Dokumentacja Projektu		
Przedmiot	Mikroprocesory	POLITECHNIKA BYDGOSKA Wydział Telekomunikacji, Informatyki i Elektrotechniki
Student	Mikołaj Kołodziejski	
Indeks	121192	

Spis treści

1. Specyfikacja Projektu

1.1. Wykorzystane urządzenia

1. płytka - STM32 NUCLEO – F401RE,
2. Listwa LED RGB WS2812 5050 x 8 diod

1.2. Oprogramowanie komunikacji z PC z wykorzystaniem przerwań i buforów kołowych poprzez interfejs asynchroniczny

1.3. Zaprojektowanie i zaimplementowanie protokołu komunikacyjnego pozwalającego na:

- adresowanie ramek,
- przekazywanie dowolnych danych,
- weryfikację poprawności przesyłanych danych z uwzględnieniem ich kolejności.

1.4. Działanie aplikacji

1. Wyświetlanie efektów świetlnych takich jak pulsacyjny
2. Komunikacja płytki z Pierścieniem za pomocą GPIO
3. Przesyłanie danych na diody WS2812B za pomocą DMA i timera PWM Out. Ciągła aktualizacja danych w buforze.
4. Implementacja logiki dla zmiany efektów świetlnych diod WS2812B. Zapewnienie ciągłej pracy aplikacji i możliwość dynamicznych modyfikacji kolorów i efektów.

2. Protokół komunikacyjny

2.1. Wstęp

Komunikacja pomiędzy użytkownikiem, a urządzeniem będzie odbywać się poprzez program terminal, za pomocą którego użytkownik będzie mógł wysyłać odpowiednie komunikaty do urządzenia STM32.

Protokół będzie służył do odbierania poleceń użytkownika, zwracał będzie informacje zwrotne (ramkę), takie jak, czy komenda zawarta w ramce będzie prawidłowa.

Użytkownik będzie wprowadzał dane w terminalu znaki, które będą odbierane przez urządzenie, które będzie przechowywać je w formacie ASCII. Po prawidłowym odebraniu ramki i rozpoznaniu komendy, urządzenie wyśle ramkę zwrotną z zamienionymi adresami nadawcy i odbiorcy oraz ze zmienionymi danymi (w otrzymanej ramce dane to komenda, a w zwracanej, informacje o rozpoznaniu lub nierozpoznaniu komendy).

Obsługa błędów:

- Jeżeli zostanie wysłanych kilka znaków rozpoczęcia ramki pod rząd, to ramka rozpocznie się od ostatniego znaku w powtórzeniu, np. :::::, to ramka rozpocznie się od piątego znaku ‘:’, a reszta zostanie zignorowana.

- Jeżeli zostanie wysłanych kilka znaków końca ramki pod rząd, to ramka zakończy się na pierwszym takim znaku w powtórzeniu, np. ;;;;;, to ramka zakończy się na pierwszym znaku ‘;’, a, reszta zostanie zignorowana.

- Jeżeli ramka zostanie wprowadzona prawidłowo (z prawidłowym adresem odbiorcy i będzie miała prawidłową konstrukcję), a komenda nie zostanie rozpoznana, zostanie zwrócona ramka informująca o wpisaniu błędnej komendy.

- W sytuacji, gdy do urządzenia zostanie wysłana prawidłowa ramka, w środku której znajdą się znaki rozpoczęcia lub zakończenia ramki (np. w danych lub adresach), nie zostanie wysłana informacja zwrotna, co trzeba przyjąć jako błędną ramkę.

- Jeżeli użytkownik poda dłuższą długość komendy, niż wpisana podana komenda będzie posiadała w rzeczywistości oraz gdy jednocześnie zostaną podane dane w ilości większej, niż długość komendy, wtedy urządzenie wykryje błąd i zwróci komunikat o błędnej długości komendy.

2.2. RAMKA PROTOKOŁU

Początek Ramki	Nadawca	Odbiorca	Długość Komendy	Dane	Koniec Ramki
:	000	001	000	[znaki ASCII]	;
0x3A	wszystkie znaki poza 0x3A i 0x3B	wszystkie znaki poza 0x3A i 0x3B	znaki od 0x30 do 0x39	wszystkie znaki poza 0x3A i 0x3B	0x3B
1	3	3	3	0-256	1

- **Maksymalna długość komendy:** Wynosi 256 bajtów (znaków). Jest to maksymalna długość pola "Dane".

