report includes testing results and team collaboration details.

Applications and Their Importance

This project can be applied in several fields, including education, cognitive training, and interactive entertainment. Memory games have been used to improve cognitive abilities such as attention, pattern recognition, and memory retention. As such, its adaptability makes it suitable for educational environments and various interactive applications.

What Was Done and What Was Not

In this project, we were able to successfully implement the following features:

- *Menus*: A user-friendly menu to start the game, view instructions, and handle the game over screen that also displays the total number of rounds completed.
- *Randomized Pattern Generation*: Every round, a pattern is randomly generated for the user to mimic. The pattern's length grows each round that has been completed successfully.
- *LEDs*: There are a total of seven LEDs used for this game: three external and four built into the NUCLEO board. The external LEDs are used to show the player how many remaining lives they have while the other four LEDs create a sequence that must be mimicked.
- *User Input Comparison*: The keypad is used to start or restart the game while SW2-SW5 are used to recreate the shown pattern. Our project is then able to take that input and compare it to the generated pattern. From this, it will determine if they failed or passed the round.
- *Buzzer Based Audio*: A buzzer is used to signal correct and incorrect inputs, play a song during the main menu and another song on the Game Over screen.

However, there were some features we were unable to implement due to time and the limitations of our own knowledge in association to coding the NUCLEO board:

- *Time Limitations*: It was originally planned to have a 10 second timer for each round that counted down to add another element of urgency but ran into constant bugs. Due to the time limit, it was decided to opt out of continuing with this want.

- *Increased Sequence Speed*: This was an option that was thought about and had discussed whether or not to include but it was decided that the gameplay was already difficult enough with the increasing pattern length so it was left out. There were also issues testing this function such as the game crashing if it moved past more than 4 rounds as the speed was too fast for it to handle.

Although these were left out of the code, they were very beneficial in helping push towards the completed project.

Challenges Faced and the Solutions

We have faced many different challenges throughout this project which has made this a great learning experience for all of us. The three biggest challenges were:

- *External LEDs*: Throughout this class and its lab, we mostly stuck to dealing with what was built into the board such as the LCD, LEDs 0-3 and the buzzer. To learn how to integrate the three life counting external LEDs, we had to research and study the NUCLEO's user manual, experiment with how to properly wire them to the board, and learn what commands were needed for them to be used.
- *Randomizing Pattern and Comparison*: The hardest thing about our code when we first started was the game itself. None of us knew how to create a randomizer in C so it took a lot of research. When it was finally testable, we ran into issues such as the pattern being identical in the next round, which is the opposite of it being random. This was fixed by adding a delay and making sure the rand function wasn't defaulting to 1 every start up. Another issue was the code reading multiple inputs if a single button was pushed. This

was fixed by adding a spot where the code would do nothing until one of the SW buttons were pressed. We also edited where it would only read in the input after release.

- *Integration of Code*: When we first began coding "MIMIC", we broke it up into three areas of code: LCD comments, music, and the random pattern generator. When we finally came together to combine everything, there were a few errors that needed to be worked through and a lot of code in one place. Like most coding projects, if one error is fixed, two more are revealed. Our solution to working through all of this was to slowly integrate the code and break up all of the bulk in the main function to smaller functions for readability and troubleshooting since it was easier to find the issue if it is isolated in its own area.

Functional Description

The memory game begins by displaying a welcome screen with the startup song, followed by a countdown timer on the LCD. After the countdown, the game generates a sequence of LED blinks that increase in length each round. The user must replicate the sequence using the push buttons SW2-SW5. If the user's input matches the pattern, they move on to the next round. If not, they lose a "life" and stay on the same round until it is successfully passed. After three unsuccessful attempts, the game ends, displaying a "Game Over" screen with the "Score" which shows how many rounds were passed successfully. It will then pull up a screen prompting the user to push any keypad button to play again.

Project Specifications

This project is a memory game, titled “Mimic”, that involves user interaction through the keypad and switch buttons. Visual aids such as the board’s LEDs, LCD screen, and external LEDs are used to guide the user through the instructions of the game, and to track their progress while playing. In addition, the board’s buzzer is used to play melodic tones, enhancing the user experience through sound. It emphasizes moments like winning and losing, and adds excitement during the countdown before the game begins. It was also used to create a theme song for the game.

