Politechnika Białostocka

Wydział Informatyki

Marcin Kondrat
nr albumu: 115530

Jowita Ochrymiuk
nr albumu: 115538

Projekt zaliczeniowy

Realizacja projektu gry „Simon Says” na mikrokontrolerze STM32 NUCLEO-L476RG

Przedmiot: Architektura Komputerów
Prowadzący: dr inż. Mirosław Omieljanowicz

Białystok 2025

Spis treści

Wstęp

Celem projektu było zaprojektowanie i zaimplementowanie gry „Simon Says” na platformie mikrokontrolera STM32 NUCLEO-L476RG. Gra ta rozwija pamięć oraz koncentrację graczy, wymagając odtwarzania zaprezentowanych sekwencji świetlnych w odpowiedniej kolejności.

Projekt został zrealizowany w ramach przedmiotu „Architektura Komputerów” na Politechnice Białostockiej jako praktyczne zastosowanie wiedzy zdobytej w trakcie zajęć. W ramach realizacji projektu wykorzystano platformę NUCLEO-L476RG, która została dostosowana do obsługi gry oraz rozszerzona o dodatkowe elementy sprzętowe, takie jak zaprojektowana płytka PCB, aby pogłębić praktyczne umiejętności projektowe.

Realizacja obejmowała dwa kluczowe aspekty:

- Projektowanie układu elektrycznego i płytki PCB, umożliwiającej połączenie mikrokontrolera z peryferiami, takimi jak wyświetlacz LCD, diody LED oraz przyciski tact switch.
- Programowanie systemu gry w języku C, wraz z implementacją mechanizmów generowania sekwencji, obsługi wieloosobowej rozgrywki oraz wizualizacji wyniku na ekranie wyświetlacza.

Projekt opiera się na platformie STM32L476RG, która została wybrana ze względu na jej:

- Niskie zużycie energii.
- Szerokie możliwości konfiguracyjne GPIO, SPI oraz UART.
- Dostępność platformy NUCLEO, ułatwiającej implementację rozwiązań mikrokontrolerowych.

Zakres pracy obejmował projektowanie i wdrażanie rozwiązań sprzętowych oraz programowych, w tym:

- Projektowanie płytki PCB.
- Integrację komponentów, takich jak wyświetlacz LCD ST7735, diody LED oraz przyciski.
- Implementację gry z możliwością obsługi wielu graczy i zapamiętywania wyników.

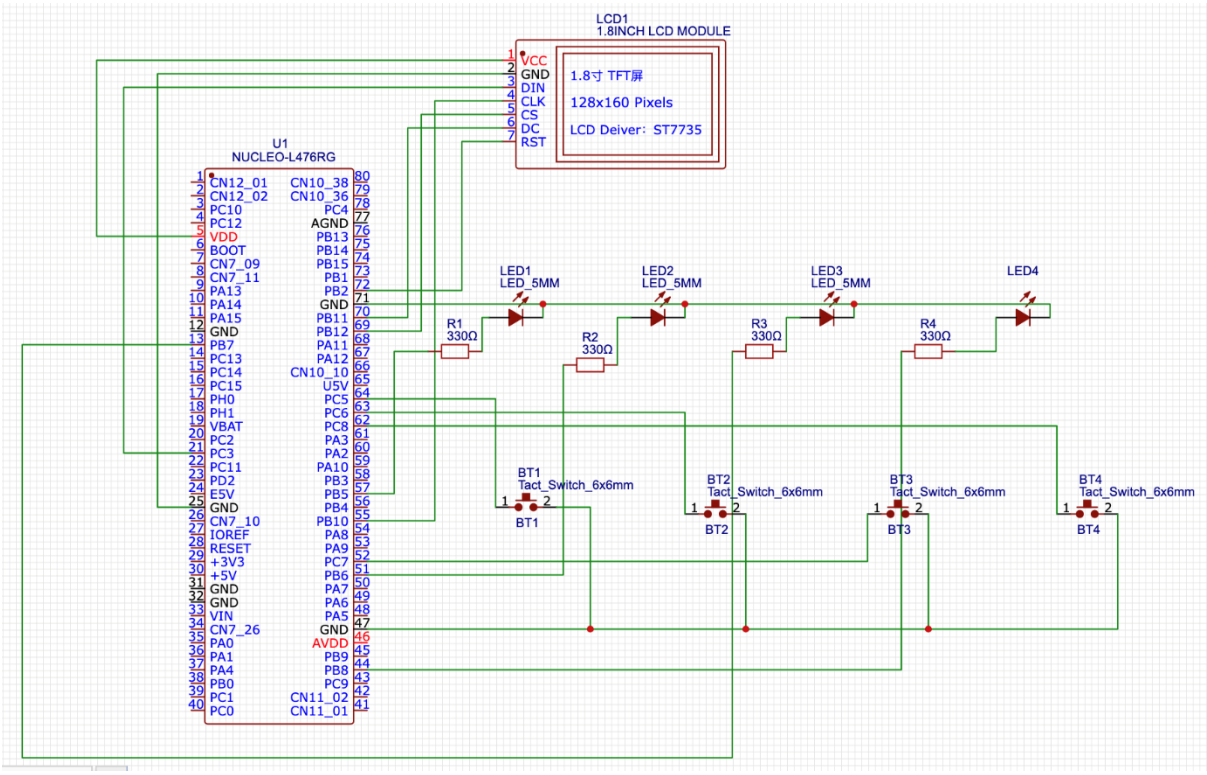
Celem było stworzenie interaktywnej i dynamicznej gry, która łączy elementy rozrywki z praktycznym wykorzystaniem wiedzy o mikrokontrolerach, programowaniu oraz projektowaniu układów elektronicznych.

