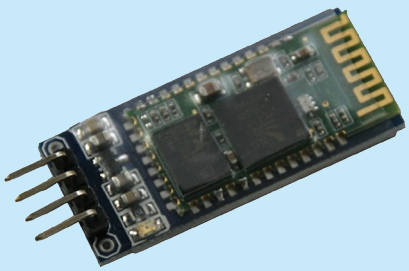
Programowanie STM32F4 (10)

W artykule omówiono obsługę modułu Bluetooth. Na warsztat weźmiemy popularny układu HC-06 i wykorzystamy go do rozbudowy projektu z numeru marcowego – poprzez połączenie Bluetooth sterować będziemy kolorami świecenia adresowalnych diod LED RGB WS2812b na pasku z giętkiego laminatu.

Moduł HC06 (**fotografia 1**) jest nieskomplikowany i przez to bardzo łatwy w obsłudze. Wspiera on tylko jeden profil Bluetooth – profil portu szeregowego. Moduł ma cztery wyprowadzenia: piny zasilania (VCC i GND) oraz piny transmisji i odbioru danych interfejsu UART (TX i RX). Po przyłączeniu do nich mikrokontrolera oraz sparowaniu z modułem urządzenia Bluetooth i otwarciu wirtualnego portu szeregowego na tym urządzeniu możliwa jest między nimi wymiana danych na podobnej zasadzie, jak gdyby były one połączone kablem poprzez interfejs RS232.



Fotografia 1. Fotografia modułu HC-06

Konfiguracja modułu – nadanie mu nazwy, wybór szybkości pracy wirtualnego portu szeregowego czy nadanie kodu pin odbywa się również za pośrednictwem interfejsu UART za pomocą specjalnych poleceń, przed sparowaniem urządzeń i otwarciem portu szeregowego. Obsługiwane są następujące polecenia konfiguracyjne:

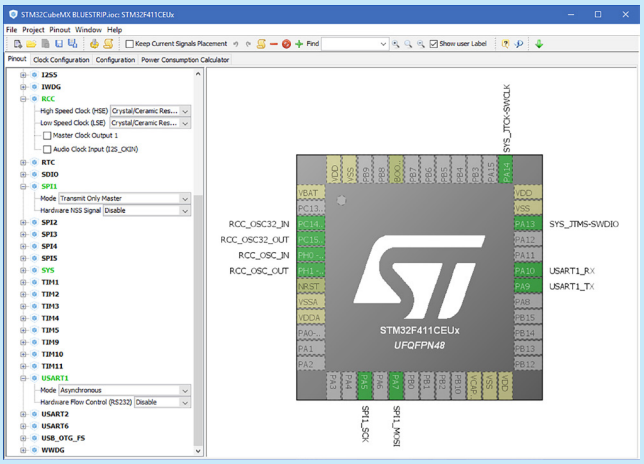
- **AT+BAUD[Liczba z zakresu od 1 do 8]** ustawiające szybkość transmisji wirtualnego portu szeregowego. Liczba z zakresu 1…8 odpowiada jednej z wymienionych szybkości pracy: 1 – 1200 bps, 2 – 2400 bps, 3 – 4800 bps, 4 – 9600 bps, 5 – 19200 bps, 6 – 38400 bps, 7 – 57600 bps, 8 – 115200 bps. Przykładowo, polecenie „AT_BAUD4” ustawi szybkość połączenia na 9600 bps.
- **AT+NAME[Nazwa]** ustawia nazwę rozgłaszaną przez urządzenie. Na przykład, „AT+NAMEBT_TEST”.
- **AT+PIN[4 cyfry]** ustawia kod pin, którego podanie jest wymagane do sparowania urządzeń. Na przykład, „AT+PIN1234”.

Domyślne parametry to nazwa „HC-06”, szybkość połączenia 9600 bps oraz PIN „1234”. Parametry po zmianie są przechowywane w pamięci EEPROM, więc konfiguracja nie jest tracona po odłączeniu zasilania.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86735, pass: 6mqh264k

Listing 1. Ramka danych włączająca na kolejnych 30 diodach na pasku naprzemiennie kolory czerwony, zielony i niebieski

```
@255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0
,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0
,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255
,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255\n\r
```



