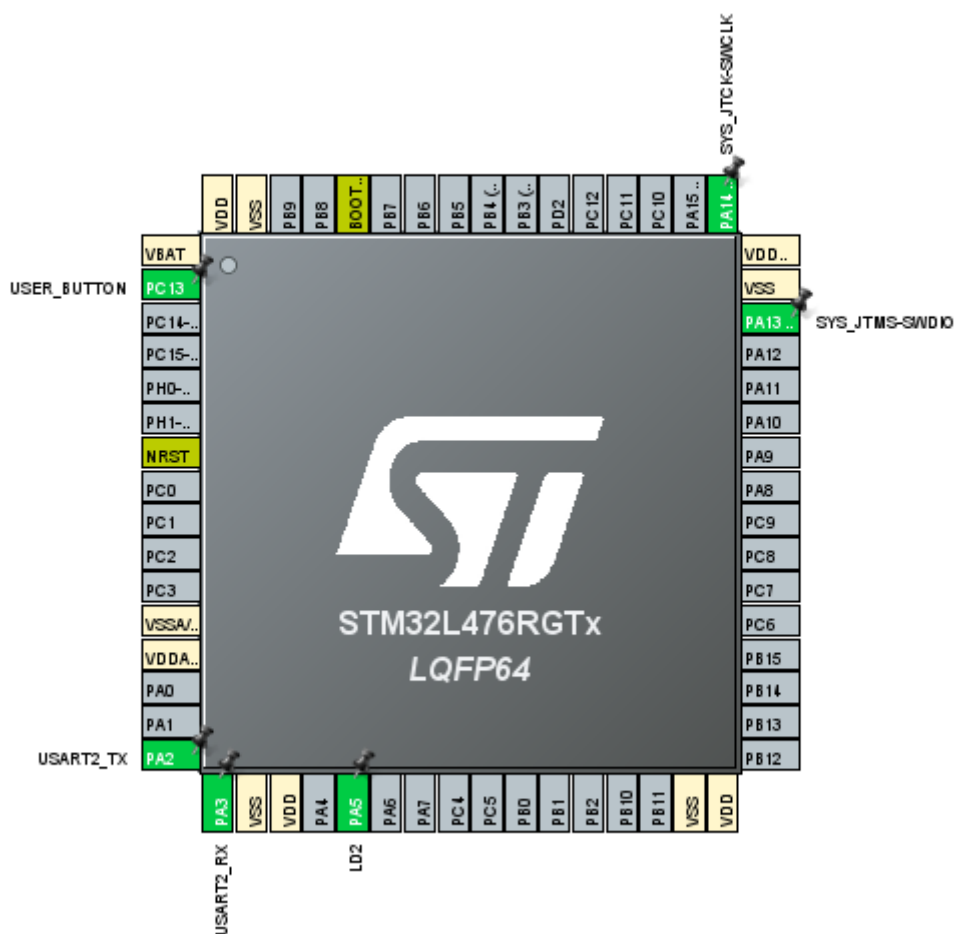


Wydział Informatyki Politechniki Białostockiej Laboratorium Architektura Komputerów	Data: 07.11.2023r.
Ćwiczenie 2 Temat: Transmisja szeregową za pomocą protokołu RS232 Grupa PS.02 Wykonał: Dominik Gąsowski	Prowadzący: dr inż. Mirosław Omieljanowicz Ocena:

Do wykonania zadań 1-6 będzie potrzebować tych konkretnych pinów. Pin PA5 odpowiada za działanie diody. Wybieramy opcję GPIO_Output i zmieniamy nazwę na „LD2”, aby łatwiej było posługiwać się tą diodą w kodzie. Pin PC13 odpowiada za działanie przycisku. Wybieramy opcję GPIO_Input i zmieniamy nazwę na „USER_BUTTON”. Jeśli chodzi o piny PA13 i PA14 to system sam je ustawi, jeśli wybierzemy dobrą opcję. Po lewej stronie gui wybieramy opcję „SYS”, w zakładce system core, a następnie w oknie „Debug” wybieramy Serial Wire. Dzięki wybraniu opcji Serial Wire od teraz STM32CubeMX będzie dbał o to, żebyśmy przypadkiem nie zmienili ustawień pinów programatora, co mogłoby doprowadzić do problemów z komunikacją. Potrzebujemy jeszcze interfejsu USART2, ponieważ jest on domyślnie połączony z przejściówką USB, która wbudowana jest w programator. Dzięki temu będziemy mogli łatwo wysyłać dane do PC. Po lewej stronie gui wybieramy zakładkę „Connectivity”, wchodzimy w „USART2” i ustawiamy mode na „asynchronous”. Po włączeniu tej opcji, piny PA3 i PA2 same zrobią się zielone. To właśnie one będą odpowiadać za transmisję.



Zadanie 1.

Polecenie:

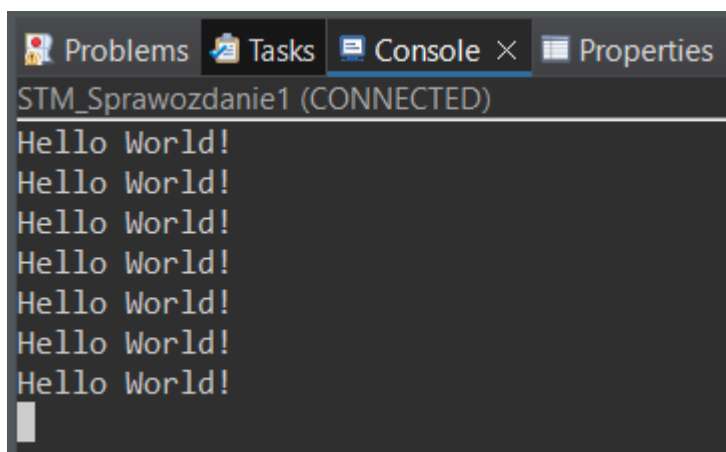
Skonfiguruj moduł UART uwzględniając następujące parametry: prędkością transmisji 115200 b/s, rozmiar słowa 8 b, brak bitu parzystości, 1 bit stopu. Następnie napisz program, który co sekundę wysyła ciąg znaków „Hello world” poprzez interfejs UART. Zweryfikuj poprawność komunikacji odbierając ciągi znaków w terminalu komputera podłączonego do płytki Nucleo. Program powinien działać w taki sposób, aby po wysłaniu ciągu „Hello world” kursor terminala przechodził do początku nowej linii.

Wykonanie:

Do wykonania tego zadania potrzebowałem tablicy znaków, w której podałem napis „Hello World” wymagany w zadaniu. W pętli while, która została domyślnie wygenerowana użyłem funkcji HAL_UART_Transmit, która służy do przesyłania danych przez interfejs UART na mikrokontrolerze do mojego PC. Na końcu podałem jeszcze opóźnienie równe 1 sekundzie.

```
/* USER CODE BEGIN 2 */
const char message[] = "Hello World!\r\n";
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    HAL_UART_Transmit(&huart2, message, strlen(message), HAL_MAX_DELAY);
    HAL_Delay(1000);
    /* USER CODE BEGIN 3 */
}
```



The screenshot shows the STM32CubeIDE interface with the 'Console' tab selected. The title bar of the console window reads 'STM_Sprawozdanie1 (CONNECTED)'. The output area displays the text 'Hello World!' repeated seven times, each on a new line. A white cursor is visible at the end of the last line of output.

Zadanie 2.