Projekt układu

W ramach projektu zaprojektowano układ elektryczny oraz zamówiono dedykowaną płytkę PCB, która integruje mikrokontroler STM32 NUCLEO-L476RG z peryferiami, takimi

jak wyświetlacz LCD ST7735, cztery diody LED oraz cztery przyciski typu tact switch. Płytka została zaprojektowana z myślą o łatwości montażu, minimalizacji zakłóceń oraz zapewnieniu stabilności działania systemu.

Układ elektroniczny



Układ wykorzystuje następujące połączenia:

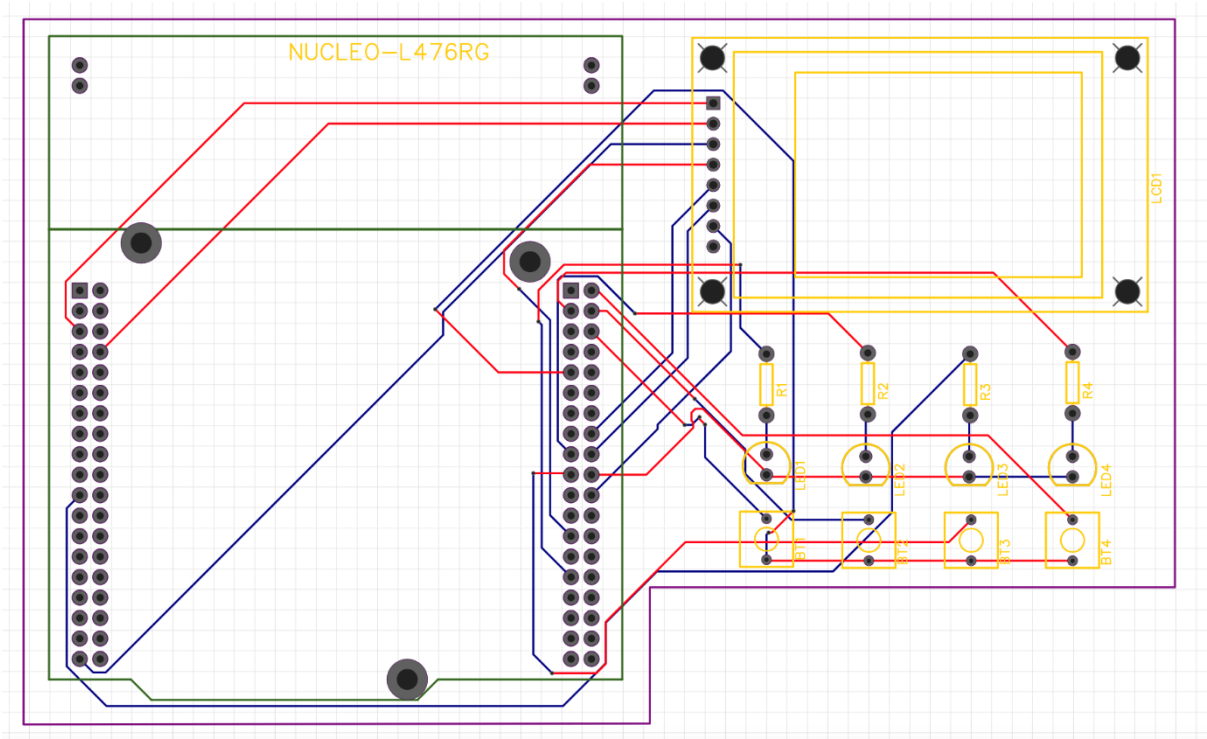
1. **Wyświetlacz LCD ST7735**
    a. **VCC (zasilanie):** podłączone do VDD mikrokontrolera.
    b. **GND:** połączone z masą systemu.
    c. **DIN:** podłączone do pinu *PC3 (interfejs SPI)*.
    d. **CLK:** podłączone do pinu *PB10 (zegar SPI)*.
    e. **CS:** podłączone do pinu *PB12 (wybór urządzenia SPI)*.
    f. **DC:** podłączone do pinu *PB11 (wybór trybu danych/komendy)*.
    g. **RST:** podłączone do pinu *PB2 (reset wyświetlacza)*.
    h. **BL:** nie zostało podłączone (*kontrola jasności; moduł ma wbudowany rezystor podciągający tę linię do zasilania, więc wyświetlacz będzie świecił z maksymalną jasnością*).
2. **Diody LED**
    a. **LED1:** anoda połączona do *PB5*, katoda do *GND*.
    b. **LED2:** anoda połączona do *PB6*, katoda do *GND*.
    c. **LED3:** anoda połączona do *PB6*, katoda do *GND*.
    d. **LED4:** anoda połączona do *PB7*, katoda do *GND*.
3. **Przyciski tact switch**

a. **BT1:** jeden styk do *PC5*, drugi do *GND*.
b. **BT2:** jeden styk do *PC6*, drugi do *GND*.
c. **BT3:** jeden styk do *PC7*, drugi do *GND*.
d. **BT4:** jeden styk do *PC8*, drugi do *GND*.