Rysunek 2. Konfiguracja wyprowadzeń w generatorze konfiguracji STM32CubeMX

Listing 2. Szablon modyfikacji pliku main.c

```
/* USER CODE BEGIN Includes */
#include „frame_parser.h”
#include „ws2812b.h”
/* USER CODE END Includes */

/* USER CODE BEGIN 0 */
volatile ws2812b_config led_strip_handler;
uint8_t diodes_count = 30;

volatile struct frame_parser_state frame_parser_handler;
volatile uint8_t recv_char;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef * uart)
{
    if (uart == &huart1)
    {
        frame_parser_recv_char(&frame_parser_handler, recv_
char);
        HAL_UART_Receive_IT(&huart1, &recv_char, 1);
    }
}

void setup_uart(UART_HandleTypeDef * uart)
{
    HAL_Delay(1000);
    HAL_UART_Transmit(uart, „AT+BAUD4”, strlen(„AT+BAUD4”),
100);
    HAL_Delay(100);
    HAL_UART_Transmit(uart, „AT+NAMEep_bt”, strlen(„AT+NAMEep_
bt”), 100);
    HAL_Delay(100);
    HAL_UART_Transmit(uart, „AT+PIN3498”, strlen(„AT+PIN3498”),
100);
    HAL_Delay(100);
}
/* USER CODE END 0 */

int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration */
    /* Reset of all peripherals, Initializes the Flash interface
and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_SPI1_Init();
    MX_USART1_UART_Init();
    /* USER CODE BEGIN 2 */
    ws2812b_config led_strip_handler = ws2812b_init(&hspi1, di-
odes_count);
    setup_uart(&huart1);
    frame_parser_handler = frame_parser_init(&led_strip_handler,
diodes_count);
    HAL_UART_Receive_IT(&huart1, &recv_char, 1);
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        frame_parser_process_frame(&frame_parser_handler);
    }
    /* USER CODE END 3 */
}
```

Tworzymy projekt

Program, który utworzymy w kolejnych krokach, będzie korzystał z dwóch modułów peryferyjnych: **SPI** do wygenerowania sygnału sterującego dla diod WS2812B oraz interfejsu **UART** do wymiany danych z modułem. Z urządzenia bezprzewodowego przyłączonego do modułu Bluetooth będą przesyłane polecenia zmiany koloru diod na pasku w postaci ramek danych. Ramka ta zaczyna się od znaku „@”, a kończy znakiem nowej linii „\r\n”. Wewnątrz są umieszczone wartości numeryczne z zakresu od 0 do 255, rozdzielone przecinkami i zgrupowane po trzy. Każde kolejne grupy wartości opisują kolor kolejnej diody na pasku. Pierwsza liczba w grupie opisuje jasność składowej czerwonej, druga – zielonej, a trzecia – niebieskiej.

Na **listingu 1** pokazano ramkę danych włączającą na kolejnych 30 diodach na pasku naprzemiennie kolory czerwony, zielony i niebieski. Format powyższej ramki nie pozwala na wygodne ręczne sterowanie diodami przy pomocy takiej aplikacji jak np. PuTTY, ułatwia jednak napisanie własnej aplikacji na telefon, tablet lub komputer.

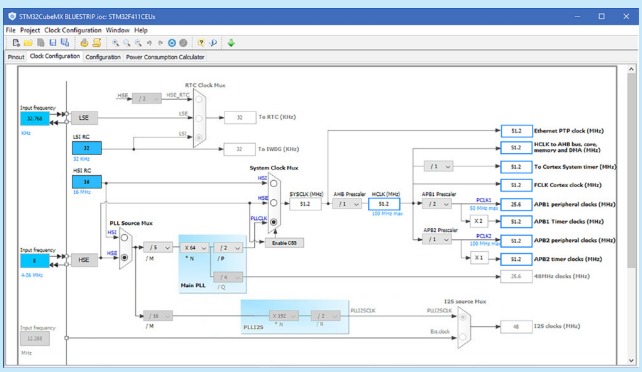
Zatem do dzieła! Standardowo już włączamy program STM32CubeMX i tworzymy w nim nowy projekt, wybierając posiadany przez nas mikrokontroler (u mnie, na płytce KA-NUCELO-F411CE, jest to układ STM32F411CEU6).

Na pierwszym ekranie konfiguratora, zatytułowanym „Pinout”, ustawiamy źródło sygnału taktującego. Na liście po lewej stronie ekranu rozwijamy pozycję „RCC” i w polu „High Speed Clock (HSC)” wybieramy „Crystal/Ceramic Resonator”. Następnie musimy wybrać i uruchomić interfejs SPI. Używany przeze mnie układ dysponuje aż pięcioma takimi interfejsami. Nie ma znaczenia, który z nich wybierzemy, jednak moduły SPI1, SPI4 i SPI5 są taktowane z innej szyny niż SPI2 i SPI3, więc na potrzeby tego projektu wybierzmy któryś z tych pierwszych. W przykładzie użyję interfejsu SPI1 oraz pinów PA5 oraz PA7, odpowiednio w roli SPI1_SCK (sygnału zegara) oraz SPI1_MOSI (wyjścia danych). Jako tryb pracy wybieramy pozycję: „Transmit Only Master”.

Oprócz modułu SPI uruchamiamy również interfejs UART. Możemy wybrać dowolny moduł – USART1, USART2 lub USART6. W przykładzie użyto USART1 na pinach PA10 (RX) i PA9 (TX). Konfigurację wyprowadzeń w STM32CubeMX pokazano na **rysunku 2**.

