Bialystok University of Technology

Faculty of Computer Science

Marcin Kondrat
album no.: 115530

Jowita Ochrymiuk
album no.: 115538

Final project

Implementation of the "Simon Says" game project on the STM32 NUCLEO-L476RG microcontroller

Subject: Computer Architecture
Presenter: Dr. Eng. Mirosław Omieljanowicz

Bialystok 2025

Contents

Entry

The aim of the project was to design and implement the game "Simon Says" on the STM32 NUCLEO-L476RG microcontroller platform. This game develops the memory and concentration of players, requiring the playback of presented light sequences in the correct order.

The project was implemented as part of the Computer Architecture course at the Białystok University of Technology as a practical application of knowledge acquired during classes. The project used the NUCLEO-L476RG platform, which was adapted to support the game and expanded with additional hardware elements, such as a designed PCB board, to deepen practical design skills.

The implementation included two key aspects:

- Designing an electrical circuit and PCB to connect a microcontroller to peripherals such as an LCD display, LEDs, and tact switch buttons.
- Programming the game system in C, including implementing mechanisms for generating sequences, supporting multi-player gameplay, and visualizing the result on the display screen.

The project is based on the STM32L476RG platform, which was selected due to its:

- Low energy consumption.
- Extensive GPIO, SPI and UART configuration options.
- Availability of the NUCLEO platform, facilitating the implementation of microcontroller solutions.

The scope of work included the design and implementation of hardware and software solutions, including:

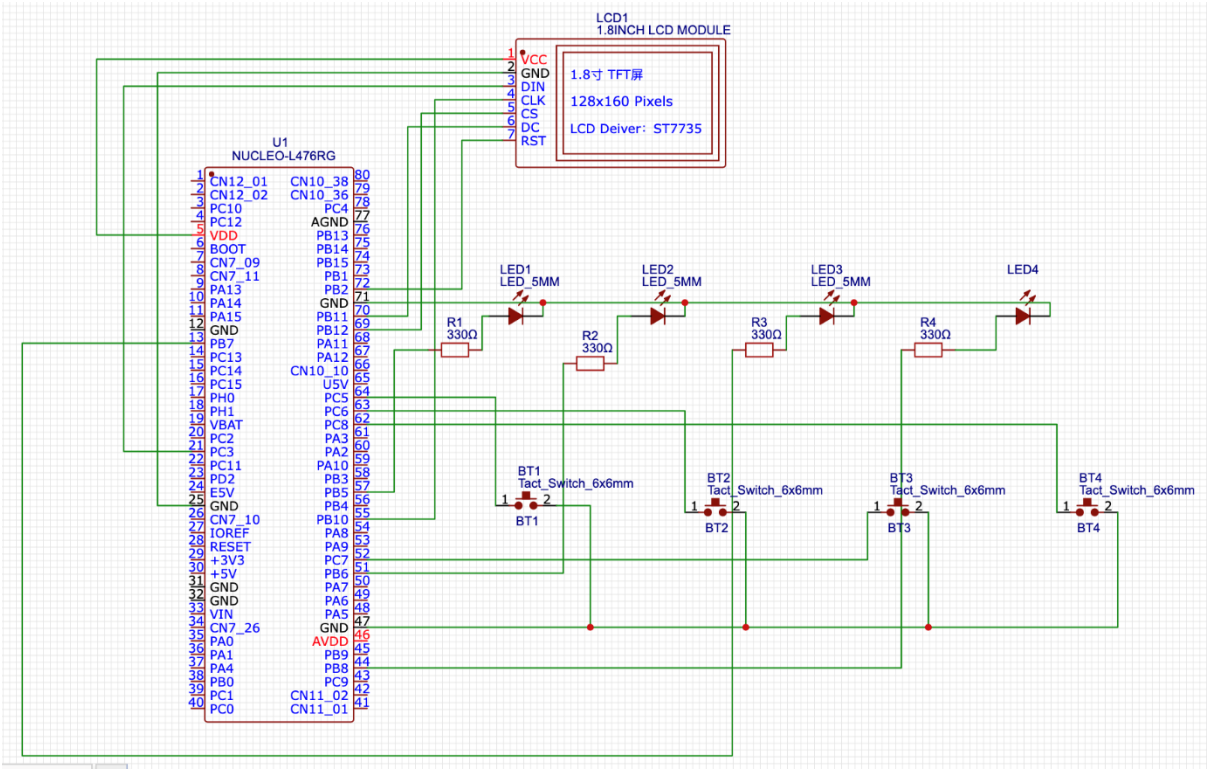
- PCB design.
- Integration of components such as the ST7735 LCD display, LEDs and buttons.
- Implementation of a game with the ability to support multiple players and remember results.