Płytka PCB

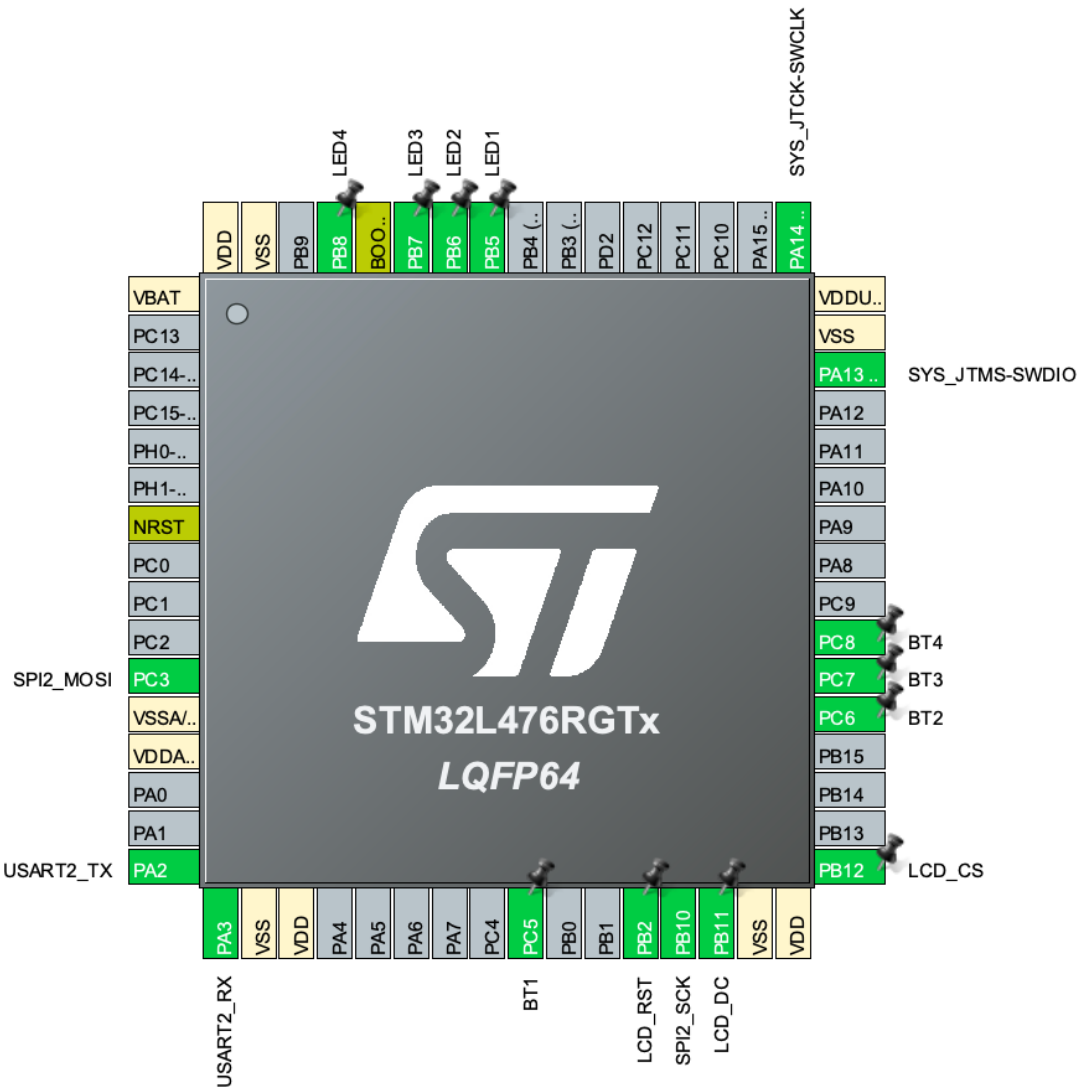
Układ został zaprojektowany z uwzględnieniem:

- Optymalizacji ścieżek na płytce PCB w celu minimalizacji zakłóceń i poprawy integralności sygnału.
- Rozmieszczenia komponentów, aby zapewnić łatwy dostęp do przycisków oraz wyraźną widoczność wyświetlacza.
- Minimalizacji połączeń przewodowych, aby uprościć proces montażu i testowania.



Obsługa peryferiów mikrokontrolera

Do realizacji projektu niezbędne było skonfigurowanie szeregu peryferiów, takich jak GPIO, NVIC, UART oraz SPI. Poniżej przedstawiono szczegóły konfiguracji kluczowych elementów, które umożliwiły poprawne działanie systemu.