Polecenie:

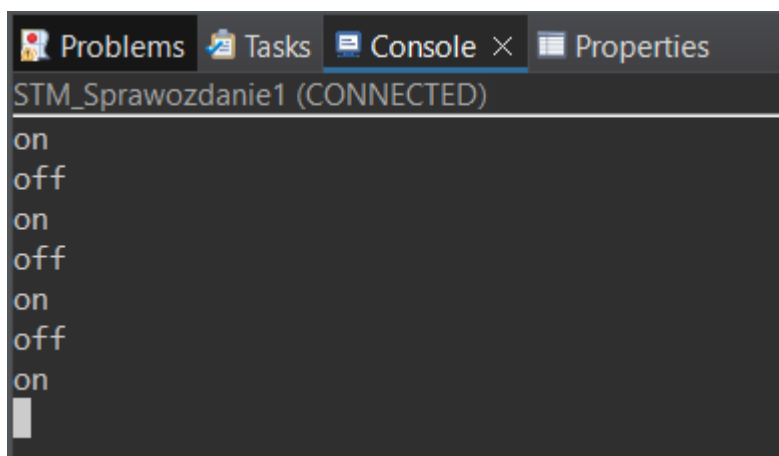
Napisz program, który będzie zapalał diodę LD2 po wciśnięciu przycisku BT1 i gasił ją po ponownym jego wciśnięciu. Proces ten powinien się powtarzać przy każdym wciśnięciu przycisku. Za każdym razem, gdy dioda jest zapalana lub gaszona, program powinien wysyłać komunikat tekstowy, odpowiednio, „on” lub „off” na interfejs UART.

Wykonanie:

Stworzyłem dwie tablice znakowe: jedna przechowująca napis „on”, druga „off”. W pętli while sprawdzam, czy przycisk został naciśnięty. Jeśli tak, to sprawdzam czy dioda się pali, czy nie. Jeśli tak, to ją gaszę i wypisuje „off”, że została zgaszona. W przeciwnym wypadku ją zapalam i wypisuje „on”.

```
/* USER CODE BEGIN 2 */
const char message[] = "on\r\n";
const char message2[] = "off\r\n";
HAL_UART_Transmit(&huart2, message, strlen(message), HAL_MAX_DELAY);
/* USER CODE END 2 */
```

```
while (1)
{
    /* USER CODE END WHILE */
    if (HAL_GPIO_ReadPin(USER_BUTTON_GPIO_Port, USER_BUTTON_Pin) == GPIO_PIN_RESET)
    {
        if (HAL_GPIO_ReadPin(LD2_GPIO_Port, LD2_Pin) == GPIO_PIN_SET)
        {
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
            HAL_UART_Transmit(&huart2, message2, strlen(message2), HAL_MAX_DELAY);
            HAL_Delay(300);
        }
        else
        {
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
            HAL_UART_Transmit(&huart2, message, strlen(message), HAL_MAX_DELAY);
            HAL_Delay(300);
        }
    }
}
```



Zadanie 3.

Polecenie:

Napisz program, który po każdym wciśnięciu przycisku BT1 będzie wysyłał poprzez interfejs UART ciąg znaków reprezentujący kolejną liczbę naturalną, zaczynając od 1. Zatem po pierwszym wciśnięciu przycisku powinien zostać wysłany ciąg „1”, po drugim „2”, i tak dalej. Do realizacji zadania napisz własną funkcję konwertującą liczbę całkowitą na odpowiedni ciąg znaków, a następnie wysyłającą ten ciąg poprzez interfejs UART.

Wykonanie:

Do wykonania tego zadania napisałem funkcję „konwertuj”, która zamienia liczbę całkowitą na ciąg znaków i zwraca go.

```
char *konwertuj(int number)
{
    char *str;
    int n = 1;
    int temp = number;

    // Obliczamy długość ciągu znaków
    while (temp / 10 != 0) {
        temp /= 10;
        n++;
    }

    // Alokujemy pamięć dla ciągu znaków
    str = (char *)malloc((n + 3) * sizeof(char));

    // Konwertujemy liczbę na ciąg znaków, od końca do początku
    for (int i = n - 1; i >= 0; i--) {
        str[i] = (char)((number % 10) + '0');
        number /= 10;
    }

    // Null-terminate ciąg znaków
    str[n] = '\r';
    str[n+1] = '\n';
    str[n+2] = '\0';

    return str;
}
```

Następnie w funkcji main zadeklarowałem licznik, który zwiększa się za każdym razem, gdy wciśnie przycisk na mikrokontrolerze i wypisuje jego stan do konsoli.