The goal was to create an interactive and dynamic game that combines elements of entertainment with practical use of knowledge about microcontrollers, programming and design of electronic systems.

Layout Design

The project involved designing an electrical system and ordering a dedicated PCB that integrates the STM32 NUCLEO-L476RG microcontroller with peripherals such as an ST7735 LCD display, four LEDs, and four tact switch buttons. The board was designed for ease of assembly, minimal interference, and stability of system operation.

Electronic system



The system uses the following connections:

1. **LCD Display ST7735**
    a. **VCC (power supply):**connected to VDD of the microcontroller.
    b. **GND:**connected to the system ground.
    c. **DIN:**connected to pin PC3 (SPI interface).
    d. **CLK:**connected to pin PB10 (SPI clock).
    e. **CS:**connected to pin PB12 (SPI device selection).
    f. **DC:**connected to pin PB11 (data/command mode selection).
    g. **RST:**connected to pin PB2 (display reset).
    h. **BL:**was not connected (brightness control; the module has a built-in resistor that pulls up this line to the power supply, so the display will shine at maximum brightness).
2. **LEDs**
    a. **LED1:**anode connected to PB5, cathode to GND.
    b. **LED2:**anode connected to PB6, cathode to GND.

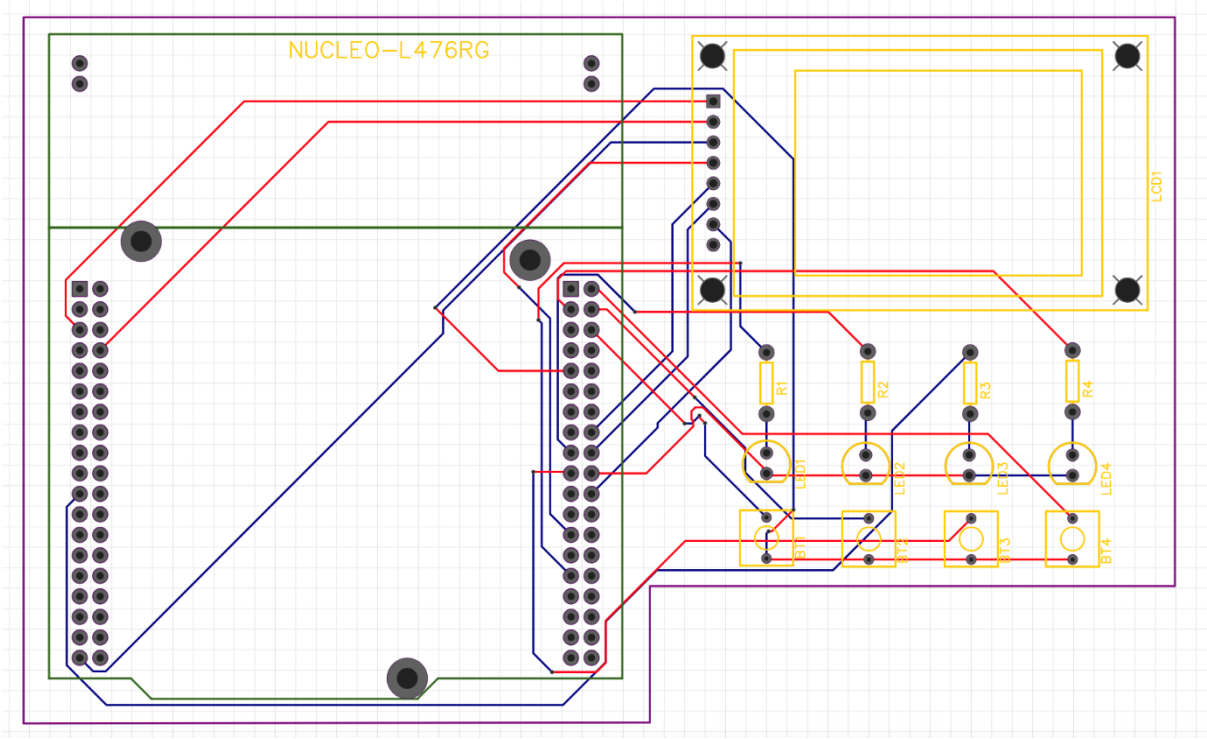
c. **LED3:**anode connected to PB6, cathode to GND.
d. **LED4:**anode connected to PB7, cathode to GND.