The game begins with a welcome message and theme song, then prompts the user to press any keypad button to begin. Once a keypad button is pressed, instructions for how to play are then displayed on the LCD screen. To play the game, the user watches a sequence of LEDs turning on and off, and then must repeat the pattern by pressing the corresponding buttons located beneath each LED once the sequence has concluded. These instructions are compacted into a phrase short enough to fit on the screen, as “Repeat LED cycle using SW2 - SW5”. Then the screen is cleared, and the message “Game begins in” displays, with a countdown of 3, 2, 1 on the second line, then the phrase “GO!!!”. This marks the start of the LED sequence process.

The rounds are kept track of via the LCD screen. It will stay on the current round if the user makes a mistake, and will increment the round by one if the user wins the round. There are an infinite amount of rounds available to play, as a variable called “patternLength” begins at one, and continuously increments by one as the user wins a round. However, it is highly unlikely for the user to get into a double-digit round number, because as each round finishes, the next one is made increasingly difficult. The rise in difficulty is gauged by the amount of LEDs that flash

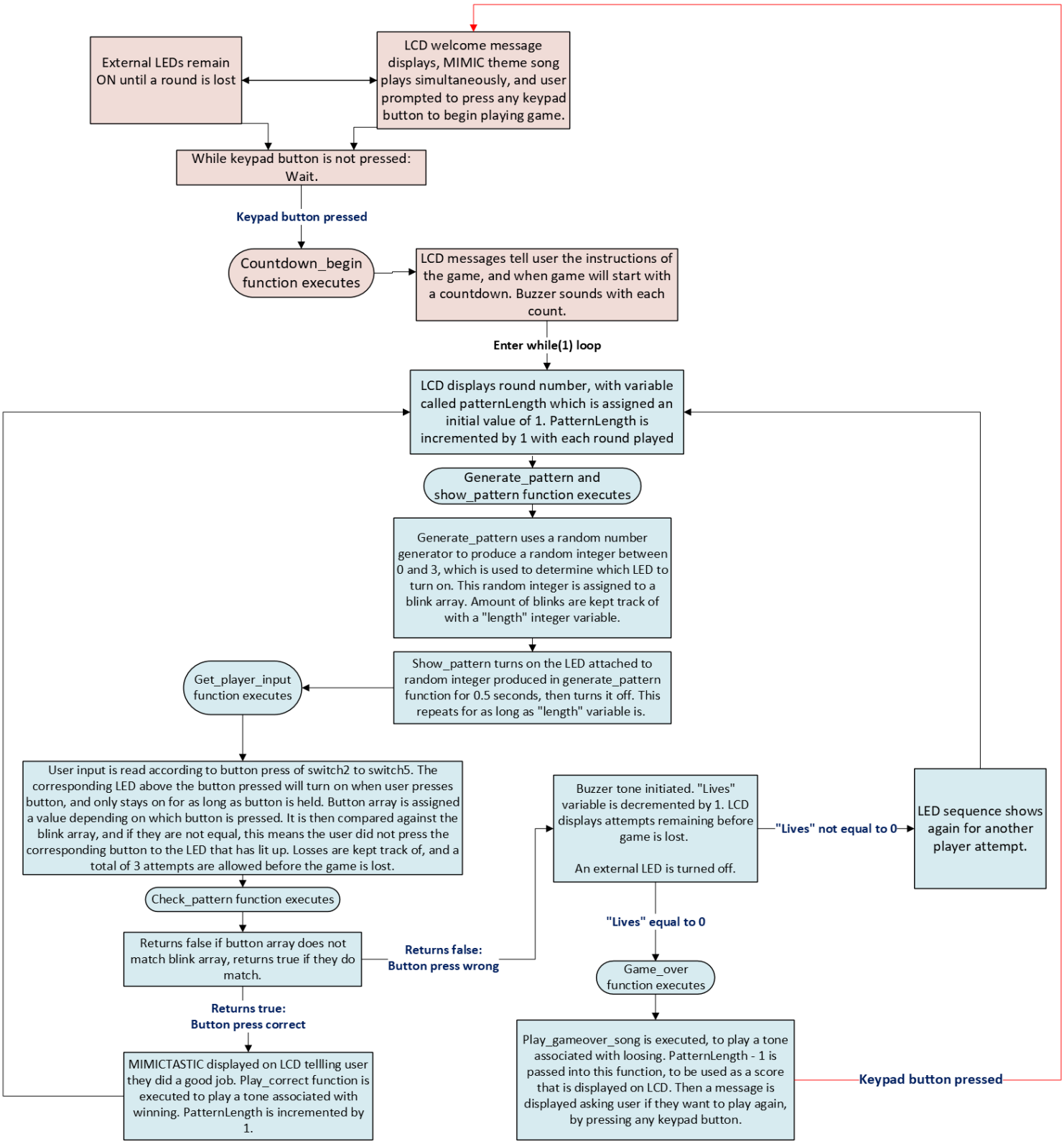
within one round. As the rounds progress, the LED sequences become longer, with more LEDs flashing in each new round.