- **Minimalna długość komendy:** Najkrótsza komenda musi zawierać co najmniej jeden znak. Zatem, minimalna niezerowa długość komendy, to 001. Ilość znaków w danych (czyli w polu "Dane") nie może wynosić 0, jeżeli długość komendy jest różna od 0.

- **Minimalna długość ramki:** Minimalna długość całej ramki (bez znaków początku i końca) wynosi 8 znaków ASCII: 3 (Nadawca) + 3 (Odbiorca) + 3 (Długość) = 9. Znak początku i końca to dodatkowe dwa znaki. *Jednakże*, jeśli pole "Długość Komendy" ma wartość "000", cała ramka jest traktowana jako *pusta* i jest ignorowana (z wyjątkiem wysłania komunikatu o błędzie).

- **Adres urządzenia (płytki STM32):** Na potrzeby tego projektu adres płytki STM32 to "MTW" (zamiast numerycznego "001" czy innego). To ułatwia identyfikację urządzenia. Adres komputera PC to "000".

- **Budowa komendy:** Komenda (zawarta w polu "Dane") składa się z *nazwy komendy*. W tej uproszczonej wersji protokołu, komendy *nie mają* dodatkowych parametrów w polu "Dane". Przykładowe komendy to: SNAKE, PULSE, BLINK, SET. Komenda SET ma parametry oddzielone przecinkami: SET,R,G,B. Gdzie R,G,B to trzypozycyjne wartości od 000 do 255.

Dozwolone znaki:

- **Dane:** Dowolne znaki ASCII *z wyjątkiem* znaków początku ramki (':', 0x3A) i końca ramki (';', 0x3B).
- **Nadawca i Odbiorca:** Dowolne znaki ASCII *z wyjątkiem* znaków początku ramki (':', 0x3A) i końca ramki (';', 0x3B). Zalecane jest używanie adresów numerycznych ("000" - "999") lub symbolicznych jak "MTW".
- **Długość Komendy:** Wyłącznie cyfry ASCII ('0' - '9', 0x30 - 0x39).

2.3. Tabela Rozkazów

Komenda	Długość w ramce [ASCII]	Opis	Przykład Ramki	Przykład ramki zwrotnej
SNAKE	**005**	Uruchamia efekt "węża" świetlnego z domyślnymi parametrami (kolor, prędkość).	:000MTW005SNAKE;	:MTW000002OK; lub :MTW000005ERROR; w przypadku błędu
PULSE	**005**	Uruchamia efekt pulsowania światła z domyślnymi parametrami.	:000MTW005PULSE;	:MTW000002OK; lub :MTW000005ERROR;
BLINK	**005**	Uruchamia efekt migania diod LED z domyślnymi parametrami.	:000MTW005BLINK;	:MTW000002OK; lub :MTW000005ERROR
SET	**Zmienna**	Ustawia statyczny kolor dla wszystkich diod LED. Przyjmuje parametry R, G, B (czerwony, zielony, niebieski) jako trzy 3-znakowe liczby ASCII (000-255).	:000MTW012SET,255,000,000; (czerwony)	:MTW000002OK; lub :MTW000005ERROR; (np. ERR_RGB dla zlych wartosci RGB)
STATUS	**006**	Żądanie statusu urządzenia. STM32 może zwrócić informacje o aktualnym trybie, kolorze, itp. (szczegóły odpowiedzi do zdefiniowania).	:000MTW006STATUS;	:MTW000xxx...; (gdzie xxx... to dane statusu, np. 008RUN,SNAKE
RESET	**005**	Resetuje urządzenie (STM32). Może wyłączyć wszystkie efekty i przywrócić domyślne ustawienia.	:000MTW005RESET;	:MTW000002OK;
PING	**004**	Sprawdza, czy urządzenie jest dostępne.	:000MTW004PING;	:MTW000004PONG;