Teraz możemy już przyłączyć do płytki pasek diod oraz moduł Bluetooth. Pin VCC (czerwony kabel) z paska diod WS2812b łączymy z wyprowadzeniem 5 V płytki, GND (czarny) do jednego z GND na płytce oraz sygnałowy (zielony) do pinu D11. Wyprowadzenia modułu Bluetooth łączymy w następujący sposób: VCC (modułu) do pinu 3V3 (płytki KA-Nucleo), GND do pinu GND oraz piny TX/RX modułu do pinów D2/D8.

W zakładce „Clock Configuration” zwyczajowo ustawiamy częstotliwość pracy wejściowego oscylatora kwarcowego (u mnie

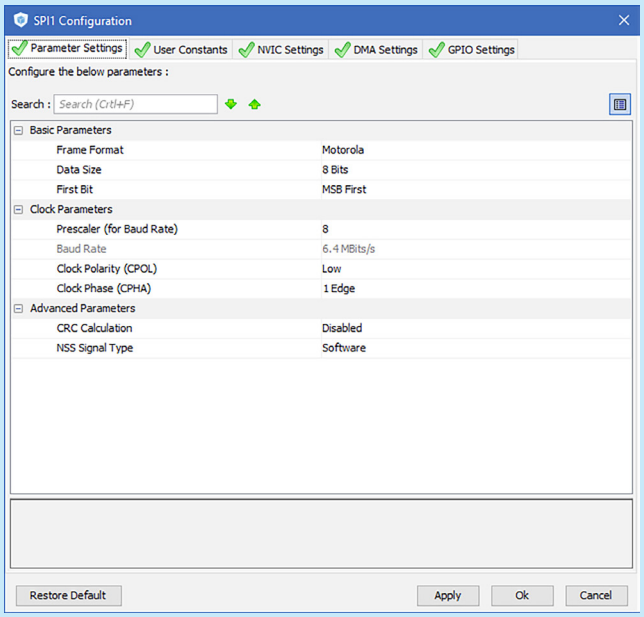


Rysunek 3. Konfiguracja sygnału taktującego w generatorze konfiguracji STM32CubeMX

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86735**, pass: **6mqh264k**

Listing 3. Plik Inc/frame_parser.h

```
#ifndef frame_parser_header
#define frame_parser_header
#include „stdint.h”
#include „stdlib.h”
#include „string.h”
#include „main.h”
#include „ws2812b.h”

struct frame_parser_state
{
    ws2812b_config * led_strip_handler;
    uint8_t diodes_count;
    uint8_t line_buffer[800];
    uint8_t writer_position;
    uint8_t reader_position;
    uint8_t field_buffer[30];
    uint8_t field_position;
    uint8_t process_frame;
};

struct frame_parser_state frame_parser_init(ws2812b_config * _
led_strip_handler, uint8_t _diodes_count);
void frame_parser_recv_char(struct frame_parser_state * state,
uint8_t recv_char);
void frame_parser_read_field(struct frame_parser_state * state);
void frame_parser_process_frame(struct frame_parser_state *
state);

#endif
```



Rysunek 4. Konfiguracja peryferialu SPI w generatorze konfiguracji STM32CubeMX

8 MHz), przełączamy źródło sygnału podawanego na główną pętlę PLL – „PLL Source Mux” na „HSE”, jako źródło sygnału taktującego dla całego układu wybierzmy pętlę PLL – przełącznik „System Clock Mux” ustawiamy na pozycję „PLLCLK”. Teraz musimy jeszcze ustawić pożądaną częstotliwość taktowania naszego układu. Nie będzie to jednak maksymalna, dozwolona wartość. Musimy dobrać ją wspólnie z dzielnikiem częstotliwości taktującej interfejs SPI, aby możliwa była transmisja sygnału „jedynki” i „zera” dla diod w odpowiednim czasie. Robimy to identycznie. jak opisano w EP 3/2017, ponieważ transmisja pojedynczego bitu sygnału sterującego diodami trwa 1,25 µs, a jego sygnał jest generowany przez 8 bitów nadawanych interfejsem SPI. Każdy taki bit powinien trwać 0,15625 µs, co przekłada się na częstotliwość taktowania peryferialu SPI równą 6,4 MHz. Aby uzyskać taką wartość, ustawiamy wartość częstotliwości „HCLK (MHz)” na 51,2 MHz oraz wartość dzielnika, w kolejnym etapie konfiguracji na 8 (**rysunek 3**).

Teraz przechodzimy do zakładki „Configuration”. Z pola „Connectivity” wybieramy pozycję „SPIx”, gdzie „x” odpowiada numerowi wybranego interfejsu SPI i przechodzimy do jego konfiguracji. Opcją, która zmieniamy, jest wartość preskalera wybrana w poprzednim kroku (**rysunek 4**).