If the player wins a round, “MIMICTASTIC” is displayed on the LCD screen, as a way to say good job that aligns with our game’s theme, as well as a bright and positive sounding tone from the buzzer. The round is then incremented after a short delay, giving the user a break between rounds while informing them of their win.

If a player loses a round by pressing mismatching buttons, then a message displaying “OH NO…” is shown, along with a somber sounding buzzer tone, reinforcing the fact that the player lost the round. In addition, three external LEDs have been included in the project, wired from the breadboard to pins within the CN8 connector region on the NUCLEO-L476RG board. These LEDs are set to be on when the game is first powered up. These three LEDs have been chosen to correspond with the amount of losses that are allowed, before you lose the game in totality. This means that a player is given three attempts in total to guess any LED pattern correctly. Only three attempts are given for the entire game. If the player presses an incorrect button, one external LED turns off to indicate the loss of one of their three lives. The remaining two LEDs will turn off with each additional mistake, reflecting the loss of the remaining lives.

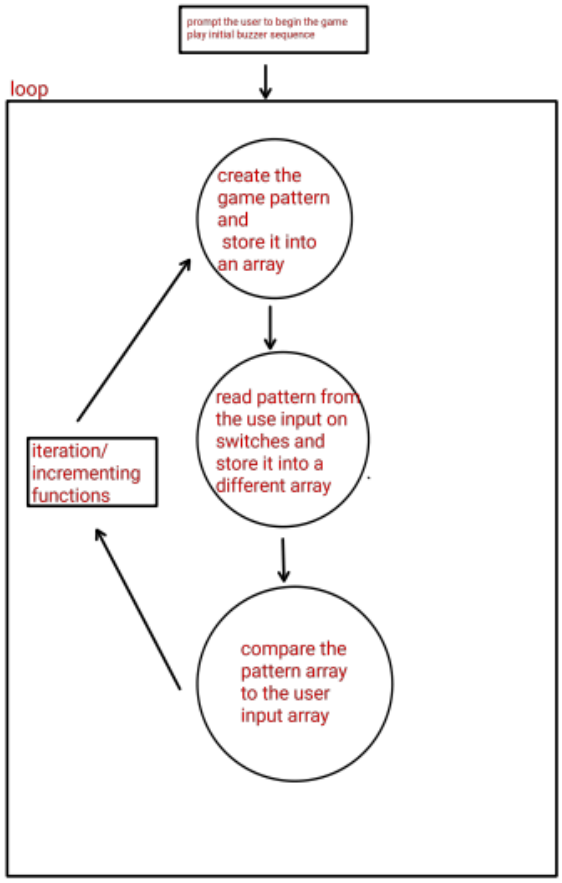
Once three attempts have surpassed, then a “Game Over” message is displayed, along with the player’s score on the line below it. The score is calculated based on how many rounds the player completes. After a 2 second delay for the player to see their score, an LCD message prompts the user to press any keypad button to replay the game. If a keypad button is pressed, the game restarts.

Below is a flowchart that describes the process flow of our game, “MIMIC”.