GPIO – Przyciski i Diody LED

1. **Przyciski:** Przyciski typu tact switch zostały skonfigurowane jako wejścia GPIO z aktywnymi rezystorami podciągającymi (*pull-up*). Dzięki temu uniknięto nieokreślonych stanów na wejściu, gdy przyciski nie są wciśnięte. Dodatkowo, wykorzystano przerwania do obsługi przycisków, co pozwala na natychmiastowe reagowanie na ich naciśnięcie bez konieczności ciągłego odpytywania stanu wejścia.
2. **Diody LED:** Diody LED zostały skonfigurowane jako wyjścia GPIO z podstawowymi ustawieniami. Sterowanie diodami polega na ustawianiu stanu wysokiego (*HIGH*) na odpowiednich pinach, co powoduje ich zaświecenie.

Przerwania (NVIC)

W projekcie skonfigurowano dwa kluczowe źródła przerwań: USART2 oraz GPIO (przyciski).

- **USART2:** Nadano *priorytet 12*, co zapewnia, że przerwania związane z komunikacją UART mają niższy priorytet niż przerwania GPIO.
- **GPIO:** Nadano *priorytet 8*, aby obsługa przycisków była szybsza i bardziej responsywna. Dzięki temu reakcja na wciśnięcie przycisku odbywa się niemal natychmiastowo.

Moduł SYS – Serial Wire Debug (SWD)

Moduł UART (USART2) został skonfigurowany w trybie asynchronicznym, co umożliwia przesyłanie danych między mikrokontrolerem a komputerem PC w celu wprowadzania nazw graczy oraz debugowania.

Główne parametry konfiguracji:

- **Prędkość transmisji:** 115200 baud.
- **Liczba bitów danych:** 8.
- **Parzystość:** brak.
- **Liczba bitów stopu:** 1.

SPI – Komunikacja z Wyświetlaczem LCD

Moduł SPI został skonfigurowany w celu komunikacji z wyświetlaczem LCD ST7735. SPI umożliwia szybki i niezawodny transfer danych, co jest kluczowe przy wyświetlaniu dynamicznych informacji, takich jak menu gry czy animacje.

Główne parametry konfiguracji:

- **Tryb pracy:** Master.
- **Format danych:** 8 bitów.
- **Częstotliwość zegara SPI:** 10 MHz.
- **Faza i polaryzacja zegara:** CPOL = 0, CPHA = 0.
- **Preskaler:** 8.

Ze względu na to, że płytka mikrokontrolera pracuje z zegarem systemowym *80 MHz*, a wyświetlacz LCD ST7735 obsługuje maksymalną częstotliwość zegara SPI równą *15 MHz*, wartość preskalera obliczamy za pomocą poniższego wzoru:

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

gdzie:

- F_{sys} to częstotliwość zegara systemowego płytki (*80 MHz*),
- F_{SPI_max} to maksymalna częstotliwość zegara SPI, którą wyświetlacz LCD jest w stanie obsłużyć (*15 MHz*).

Podstawiając wartości:

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

Ponieważ STM32CubeIDE obsługuje tylko preskalery będące potęgami liczby 2, najbliższą większą mocą 2 jest 8. W związku z tym **ustawiamy preskaler na wartość 8**, co daje **częstotliwość SPI** równą:

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

Taktowanie mikrokontrolera

W ramach projektu skonfigurowano mikrokontroler STM32 NUCLEO-L476RG do pracy z maksymalną częstotliwością 80 MHz. Dzięki temu zwiększono szybkość działania wyświetlacza LCD oraz całego systemu, co znacząco wpłynęło na płynność animacji i responsywność gry.

W konfiguracji wykorzystano:

- **MSI** (*Multi-Speed Internal*) jako źródło bazowe o częstotliwości 4 MHz.
- **PLL** (*Phase-Locked Loop*) do mnożenia częstotliwości sygnału do 80 MHz.

Taktowanie zostało dostosowane za pomocą narzędzia STM32CubeMX, w którym ustawiono odpowiednie dzielniki oraz źródło sygnału. Dzięki zastosowaniu tej konfiguracji, system działa z maksymalną wydajnością, jednocześnie zachowując stabilność i precyzję sygnału zegarowego.

Rozwiązania techniczne

Kod systemu gry został napisany w języku C z wykorzystaniem środowiska STM32CubeIDE. Implementacja obejmuje obsługę wyświetlacza LCD, diod LED, przycisków tact switch, a także logikę gry opartą na mechanizmie stanów (`game states`).

Obsługa wyświetlacza LCD ST7735

Wyświetlacz LCD pełni kluczową rolę w grze, prezentując informacje, takie jak menu, nazwy graczy, wyniki oraz komunikaty. Implementacja obejmuje:

- **Inicjalizację wyświetlacza** za pomocą funkcji `lcd_init()`.
- **Obsługę grafiki** z wykorzystaniem biblioteki HAGL. Biblioteka została dostosowana do współpracy ze sterownikiem wyświetlacza ST7735, co umożliwiło wygodne rysowanie tekstu, kształtów, animacji oraz innych elementów graficznych. Dodano własne funkcje inicjalizacyjne i sterujące, które zapewniły kompatybilność z niskopoziomowym protokołem SPI używanym przez sterownik.

- **Wyświetlanie komunikatów**, takich jak menu wyboru liczby graczy, wyniki gry, aktualna tura gracza oraz wskazówki.