3. **Tact switch buttons**
   a. **BT1:**one pin to PC5, the other to GND.
   b. **BT2:**one pin to PC6, the other to GND.
   c. **BT3:**one pin to PC7, the other to GND.
   d. **BT4:**one pin to PC8, the other to GND.

PCB board

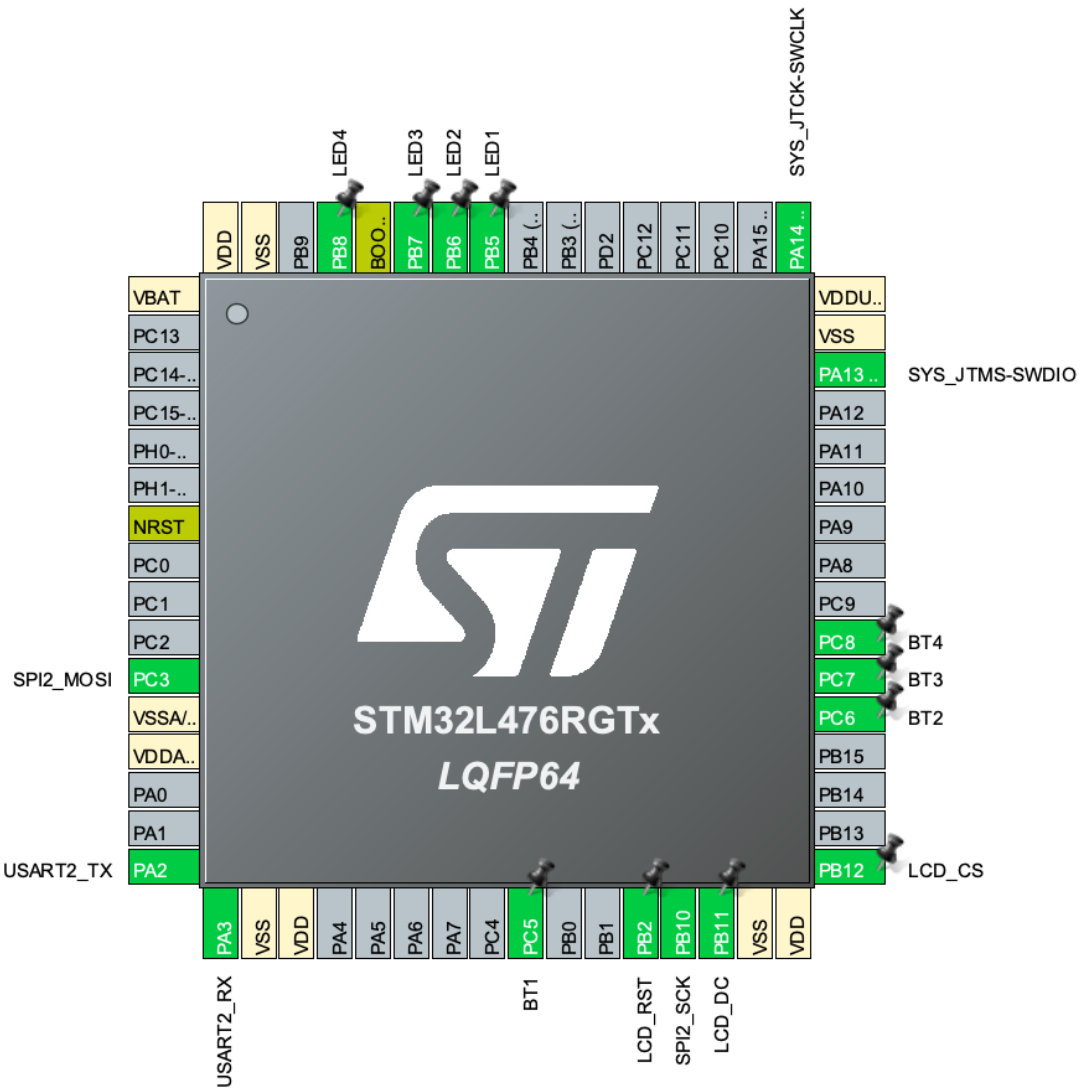
The system was designed taking into account:

- Optimizing PCB traces to minimize interference and improve signal integrity.
- Arrangement of components to ensure easy access to buttons and clear visibility of the display.
- Minimizing wired connections to simplify the assembly and testing process.



Microcontroller peripheral support

To implement the project, it was necessary to configure a number of peripherals, such as GPIO, NVIC, UART and SPI. Below are the details of the configuration of key elements that allowed the system to function correctly.



GPIO – Buttons and LEDs

1. **Buttons:**The tact switch buttons were configured as GPIO inputs with active pull-up resistors. This avoided undefined input states when the buttons were not pressed. Additionally, interrupts were used to handle the buttons, which allows for immediate response to their pressing without the need to constantly poll the input state.
2. **LEDs:**The LEDs have been configured as GPIO outputs with basic settings. Controlling the LEDs involves setting the high state (HIGH) on the appropriate pins, which causes them to light up.

Interrupts (NVIC)

The project has two key interrupt sources configured: USART2 and GPIO (buttons).

- **USART2:**Set to priority 12, ensuring that UART communication related interrupts have a lower priority than GPIO interrupts.

- **GPIO:**Priority 8 has been assigned to make button operation faster and more responsive. This makes the reaction to pressing a button almost instantaneous.

SYS Module – Serial Wire Debug (SWD)

The UART module (USART2) has been configured in asynchronous mode, which allows data transfer between the microcontroller and the PC for player naming and debugging.

Main configuration parameters:

- **Transmission speed:**115200 baud
- **Number of data bits:**8.
- **Parity:**lack.
- **Number of stop bits:**1.

SPI – Communication with LCD Display

The SPI module has been configured to communicate with the ST7735 LCD display. SPI enables fast and reliable data transfer, which is crucial for displaying dynamic information such as game menus or animations.

Main configuration parameters:

- **Operating mode:**Master.
- **Data format:**8 bit.
- **SPI clock frequency:**10MHz.
- **Clock Phase and Polarity:**CPOL = 0, CPH = 0.
- **Prescaler:**8.

Since the microcontroller board operates with a system clock of 80 MHz and the ST7735 LCD display supports a maximum SPI clock frequency of 15 MHz, the prescaler value is calculated using the following formula:

$$Preskaler = \frac{F_{sys}}{F_{SPI_{max}}}$$

Where:

- F_{sys} is the system clock frequency of the board (80 MHz),
- $F_{SPI_max}$is the maximum SPI clock frequency that the LCD can support (15 MHz).

Substituting the values:

$$Preskaler = \frac{80\ \text{MHz}}{15\ \text{MHz}} \approx 5.33$$

Since STM32CubeIDE only supports prescalers that are powers of 2, the closest higher power of 2 is 8. Therefore, we set the prescaler to 8, which gives an SPI frequency of:

$$F_{SPI} = \frac{80\ \text{MHz}}{8} = \mathbf{10\ MHz.}$$

Microcontroller clock speed

As part of the project, the STM32 NUCLEO-L476RG microcontroller was configured to work at a maximum frequency of 80 MHz. This increased the speed of the LCD display and the entire system, which significantly affected the smoothness of the animation and the responsiveness of the game.

