Programowanie STM32F4 (6)

Zajmiemy się sterowaniem adresowalnymi paskami diod LED WS2812B. Mowa o elastycznych listwach oświetleniowych z giętkiego laminatu, na których znajdują się połączone równolegle diody LED z wbudowanym kontrolerem. Listwy takie możemy dowolnie docinać (w wyznaczonych miejscach), łączyć i przyklejać w miejscach, które chcemy oświetlić.

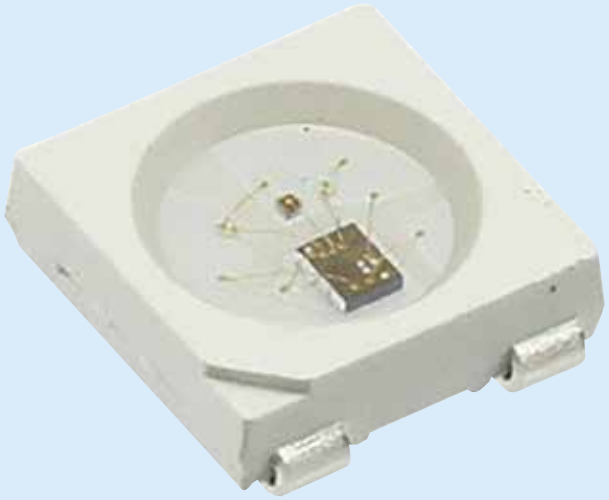
Na paskach są montowane diody świecące w jednym kolorze, najczęściej białym, choć występują one także w innych kolorach. Istnieją też paski z diodami RGB, pozwalające na sterowanie kolorem emitowanego światła. Sterujemy wtedy kolorem całego paska, a nie poszczególnych diod. Mają one wyprowadzone (na łączeniach i w miejscach cięcia) styki wspólnej anody oraz trzech katod, diod odpowiadających za świecenie w trzech podstawowych kolorach – czerwonym, zielonym i niebieskim, znajdujących się w każdej „diodzie RGB". Kolorem tych pasków możemy sterować przy użyciu tranzystorów MOSFET oraz generatora sygnału PWM, opisanego w części drugiej tego cyklu.

Paski, którymi zajmiemy się w tej części, pozwalają na indywidualne ustawianie koloru każdej diody. Dawniej, paski „adresowalne" miały osobne układy scalone sterujące barwą, montowane pomiędzy diodami. Zazwyczaj takie układy obsługiwały po kilka diod, ustawiając na nich jeden i ten sam kolor. Obecnie, chipy te montowane są w samych diodach, co pozwoliło na obniżenie ich ceny i umożliwia nam sterowanie każdą diodą osobno. Diody WS2812B z chipami (**fotografia 1**) pozwalają na ustawianie natężenia każdej z barw składowych (RGB) w zakresie od 0 do 255, co daje nam przestrzeń 16777216 różnych kolorów ustawianych na każdej diodzie. Częstym ich zastosowaniem jest podświetlenie typu *ambient light* za telewizorem lub monitorem, wizualizacje w rytm muzyki, czy lampki choinkowe.

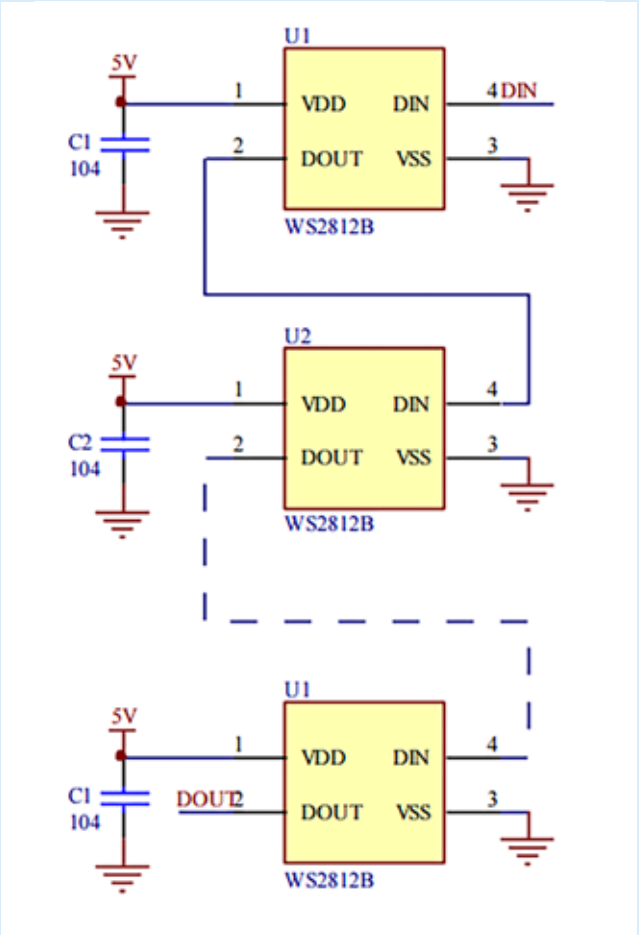
Jak sterować diodami?