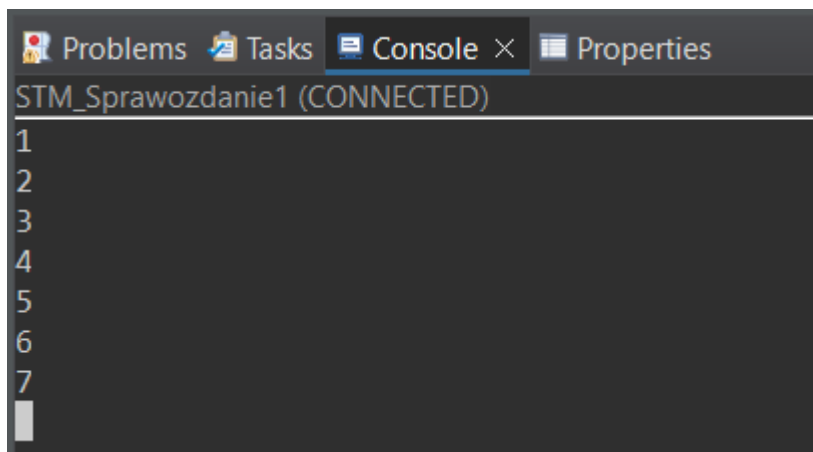
```
int licznik = 0;

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    if(HAL_GPIO_ReadPin(USER_BUTTON_GPIO_Port, USER_BUTTON_Pin) == GPIO_PIN_RESET)
    {
        licznik++;
        char *message = konwertuj(licznik);
        HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message), HAL_MAX_DELAY);
        HAL_Delay(300);
    }

    /* USER CODE BEGIN 3 */
}
```



Zadanie 4.

Polecenie:

Napisz program, który będzie odczytywał z interfejsu UART ciąg znaków zakończony znakiem końca linii. Jeśli odczytany ciąg będzie równy „on”, dioda powinna się zapalić. Jeśli będzie równy „off”, dioda powinna zgasnąć. W przypadku innych ciągów znaków program powinien wysłać komunikat o błędzie. Aby zrealizować zadanie, napisz funkcję, która wczytuje pojedynczo znaki i dodaje je do bufora aż do momentu napotkania znaku końca linii.

Wykonanie:

Do wykonania tego zadania napisałem funkcję putchar, abym wysyłał wprowadzone znaki do konsoli i je widział. Dodatkowo dzięki niej mogę używać funkcji „printf”

```
int __io_putchar(int ch)
{
    if (ch == '\n') {
        uint8_t ch2 = '\r';
        HAL_UART_Transmit(&huart2, &ch2, 1, HAL_MAX_DELAY);
    }

    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return 1;
}
```

Zdefiniowałem maksymalną długość ciągu znaków oraz bufor, który będzie przechowywał moje znaki. W zależności od potrzeb można zmienić długość.

```
#define LINE_MAX_LENGTH 80

static char line_buffer[LINE_MAX_LENGTH + 1];
static uint32_t line_length;
```