External LEDs remain ON until a round is lost
LCD welcome message displays, MIMIC theme song plays simultaneously, and user prompted to press any keypad button to begin playing game.
While keypad button is not pressed: Wait.
Keypad button pressed
Countdown_begin function executes
LCD messages tell user the instructions of the game, and when game will start with a countdown. Buzzer sounds with each count.
Enter while(1) loop
LCD displays round number, with variable called patternLength which is assigned an initial value of 1. PatternLength is incremented by 1 with each round played
Generate_pattern and show_pattern function executes
Generate_pattern uses a random number generator to produce a random integer between 0 and 3, which is used to determine which LED to turn on. This random integer is assigned to a blink array. Amount of blinks are kept track of with a "length" integer variable.
Show_pattern turns on the LED attached to random integer produced in generate_pattern function for 0.5 seconds, then turns it off. This repeats for as long as "length" variable is.
Get_player_input function executes
User input is read according to button press of switch2 to switch5. The corresponding LED above the button pressed will turn on when user presses button, and only stays on for as long as button is held. Button array is assigned a value depending on which button is pressed. It is then compared against the blink array, and if they are not equal, this means the user did not press the corresponding button to the LED that has lit up. Losses are kept track of, and a total of 3 attempts are allowed before the game is lost.
Check_pattern function executes
Returns false if button array does not match blink array, returns true if they do match.
Returns false: Button press wrong
Buzzer tone initiated. "Lives" variable is decremented by 1. LCD displays attempts remaining before game is lost.
An external LED is turned off.
"Lives" not equal to 0
LED sequence shows again for another player attempt.
"Lives" equal to 0
Game_over function executes
Play_gameover_song is executed, to play a tone associated with loosing. PatternLength - 1 is passed into this function, to be used as a score that is displayed on LCD. Then a message is displayed asking user if they want to play again, by pressing any keypad button.
Keypad button pressed
Returns true: Button press correct
MIMICTASTIC displayed on LCD tellling user they did a good job. Play_correct function is executed to play a tone associated with winning. PatternLength is incremented by 1.

Detailed Implementation

To implement our game, we divided the development tasks into three main areas: LCD control, audio feedback through buzzers, and the core game logic. Each team member was responsible for one of these areas. After completing our individual components, we successfully integrated our work into the final cohesive project. The overall program structure is illustrated by the flowchart shown below.



Our design strategy emphasized modularity, with separate functions created for each step of the program's flow. We also enhanced gameplay by generating a new random pattern each time the game started, ensuring that every playthrough offered a unique experience. Using the modular functions, we were able to build game prompts and manage LCD output around the core game logic with clarity and flexibility. In every loop we added to the pattern length to increase difficulty.

A critical early phase of development involved initializing all necessary hardware components. This setup largely built upon techniques learned in previous course labs. However, configuring the LEDs and switches

required additional effort. Identifying the correct GPIO pins involved referencing datasheets and consulting instructors. Through this process, we determined that:

- LEDs 0 and 1 were connected to GPIOA
- LEDs 7 and 8 were connected to GPIOC,
- All four switches were connected to GPIOB.

We then wrote a hardware initialization function that configured these pins appropriately for inputs and outputs as needed. Our code looked like this:

```
// Set up the GPIO pins
static void MX_GPIO_Init(void)
{
  // TODO: Initialize any input or output pins you'll use here
      GPIO_InitTypeDef GPIO_InitStruct = {0};

    // Enable GPIOA clock
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();

    // Configure PA1 (LED) as output
    GPIO_InitStruct.Pin = GPIO_PIN_1 | GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_7 |GPIO_PIN_8 ;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    // Configure PA0 (Button) as input with pull-up
    GPIO_InitStruct.Pin = GPIO_PIN_8 |  GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 ;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  // To initialize a pin as an input with the HAL, use
  //GPIO_InitStruct.Mode = GPIO_PIN_1
  //GPIO_InitStruct.Mode = GPIO_MODE_INPUT
  // when initializing the pin
}
```

Game Play

After we had set up inputs and outputs it was time to create the pattern, read a pattern from the user, and compare the two. Here is what the pattern generation was like:

```
  int patternlength = 1;
  // TODO: Put any other initialization code here
  uint8_t blink[patternlength];  //array of the blinking LEDs created
  uint8_t button[patternlength]; //array of the four buttons pressed
// Variable to track LED state

while (1)
{
    // 1) Generate random code and put it into the LEDs
    for (int i = 0; i < patternlength; i++)
  {
      blink[i] = rand() % 4;
        //blink the corrosponding LED to each generated number
        if (blink[i] == 0)
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
        //buzzer 1 write pin high
        else if (blink[i] == 1)
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
        else if (blink[i] == 2)
          HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
        else if (blink[i] == 3)
          HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);

        //keep the blink on for a certain amount of time
        HAL_Delay(500);
```