Każda dioda na pasku ma 4 wyprowadzenia: piny zasilania i masy oraz pin wejściowy i wyjściowy przebiegu sterującego. Diody podłączone są w szeregu w taki sposób, że pin wejściowy sygnału sterującego pierwszej z nich (DIN) połączony będzie do początku paska i dalej do naszego układu, pin DIN kolejnej, i kolejnych w szeregu, do pinu wyjściowego (DOUT) poprzedniej, jak na **rysunku 2**.

Diody należy zasilać napięciem od 3,5 V do 5,3 V. Krótki pasek składający się z kilkunastu diod możemy zasilić z pinu „5V" płytki rozwojowej KA-NUCELO lub portu USB komputera. Dla dłuższych potrzebujemy zaopatrzyć się w zewnętrzny zasilacz (najlepiej +5 V). Należy też wtedy pamiętać o spadku napięcia występującym na pasku, w miarę zwiększania się odległości od źródła zasilania. Już przy 5 metrach jesteśmy w stanie zaobserwować spadek jasności świecenia diod spowodowany niższym napięciem zasilania przy końcu paska. Zatem dłuższe paski należy zasilać w kilku punktach. Jeśli korzystamy z osobnego zasilacza, musimy także połączyć masy każdego z obwodów (należy to zrobić przed podłączeniem zasilania).



Fotografia 1. Dioda z chipem WS2812B. Chip jest widoczny w postaci dużego elementu, od którego odchodzą połączenia do diod, na „tarczy" diody



Rysunek 2. Schemat podłączenia diod na pasku LED

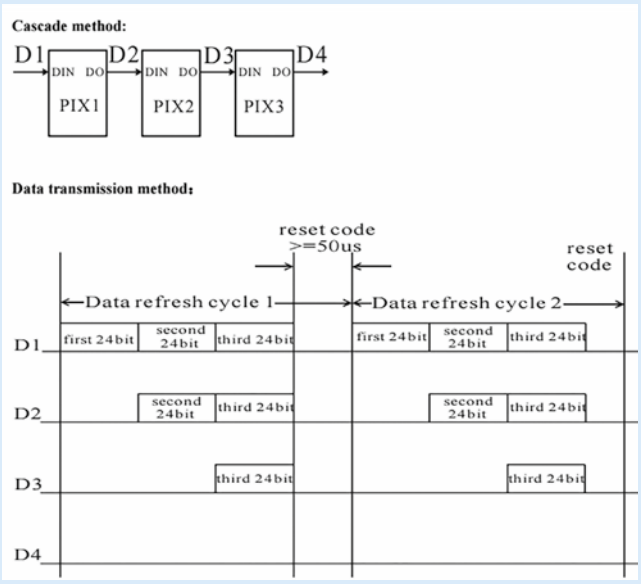
Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77322, pass: 8qxonzsb

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

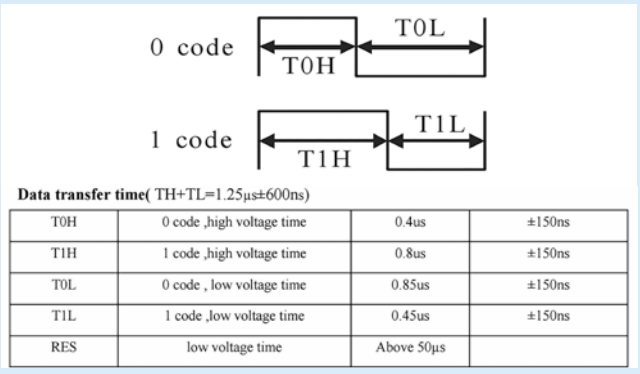
Note: Follow the order of GRB to sent data and the high bit sent at first.

Rysunek 3. Struktura bitów kodujących kolor poszczególnych diod LED w transmitowanym sygnale



Rysunek 4. Cykle odświeżania

Na pinach sterujących, diody odbierają i wysyłają dane z logiką 5 V – tzn. 5 V oznacza poziom logiczny wysoki, a 0 V – niski. Mimo tego możemy transmitować przebieg sterujący z napięciem 3,3 V, występującym na płytce rozwojowej KA-NUCLEO.