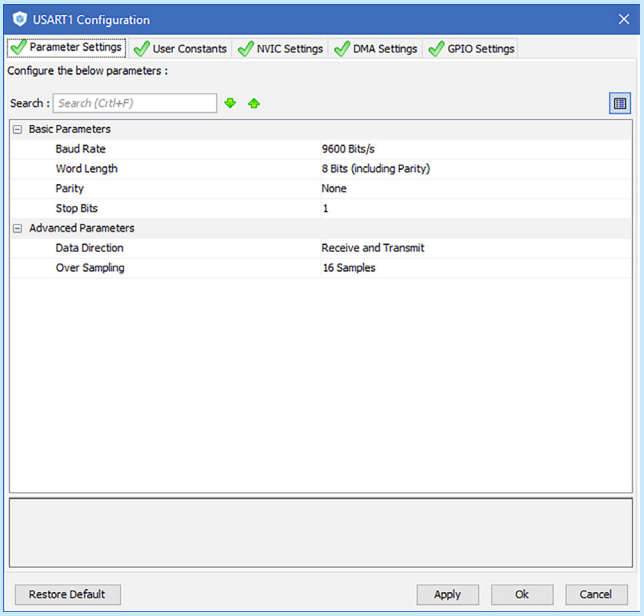
Listing 4. Plik Src/frame_parser.c

```
#include „frame_parser.h”

struct frame_parser_state frame_parser_init(ws2812b_config * _
led_strip_handler, uint8_t _diodes_count)
{
    struct frame_parser_state state;
    state.led_strip_handler = _led_strip_handler;
    state.diodes_count = _diodes_count;
    for(uint16_t i=0; i<800; i++) state.line_buffer[i] = ,\0’;
    state.writer_position = 0;
    state.reader_position = 0;
    for(uint8_t i=0; i<30; i++) state.field_buffer[i] = ,\0’;
    state.field_position = 0;
    state.process_frame = 0;
    return state;
}

void frame_parser_recv_char(struct frame_parser_state * state,
uint8_t recv_char)
{
    if (state->writer_position == 0 && recv_char == ,@’)
    {
        state->writer_position++;
    } else if (state->writer_position >= 1 && state->writer_po-
sition < 799)
    {
        if (recv_char == ,\r’ || recv_char == ,\n’)
        {
            state->line_buffer[state->writer_position - 1] =
,\0’;
            state->writer_position = 0;
            state->process_frame = 1;
        } else
        {
            state->line_buffer[state->writer_position - 1] =
recv_char;
            state->writer_position++;
        }
    } else
    {
        state->writer_position = 0;
    }
}

void frame_parser_read_field(struct frame_parser_state * state)
{
    state->field_position = 0;
    while(state->line_buffer[state->reader_position] != ,,’ &&
state->line_buffer[state->reader_position] != ,\0’
          && state->field_position < 29) {
        state->field_buffer[state->field_position] = state->line_
buffer[state->reader_position];
        state->reader_position++;
        state->field_position++;
    }
    state->field_buffer[state->field_position] = ,\0’;
    state->reader_position++;
}

void frame_parser_process_frame(struct frame_parser_state *
state)
{
    if(state->process_frame == 0) return;
    state->process_frame = 0;
    state->reader_position = 0;
    for(int i=0; i<30; i++)
    {
        ws2812b_color rgb;
        frame_parser_read_field(state);
        sscanf(state->field_buffer, „%d”, &(rgb.red));
        frame_parser_read_field(state);
        sscanf(state->field_buffer, „%d”, &(rgb.green));
        frame_parser_read_field(state);
        sscanf(state->field_buffer, „%d”, &(rgb.blue));
        ws2812b_set_diode_color(state->led_strip_handler, i,
rgb);
    }
    ws2812b_refresh(state->led_strip_handler);
}
```

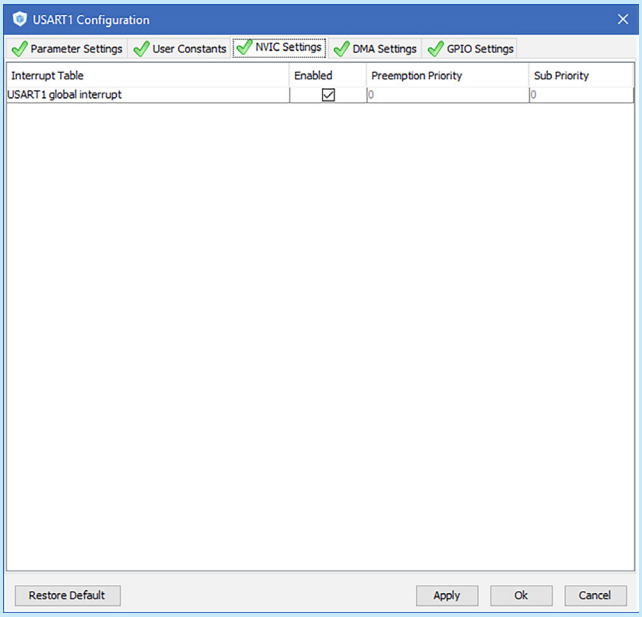
Teraz przechodzimy do konfiguracji UART. Ponownie z pola „Connectivity” wybieramy pozycję „USARTx”. Tym razem jednak musimy zmienić inny parametr – „Baud Rate”. W tym polu wprowadzamy „9600 Bits/s” (**rysunek 5**). Następnie, nie wychodząc z okna konfiguracji peryferialu USART, przechodzimy do zakładki „NVIC Settings” i zaznaczamy tam jedyne znajdujące się na liście przerwanie (**rysunek 6**). Zatwierdzamy zmiany przyciskiem „OK”.