After this was done we read the input from the user:

```
for (int i = 0; i < patternlength; i++)
{
    // 1) Wait for *any* button to go high
while (
    HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8) == GPIO_PIN_RESET &&
    HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9) == GPIO_PIN_RESET &&
    HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10) == GPIO_PIN_RESET &&
    HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11) == GPIO_PIN_RESET)
{
  //do nothing
}

// Button 1 (PB8) LED on PA1
if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8) == GPIO_PIN_RESET)
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // OFF
else
{
  //if button is pressed display the LED and store 0 into array
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);   // ON
    button[i] = 0;
}


// Button 2 (PB9) LED on PA0
if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9) == GPIO_PIN_RESET)
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
else
  {
  //if button is pressed display the LED and store 1 into array
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
    button[i] = 1;
  }

// Button 3 (PB10) LED on PC7
if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10) == GPIO_PIN_RESET)
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
else
  {
  //if button is pressed display the LED and store 2 into array
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
    button[i] = 2;
  }
```

It is also worth noting that in our final code implementation, each time a button was pressed, the corresponding LED would light up. While this step was not strictly necessary for the core functionality of the game, it significantly enhanced the user experience. By providing immediate visual feedback that linked each switch to a specific LED, players could more easily associate the

controls with the game's responses, making the gameplay more intuitive and engaging.

The comparing function then looked like so:

```
Write_Instr_LCD(0x01); //Clears the LCD
int correct = 1;
for (int i = 0; i < 4; i++)
{
    if (blink[i] != button[i])
    {
        correct = 0;
        break;
    }
}
Write_Instr_LCD(0x01);              // clear
if (correct == 1)
    Write_String_LCD("Pattern OK");

else
    Write_String_LCD("Try again!");

HAL_Delay(1000);
patternlength ++;
```

This functionality was developed during the early stages of the project, before the LCD and buzzer systems were fully integrated into the code. Initially, the game displayed simple indicators such as "OK" and "Try Again" through basic outputs. As development progressed, we refined these prompts to display more descriptive messages on the LCD, enhancing clarity and improving the overall user experience.

The LCD display was later added with code similar to this:

```
Write_String_LCD("Welcome to MIMIC");
HAL_Delay(4000);
Write_Instr_LCD(0x01);
Write_String_LCD("Press any keypad ");
Write_Instr_LCD(0xC0);
Write_String_LCD("button to start!");

while(!is_key_pressed()) {}
HAL_Delay(200); // debounce delay
Write_Instr_LCD(0x01);

Write_String_LCD("Repeat LED cycle");  // then write line 4 "Repeat LED cycle" on first line
Write_Instr_LCD(0xC0);  // move cursor to bottom line of LCD screen
Write_String_LCD("using SW2 - SW5.");  // write line 5 "using SW2 - SW5." on second line
HAL_Delay(3000);

Write_Instr_LCD(0x01);
Write_String_LCD("Game begins in");  // then write line 6 "Game begins in" on first line
Write_Instr_LCD(0xC0); // move to next line for countdown
HAL_Delay(500);
Write_Instr_LCD(0xC0 + 7); // center of 16 char screen on second line so that "Game begins in" still displays on first line
Write_String_LCD("3");
HAL_Delay(1000);
Write_Instr_LCD(0xC0 + 7);
Write_String_LCD("2");
```

The LCD primarily utilized ASCII coding to control character placement and display the correct information. Refining these lines of code provided valuable insight into how individual characters are mapped and displayed on the screen.

Audio - Buzzer

The final feature we integrated into the project was audio output through a buzzer. This was accomplished by using numerical frequency values to represent musical notes. Each number in the melody array corresponds to a specific note on the musical scale — for example, the note C5 is represented by the frequency 523 Hz. In parallel, a second array was used to define the duration each note should be played. These values allowed us to create recognizable tones and rhythms by sending the appropriate frequency and timing information to the buzzer through a GPIO pin.

```
int melody[] = {
    523, 659, 784, 523, 659, 784, 523, 392,
    440, 523, 659, 523, 440, 523, 440, 392,
    659, 523, 440, 392, 523, 659, 784, 523
};
int noteDurations[] = {
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8
};
void tone(GPIO_TypeDef *port, uint16_t pin, uint32_t freq, uint32_t duration_ms);
```

During development, two main challenges emerged: **button debouncing** and **truly random pattern generation**. Initially, although the game appeared to generate random patterns, it would consistently produce the same sequence after every board reset. This was due to the random number generator being seeded with a fixed value at startup. To resolve this, we seeded the random number generator with the system clock, ensuring different patterns with each reset.