2.4. Tabela błędów

Kod Błędu	Długość w ramce [ASCII]	Opis	Przyczyna
OK	**002**	Operacja zakończona sukcesem.	Komenda została poprawnie odebrana, zinterpretowana i wykonana.
ERROR	**005**	Ogólny błąd. Używany, gdy wystąpił błąd, ale nie można go zakwalifikować do bardziej szczegółowej kategorii. Zaleca się unikanie tego kodu na rzecz bardziej precyzyjnych kodów błędów.	Niesprecyzowany błąd.
ERR_CMD	**007**	Nieznana komenda. STM32 nie rozpoznało nazwy komendy przesłanej w polu "Dane".	W polu "Dane" ramki z komendą znajduje się ciąg znaków, który nie odpowiada żadnej zdefiniowanej nazwie komendy (np. literówka, nieprawidłowa komenda).
ERR_ARG	**007**	Nieprawidłowe argumenty komendy. Używane, gdy komenda (np. SET) przyjmuje argumenty, a podane argumenty są nieprawidłowe (zła liczba argumentów, nieprawidłowy format, brakujące argumenty).	Dla komendy SET: mniej lub więcej niż 3 argumenty (R, G, B), argumenty nie są oddzielone przecinkami, argumenty nie są 3-znakowymi liczbami ASCII.
ERR_RGB	**007**	Nieprawidłowe wartości RGB. Specyficzny błąd dla komendy SET, gdy wartości R, G lub B są poza zakresem 0-255.	Wartość R, G lub B w komendzie SET jest mniejsza niż 0 lub większa niż 255, lub nie jest liczbą.
ERR_LEN	**007**	Nieprawidłowa długość komendy. Wartość w polu "Długość Komendy" nie zgadza się z faktyczną długością pola "Dane" lub wartość w polu długości jest większa niż 256.	Wartość w polu "Długość Komendy" jest mniejsza lub większa niż liczba znaków w polu "Dane", w polu długości podano więcej niż 3 znaki, podana wartość jest większa niż 256.
ERR_EMP	**007**	Pusta ramka	Wysłano pustą ramkę z wartością 000 w polu "Długość Komendy".

2.4.1. Działanie ramek zwrotnych

Ramki zwrotne to komendy które będą wysyłane z odbiory do nadawcy (STM32 → PC). Są one według mnie niezbędne w każdym protokole komunikacji ze względu na potwierdzenie czy ramka została poprawnie odebrana.

2.4.2. Konstrukcja ramki nadawczej (PC→ STM32)

To jest proces, który musi wykonać oprogramowanie na PC *przed* wysłaniem komendy do STM32.

1. **Wybierz Komendę:** Zdecyduj, którą komendę chcesz wysłać (np. SNAKE, SET, STATUS, RESET).
2. **Przygotuj Pole "Dane":**
   - **Dla komend bez parametrów (SNAKE, PULSE, BLINK, STATUS, RESET, PING):** Pole "Dane" zawiera *tylko* nazwę komendy (np. "SNAKE").
   - **Dla komendy SET:** Pole "Dane" zawiera nazwę komendy i parametry, oddzielone przecinkami: SET,R,G,B. Wartości R, G, B muszą być 3-znakowymi liczbami ASCII (np. "255", "000", "128"). *Przykład:* "SET,255,000,100"
   - **Dla komendy PONG:** PONG.
3. **Oblicz Długość:** Oblicz *liczbę znaków* w polu "Dane". To będzie wartość pola "Długość Komendy".
   - **Przykład (SNAKE):** Długość = 5 (bo "SNAKE" ma 5 znaków).
   - **Przykład (SET,255,000,100):** Długość = 12 (bo "SET,255,000,100" ma 12 znaków).
4. **Sformatuj Pole "Długość Komendy":** Zapisz obliczoną długość jako 3-znakową liczbę ASCII, z zerami wiodącymi, jeśli trzeba.
   - **Przykład (Długość = 5):** "005"
   - **Przykład (Długość = 12):** "012"
   - **Przykład (Długość = 256):** "256"
5. **Złóż Ramkę:** Połącz wszystkie pola w jeden ciąg znaków, w odpowiedniej kolejności:

   `Ramka = ":" + Nadawca + Odbiorca + Długość_Komendy + Dane + ";"`

   - **Nadawca:** "000" (zakładamy, że PC to "000").
   - **Odbiorca:** "MTW" (adres STM32).