Na tym etapie możemy już zapisać ustawienia i wygenerować projekt, a następnie zaimportować go w środowisku IDE w sposób opisany w poprzednich artykułach – klikamy ikonę zębatki na pasku narzędziowym, wybieramy w polu „Toolchain / IDE” opcję „SW4STM32” i zapisujemy pliki projektu w wybranym miejscu (**rysunek 7**). Następnie otwieramy IDE System Workbench for STM32, zamykamy planszę powitalną (X), w ramce „Project Explorer” klikamy prawym przyciskiem myszy i z menu kontekstowego wybieramy opcję „Import...”, w nowym oknie, klikamy kolejno: „General”, „Existing Projects into Workspace”, „Next”,

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86735, pass: 6mqh264k



Rysunek 5. Ustawianie parametrów pracy peryferialu USART w generatorze konfiguracji STM32CubeMX



Rysunek 6. Włączanie przerwania peryferialu USART w generatorze konfiguracja STM32CubeMX

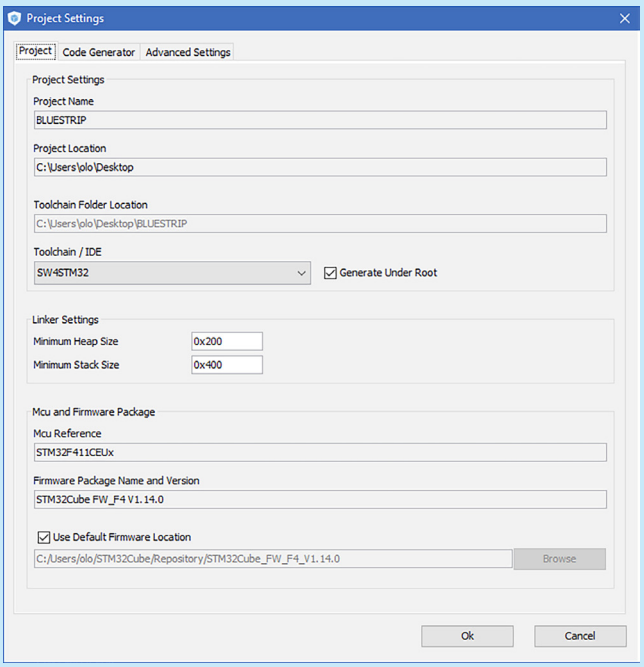
wybieramy lokalizację plików projektu oraz zatwierdzamy import przyciskiem „Finish”.

Aby możliwe było korzystanie z wartości zmiennoprzecinkowych, w funkcjach *printf()*, *sprintf()*, *scanf()* oraz *sscanf()*, konieczne jest dodanie parametrów „ -u _printf_float -u _scanf_float” do linii polecenia linkera. Robimy to, klikając prawym przyciskiem myszy na nazwę nowego projektu, z menu kontekstowego, wybierając pozycję „Properties” oraz nawigując do „C/C++ Build” -> „Settings” -> „Miscellaneous” i dopisując do pola „Linker flags” wartość: „ -u _printf_float -u _scanf_float” (**rysunek 8**).

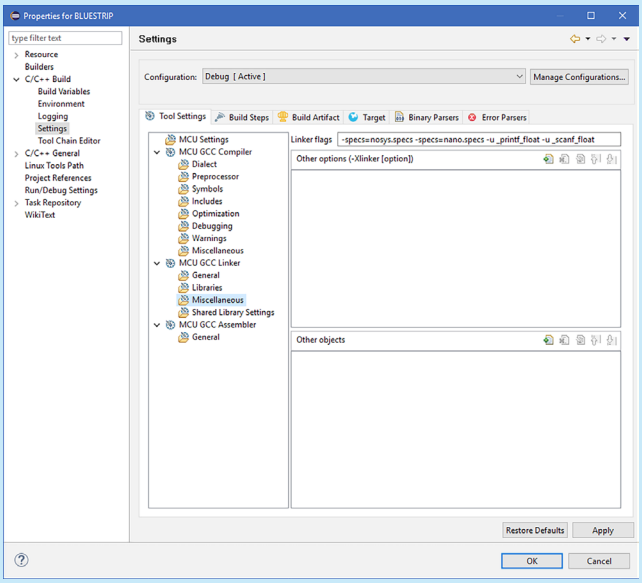
Teraz dodamy do projektu cztery pliki, które wykorzystamy do obsługi diod oraz interpretacji opisanych wcześniej ramek (**rysunek 9**). W tym celu, w „Project Explorerze” rozwijamy katalog projektu, klikamy PPM na znajdujący się w nim podkatalog „Inc” i z menu kontekstowego wybieramy opcję „New” -> „File”. W nowym oknie podajemy nazwę pliku, który chcemy utworzyć – „ws2812b.h”, klikamy „Finish”. Czynności ponawiamy również dla pliku „frame_parser.h”. Następnie, w podobny sposób, do katalogu „Src” dodajemy dwa kolejne pliki: „ws2812b.c” oraz „frame_parser.c”.