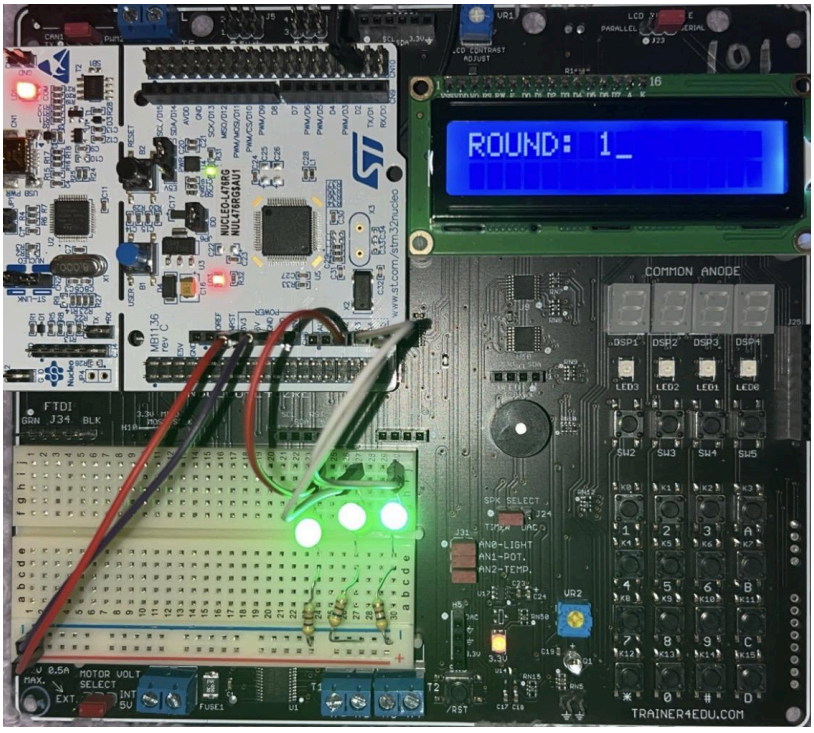
For debouncing the push buttons, we added appropriate delays to allow input signals to stabilize before processing them. Additionally, we cleared all input pins not currently in use, reducing the chance of false inputs or glitches during gameplay.

Through careful task division, hardware integration, and refinement of system timing, we achieved a fully functioning and engaging game experience.

Project Hardware

The hardware setup for the project involves the NUCLEO board with the following components:

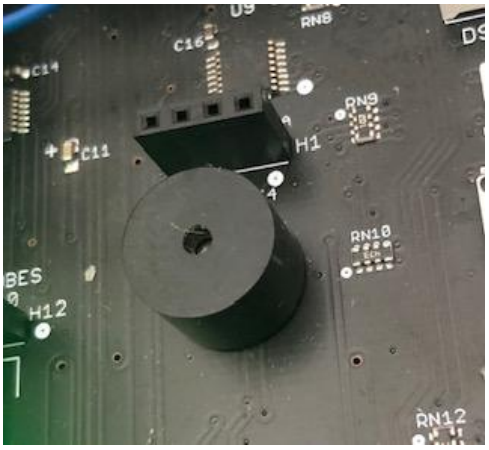
- *Built-in LEDs (LED0–LED3)*: Connected to GPIOA and GPIOC, these LEDs are used to display the generated pattern and provide feedback to the player.
- *External LEDs*: Three additional external LEDs are used to enhance visual feedback, with each LED assigned to specific GPIO pins for user interaction and pattern display.
- *Switches*: Four buttons (SW2–SW5) connected to GPIOB (pins PB8–PB11) are used for user input to replicate the generated LED pattern.
- *Buzzer*: Connected to PC9, the buzzer provides audio feedback (success or failure sounds).
- *LCD*: An LCD connected via I2C displays game information such as prompts, countdown timers, and results.