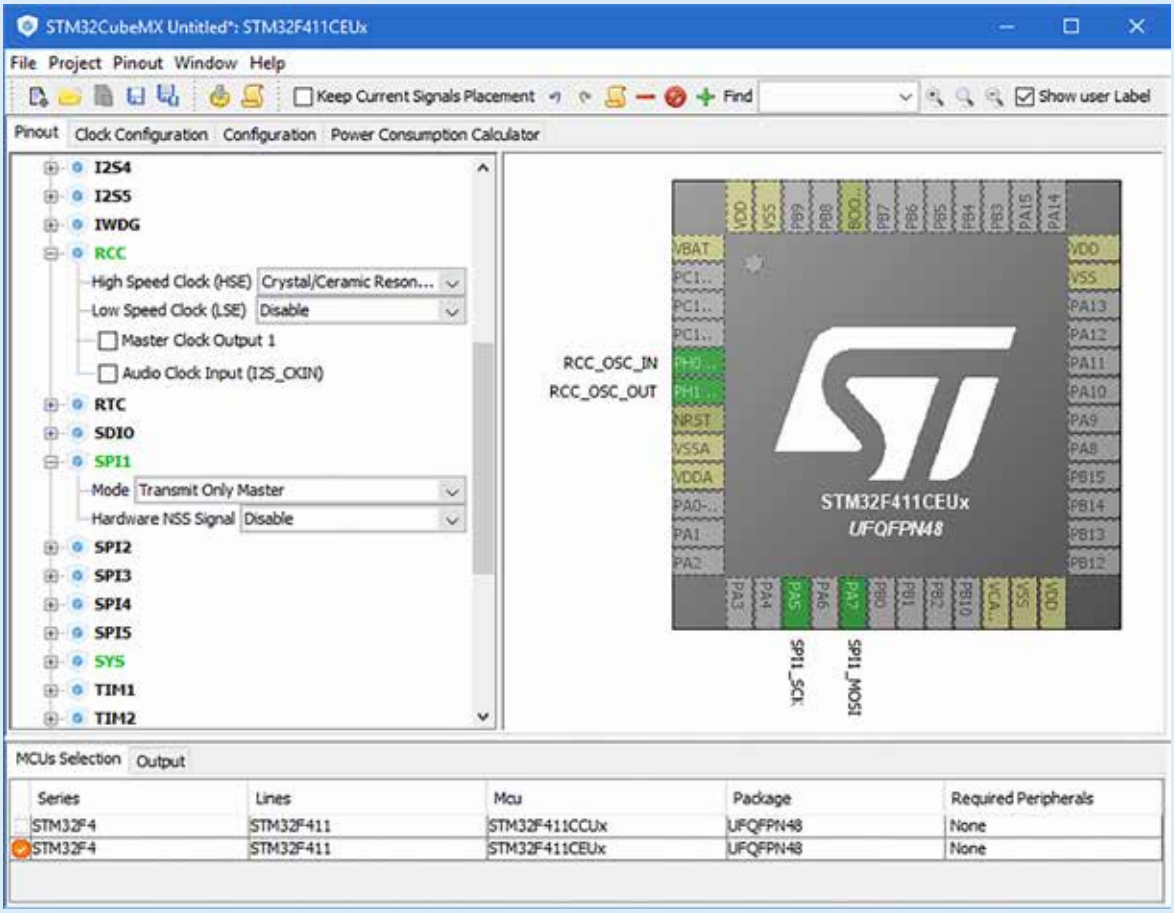


Data transfer time(TH+TL=1.25μs±600ns)

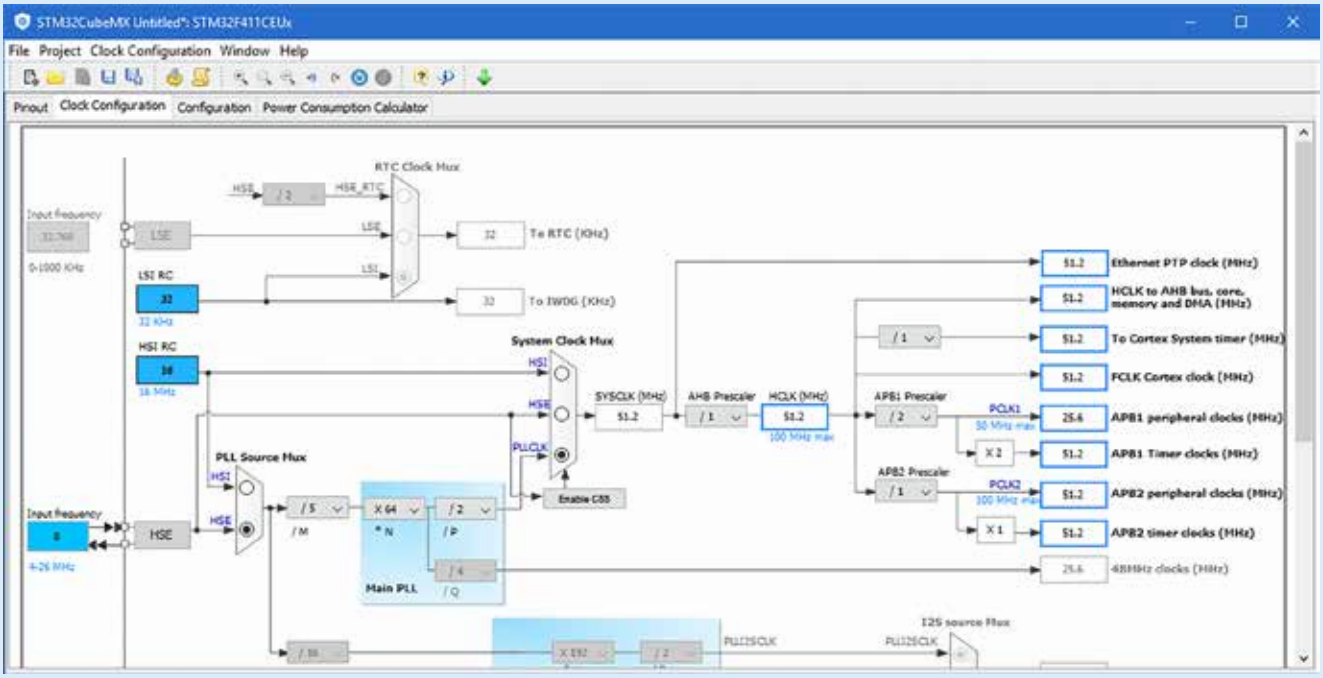
T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Rysunek 5. Czasy transmisji sygnałów kodujących bity