Listing 5. Plik Inc/ws2812b.h

```
#ifndef ws2812b_header
#define ws2812b_header
#include „stm32f4xx_hal.h”
#include „spi.h”

typedef struct ws2812b_color
{
    uint8_t red, green, blue;
} ws2812b_color;

typedef struct ws2812b_config
{
    SPI_HandleTypeDef * spi_handler;
    uint16_t diodes_count;
    ws2812b_color * colors_array;
} ws2812b_config;

ws2812b_config ws2812b_init(SPI_HandleTypeDef * spi_handler,
uint16_t diodes_count);
void ws2812b_set_diode_color(ws2812b_config * config, uint16_t di-
ode_id, ws2812b_color color);
void ws2812b_refresh(ws2812b_config * config);

#endif
```



Rysunek 7. Eksport projektu z generatora konfiguracji STM32CubeMX



Rysunek 8. Dodawanie parametrów wywołania linkera w opcjach projektu programu System Workbench for STM32

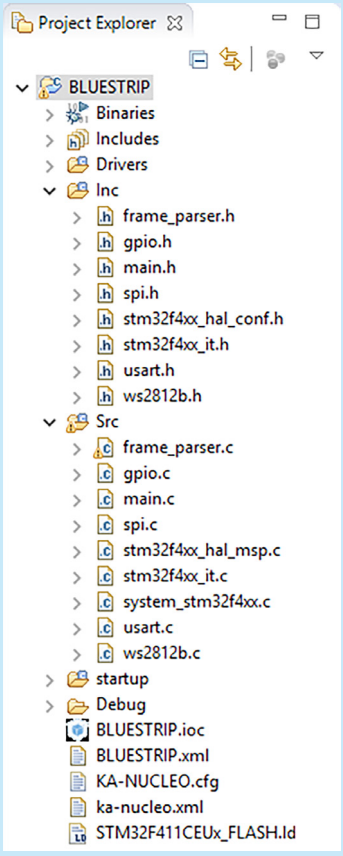
Listing 6. Plik Src/ws2812b.c

```
#include „ws2812b.h”

ws2812b_config ws2812b_init(SPI_HandleTypeDef * spi_handler,
uint16_t diodes_count)
{
    ws2812b_config config;
    config.spi_handler = spi_handler;
    config.diodes_count = diodes_count;
    config.colors_array = calloc(diodes_count, sizeof(ws2812b_
color));
    return config;
}

void ws2812b_set_diode_color(ws2812b_config * config, uint16_t di-
ode_id, ws2812b_color color)
{
    config->colors_array[diode_id] = color;
}

void ws2812b_refresh(ws2812b_config * config)
{
    const uint8_t zero = 0b00011111;
    const uint8_t one = 0b00000111;
    uint8_t buffer[30 * 24];

    for (uint16_t i = 0, j = 0; i < config->diodes_count; i++)
    {
        // Green
        for (int16_t k = 7; k >= 0; k--)
        {
            if ((config->colors_array[i].green & (1 << k)) == 0)
buffer[j] = one;
            else buffer[j] = zero;
            j++;
        }
        // Red
        for (int16_t k = 7; k >= 0; k--)
        {
            if ((config->colors_array[i].red & (1 << k)) == 0)
buffer[j] = one;
            else buffer[j] = zero;
            j++;
        }
        // Blue
        for (int16_t k = 7; k >= 0; k--)
        {
            if ((config->colors_array[i].blue & (1 << k)) == 0)
buffer[j] = one;
            else buffer[j] = zero;
            j++;
        }
    }
    HAL_SPI_Transmit(config->spi_handler, &buffer, config->diodes_
count * 24, 1000);
    HAL_Delay(1);
}
```

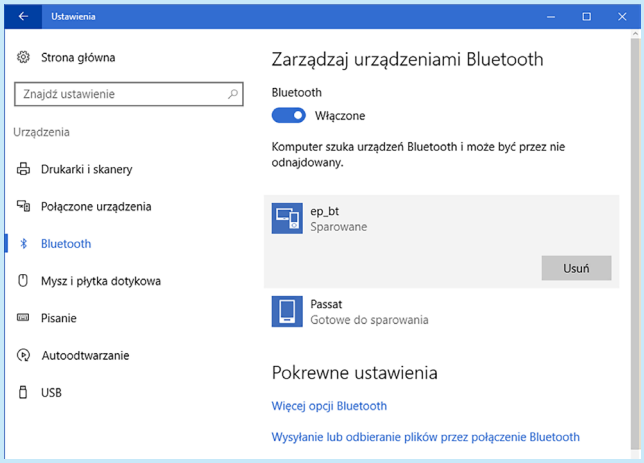
Dalej, uzupełniamy nowo dodane pliki zawartością **listingów 3...6** oraz modyfikujemy zawartość pliku „main.c”, zgodnie z **listingiem 2**. Kompilujemy program i wgrywamy do pamięci mikrokontrolera (ikony młotka i robaka na pasku narzędziowym).