6. **Wyślij Ramkę:** Wyślij utworzony ciąg znaków przez port szeregowy (UART) do STM32.

Przykład Ramki zwrotnej:

`:000MTW005BLINK;`

2.4.3. Konstrukcja ramki nadawczej (STM32→ PC)

To jest proces, który wykonuje STM32 *po* odebraniu i przetworzeniu komendy od PC.

1. **Określ Wynik:** Po przetworzeniu komendy, ustal, czy operacja się powiodła (np. "OK"), czy wystąpił błąd (np. "ERR_CMD", "ERR_RGB").
2. **Przygotuj Pole "Dane":** Wpisz odpowiedni kod wyniku do pola "Dane" (np. "OK", "ERR_CMD").
3. **Oblicz Długość:** Oblicz długość pola "Dane" (np. "OK" ma długość 2, "ERR_CMD" ma długość 7).
4. **Sformatuj Pole "Długość Komendy":** Zapisz długość jako 3-znakową liczbę ASCII (np. "002", "007").
5. **Złóż Ramkę:** Połącz wszystkie pola w jeden ciąg znaków, w odpowiedniej kolejności:

Ramka = ":" + Nadawca + Odbiorca + Długość_Komendy + Dane + ";"

- **Nadawca:** "000" (zakładamy, że PC to "000").
- **Odbiorca:** "MTW" (adres STM32).

6. **Wyślij Ramkę:** Wyślij utworzony ciąg znaków przez port szeregowy (UART) do STM32.

Przykład Ramki zwrotnej:

:MTW000002OK; lub :MTW000005ERROR;

3. Parametry komunikacji

3.1. Konfiguracja USART

- Baud Rate – 115200 Bits/s
- Word Length – 8 bitów
- Parity – brak
- Stop Bits – wartość 1
- Data Direction – Receive and Transmit
- Over Sampling – 16 sampli

3.2. Konfiguracja PWM

- Częstotliwość PWM: Około 2.47 MHz
- Prescaler (TIMx->PSC): 0
- Period (ARR - Auto Reload Register): 16
- Timer: TIM2
- Kanał: Channel 1 (TIM2_CH1)
- Zegar APB1 (taktujący TIM2): 42 MHz

4. Protokół komunikacyjny – kod i działanie

4.1. Definicja zmiennych

```
// Plik: definitions.h

#ifndef DEFINITIONS_H
#define DEFINITIONS_H

#include "stm32f4xx_hal.h"
#include <stdbool.h>

// ----- Konfiguracja Sprzętowa -----
#define LED_COUNT 8       // Liczba diod LED (DOSTOSUJ!)
#define UART_RX_BUFFER_SIZE 256 // Rozmiar bufora kołowego UART

// ----- Protokół Ramki -----
#define START_BYTE ':'
#define END_BYTE ';'
#define PC_ADDRESS "000"
#define STM32_ADDRESS "MTW"

// ----- Diody WS2812B -----
#define BITS_PER_LED 24    // 24 bity na kolor (GRB)
#define BUFFER_SIZE (LED_COUNT * BITS_PER_LED)     // Rozmiar bufora DMA


#define PWM_LOW  5   // Wartość CCR dla bitu "0"
#define PWM_HIGH 11  // Wartość CCR dla bitu "1"

// ----- Kody Błędów (Ramki Zwrotne) -----
#define OK_RESPONSE "OK"
#define ERROR_RESPONSE "ERROR"
#define ERROR_COMMAND "ERR_CMD"
#define ERROR_ARGUMENT "ERR_ARG"
#define ERROR_RGB "ERR_RGB"
#define ERROR_LENGTH "ERR_LEN"
#define ERROR_EMPTY "ERR_EMP"
#define PING_RESPONSE "PONG"

//Struktura przechowująca wskaźniki na funkcję obsługi komend
typedef struct {
    const char* name;
    void (*handler)(const char*, const char*, const char*);
} CommandHandler;

#endif // DEFINITIONS_H
```