The configuration uses:

- **MSI**(Multi-Speed Internal) as a base source with a frequency of 4 MHz.
- **PLL**(Phase-Locked Loop) to multiply signal frequencies up to 80 MHz.

The clocking was adjusted using the STM32CubeMX tool, where the appropriate dividers and signal source were set. By using this configuration, the system operates at maximum efficiency while maintaining the stability and precision of the clock signal.

Technical solutions

The game system code was written in C using the STM32CubeIDE environment. The implementation includes support for the LCD display, LEDs, tact switch buttons, and game logic based on the game states mechanism.

LCD Display Support ST7735

The LCD display plays a key role in the game, presenting information such as menus, player names, scores, and messages. The implementation includes:

- **Initializing the display**using the lcd_init() function.
- **Graphics support**using the HAGL library. The library was adapted to work with the ST7735 display controller, which enabled convenient drawing of text, shapes, animations and other graphic elements. Own initialization and control functions were added, which ensured compatibility with the low-level SPI protocol used by the controller.
- **Displaying messages**, such as the player selection menu, game results, current player turn, and hints.

Using the HAGL library

The HAGL library was used to simplify the display operation by introducing higher level functions such as:

- `void hagl_put_text(void *display, const wchar_t *text, uint16_t x, uint16_t y, uint16_t color, const font_t *font)` – display text on the screen.

- `void hagl_fill_circle(void *display, uint16_t x, uint16_t y, uint16_t radius, uint16_t color)` – drawing filled circles.

  - `display` – pointer to the display control structure (void).
  - `text` – text to display (const wchar_t).
  - `x` – horizontal coordinate (uint16_t).
  - `y` – vertical coordinate (uint16_t).
  - `radius` – radius of the circle (uint16_t).
  - `color` – color in RGB565 (uint16_t) format.
  - `font` – pointer to a font structure (const font_t).

Example code snippet:

```
// Display username
hagl_put_text(display, username_label, 10, 40 + (current_player * 20), WHITE, font6x9);
// Update the display content
lcd_copy();
```

Customizing the HAGL library

The customization included:

- **Introduction of SPI initialization and communication functions**with ST7735 controller.
- **Rendering speed optimization**by increasing the SPI clock frequency and using a buffer to minimize the number of write operations to the display.
- **Added support for 16-bit color palette (RGB565)**, used by the ST7735 driver.

This integration allows the display to operate at high performance, allowing for smooth animations and dynamic screen updates.

LED support

LEDs indicate the game sequence and player reactions. Each LED is controlled by a separate GPIO pin. LED support functions:

- `all_leds_on()`: turns on all leds.
- `all_leds_off()`: turns off all leds.
- `all_leds_blink(uint8_t times, uint32_t delay_ms)`: turns all LEDs on and off with the specified delay, creating a blinking effect.
- `all_leds_off_but(int num)`: turns off all LEDs except the specified one.

- `all_leds_on_but(int num)`: turns on all LEDs except the specified one.

    - `times` – number of LED blinks (`uint8_t`).
    - `delay_ms` – time in milliseconds between turning the LEDs on and off (`uint32_t`).
    - `num` – diode number (1-4; `int`).

Example of turning on a specific diode:

```
HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin,GPIO_PIN_SET);
```

Tact switch button support

The buttons are responsible for navigating the menu and selecting options. GPIO interrupts are used, which respond to pressing the button and perform the appropriate action depending on the current state of the game.

Example of button handling in the HAL_GPIO_EXTI_Callback() function:

```
if(GPIO_Pin == BT4_Pin) {
     game_state = 1;
     HAL_UART_Transmit(&huart2, (uint8_t*)"Back to Menu\r\n",
14, HAL_MAX_DELAY);
     all_leds_on_but(4);
     display_menu();
     return;
}
```