Wykorzystanie biblioteki HAGL

Biblioteka HAGL została użyta do uproszczenia obsługi wyświetlacza poprzez wprowadzenie funkcji wyższego poziomu, takich jak:

- `void hagl_put_text(void *display, const wchar_t *text, uint16_t x, uint16_t y, uint16_t color, const font_t *font)` – wyświetlanie tekstu na ekranie.
- `void hagl_fill_circle(void *display, uint16_t x, uint16_t y, uint16_t radius, uint16_t color)` – rysowanie wypełnionych kółek.

  - `display` – wskaźnik do struktury sterującej wyświetlaczem (*void*).
  - `text` – tekst do wyświetlenia (*const wchar_t*).
  - `x` – współrzędna pozioma (*uint16_t*).
  - `y` – współrzędna pionowa (*uint16_t*).
  - `radius` – promień koła (*uint16_t*).
  - `color` – kolor w formacie RGB565 (*uint16_t*).
  - `font` – wskaźnik do struktury czcionki (*const font_t)*.

Przykładowy fragment kodu:

```
// Wyświetlenie nazwy użytkownika
hagl_put_text(display, username_label, 10, 40 + (current_player * 20), WHITE, font6x9);
// Zaaktualizowanie zawartości wyświetlacza
lcd_copy();
```

Dostowanie biblioteki HAGL

Dostosowanie obejmowało:

- **Wprowadzenie funkcji inicjalizacji i komunikacji SPI** ze sterownikiem ST7735.
- **Optymalizację prędkości renderowania** poprzez zwiększenie częstotliwości zegara SPI oraz wykorzystanie bufora w celu minimalizacji liczby operacji zapisu na wyświetlacz.
- **Dodanie wsparcia dla palety kolorów 16-bitowych (RGB565)**, używanej przez sterownik ST7735.

Dzięki tej integracji wyświetlacz działa z wysoką wydajnością, co pozwala na płynne animacje i dynamiczne aktualizowanie ekranu.

Obsługa diod LED

Diody LED sygnalizują sekwencję gry oraz reakcje gracza. Każda dioda jest sterowana oddzielnym pinem GPIO. Funkcje obsługi diod:

- `all_leds_on()` : włącza wszystkie diody.
- `all_leds_off()` : wyłącza wszystkie diody.
- `all_leds_blink(uint8_t times, uint32_t delay_ms)` : włącza i wyłącza wszystkie diody z określonym opóźnieniem, realizując efekt migania.
- `all_leds_off_but(int num)` : wyłącza wszystkie diody oprócz wskazanej.
- `all_leds_on_but(int num)` : włącza wszystkie diody oprócz wskazanej.

    - `times` – liczba mignięć diod (*uint8_t*).
    - `delay_ms` – czas w milisekundach między włączeniem i wyłączeniem diod (*uint32_t*).
    - `num` – numer diody (1-4; *int*).

Przykład włączenia określonej diody:

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

Obsługa przycisków tact switch

Przyciski odpowiadają za nawigację w menu oraz wybór opcji. Wykorzystano przerwania GPIO, które reagują na wciśnięcie przycisku i wykonują odpowiednią akcję w zależności od aktualnego stanu gry.