4.2. Bufor kołowy

```h
// Plik: circular_buffer.h

#ifndef CIRCULAR_BUFFER_H
#define CIRCULAR_BUFFER_H

#include <stdint.h>
#include "definitions.h" // Dla UART_RX_BUFFER_SIZE

// Bufor kołowy (deklaracje zmiennych)
extern volatile uint8_t uart_rx_buffer[UART_RX_BUFFER_SIZE];
extern volatile int uart_rx_head;
extern volatile int uart_rx_tail;

// Deklaracje funkcji
void uart_rx_put(uint8_t data);
int  uart_rx_get();

#endif // CIRCULAR_BUFFER_H
```

```c
// Plik: circular_buffer.c

#include "circular_buffer.h"

// Bufor kołowy
volatile uint8_t uart_rx_buffer[UART_RX_BUFFER_SIZE];
volatile int uart_rx_head = 0;
volatile int uart_rx_tail = 0;

// Dodawanie do bufora
void uart_rx_put(uint8_t data) {
    int next_head = (uart_rx_head + 1) % UART_RX_BUFFER_SIZE;
    if (next_head != uart_rx_tail) { // Sprawdź, czy nie pełny
        uart_rx_buffer[uart_rx_head] = data;
        uart_rx_head = next_head;
    }
}

// Pobieranie z bufora
int uart_rx_get() {
    if (uart_rx_head == uart_rx_tail) {
        return -1; // Bufor pusty
    }
    uint8_t data = uart_rx_buffer[uart_rx_tail];
    uart_rx_tail = (uart_rx_tail + 1) % UART_RX_BUFFER_SIZE;
    return data;
}
```

4.3. Funkcja parsowania ramki i obsługa komend

```
void Process_SetColor(const char* args, const char* sender, const char* receiver);

void Process_Frame(char* frame) {
  // 1. Szukaj znaku początku i końca
    char* start = strchr(frame, ':');
    char* end = strchr(frame, ';');

    if (!start || !end) {
        return;  // Nieprawidłowa ramka
    }
    if (start >= end) {
      return; // Błąd: znak końca przed znakiem początku
    }

    // 2. wyodrębnianie pól
    char sender[4];
    char receiver[4];
    char length_str[4];
    char data[257]; // +1 na '\0'

   int parsed = sscanf(start, ":%3[^:]%3[^:]%3[^:]", sender, receiver, length_str);

    if (parsed != 3) {
        return; // Błąd parsowania
    }
    //zabezpieczenie przed atakiem
    sender[3] = '\0';
    receiver[3] = '\0';
    length_str[3] = '\0';


    // 3. Sprawdź Adresy
    if (strcmp(receiver, STM32_ADDRESS) != 0) {
        return; // Ramka nie do nas
    }

    // 4. Konwertuj długość
    int length = atoi(length_str);

    // 5. Sprawdzamy czy w danych nie ma znaków końca i początku
    for(int i = 0; i < length; i++){
        if(start[10 + i] == ':' || start[10 + i] == ';'){
            return; // Błąd, znaki specjalne w danych
        }
    }

    //6. Sprawdzamy czy ramka nie jest pusta
    if(length == 0){
        send_error_frame(receiver,sender, ERROR_EMPTY);
        return;
    }

    // 7. Sprawdź długość
    int actual_data_length = end - (start + 10);
    if (length != actual_data_length || length > 256) {
        send_error_frame(receiver, sender, ERROR_LENGTH);
        return;
    }

    // 8. Kopiuj dane
    strncpy(data, start + 10, length);
    data[length] = '\
```

```
// Plik: command_parser.h

#ifndef COMMAND_PARSER_H
#define COMMAND_PARSER_H

#include "definitions.h"

void Process_Frame(char* frame);

#endif // COMMAND_PARSER_H
```