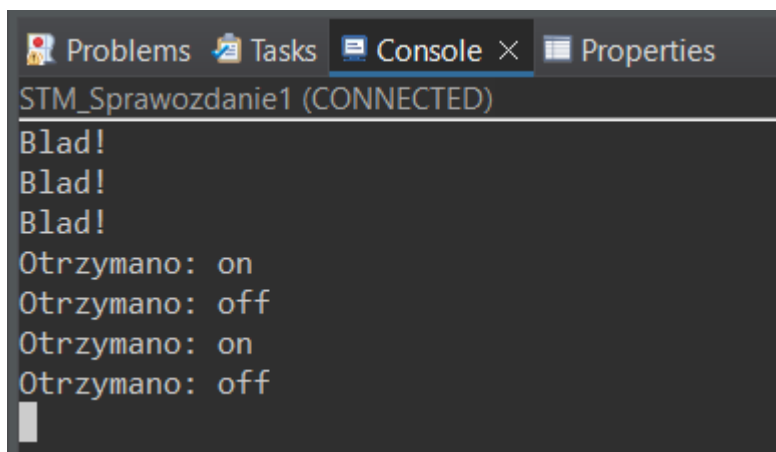
Napisałem funkcję która sprawdza co wpisuję w konsoli i w zależności od tego co wpiszę, drukuje komunikat.

```
void line_append(uint8_t value)
{
    if (value == '\r' || value == '\n') {
        if (line_length > 0) {
            line_buffer[line_length] = '\0';
            if (line_buffer[0] == 'o' && line_buffer[1] == 'n' && line_length == 2)
            {
                HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
                printf("Otrzymano: %s\n", line_buffer);
            }
            else if (line_buffer[0] == 'o' && line_buffer[1] == 'f' && line_buffer[2] == 'f' && line_length == 3)
            {
                HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
                printf("Otrzymano: %s\n", line_buffer);
            }
            else
            {
                printf("Bład! \n");
            }
            line_length = 0;
        }
    }
    else {
        if (line_length >= LINE_MAX_LENGTH) {
            line_length = 0;
        }
        line_buffer[line_length++] = value;
    }
}
```

W funkcji main za pomocą funkcji HAL_UART_Receive odbieram dane przez interfejs UART i sprawdzam czy odebranie tych danych zakończyło się sukcesem. Jeśli tak, wywołuję funkcję wyżej.

```
while (1)
{
    /* USER CODE END WHILE */
    uint8_t value;
    if (HAL_UART_Receive(&huart2, &value, 1, 0) == HAL_OK)
        line_append(value);

    /* USER CODE BEGIN 3 */
}
```



```
Problems Tasks Console × Properties
STM_Sprawozdanie1 (CONNECTED)
Bład!
Bład!
Bład!
Otrzymano: on
Otrzymano: off
Otrzymano: on
Otrzymano: off
```

Zadanie 5.

Polecenie:

Napisz program, który poprzez interfejs UART pobierze od użytkownika dwie liczby, a następnie wypisze tekst „{a} + {b} = {c}”, gdzie {a} i {b} oznaczają dwie pobrane liczby, a {c} oznacza sumę pobranych liczb. Przykładowo, dla liczb 3 i 5 program powinien wypisać tekst „3 + 5 = 8”. Program zaimplementuj wykorzystując wysokopoziomowe funkcje do komunikacji ze standardowym wejściem i wyjściem z biblioteki stdio.h, tj. printf i scanf.

Zaimplementuj te funkcje korzystając z HAL_UART_Transmit oraz HAL_UART_Receive, tak aby odpowiednio wysyłały i pobierały znaki z interfejsu UART. Tak zdefiniowane funkcje powinny zostać automatycznie „zlinkowane” z biblioteką stdlib, co umożliwi działanie funkcji operujących na standardowym wejściu/wyjściu. Jeśli użytkownik wprowadzi niepoprawne dane, program powinien wykryć ten błąd i poprosić użytkownika o ponowne wprowadzenie danych. Wykorzystaj funkcję scanf do odczytu danych i fflush do opróżnienia bufora z błędnymi danymi. Aby polecenie użytkownika mogło zostać przetworzone po podaniu znaku końca linii, a nie po wypełnieniu całego bufora, wyłącz buforowanie

Wykonanie:

Jeśli chodzi o funkcję „printf”, opisałem ją w zadaniu 4. Do wykonania tego zadania dopisałem funkcję „getchar”, dzięki której mogę używać „scanf”.

```
int __io_getchar(void)
{
    uint8_t ch;
    HAL_UART_Receive(&huart2, &ch, 1, HAL_MAX_DELAY);
    return ch;
}
```

Przed napisaniem funkcji main wyczyściłem bufor wejścia i wyjścia.

```
setvbuf(stdin, NULL, _IONBF, 0);
setvbuf(stdout, NULL, _IONBF, 0);
```

W funkcji main zainicjowałem zmienne a i b. Wczytuje je od użytkownika z konsoli czyszcząc między czasie bufor wejściowy, aby nie było tam wcześniej wpisanych liczb. Następnie dodaje je i zapisuje do zmiennej c, którą wypisuję w konsoli.

```
while (1)
{
    /* USER CODE END WHILE */
    int a, b;
    printf("Podaj pierwsza liczbe: ");
    if (scanf("%d", &a) != 1)
    {
        printf("Nieprawidlowe dane. Podaj prawidlowa liczbe.\n");
        fflush(stdin); // Wyczyść bufor wejściowy
        continue;
    }

    printf("Podaj druga liczbe: ");
    if (scanf("%d", &b) != 1) {
        printf("Nieprawidlowe dane. Podaj prawidlowa liczbe.\n");
        fflush(stdin); // Wyczyść bufor wejściowy
        continue;
    }

    int c = a + b;
    printf("%d + %d = %d\n", a, b, c);
}
```



```
Problems Tasks Console × Properties
STM_Sprawozdanie1 (CONNECTED)
Podaj pierwsza liczbe: Podaj druga liczbe: 1 + 2 = 3
Podaj pierwsza liczbe: Podaj druga liczbe: 5 + 6 = 11
Podaj pierwsza liczbe: Podaj druga liczbe: 11 + 123 = 134
Podaj pierwsza liczbe: Podaj druga liczbe: 81 + 1 = 82
Podaj pierwsza liczbe: Podaj druga liczbe: 100 + 200 = 300
Podaj pierwsza liczbe: Podaj druga liczbe: 200 + 12 = 212
Podaj pierwsza liczbe: Podaj druga liczbe: 212 + 0 = 212
Podaj pierwsza liczbe: 
```

Zadanie 6.

Polecenie:

Korzystając z funkcji z biblioteki stdlib, napisz program imitujący prostą powłokę systemową. Program powinien przyjmować komendy na swoje wejście (poprzez interfejs UART) i reagować na nie w odpowiedni sposób. Program powinien wypisywać prompt w postaci „>” zachęcający użytkownika do wpisania komendy. Program powinien obsługiwać następujące polecenia:

o help – wypisuje wszystkie dostępne komendy wraz z krótkimi opisami;

o turn on|off – zapala lub gasi diodę LD2;

o blink – włącza „miganie” diodą LD2 z okresem ms przez podany czas ms;

o state – wypisuje aktualny stan przycisku BT1, tj. 0 lub 1, w zależności od jego wciśnięcia.

W przypadku podania błędnej komendy, program powinien wysłać komunikat o błędzie. Przykładowa interakcja z programem została przedstawiona poniżej. Po prawej stronie podano czynności powodowane przez komendy.

Wykonanie:

Do wykonania tego zadania „ulepszyłem funkcję „getchar”, dzięki której widzę co wpisuje w konsoli.

```
int __io_getchar(void)
{
    uint8_t ch;
    HAL_UART_Receive(&huart2, &ch, 1, HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, &ch, 1, HAL_MAX_DELAY);
    return ch;
}
```

W funkcji main, przed pętlą while wyczyściłem bufor wejścia i wyjścia.

```
setvbuf(stdin, NULL, _IONBF, 0);
setvbuf(stdout, NULL, _IONBF, 0);
```