Sygnał sterujący przesyłany do diody składa się z serii impulsów kodujących zera i jedynki. Każde przesłane 24 bity kodują kolor kolejnej diody w szeregu, przenosząc 8-bitowe wartości jasności poszczególnych jego barw składowych, kolejno: zielonej, czerwonej oraz niebieskiej. Każda dioda odbiera i interpretuje pierwsze otrzymane 24 bity strumienia, pozostałe wartości przesyłając do kolejnej diody (**rysunek 3**). Tak, więc, pierwsza nadana struktura kodująca kolor odnosi się do pierwszej diody na pasku, druga do kolejnej, itd. Aby ponownie ustawić kolor na pierwszej i kolejnych diodach w szeregu, należy zrobić przerwę w transmisji, trwającą co najmniej 50 mikrosekund (**rysunek 4**).



Rysunek 6. Konfiguracja wyprowadzeń w generatorze konfiguracji STM32CubeMX



Rysunek 7. Konfiguracja sygnału taktującego w generatorze konfiguracji STM32CubeMX

Wysyłanie danych nie jest jednak takie łatwe. Nie wystarczy ustawić poziomu logicznego wysokiego lub niskiego na ustalony czas, aby nadać jedynkę, czy zero. Diody z chipami WS2812B wymagają stosowania konkretnego kodowania. Przesyłanie każdego bitu trwa 1,25 mikrosekundy, przy czym dopuszczalne odchylenie wynosi 0,6 mikrosekundy. Aby nadać logiczną jedynkę, musimy ustawić pin na czas 0,8 ±0,15 µs, po czym wyzerować go i utrzymać przez 0,45 ±0,15 µs. Dla logicznego zera, czasy te wynoszą odpowiednio: 0,4 ± 0,15 µs poziomu wysokiego oraz 0,85 ±0,15 µs poziomu niskiego (**rysunek 5**).

Jak to wykonamy?

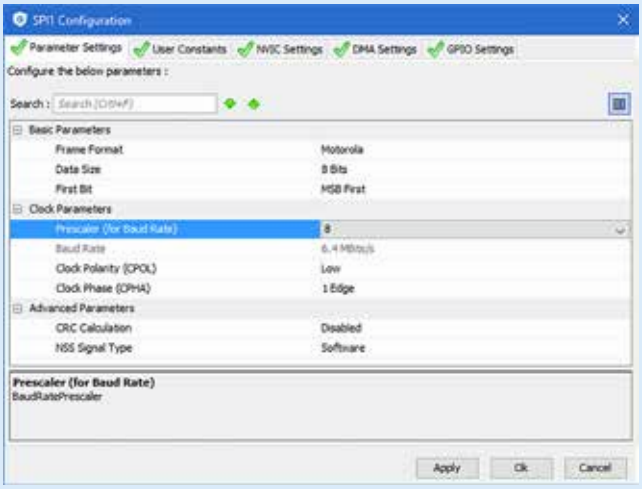
Brzmi dość łatwo, jeśli jednak zechcemy zrobić to w typowy, „arduinowy" sposób, ustawiając lub zerując pin, a następnie odczekując przez ustalony czas za pomocą funkcji *HAL_Delay()*, szybko zorientujemy się, że ten czas jest zbyt krótki. Diody oczekują od nas transmisji ze stosunkowo dużą szybkością i nie stosują żadnego typowego interfejsu, jak UART, SPI itp. W ten sposób – dodając aktywne oczekiwanie – tracimy całą moc mikrokontrolera w trakcie czekania, a przychodzące przerwanie, jest w stanie zepsuć nam całą transmisję.

Co więc możemy zrobić? Możemy użyć interfejsu SPI. Nie użyjemy go jednak do transmisji danych, a posłużymy się do wygenerowania sekwencji zera (11100000) i jedynki (11111000) oczekiwanych przez diodę. Każde 8 bitów nadane interfejsem SPI będzie generowało sygnał odpowiadający pojedynczemu bitowi dla diod – „jedynce" lub „zeru". Z pewnymi zmianami w kodzie, możliwe by było też nadawanie jednego bitu dla diody za pomocą 4 lub 3 bitów przesyłanych przez SPI, byłoby to jednak trochę bardziej skomplikowane do wykonania i wyjaśnienia.

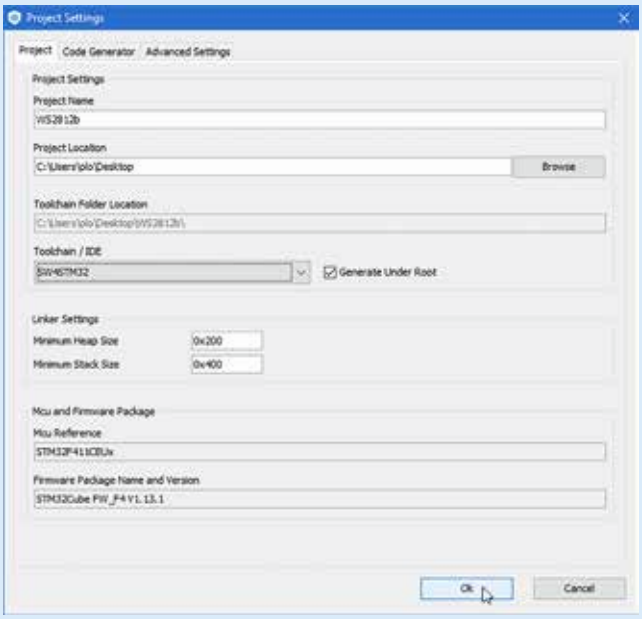
Interfejs SPI może pracować z dowolnie wybraną szybkością, przesyłając oprócz sygnału danych, również sygnał zegarowy. Dzięki jego obecności nie ma konieczności stosowania ramek, bitów stopu, ani startu – dane przesyłane są ciągle, a to pozwala nam na transmisję dowolnych ciągów zapisanych uprzednio w buforze.