Analysis

Testing for “MIMIC” came with plenty of trial and error. Most of the code was written based on our existing knowledge, with additional research conducted whenever we encountered challenges or uncertainties. Along with multiple components being used on the board, comes lots of time invested in testing to ensure the program is executing as expected. We approached this project by first developing and testing individual sections separately, ensuring each worked as intended, and then integrating them together at the end.

Buzzer



Research was conducted to be able to understand how to get the buzzer to operate. After compiling code in an attempt to play music, it was found that a better approach would be to create a commonly used “tone()” function in order to manipulate different melody notes. The tone function allowed for the desired frequency and duration in milliseconds to be entered in, and it would assign these values to the corresponding buzzer pins. As for the melodies in this project, after the tone() function was defined, we used a trial-and-error approach

to experiment with different frequencies and adjust them based on how they sounded. We found that lower frequencies tended to create a more somber tone, which worked well for indicating a game over or a loss, while higher frequencies produced a lighter, more cheerful sound that was better suited for signaling a win. From there, the focus shifted to combining the selected frequencies in a way that formed a coherent and recognizable melody, enhancing the overall experience of the game. This included experimenting with the duration of the tone, as well as the frequency of each individual tone.

LCD Screen

Due to past labs, writing to the LCD screen was a familiar task, and we were able to reference past work to implement it effectively in this project. How the text looked was a concern of ours, especially when it came to centering text. It was previously learned that using

"Write_Instr_LCD(0xC0)" after all proper initializations, would move the cursor to the second line. It was observed that 16 chars can fit in one line on the screen. Eight chars in would mean that the cursor would lie in the center of the screen. To compensate for the space that a number would take up, being centered in the screen for a countdown, it was found that using "Write_Instr_LCD(0xC0 + 7)" would move the cursor over 7 chars, effectively displaying the digit in the center of the screen. For words that took up more space, the same logic was applied after figuring out that it works the way we expected, and we were able to effectively get text centered on the screen.

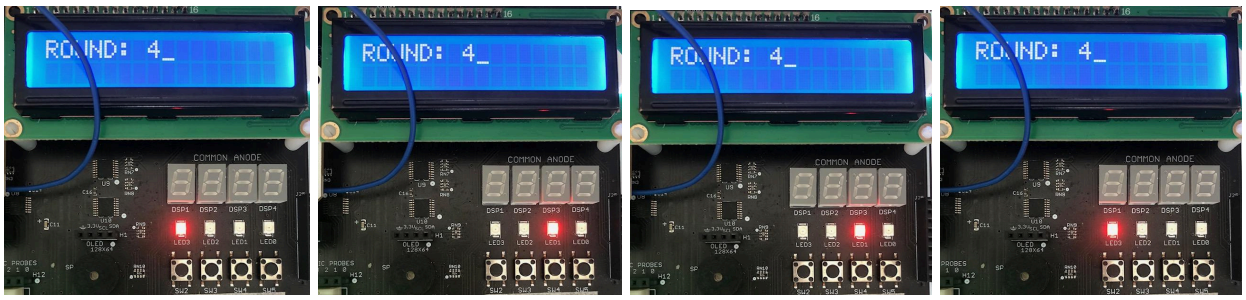
Order of operations was a big part of testing our project. There were times when certain lines of code were just in the wrong spot, although they were typed correctly. This occurred with all sections of our code. For example, clearing the screen at the correct time so that text does not overlap each other, and creating delays in the right spots so that the messages were both readable and displayed long enough for the user to see clearly. When these things would happen, most of the time we would reread our code, and recognize that it makes more sense for this line to be here, instead of there.

Random Number Generator

After some research and with prior knowledge from previous classes, it was found that a rand() function would need to be used to create our random LED pattern. It was taken note of that we would need to use rand() mod 4, to be able to limit the integer value, from zero to three, representing the four different LEDs on the board. After the base code was written, it was found that the same pattern would display, all the time. Multiple fixes were attempted, but the only one that worked was seeding the random function to the system clock. “srand(HAL_GetTick());” uses the system clock to create random numbers based on time, which results in a varying number production from the random function.

“Length” and “patternLength” were very important variables used in our code. “Length” was often passed into multiple functions, carrying the value of “patternLength”. These variables were also used in for-loops to keep track of round number, player score, as well as how many LEDs will flash in a given round. Before finalizing the code, we initially did not use shared variables for these events, which made the logic harder to follow. During integration, we addressed this by introducing mutual variables, which not only improved the readability of our

code but also helped everything function more smoothly and in line with our intended design. This also creates a smoother experience for the user. It makes sense that if you successfully complete 4 rounds, your score will be a 4. Also, a pattern of 4 LEDs flashes for that round. This simplifies the code in an effective way. See example of a randomly generated LED sequence during round 4.