4.4. Funkcja ramek zwrotnych

```c
// Plik: response_frames.c
#include "response_frames.h"
#include "definitions.h"
#include "stm32f4xx_hal.h"
#include <stdio.h>
#include <string.h>

extern UART_HandleTypeDef huart2;

void send_ok_frame(const char* receiver, const char* sender) {
    char frame[16];
    sprintf(frame, ":%s%s002OK;", receiver, sender);
    HAL_UART_Transmit(&huart2, (uint8_t*)frame, strlen(frame), HAL_MAX_DELAY);
}

void send_error_frame(const char* receiver, const char* sender, const char* error_code) {
    char frame[32];
    sprintf(frame, ":%s%s%03d%s;", receiver, sender, (int)strlen(error_code), error_code);
    HAL_UART_Transmit(&huart2, (uint8_t*)frame, strlen(frame), HAL_MAX_DELAY);
}

void send_pong_frame(const char* receiver, const char* sender){
    char frame[16];
    sprintf(frame, ":%s%s004PONG;", receiver, sender);
    HAL_UART_Transmit(&huart2, (uint8_t*)frame, strlen(frame), HAL_MAX_DELAY);
}
```

```c
// Plik: response_frames.h
#ifndef RESPONSE_FRAMES_H
#define RESPONSE_FRAMES_H
void send_ok_frame(const char* receiver, const char* sender);
void send_error_frame(const char* receiver, const char* sender, const char* error_code);
void send_pong_frame(const char* receiver, const char* sender);
#endif
```

4.5. Funkcja obsługi diod WS2812B

```c
// Plik: ws2812b.h

#ifndef WS2812B_H
#define WS2812B_H

#include "definitions.h"
#include "stm32f4xx_hal.h" //jeśli potrzebujemy

extern TIM_HandleTypeDef htim2; //timer do PWM
extern uint16_t pwm_buffer[BUFFER_SIZE]; //bufor PWM

void set_led_color(int led_index, uint8_t r, uint8_t g, uint8_t b);
void reset_leds(void);


#endif // WS2812B_H
```

```c
// Plik: ws2812b.c
#include "ws2812b.h"


// Ustawia kolor *pojedynczej* diody LED (w buforze DMA)
void set_led_color(int led_index, uint8_t r, uint8_t g, uint8_t b) {
    if (led_index < 0 || led_index >= LED_COUNT) {
        return;  // Nieprawidłowy indeks
    }

    int buffer_offset = led_index * BITS_PER_LED;

    // Kolejność bitów GRB (nie RGB!)
    for (int i = 7; i >= 0; i--) { // Zielony
        pwm_buffer[buffer_offset++] = ((g >> i) & 1) ? PWM_HIGH : PWM_LOW;
    }
    for (int i = 7; i >= 0; i--) { // Czerwony
        pwm_buffer[buffer_offset++] = ((r >> i) & 1) ? PWM_HIGH : PWM_LOW;
    }
    for (int i = 7; i >= 0; i--) { // Niebieski
        pwm_buffer[buffer_offset++] = ((b >> i) & 1) ? PWM_HIGH : PWM_LOW;
    }
}

//Zerowanie bufora
void reset_leds(void)
{
  for (int i = 0; i<BUFFER_SIZE;i++)
  {
    pwm_buffer[i] = 0;
  }
}
```

4.6. Funkcja efektów LED

```c
// Plik: led_effects.h

#ifndef LED_EFFECTS_H
#define LED_EFFECTS_H
#include "stm32f4xx_hal.h"
#include <stdbool.h>

extern volatile bool should_stop_effects; // Flaga do zatrzymywania efektów
void Stop_Effects(); // Zatrzymuje wszystkie efekty
void Start_Snake_Effect();
void Start_Pulse_Effect();
void Start_Blink_Effect();

#endif // LED_EFFECTS_H
```