Tworzymy nowy projekt

Po pierwsze, za pomocą STM32CubeMX tworzymy nowy projekt wybierając posiadany przez nas mikrokontroler (mój to STM32F411CEU6). Na pierwszym ekranie konfiguratora, zatytułowanym „Pinout", ustawiamy źródło sygnału taktującego dołączonego



Rysunek 8. Ustawienia interfejsu SPI w STM32CubeMX



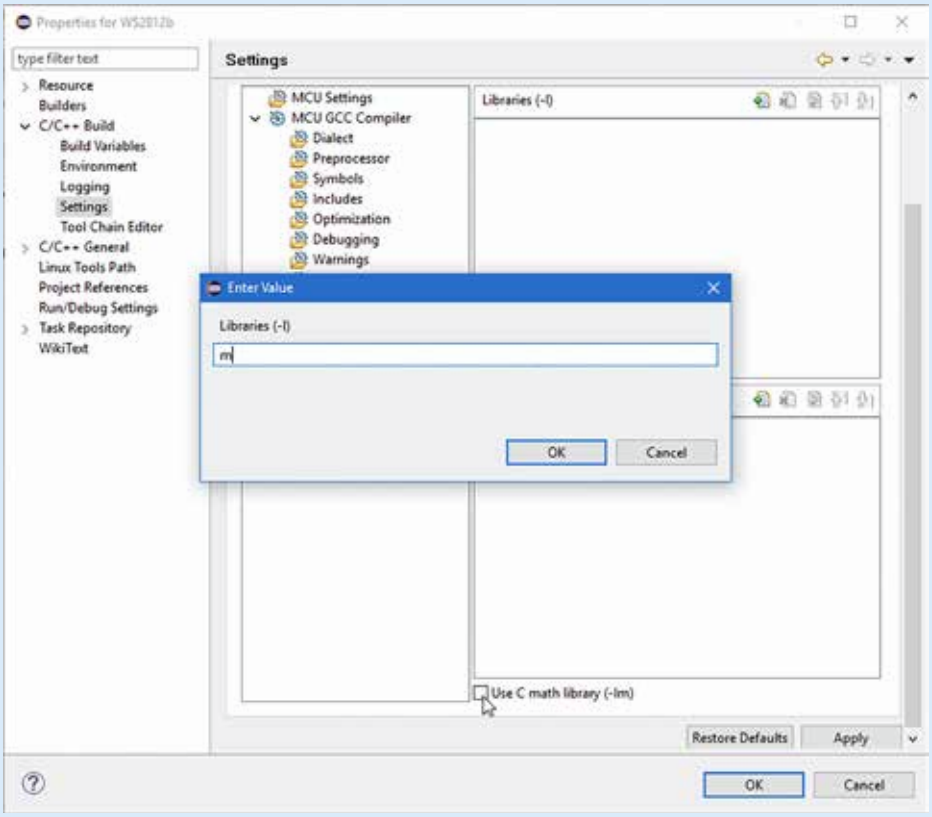
Rysunek 9. Ustawienia projektu w programie STM32CubeMX

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77322**, pass: **8qxonzsb**

do układu – 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. Wykorzystywany przeze mnie układ dysponuje aż 5 takimi interfejsami. Nie ma znaczenia, który z nich wybierzemy, trzeba jednak pamiętać, że peryferiale SPI1, SPI4 i SPI5 otrzymują taktowanie z szyny APB1, a SPI2 oraz SPI3 z szyny APB2. W przykładzie, użyję interfejsu SPI1 oraz pinów PA5 oraz PA7, odpowiednio w roli SPI1_SCK – sygnału zegara oraz SPI1_MOSI – wyjścia danych, ponieważ są one dostępne do dyspozycji użytkownika na płytce rozwojowej KA-NUCLEO.