Przykład obsługi przycisku w funkcji `HAL_GPIO_EXTI_Callback()`:

```
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

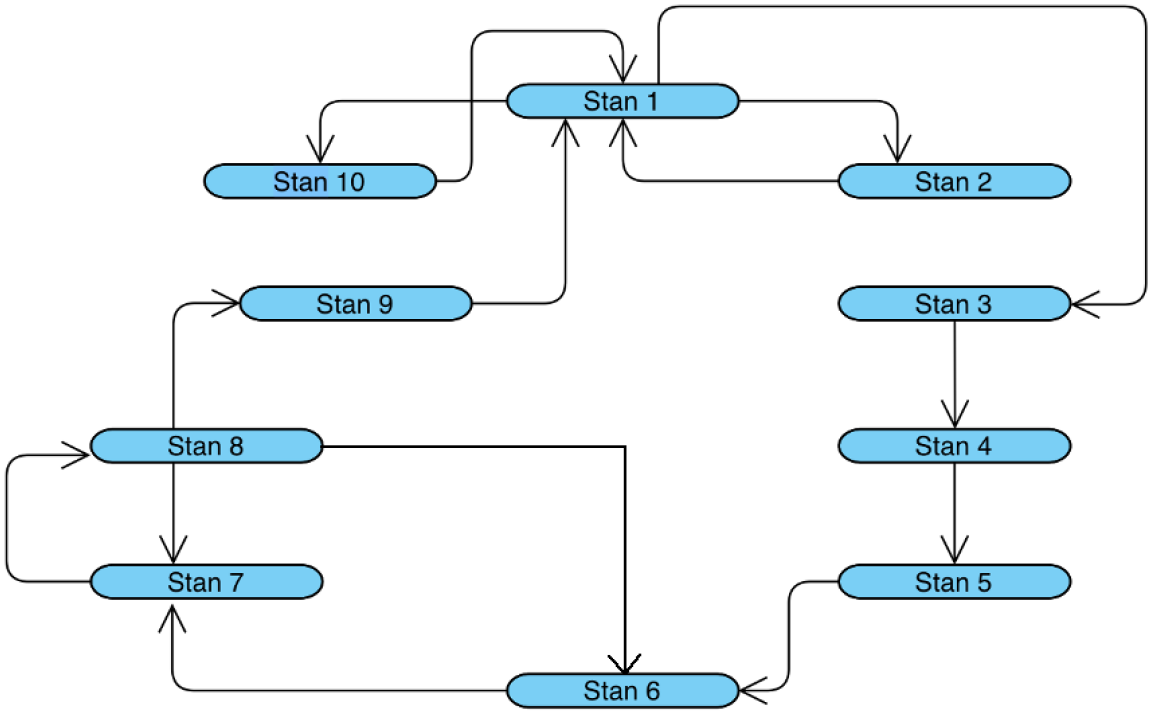
Mechanizm stanów gry (`game_state`) pozwala na płynne przechodzenie między różnymi etapami gry, takimi jak menu, wybór liczby graczy, prezentacja sekwencji, rozgrywka oraz wyświetlanie wyników.

1. **Stan 1 (Menu)**
    a. Wyświetla menu główne gry z trzema opcjami do wyboru:
        i. *Start Game* (przejście do *stanu 3*).
        ii. *Global Ranking* (przejście do *stanu 2*).
        iii. *Credits* (przejście do *stanu 10*).
    b. Wybór dokonywany jest za pomocą przycisków *BT1-BT3*.
2. **Stan 2 (Ranking globalny)**

a. Wyświetla listę graczy oraz ich najlepsze wyniki zapisane w pamięci globalnej.
b. Gracz może wrócić do menu głównego (*stan 1*) po wciśnięciu przycisku *BT4*.

3. **Stan 3 (Wybór liczby graczy)**
   a. Umożliwia wybór liczby uczestników gry (od 1 do 4).
   b. Wybór liczby graczy odbywa się za pomocą przycisków *BT1-BT4*, co prowadzi do przejścia do *stanu 4*.
4. **Stan 4 (Potwierdzenie wyboru liczby graczy)**
   a. Wyświetla informację o wybranej liczbie graczy.
   b. Po krótkim czasie automatycznie przechodzi do *stanu 5*, gdzie definiowane są nazwy graczy.
5. **Stan 5 (Wprowadzenie nazw graczy)**
   a. Umożliwia wprowadzanie nazw graczy za pomocą interfejsu UART.
   b. Po wprowadzeniu nazw wszystkich graczy gra automatycznie przechodzi do *stanu 6*.
6. **Stan 6 (Oczekiwanie na potwierdzenie gotowości gracza)**
   a. Wyświetla ekran z prośbą o potwierdzenie gotowości gracza.
   b. Po kliknięciu przycisku *BT4*, gra rozpoczyna się (*stan 7*).
7. **Stan 7 (Wyświetlanie sekwencji)**
   a. Generuje i wyświetla nową sekwencję.
8. **Stan 8 (Oczekiwanie na powtórzenie sekwencji przez gracza)**
   a. Sprawdza, czy aktywny gracz odtwarzył sekwencję w odpowiedniej kolejności.
   b. W przypadku poprawnej odpowiedzi gra kontynuuje na wyższym poziomie (*stan 7*).
   c. W przypadku błędu gra zapisuje wynik i przechodzi do następnego gracza lub kończy grę (*stan 9*).
9. **Stan 9 (Koniec gry)**
   a. Wyświetla wyniki rozgrywki dla wszystkich graczy.
   b. Zapisuje wyniki w pamięci globalnej.
   c. Umożliwia powrót do menu głównego (*stan 1*) po wciśnięciu przycisku *BT4*.
10. **Stan 10 (Credits)**
    a. Wyświetla informacje o autorach projektu oraz uczelni.
    b. Umożliwia powrót do menu głównego (*stan 1*) po wciśnięciu przycisku *BT4*.

Schemat przejść stanów:



Przykład zmiany stanu gry:

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

Generowanie sekwencji i rozgrywka

Sekwencja świetlna jest generowana losowo, a jej długość zależy od aktualnego poziomu gry. Funkcje:

- `generate_sequence(uint8_t level)`: generuje losową sekwencję.
- `play_sequence(uint32_t base_delay)`: odtwarza wygenerowaną sekwencję za pomocą diod LED.
- `display_replay_sequence()`: prosi gracza o odtworzenie sekwencji.

Przykład generowania i odtwarzania sekwencji:

```
// Generowanie nowej sekwencji
current_sequence_check_index = 0;
current_level += 1;
generate_sequence(current_level);

// Wyświetlenie sekwencji za pomocą LED-ów
uint32_t base_delay = 1000 - (current_level * 100);  //
Skracanie opóźnienia z każdym poziomem
play_sequence(base_delay);
```

Integracja UART

UART jest używany do komunikacji z graczem, m.in. do wprowadzania nazw graczy. Odbierane dane są przetwarzane w funkcji `line_append()`, a wyniki wyświetlane na ekranie LCD.

Obsługa rankingów

System rankingów pozwala na zapisywanie i wyświetlanie wyników graczy, dzieląc je na dwa poziomy: ranking globalny oraz ranking gry. Dzięki temu można śledzić zarówno wyniki z konkretnej rozgrywki, jak i najlepsze osiągnięcia graczy zapisane w pamięci globalnej.

1. **Ranking gry**
    a. Zawiera wyniki z bieżącej rozgrywki, sortowane według liczby zdobytych punktów.
    b. Jest tymczasowy i wyświetlany po zakończeniu gry.

    Przykład wyświetlania:

```
for (uint8_t i = 0; i < player_count; i++) {
  wchar_t player_label[20];
  swprintf(player_label, sizeof(player_label) /
  sizeof(wchar_t),
  L"Player %u:", i + 1);
  hagl_put_text(display, player_label, 10, 40 + (i *
  20), WHITE, font6x9);
}
```

2. **Ranking globalny**
    a. Przechowuje najlepsze wyniki każdego gracza.
    b. Po każdej grze najlepszy wynik gracza z rankingu gry jest porównywany z jego rekordem globalnym. Jeśli jest wyższy, ranking globalny zostaje zaktualizowany.

    Przykład aktualizacji:

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

Opis funkcji

Warstwa HAL dla wyświetlacza

Core/Inc/hagl_hal_color.h

Plik `hagl_hal_color.h` definiuje typ danych `hagl_color_t`, który jest używany do reprezentowania kolorów w 16-bitowej przestrzeni kolorów.

Core/Inc/hagl_hal.c

Plik `hagl_hal.c` zawiera funkcję `hagl_hal_init`, która inicjalizuje backend biblioteki HAGL. Ustawia ona wymiary wyświetlacza, głębię kolorów oraz wskazuje funkcję odpowiedzialną za rysowanie pojedynczych pikseli.

Core/Inc/hagl_hal.h

W pliku `hagl_hal.h` znajdują się definicje rozmiarów wyświetlacza (`DISPLAY_WIDTH`, `DISPLAY_HEIGHT`) oraz głębi kolorów (`DISPLAY_DEPTH`). Te parametry są zgodne z rozdzielczością i specyfikacją wyświetlacza LCD.

Core/Inc/sdkconfig.h

Plik `sdkconfig.h` obecnie pełni funkcję rezerwową. Zawiera jedynie dyrektywę `#pragma once`, która zapobiega wielokrotnemu dołączaniu tego samego pliku w trakcie kompilacji.

Core/Src/lcd.c

W pliku `lcd.c` znajdują się wszystkie funkcje obsługujące wyświetlacz ST7735. Funkcja `lcd_init` inicjalizuje wyświetlacz, przesyłając do niego odpowiednie komendy konfiguracyjne (np. włączenie wyświetlacza `ST7735S_DISPON` czy ustawienia trybu koloru `ST7735S_COLMOD`). Funkcja `lcd_cmd` wysyła pojedyncze komendy do wyświetlacza, natomiast `lcd_data` przesyła dane.

Funkcja `lcd_set_window` definiuje obszar roboczy na wyświetlaczu, w którym będą rysowane piksele. Z kolei `lcd_put_pixel` zapisuje kolor konkretnego piksela do bufora ramki, a `lcd_copy` przesyła zawartość tego bufora na ekran za pomocą SPI.

System gry

Core/Src/main.c

W poniższej sekcji opisano szczegółowo wszystkie zaimplementowane metody oraz ich funkcje w systemie gry. Każda metoda została zaprojektowana w sposób modularny, co pozwala na przejrzyste zarządzanie logiką gry, obsługą urządzeń peryferyjnych (LCD, LED, UART) oraz interakcją z użytkownikiem.

1. `line_append(uint8_t value)`

Obsługuje wejście UART w trybie tekstowym. Dodaje otrzymane znaki do bufora `line_buffer`, kończy linię po otrzymaniu znaku nowej linii (`\r` lub `\n`) i przetwarza zawartość w zależności od aktualnego stanu gry (`game_state`). Jeśli bufor jest pełny, resetuje go, aby zapobiec przepełnieniu.

2. `HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)`
   Funkcja obsługująca przerwania dla UART. Odbiera znaki w trybie przerwań, przesyła je do metody `line_append()` i ponownie włącza odbiór danych.
3. `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
   Obsługuje przerwania zewnętrzne z przycisków. Reaguje na różne piny GPIO, zmieniając stan gry (`game_state`), wyświetlając odpowiednie menu lub wprowadzając użytkownika w kolejne kroki.
4. `generate_sequence(uint8_t level)`
   Generuje sekwencję kolorów o długości zależnej od poziomu gry (`level`). Każdy kolor reprezentowany jest przez liczbę od 0 do 3.
5. `play_sequence(uint32_t base_delay)`
   Wyświetla zapamiętaną sekwencję za pomocą diod LED. Diody zapalają się i gasną w ustalonym tempie, które skraca się z każdym poziomem.
6. `game_system()`
   Główna logika gry. Generuje nową sekwencję, wyświetla ją za pomocą LED, a następnie ustawia stan gry na tryb odtwarzania sekwencji przez gracza.
7. `all_leds_blink(uint8_t times, uint32_t delay_ms)`
   Migocze wszystkimi diodami LED określoną liczbę razy z podanym opóźnieniem.
8. `all_leds_on()`
   Włącza wszystkie diody LED.
9. `all_leds_on_but(int num)`
   Włącza wszystkie diody LED z wyjątkiem wskazanej (1-4).
10. `all_leds_off()`
    Wyłącza wszystkie diody LED.
11. `all_leds_off_but(int num)`
    Wyłącza wszystkie diody LED z wyjątkiem wskazanej (1-4).