Po wykonaniu wszystkich powyższych kroków parujemy nasz komputer z modułem Bluetooth. Pod Windowsem 10 uruchamiamy „Ustawienia” i przechodzimy do zakładki „Urządzenia” → „Bluetooth’. Tam odnajdujemy urządzenie „ep_bt”. Zaznaczamy je, klikamy „Sparuj” i podajemy pin kod „3498”. Pod Linuksem jest to zależne od używanego środowiska graficznego (**rysunek 10**).

Teraz otwieramy Menedżer urządzeń i rozwijamy zakładkę „Porty (COM i LPT)”. Zapamiętujemy (lub zapisujemy) numer portu COM



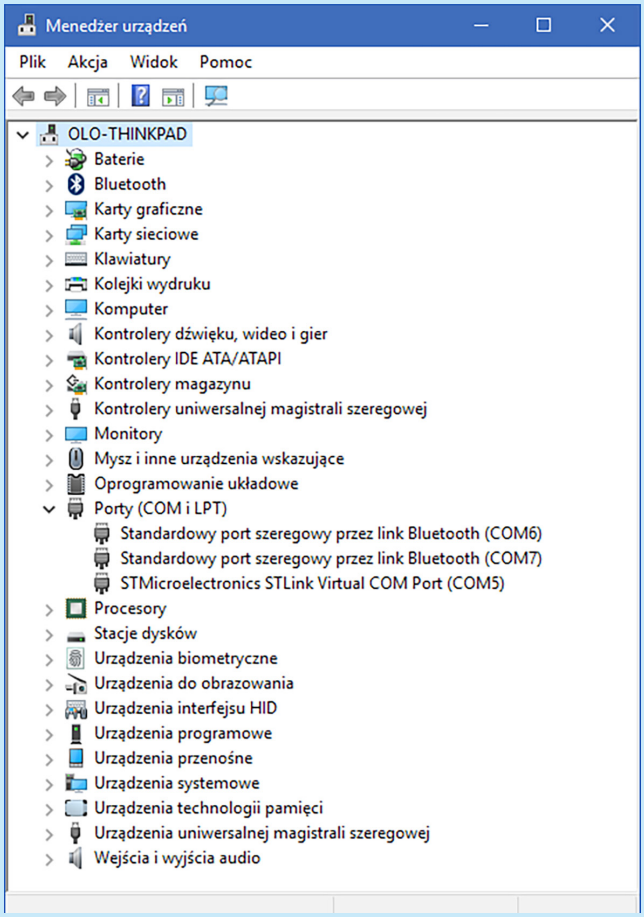
Rysunek 9. Drzewo katalogów po poprawnym dodaniu plików



Rysunek 10. Parowanie urządzeń Bluetooth w systemie Windows 10

pierwszego urządzenia Bluetooth (**rysunek 11**). Pod Linuksem identyfikator portu nowego urządzenia możemy sprawdzić poleceniem dmesg.

Nie pozostaje nam już nic innego jak otworzyć port szeregowy w programie PuTTY i przesłać do mikrokontrolera ramkę danych. W polu „Connection Type:” zaznaczamy opcję „Serial”, w polu „Serial line” podajemy identyfikator portu szeregowego, a w polu „Speed”, szybkość połączenia 9600 bps. Następnie, klikamy przycisk „Open” i w oknie konsoli, które zostanie wyświetlone po chwili oczekiwania wklejamy testową ramkę danych (**listing 7**) oraz klikamy „Enter” (**rysunek 12**). Na pasku LED, na kolejnych diodach, powinny pojawić się na przemian kolory: czerwony, zielony i niebieski.

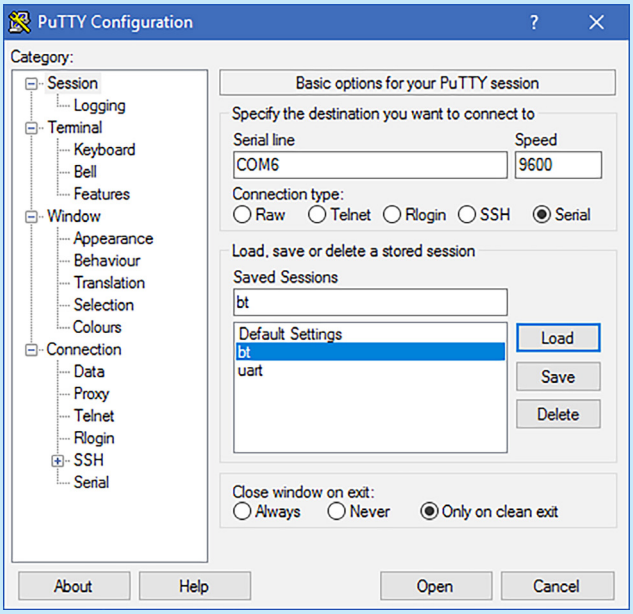


Rysunek 11. Wirtualne portu szeregowe profilu portu szeregowego widoczne w Menedżerze urządzeń w systemie Windows 10