Oprócz tych pinów, interfejs SPI może korzystać również z pinu SPI1_MISO – wejścia danych, a także pracować w wielu trybach – my korzystać będziemy z trybu „Transmit Only Master”, co oznacza, że nasze urządzenie będzie urządzeniem głównym w transmisji – dostarczy sygnał zegara urządzeniom podrzędnym (nie jest on potrzebny diodom) i będziemy tylko nadawać. W tym celu, z listy po lewej stronie okna, wybieramy odpowiedni interfejs SPI oraz w polu „Mode”, ustawiamy opcję „Transmit Only Master”. Pozostałe tryby pracy pozwalają na: tylko odbiór od urządzeń podrzędnych – „Receive Only Master”, nadawanie i odbiór na tym samym pinie – „Half-Duplex Master”, nadawanie i odbiór na różnych pinach (MISO – Master Input Slave Output i MOSI – Master Output Slave Input) – „Full-Duplex Master”, a także na pracę w charakterze urządzenia podrzędnego, korzystającego z sygnału zegara dostarczanego przez inne urządzenie – wszystkie opcje z „Slave” w nazwie. Jeśli interfejs SPI wykorzystywany jest do komunikacji z wieloma urządzeniami podrzędnymi, często stosuje się także piny „Slave Select”. Są to zwykłe piny GPIO ustawione w tryb pracy wyjścia. Gdy na pinie SS, podłączonym do urządzenia podrzędnego, pojawi się stan logiczny wysoki, urządzenie to może odbierać i nadawać dane. Pinów takich potrzebujemy, zatem tyle ile urządzeń podłączamy do pojedynczego interfejsu (**rysunek 6**).

W kolejnej zakładce – „Clock Configuration”, zwyczajowo ustawiamy częstotliwość pracy wejściowego oscylatora kwarcowego – u mnie 8 MHz, przełączamy źródło sygnału wchodzącego 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”.



Rysunek 10. Dodawanie biblioteki „m” w ustawieniach linkera

Teraz musimy jeszcze ustawić pożądaną częstotliwość taktowania naszego układu. Nie będzie to jednak, jak poprzednio, maksymalna dozwolona wartość, ani najniższa potrzebna. Musimy dobrać ją, wspólnie z dzielnikiem częstotliwości wchodzącej na peryferial SPI, tak, aby możliwa była transmisja sygnału jedynki i zera, dla diod, w odpowiednim czasie.

Częstotliwość sygnału sterującego

Transmisja sygnału pojedynczego bitu dla diody powinna trwać 1,25 µs. Jednak, ponieważ sygnał ten generowany jest przez 8 bitów przesyłanych interfejsem SPI, pojedynczy bit nadawany przez SPI powinien być ustawiony na pinie przez 0,15625 µs. To przekłada się na częstotliwość pracy interfejsu SPI wynoszącą aż 6,4 MHz – w ciągu sekundy nadać musimy 6,4 mln bitów.

Jak już wspominałem, peryferiale SPI1, SPI4 i SPI5 otrzymują sygnał taktujący z szyny APB1, a SPI2 oraz SPI3 z szyny APB2. Maksymalna częstotliwość obsługiwana przez szynę APB1, w wykorzystywanym przeze mnie mikrokontrolerze, to 50 MHz, peryferiale taktowane z szyny APB2 mogą być taktowane z maksymalną częstotliwością pracy układu. Każdy interfejs SPI posiada również wbudowany dzielnik, pozwalający podzielić wejściową częstotliwość przez liczby będące potęgami liczby 2, z zakresu od 2 do 256. Ponieważ musimy uzyskać częstotliwość 6,4 MHz, możliwe do ustawienia częstotliwości wchodzące na szynę dostarczającą taktowanie do układu (równą zazwyczaj głównej częstotliwości taktowania lub jej połowie) oraz wartości podzielnika to:

- PCLK1/PCLK2=102,4 MHz; Prescaler=16,
- PCLK1/PCLK2=51,2 MHz; Prescaler=8,
- PCLK1/PCLK2=25,6 MHz; Prescaler=4,
- PCLK1/PCLK2=12,8 MHz; Prescaler=2.

W omawianym przykładzie użyto częstotliwości 51,2 MHz oraz wartość dzielnika równa 8, z tego względu, że 100 MHz

Listing 1. Plik nagłówkowy ws2812b.h

```
#ifndef ws2812b_header
#define ws2812b_header

#include „stm32f4xx_hal.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
```

Listing 2. Plik biblioteki 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 diode_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++) {
        // Zielony
        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++;
        }
        // Czerwony
        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++;
        }
        // Niebieski
        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);
}
```

Listing 3. Modyfikacja pliku *main.c*