12. `animate_circles()`
    Wyświetla animację rosnących kół na ekranie LCD. Każde koło ma inny kolor i pojawia się w jednym z rogów ekranu.
13. `display_menu()`
    Wyświetla główne menu gry z opcjami: rozpoczęcie gry, globalny ranking i informacje o projekcie (*Credits*).
14. `display_player_count_selection()`
    Wyświetla ekran wyboru liczby graczy. Każdy wybór jest reprezentowany przez kolorowe koło i liczbę graczy.
15. `display_player_count_selected()`
    Informuje użytkownika o wybranej liczbie graczy i przygotowuje do wprowadzenia nazw użytkowników.

16. `display_player_introduction()`
    Wyświetla ekran do wprowadzenia nazw użytkowników. Nazwy są przypisywane przez UART.
17. `display_players_turn_text()`
    Informuje o kolejności gracza, którego tura jest aktualnie aktywna.
18. `display_remember_sequence()`
    Wyświetla komunikat o zapamiętywaniu sekwencji przez gracza.
19. `display_replay_sequence()`
    Wyświetla ekran, który zachęca gracza do odtworzenia zapamiętanej sekwencji. Prezentuje również poziom gry oraz krok sekwencji.
20. `display_wrong()`
    Informuje o błędnym odtworzeniu sekwencji. Wyświetla komunikat o ukończonych poziomach, zatrzymuje grę dla aktualnego gracza i przechodzi do kolejnego lub do podsumowania.
21. `update_global_score(const char *username, uint8_t score)`
    Aktualizuje globalny ranking graczy. Jeśli wynik gracza jest wyższy niż dotychczasowy, zostaje zaktualizowany. Nowi gracze są dodawani do rankingu.
22. `update_score(uint8_t score)`
    Zapisuje wynik bieżącego gracza.
23. `display_global_ranking()`
    Wyświetla globalny ranking graczy, sortując wyniki w kolejności malejącej.
24. `display_game_ranking()`
    Wyświetla ranking wyników dla bieżącej gry, sortując graczy według zdobytych punktów.
25. `display_credits()`
    Wyświetla informacje o twórcach projektu oraz uczelni.

Wnioski

Realizacja projektu pozwoliła na zdobycie i pogłębienie wiedzy z zakresu projektowania układów elektronicznych oraz programowania mikrokontrolerów. Stworzenie gry „Simon Says” z wykorzystaniem platformy STM32 NUCLEO-L476RG wymagało integracji różnych komponentów sprzętowych i programowych, co przełożyło się na rozwój umiejętności w zakresie:

1. **Projektowania układów elektronicznych**
   Zaprojektowanie płytki PCB, która łączy mikrokontroler z wyświetlaczem LCD, diodami LED oraz przyciskami, było kluczowym elementem projektu. Umożliwiło to praktyczne zastosowanie wiedzy o układach cyfrowych, projektowaniu ścieżek oraz minimalizacji zakłóceń.

2. **Programowania systemów wbudowanych**
   Implementacja kodu w języku C umożliwiła zapoznanie się z obsługą interfejsów SPI, GPIO oraz UART. Mechanizm stanów gry (`game_state`) pozwolił na efektywne zarządzanie logiką gry, a wykorzystanie przerwań GPIO przyczyniło się do szybkiej i responsywnej obsługi przycisków.

3. **Rozwoju aplikacji interaktywnych**
   Integracja wyświetlacza LCD oraz dynamicznie generowanych animacji i komunikatów pozwoliła na stworzenie intuicyjnego interfejsu użytkownika. Gra stała się bardziej angażująca dzięki wizualnym i świetlnym efektom, które zwiększyły interaktywność.

4. **Pracy zespołowej i organizacji projektu**
   Realizacja projektu wymagała umiejętności planowania, podziału zadań oraz testowania zarówno układu sprzętowego, jak i kodu. Przeprowadzenie testów w środowisku laboratoryjnym pozwoliło na eliminację błędów oraz optymalizację rozwiązania.

5. **Rozwiązywania problemów praktycznych**
   Podczas realizacji projektu napotkano trudności, takie jak synchronizacja działania wyświetlacza LCD z systemem gry czy minimalizacja zakłóceń w układzie PCB. Wszystkie problemy zostały pomyślnie rozwiązane dzięki iteracyjnemu podejściu oraz testom.

Podsumowując, projekt nie tylko spełnił założenia funkcjonalne, ale również pozwolił na zdobycie cennego doświadczenia w dziedzinie mikrokontrolerów, projektowania PCB oraz tworzenia interaktywnych aplikacji wbudowanych. Stworzona gra „Simon Says” łączy elementy edukacyjne i rozrywkowe, jednocześnie ukazując praktyczne zastosowanie teorii zdobytej w trakcie zajęć.