Efekt Węża:

```c
// Plik: led_effects.c

#include "led_effects.h"
#include "ws2812b.h"


volatile bool should_stop_effects = false; // Inicjalizacja flagi

void Stop_Effects() {
    should_stop_effects = true;
}
//Wonyszy
void Start_Snake_Effect() {
    should_stop_effects = false;

    int head = 0;
    uint8_t colors[][3] = {
        {255, 0, 0},   // Czerwony
        {0, 255, 0},   // Zielony
        {0, 0, 255},   // Niebieski
        {255, 255, 0}, // Żółty
        {255, 0, 255}, // Magenta
        {0, 255, 255}  // Cyjan
    };
    int color_index = 0;
      reset_leds(); // Zerowanie bufora
    HAL_TIM_PWM_Start_DMA(&htim2, TIM_CHANNEL_1, (uint32_t*)pwm_buffer, BUFFER_SIZE);  // Uruchomienie DMA

    while (!should_stop_effects) {
        // Wyłącz wszystkie diody
        for(int i = 0; i< LED_COUNT; i++){
            set_led_color(i, 0, 0, 0);
        }
        // Zapal diodę na pozycji "head"
        set_led_color(head, colors[color_index][0], colors[color_index][1], colors[color_index][2]);

        head = (head + 1) % LED_COUNT; // Przesuń "głowę" węża
        if (head == 0) {
            color_index = (color_index + 1) % (sizeof(colors) / sizeof(colors[0])); // Zmień kolor
        }

        HAL_Delay(200); // Czekaj
         if (should_stop_effects) break; // Zatrzymanie efektu, gdy flaga ustawiona
    }

      reset_leds();// Wyłącz wszystkie po zakończeniu
     HAL_TIM_PWM_Stop_DMA(&htim2, TIM_CHANNEL_1);
```

Efekt Pulsacji:

```c
//Pulsacja
void Start_Pulse_Effect() {
    should_stop_effects = false; //Zerujemy flage
    reset_leds(); // Zerowanie bufora
    HAL_TIM_PWM_Start_DMA(&htim2, TIM_CHANNEL_1, (uint32_t*)pwm_buffer, BUFFER_SIZE);

    )
    while (!should_stop_effects) {
        for (int i = 0; i < 256; i++) { // Zwiększaj jasność
            for (int led = 0; led < LED_COUNT; led++) {
                set_led_color(led, i, 0, 0);
            }
            HAL_Delay(5); // Czekaj
            if(should_stop_effects) break;
        }
        for (int i = 255; i >= 0; i--) { // Zmniejszaj jasność
            for (int led = 0; led < LED_COUNT; led++) {
                set_led_color(led, i, 0, 0);
            }
            HAL_Delay(5); // Czekaj
             if(should_stop_effects) break;
        }

    }
     reset_leds();// Zerowanie bufora
    HAL_TIM_PWM_Stop_DMA(&htim2, TIM_CHANNEL_1);
}
```

Efekt wszystko na biało:

```
// Zapal wszystkie diody na biało
void Start_Blink_Effect() {
    should_stop_effects = false;//Zerujemy flage

    reset_leds();// Zerowanie bufora
    HAL_TIM_PWM_Start_DMA(&htim2, TIM_CHANNEL_1, (uint32_t*)pwm_buffer, BUFFER_SIZE);

    while (!should_stop_effects) {
        // Zapal wszystkie diody na biało
        for (int led = 0; led < LED_COUNT; led++) {
            set_led_color(led, 255, 255, 255);
        }
        HAL_Delay(400); // Czekaj
        if(should_stop_effects) break;

        // Wyłącz wszystkie diody
        for (int led = 0; led < LED_COUNT; led++) {
            set_led_color(led, 0, 0, 0);
        }
        HAL_Delay(400); // Czekaj
        if(should_stop_effects) break;
    }
    reset_leds();// Zerowanie bufora
    HAL_TIM_PWM_Stop_DMA(&htim2, TIM_CHANNEL_1);
}
```

5. Przykładowa obsługa programu

1. **Połączenie:** Użytkownik łączy płytkę STM32 Nucleo z komputerem PC za pomocą kabla USB (co tworzy wirtualny port COM przez wbudowany ST-LINK).
2. **Uruchomienie Terminala:** Użytkownik uruchamia na komputerze PC program terminala szeregowego (np. PuTTY)
3. **Konfiguracja Terminala:** Użytkownik konfiguruje program terminala:

    - Wybiera odpowiedni port COM (ten, który został utworzony po podłączeniu STM32).
    - Ustawia prędkość transmisji (baud rate) na *115200*.
    - Ustawia pozostałe parametry transmisji (8 bitów danych, brak parzystości, 1 bit stopu) - zazwyczaj są to domyślne ustawienia.
    - Upewnia się, że terminal jest ustawiony na wysyłanie i odbieranie danych w formacie *tekstowym* (ASCII).