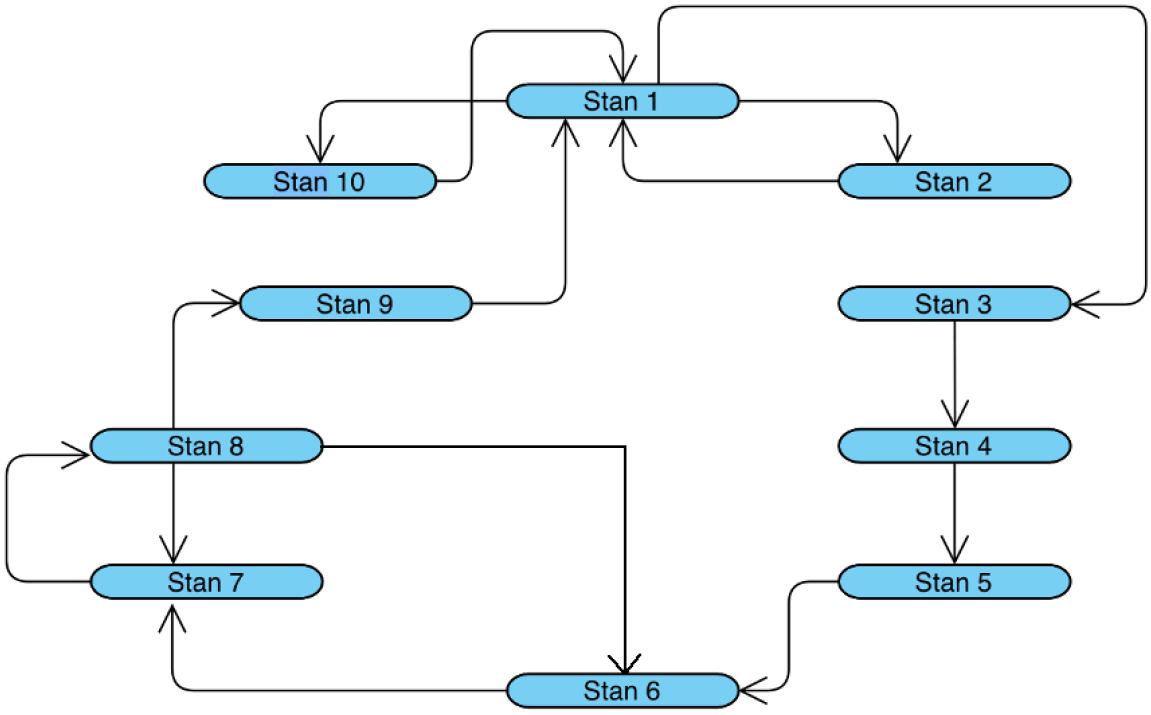
Game States

The game_state mechanism allows for smooth transitions between various stages of the game, such as the menu, selecting the number of players, presenting the sequence, gameplay and displaying the results.

1. **State 1 (Menu)**
    a. Displays the game's main menu with three options to choose from:
        i. *Start Game*(transition to state 3).
        ii. *Global Ranking*(transition to state 2).
        iii. *Credits*(transition to state 10).
    b. The selection is made using the BT1-BT3 buttons.
2. **Status 2 (Global Ranking)**
    a. Displays a list of players and their high scores saved in global memory.
    b. The player can return to the main menu (state 1) by pressing the BT4 button.
3. **State 3 (Selecting the number of players)**
    a. Allows you to select the number of game participants (1 to 4).

    b. The selection of the number of players is done using the BT1-BT4 buttons, which leads to the transition to state 4.
4. **Status 4 (Confirmation of selection of the number of players)**
    a. Displays information about the selected number of players.
    b. After a short time, it automatically moves to state 5, where player names are defined.
5. **State 5 (Introducing Player Names)**
    a. Allows you to enter player names via the UART interface.
    b. Once all player names have been entered, the game automatically moves to state 6.
6. **Status 6 (Waiting for player readiness confirmation)**
    a. Displays a screen asking you to confirm the player's readiness.
    b. After clicking the BT4 button, the game starts (state 7).
7. **Status 7 (Sequence display)**
    a. Generates and displays a new sequence.
8. **State 8 (Waiting for player to repeat sequence)**
    a. Checks if the active player has played the sequence in the correct order.
    b. If the answer is correct, the game continues at the next level (state 7).
    c. In case of an error, the game saves the result and moves on to the next player or ends the game (state 9).
9. **State 9 (Game Over)**
    a. Displays the game results for all players.
    b. Saves results to global memory.
    c. Allows you to return to the main menu (state 1) by pressing the BT4 button.
10. **Status 10 (Credits)**
    a. Displays information about the project authors and university.
    b. Allows you to return to the main menu (state 1) by pressing the BT4 button.