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86735, pass: 6mqh264k

Biblioteka ws2812b.h/ws2812b.c oraz sposób generowania sygnału sterującego dla diod zostały opisane w EP 3/2017. Funkcja *ws2812b_init()* tworzy strukturę przechowującą konfigurację, bufory danych i uchwyt na podobną strukturę opisującą używany interfejs SPI. Wywołując ją, musimy podać wskaźnik na uchwyt interfejsu SPI oraz liczbę diod, którymi chcemy sterować. Kolejna funkcja *ws2812b_set_diode_color()* manipuluje jedynie na buforach danych w tej strukturze. Przesyłamy do niej wskaźnik na uchwyt wygenerowany przez funkcję *ws2812b_init()*, numer diody oraz pożądany kolor w formie struktury *ws2812b_color* zawierającej trzy wartości składowe – natężenie barw czerwonej, zielonej i niebieskiej w formie trzech wartości jednobajtowych. Następna funkcja *ws2812b_refresh()*, do której wywołania potrzebujemy przesłać jedynie wskaźnik na strukturę konfiguracyjną, generuje właściwy sygnał sterujący i ustawia kolory zapisane w buforze, na kolejnych diodach na pasku.

Biblioteka *frame_parser.h/frame_parser.c* odpowiada za interpretację danych otrzymanych przez interfejs UART z modułu Bluetooth. W odpowiednich momentach wywołuje ona funkcje biblioteki ws2812b, ustawiając kolor konkretnej diody na podstawie danych odebranych z ramek lub wywołując funkcję generującą sygnał - *ws2812b_refresh()*, gdy nie jest przetwarzane żadne przerwanie. Podobnie jak poprzednio, funkcja *frame_parser_init()* generuje strukturę przechowującą zmienne konfiguracyjne, bufory danych oraz uchwyt biblioteki *ws2812b* (wygenerowany przez funkcję *ws2812b_init()*). Funkcja *frame_parser_recv_char()* wywoływana jest w momencie otrzymania nowego bajtu danych - znaku ASCII przez interfejs UART, dodaje on do bufora nowy znak, a w odpowiednim momencie (po odebraniu znacznika końca ramki), ustawia ona flagę wymuszającą interpretację nowej ramki przez kolejną omawianą funkcję. Funkcja *frame_parser_process_frame()* wywoływana jest ciągle, w pętli głównej programu, gdy zachodzi taka potrzeba - gdy ustawiona jest flaga wymuszająca interpretację ramki, funkcja odczytuje z odebranej ramki wartości natężeń poszczególnych składowych kolorów kolejnych diod i ustawia na diodach te kolory, a dalej - po zinterpretowaniu całej ramki, wywołuje funkcję *ws2812b_refresh()*, generująca sygnał sterujący.



Rysunek 12. Konfiguracja połączenia szeregowego w programie PuTTY

```
Listing 7. Testowa ramka danych
@255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0
,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0
,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255
,255,0,0,0,255,0,0,0,255,255,0,0,0,255,0,0,0,255
```

W pliku *main.c*, w sekcji USER CODE 0, tworzone są zmienne przechowujące uchwyty dla obu bibliotek oraz dwie funkcje - *HAL_UART_RxCpltCallback()*, wykonywana w przerwaniu, w momencie otrzymania nowego znaku przez interfejs UART oraz *setup_uart()*, wysyłająca do modułu Bluetooth, polecenia sterujące. Następnie, w sekcji USER CODE 2, inicjowane są uchwyty obu bibliotek, wywoływana jest funkcja *uart_setup()* oraz uruchamiana jest obsługa przerwań peryferialu UART. W sekcji USER CODE 3 ciągle wywoływana jest, omówiona powyżej, funkcja *frame_parser_process_frame()*.

Aleksander Kurczyk

ULUBIONY**KIOSK.PL to:**

- egzemplarze wydań papierowych w cenie okładkowej – **przesyłka zawsze gratis**
- egzemplarze **e-wydań** – wygodny sposób zamawiania wydań elektronicznych
- egzemplarze archiwalne – prosty sposób na **wyszukiwanie brakujących wydań**
- prenumerata realizowana w trybie przedpłaty oraz stałego zlecenia bankowego
- możliwość założenia **własnej „Teczki”** by zamawiać magazyny jeszcze łatwiej
- **Klub Ulubionego Kiosku** – zbieraj punkty i wymieniaj je na atrakcyjne nagrody

ULUBIONY KIOSK.PL

WWW.ULUBIONYKIOSK.PL