4. **Wysyłanie Komend:** Użytkownik wpisuje komendy w oknie terminala, *zgodnie z formatem ramki protokołu*, i wysyła je, naciskając Enter (lub odpowiedni przycisk w programie terminala). Przykłady komend:

    - **: + 000 + MTW + 005 + SNAKE + ;** (uruchomienie efektu węża)
    - **: + 000 + MTW + 012 + SET,255,000,000 + ;** (ustawienie koloru czerwonego)
    - **: + 000 + MTW + 006 + STATUS + ;** (żądanie statusu)
    - **: + 000 + MTW + 004 + PING + ;** (sprawdzenie połączenia)

Ważne:

- Ramka *musi* zaczynać się od dwukropka (:) i kończyć średnikiem (;).
- Adres nadawcy (PC) to *zawsze* 000.
- Adres odbiorcy (STM32) to *zawsze* MTW.

- Długość komendy musi być podana jako 3-znakowa liczba ASCII (z zerami wiodącymi) i *musi* odpowiadać faktycznej długości nazwy komendy (i parametrów, jeśli występują).
- Komenda musi być jedną z dozwolonych komend (np., SNAKE, PULSE, BLINK, SET, STATUS, RESET, PING).

5. **Odbieranie Odpowiedzi:** Po wysłaniu komendy użytkownik *czeka* na odpowiedź od STM32 (ramkę zwrotną) w oknie terminala. Odpowiedź będzie również w formacie ramki
6. **Interpretacja Odpowiedzi:** Użytkownik interpretuje odpowiedź od STM32:

    - Jeśli w polu "Dane" ramki zwrotnej jest OK, oznacza to, że komenda została wykonana pomyślnie.
    - Jeśli w polu "Dane" jest ERROR lub kod błędu (np. ERR_CMD, ERR_RGB), oznacza to, że wystąpił błąd. Użytkownik powinien sprawdzić, co poszło nie tak (np. literówka w komendzie, błędne wartości RGB).
    - Jeśli w polu "Dane" jest odpowiedź na komendę PING, to w polu danych pojawi się PONG
    - Jeśli w polu "Dane" jest odpowiedź na komendę STATUS, użytkownik może zobaczyć np. RUN,SNAKE (co oznacza, że urządzenie działa i aktualnie jest aktywny efekt węża).
    - Jeśli *nie ma* odpowiedzi w ciągu kilku sekund (timeout), użytkownik może założyć, że wystąpił problem z komunikacją (np. odłączony kabel, zawieszenie się STM32).

7. **Powtarzanie:** Użytkownik może wysyłać kolejne komendy (krok 4) i odbierać odpowiedzi (krok 5 i 6), aby sterować taśmą LED.

Co się dzieje "pod maską" (w skrócie):

1. **PC (Terminal):** Program terminala wysyła wpisaną przez użytkownika komendę (w formie ramki) przez port szeregowy (UART).
2. **STM32 (Odbiór):** Mikrokontroler STM32 odbiera dane przez UART (za pomocą przerwań i bufora kołowego).
3. **STM32 (Parsowanie):** STM32 parsuje odebraną ramkę (sprawdza znak początku i końca, wyodrębnia adresy, długość, dane).
4. **STM32 (Sprawdzanie Błędów):** STM32 sprawdza, czy ramka jest poprawna (długość, adres, znaki specjalne).
5. **STM32 (Interpretacja Komendy):** STM32 porównuje pole "Dane" z listą znanych komend.
6. **STM32 (Wykonanie):** Jeśli komenda zostanie rozpoznana, STM32 wykonuje odpowiednią akcję (np. uruchamia efekt świetlny, ustawia kolor, resetuje się).
7. **STM32 (Wysłanie Odpowiedzi):** STM32 wysyła ramkę zwrotną przez UART, informując PC o wyniku operacji ("OK" lub kod błędu).
8. **PC (Terminal):** Program terminala odbiera ramkę zwrotną i wyświetla ją użytkownikowi.