State transition diagram:



Example of changing the game state:

```
if(current_sequence_check_index >= sequence_length) {
      current_sequence_check_index = 0;
      game_state = 7;
      game_system();
}
```

Sequence generation and gameplay

The light sequence is randomly generated and its length depends on the current game level. Features:

- `generate_sequence(uint8_t level)`: generates a random sequence.
- `play_sequence(uint32_t base_delay)`: plays the generated sequence using LEDs.
- `display_replay_sequence()`: Asks the player to play the sequence.

Example of generating and playing a sequence:

```
// Generating a new sequence
current_sequence_check_index = 0;
current_level += 1;
generate_sequence(current_level);

// Displaying the sequence using LEDs
```

```
uint32_tbase_delay = 1000 - (current_level * 100);  //
Reducing delay with each level
play_sequence(base_delay);
```

UART Integration

UART is used to communicate with the player, including entering player names. The received data is processed in the line_append() function and the results are displayed on the LCD screen.

Ranking support

The ranking system allows you to save and display player results, dividing them into two levels: global ranking and game ranking. This allows you to track both the results from a specific game and the best player achievements saved in the global memory.

1. **Game Ranking**
    a. Contains the results of the current game, sorted by number of points scored.
    b. It is temporary and is displayed after the game ends.

    Display example:

```
for(uint8_ti = 0; and < player_count; i++) {
  wchar_tplayer_label[20];
  swprintf(player_label,sizeof(player_label)
  /sizeof(wchar_t),
  L"Player %u:", and + 1);
  hagl_put_text(display, player_label, 10, 40 + (i *
  20), WHITE, font6x9);
}
```

2. **Global ranking**
    a. Stores the high scores of each player.
    b. After each game, the player's best score from the game ranking is compared to their global record. If it is higher, the global ranking is updated.

    Update example:

```
update_global_score(player_names[current_player],
current_level - 1);
```

Function Description

HAL layer for display

Core/Inc/hagl_hal_color.h

The file hagl_hal_color.h defines the hagl_color_t data type, which is used to represent colors in a 16-bit color space.

Core/Inc/hagl_hal.c

The hagl_hal.c file contains the hagl_hal_init function, which initializes the HAGL library backend. It sets the display dimensions, color depth, and points to the function responsible for drawing individual pixels.

Core/Inc/hagl_hal.h

The hagl_hal.h file contains definitions for the display size (DISPLAY_WIDTH, DISPLAY_HEIGHT) and color depth (DISPLAY_DEPTH). These parameters are consistent with the resolution and specifications of the LCD display.

Core/Inc/sdkconfig.h

The sdkconfig.h file is currently a fallback. It only contains the #pragma once directive, which prevents the same file from being included multiple times during compilation.

Core/Src/lcd.c

The lcd.c file contains all the functions supporting the ST7735 display. The lcd_init function initializes the display by sending the appropriate configuration commands to it (e.g. turning on the ST7735S_DISPON display or setting the ST7735S_COLMOD color mode). The lcd_cmd function sends single commands to the display, while lcd_data sends data.

The lcd_set_window function defines the working area on the display where pixels will be drawn. The lcd_put_pixel function writes the color of a specific pixel to the frame buffer, and the lcd_copy transfers the contents of that buffer to the screen using SPI.

Game system

Core/Src/main.c

The following section describes in detail all implemented methods and their functions in the game system. Each method has been designed in a modular way, which allows for clear management of game logic, peripheral support (LCD, LED, UART) and user interaction.

1. `line_append(uint8_t value)`

Handles UART input in text mode. Adds received characters to line_buffer, terminates line upon receiving newline character (\r or \n) and processes contents depending on current game_state. If buffer is full, resets it to prevent overflow.