Zainicjowałem zmienną znakową, która będzie przechowywać znaki, które wprowadzę w konsoli jako komendę. Zainicjowałem również okres i czas, które będą potrzebne podczas pisania komendy „blink”

```
char command[20];
int okres, czas;
```

W pętli while używam funkcji „strcmp”, która porównuje to co wpisuje w konsoli z jakimś napisem. W zależności co wpisuje, dzieją się inne rzeczy. Jeśli chodzi o komendy „turn on” i „turn off”, to najpierw sprawdzam czy wpisany napis to „turn”, a potem wczytuje kolejny napis, decydujący o tym, czy dioda ma się zapalić, czy nie. Za pomocą funkcji HAL_GPIO_WritePin zapalam lub gaszę diodę.

```
while (1)
{
    printf("> ");
    if(scanf("%s",command) == 1)
    {
        if(strcmp(command, "help") == 0)
        {
            printf("Dostępne komendy:\n");
            printf("help - wypisuje dostępne komendy\n");
            printf("turn on|off - zapala lub gasi diode LD2\n");
            printf("blink <period> <duration> - włącza miganie dioda LD2\n");
            printf("state - wypisuje stan przycisku BT1\n");
        }
        else if (strcmp(command,"turn") == 0)
        {
            scanf("%s",command);
            if(strcmp(command,"on") == 0)
            {
                HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
            }
            else if(strcmp(command,"off") == 0)
            {
                HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
            }
            else
            {
                printf("Nieznane polecenie: turn %s",command);
            }
        }
    }
}
```

Za pomocą funkcji HAL_GetTick() mogę kontrolować interwały czasowe i wykonywać miganie diodą z dokładnością do milisekund. Za pomocą funkcji HAL_GPIO_ReadPin sprawdzam czy dioda się świeci, czy nie i wypisuje odpowiedni komunikat po wpisaniu „state”. Podobnie z przyciskiem. Sprawdzam czy jest wciśnięty i wypisuje odpowiedni napis.

```
    else if (strcmp(command, "blink") == 0)
    {
        if(scanf(" %d %d",&okres, &czas) == 2)
        {
            int start = HAL_GetTick();
            while (HAL_GetTick() - start < czas)
            {
                HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
                HAL_Delay(okres);
            }
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
        }
        else
        {
            printf("Bledna komenda");
        }
    }
    else if(strcmp(command, "state") == 0)
    {
        scanf("%s",command);
        if(strcmp(command,"LD2") == 0)
        {
            if (HAL_GPIO_ReadPin(LD2_GPIO_Port,LD2_Pin) == GPIO_PIN_SET)
            {
                printf("Dioda sie swieci\n");
            }
        }
    }
}
```

```
    }
    else
    {
        if(HAL_GPIO_ReadPin(USER_BUTTON_GPIO_Port,USER_BUTTON_Pin) == GPIO_PIN_RESET)
        {
            printf("wcisnietv\n");
        }
        else
        {
            printf("niewcisnietv\n");
        }
    }
}

else
{
    printf("nieznana komenda\n");
}

else
{
    printf("blad podczas analizy komendy");
}
printf("\n");
}
/* USER CODE BEGIN 3 */
/* USER CODE END 3 */
}
```

Wnioski:

Do wykonania tych zadań kluczowa jest konfiguracja interfejsu UART, w celu poprawnej komunikacji między mikrokontrolerem, a komputerem. W zadaniu nr.6 musiałem skorygować funkcję „getchar”, ponieważ nie widziałem komend, które wpisuje w konsolę. Pomogła w tym funkcja HAL_GPIO_Transmit. Rozumienie tej funkcji oraz funkcji HAL_GPIO_Receive jest kluczowe do poprawnej komunikacji między mikrokontrolerem a PC. Przydatna była również funkcja HAL_GetTick, dzięki której mogłem tworzyć opóźnienia i kontrolować interwały czasowe w programach.