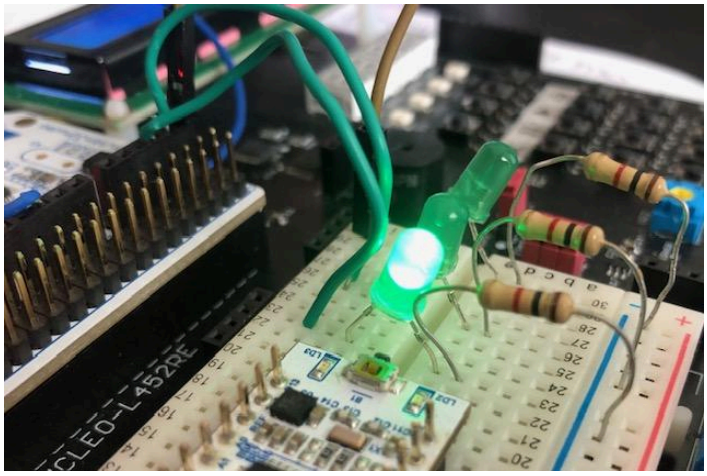


External LEDs

The addition of external LEDs was something that was not gone over in previous classes. After observing that they needed to be connected to the NUCLEO board, a pinout diagram was found and used to begin the wiring process. A simple test code was used to test the operation of

an external LED. The code snippet shows the testing of the LED connected to pin PC0.

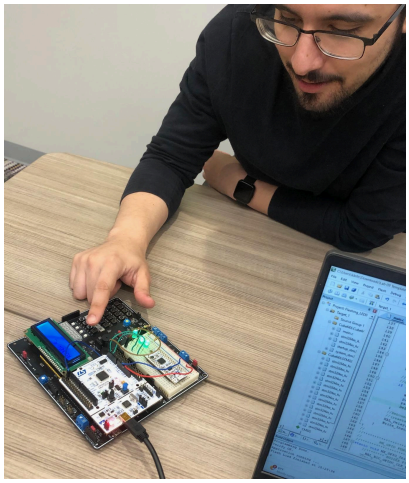
```
  while (1)
  {

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);  // Set the pin high
    HAL_Delay(1000); // Wait 1 second
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // Set the pin low
    HAL_Delay(1000); // Wait 1 second
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0}; // Declare and initialize to zero

    __HAL_RCC_GPIOC_CLK_ENABLE(); // Enable GPIOC for PC0

    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; // Push-pull mode
    GPIO_InitStruct.Pull = GPIO_NOPULL;         // No pull-up or pull-down resistors
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW; // Set speed to low
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}
```



Once it was confirmed that using similar code from previous labs still works with external LEDs the same way it does with on-board LEDs, it was then implemented into our code where needed. Again, placement was very important to pay attention to during this process, however, confidence was boosted after confirming that this test code worked as intended.

Field Testing

As our game made sense to all of our members, we wanted to ensure that if a player encountered this game and we were not there to tell them how to play it, they could still understand what to do. To accomplish this, we each found one person who was not in our class, to play the game, without giving them any instruction on what it is or how it works. We found that out of the three people we handed our boards off to, all three of them were able to successfully figure out what we were asking of them. This confirmed that the instructions were clear and concise, and the game itself was built properly as to where anybody could understand how to play it. We even had a player get a score of 9 which sadly is higher than what the creators could do.



Description of Teamwork Experience

Our team created a collaborative and inclusive environment by assigning tasks based on each member's interests, while making sure everyone's input was supported during discussions. We met every Monday to outline our goals for the week and to show the progress each person had made. These meetings helped us stay aligned and focused, and allowed us to quickly address any issues that came up.

Between meetings, we stayed in frequent contact through messaging, where we asked questions, shared updates, and offered help to one another. Leadership rotated depending on the task—whether it was programming the LCD, configuring the buzzer, or handling game logic, the

team member with the most experience in that area would guide the others. This structure allowed us to plan tasks efficiently, stay on schedule, and complete the project as a well-coordinated team.