2. `HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)`
   Interrupt handler for UART. Receives characters in interrupt mode, passes them to the line_append() method, and re-enables data reception.
3. `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
   Supports external interrupts from buttons. Responds to various GPIO pins by changing the game state (game_state), displaying the appropriate menu or guiding the user to the next steps.
4. `generate_sequence(uint8_t level)`
   Generates a sequence of colors with length depending on the game level. Each color is represented by a number from 0 to 3.
5. `play_sequence(uint32_t base_delay)`
   Displays the memorized sequence using LEDs. The LEDs light up and go out at a set rate, which gets shorter with each level.
6. `game_system()`
   Main game logic. Generates a new sequence, displays it via LED, and then sets the game state to the player's sequence playback mode.
7. `all_leds_blink(uint8_t times, uint32_t delay_ms)`
   Flashes all LEDs the specified number of times with the specified delay.
8. `all_leds_on()`
   Turns on all LEDs.
9. `all_leds_on_but(int num)`
   Turns on all LEDs except the one indicated (1-4).
10. `all_leds_off()`
    Turns off all LEDs.
11. `all_leds_off_but(int num)`
    Turns off all LEDs except the one indicated (1-4).
12. `animate_circles()`
    Displays an animation of growing circles on the LCD screen. Each circle is a different color and appears in one of the corners of the screen.
13. `display_menu()`
    Displays the main game menu with options: start the game, global ranking, and project information (Credits).
14. `display_player_count_selection()`
    Displays the player selection screen. Each selection is represented by a colored circle and the number of players.
15. `display_player_count_selected()`
    Informs the user of the selected number of players and prepares them to enter usernames.

16. `display_player_introduction()`
    Displays a screen for entering user names. Names are assigned via UART.
17. `display_players_turn_text()`
    Tells you the order of the player whose turn is currently active.
18. `display_remember_sequence()`
    Displays a message about the player memorizing the sequence.
19. `display_replay_sequence()`
    Displays a screen that prompts the player to play the memorized sequence. It also shows the game level and the sequence step.
20. `display_wrong()`
    Informs about an incorrectly played sequence. Displays a message about completed levels, pauses the game for the current player and moves on to the next or to the summary.
21. `update_global_score(const char *username, uint8_t score)`
    Updates the global player ranking. If the player's score is higher than the previous one, it is updated. New players are added to the ranking.
22. `update_score(uint8_t score)`
    Records the current player's score.
23. `display_global_ranking()`
    Displays a global player ranking, sorting scores in descending order.
24. `display_game_ranking()`
    Displays a leaderboard for the current game, sorting players by points scored.
25. `display_credits()`
    Displays information about the project creators and university.

Conclusions

The implementation of the project allowed for gaining and deepening knowledge in the field of electronic circuit design and microcontroller programming. Creating the game "Simon Says" using the STM32 NUCLEO-L476RG platform required the integration of various hardware and software components, which resulted in the development of skills in the field of:

1. **Design of electronic systems**
   Designing a PCB that connects the microcontroller to the LCD display, LEDs, and buttons was a key part of the project. It enabled practical application of knowledge about digital circuits, path design, and interference minimization.

2. **Embedded systems programming**
   The implementation of the code in C language allowed familiarization with the operation of SPI, GPIO and UART interfaces. The game state mechanism

(game_state) allowed for effective management of the game logic, and the use of GPIO interrupts contributed to fast and responsive button operation.

3. **Development of interactive applications**
   Integration of LCD display and dynamically generated animations and messages allowed for creation of intuitive user interface. Game became more engaging thanks to visual and light effects, which increased interactivity.

4. **Teamwork and project organization**
   The project required planning skills, task division, and testing of both the hardware system and the code. Conducting tests in a laboratory environment allowed for the elimination of errors and optimization of the solution.

5. **Solving practical problems**
   During the project, difficulties were encountered, such as synchronizing the LCD display with the game system and minimizing interference in the PCB layout. All problems were successfully solved thanks to an iterative approach and testing.

In summary, the project not only met the functional assumptions, but also allowed to gain valuable experience in the field of microcontrollers, PCB design and creation of interactive embedded applications. The created game "Simon Says" combines educational and entertainment elements, at the same time showing practical application of theory acquired during classes.