```
/* USER CODE BEGIN Includes */
#include „ws2812b.h”
/* USER CODE END Includes */
/* USER CODE BEGIN 2 */
ws2812b_config ws2812b = ws2812b_init(&hspi1, 30);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
    ws2812b_color color1, color2, color3;
    color1.red = 255; color1.green = 0; color1.blue = 0;
    color2.red = 0; color2.green = 255; color2.blue = 0;
    color3.red = 0; color3.green = 0; color3.blue = 255;
    ws2812b_set_diode_color(&ws2812b, 0, color1);
    ws2812b_set_diode_color(&ws2812b, 1, color2);
    ws2812b_set_diode_color(&ws2812b, 2, color3);
    ws2812b_refresh(&ws2812b);
    HAL_Delay(1000);
}
/* USER CODE END 3 */
```

to maksymalna częstotliwość obsługiwana przez wykorzystywany przeze mnie układ – STM32F411CEU6. Wybraną częstotliwość lub jej dwukrotność (w zależności od tego, z której szyny APB korzystamy), ustawiamy w polu „HCLK (MHz)” (**rysunek 7**) i przechodzimy do zakładki „Configuration”. Z pola „Connectivity”, wybieramy pozycję „SPIx”, gdzie x to nr wybranego interfejsu SPI i przechodzimy do jego konfiguracji. Opcją, która zmieniamy jest wartość preskalera, wybrana w poprzednim kroku (**rysunek 8**).

Teraz możemy już zapisać ustawienia (**rysunek 9**) i wygenerować projekt, a następnie zaimportować go w środowisku IDE, w sposób przedstawiony w poprzednich częściach – klikamy ikonę zębatki na pasku narzędziowym, wybieramy w polu „Toolchain / IDE” opcję „SW4STM32” i zapisujemy pliki projektu w wybranym miejscu. 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”, wybieramy lokalizację plików projektu oraz zatwierdzamy import przyciskiem „Finish”.

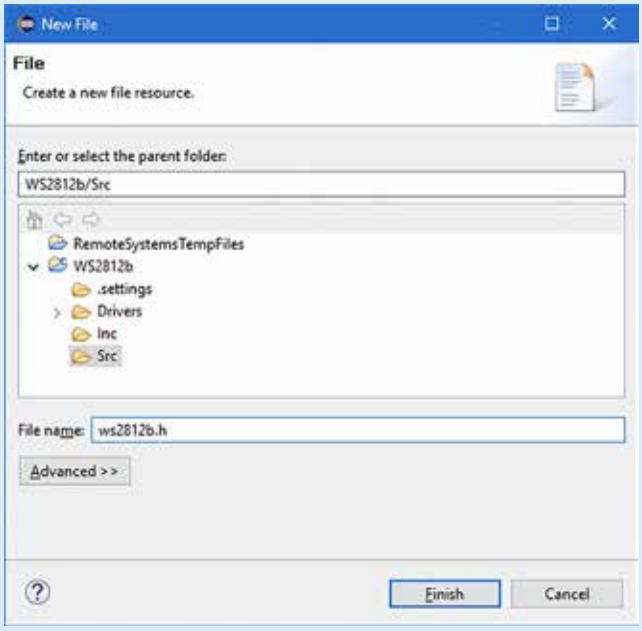
Ponieważ potrzebujemy skorzystać z biblioteki math, klikamy PPM na nazwę naszego projektu w „Project Explorer”. Z menu kontekstowego wybieramy pozycję „Properties” i w nowo otwartym oknie rozwijamy zakładkę „C++ Build” –> „Settings”, następnie, w nowej karcie: „MCU GCC Linker” –> „Libraries”. W polu Libraries klikamy przycisk „Add” i dodajemy nową bibliotekę „m”. Następnie, na samym dole karty odznaczamy opcję „Use C math library (-lm)” i zapisujemy ustawienia (**rysunek 10**).

Teraz dodamy do projektu dwa pliki, które wykorzystamy do obsługi diod. W tym celu w „Project Explorerze” rozwijamy katalog projektu, klikamy PPM na, znajdujący się w nim, podkatalog „Src” i z menu kontekstowego wybieramy opcję „New” –> „File”. W nowym oknie podajemy nazwę pliku, który chcemy utworzyć

Listing 4. Plik nagłówkowy lighting.h

```
#ifndef lighting_header
    #define lighting_header
    #include „stm32f4xx_hal.h”
    #include „ws2812b.h”

    typedef struct lighting_rgb {
        double red, green, blue;
    } lighting_rgb;

    typedef struct lighting_hsv {
        double hue, saturation, value;
    } lighting_hsv;

    typedef struct lighting_config {
        uint16_t diodes_count;
        lighting_rgb * colors_array;
    } lighting_config;

    lighting_config lighting_init(uint16_t diodes_count);
    lighting_rgb lighting_hsv2rgb(lighting_hsv in);
    void lighting_set_diode_color_rgb(lighting_config * config, int diode_no, lighting_rgb rgb);
    void lighting_set_diode_color_hsv(lighting_config * config, int diode_no, lighting_hsv hsv);
    void lighting_draw_gradient_rgb(lighting_config * config, int from_diode, int to_diode, lighting_rgb from_color, lighting_rgb to_
color);
    void lighting_draw_gradient_hsv(lighting_config * config, int from_diode, int to_diode, lighting_hsv from_color, lighting_hsv to_
color);
    double lighting_gamma_correction(double in);
    void lighting_update_ws2812b(lighting_config * lighting_c, ws2812b_config * ws2812b_c);
#endif
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77322**, pass: **8qxonzsb**



Rysunek 11. Tworzenie nowego pliku w System Workbench for STM32

– „ws2812b.h”, klikamy „Finish”. Następnie całą operację powtarzamy dla pliku „ws2812b.c” i uzupełniamy oba pliki zawartością **listingów 1 i 2**. Dalej, modyfikujemy plik „main.c”, dodając do sekcji „USER CODE Includes”, „USER CODE 2” i „USER CODE 3” kod, zgodnie z **listingiem 3**, kompilujemy program i wgrywamy go na nasz mikrokontroler.

Powyższy kod spowoduje zaświecenie na pasku trzech pierwszych diod, ustawiając na nich kolejno kolory: czerwony, zielony i niebieski. Najpierw w sekcji USER CODE 2, przy pomocy funkcji WS2812B_init(), tworzymy strukturę konfiguracyjną paska LED i umieszczamy w niej wskaźnik na strukturę konfiguracyjną interfejsu SPI, który zostanie użyty w komunikacji z diodami oraz przesyłając ilość diod podłączonych do danego pinu (paski możemy łączyć). Następnie, przy pomocy funkcji WS2812B_set_diode_color(), ustawiamy na wybranym pasku (podając jego strukturę konfiguracyjną), kolory poszczególnych diod, będące strukturą typu WS2812B_color, składającą się z trzech wartości z zakresu od 0 do 255 – natężeń poszczególnych barw składowych (czerwonej, zielonej i niebieskiej). Dalej, przy pomocy funkcji WS2812B_refresh(), generujemy i nadajemy sygnał ustawiający kolory poszczególnych diod.

To, na co należy zwrócić uwagę, to fakt, że przy domyślnych ustawieniach interfejsu SPI, sygnał jest odwracany – w miejscu gdzie

Lisitng 5. Plik biblioteki lighting.c

```
#include „lighting.h”
lighting_config lighting_init(uint16_t diodes_count) {
    lighting_config config;
    config.diodes_count = diodes_count;
    config.colors_array = calloc(diodes_count, sizeof(lighting_rgb));
    return config;
}

lighting_rgb lighting_hsv2rgb(lighting_hsv in) {
    if (in.hue >= 360) in.hue = 0;
    double c = in.value * in.saturation;
    double x = c * (1 - fabs(fmod((in.hue / 60), 2) - 1));
    double m = in.value - c;
    lighting_rgb out;
    switch ((int) (in.hue / 60)) {
        case 0: out.red = c + m; out.green = x + m; out.blue = 0; break;
        case 1: out.red = x + m; out.green = c + m; out.blue = 0; break;
        case 2: out.red = 0; out.green = c + m; out.blue = x + m; break;
        case 3: out.red = 0; out.green = x + m; out.blue = c + m; break;
        case 4: out.red = x + m; out.green = 0; out.blue = c + m; break;
        case 5: out.red = c + m; out.green = 0; out.blue = x + m; break;
    }
    return out;
}

void lighting_set_diode_color_rgb(lighting_config * config, int diode_no, lighting_rgb rgb) {
    config->colors_array[diode_no].red = rgb.red;
    config->colors_array[diode_no].green = rgb.green;
    config->colors_array[diode_no].blue = rgb.blue;
}

void lighting_set_diode_color_hsv(lighting_config * config, int diode_no, lighting_hsv hsv) {
    lighting_rgb rgb = lighting_hsv2rgb(hsv);
    lighting_set_diode_color_rgb(config, diode_no, rgb);
}

void lighting_draw_gradient_rgb(lighting_config * config, int from_diode, int to_diode, lighting_rgb from_color, lighting_rgb to_color)
{
    for (int i = from_diode; i <= to_diode; i++) {
        double percent = (double) (i - from_diode) / (double) (to_diode - from_diode);
        lighting_rgb color;
        color.red = from_color.red + percent * (to_color.red - from_color.red);
        color.green = from_color.green + percent * (to_color.green - from_color.green);
        color.blue = from_color.blue + percent * (to_color.blue - from_color.blue);
        lighting_set_diode_color_rgb(config, i, color);
    }
}

void lighting_draw_gradient_hsv(lighting_config * config, int from_diode, int to_diode, lighting_hsv from_color, lighting_hsv to_color)
{
    lighting_rgb from_rgb = lighting_hsv2rgb(from_color);
    lighting_rgb to_rgb = lighting_hsv2rgb(to_color);
    lighting_draw_gradient_rgb(config, from_diode, to_diode, from_rgb, to_rgb);
}

double lighting_gamma_correction(double in) {
    return powf(in, 2.2);
}

void lighting_update_ws2812b(lighting_config * lighting_c, ws2812b_config * ws2812b_c) {
    for (int i = 0; i < ws2812b_c->diodes_count; i++) {
        ws2812b_c->colors_array[i].red = (uint8_t) (lighting_gamma_correction(lighting_c->colors_array[i].red) * 255);
        ws2812b_c->colors_array[i].green = (uint8_t) (lighting_gamma_correction(lighting_c->colors_array[i].green) * 255);
        ws2812b_c->colors_array[i].blue = (uint8_t) (lighting_gamma_correction(lighting_c->colors_array[i].blue) * 255);
    }
}
```

nadajemy jedynkę, przesyłane jest zero i na odwrót. Konsekwencją tej zmiany jest konieczność nadawania, w kodzie programu, sygnałów „00011111" oraz „00000111", zamiast „11100000" i „11111000", w celu wygenerowania sygnału jedynki i zera dla diody.

Omówione funkcje zostały umieszczone w dwóch osobnych plikach – „ws2812b.h" i „ws2812b.c". Pierwszy z nich jest plikiem nagłówkowym – przechowuje on definicje struktur, typów danych i prototypy funkcji, których implementacje znajdują się w pliku „.c". Plik „.h" dołączamy do kodu wszędzie tam gdzie planujemy użyć funkcji z pliku „.c". Dzięki odpowiedniej instrukcji dla kompilatora, plik ten będzie przetworzony tylko raz, nawet, jeśli zainkludujemy go kilkukrotnie.

Mieszanie barw i tworzenie gradientów

Następnie, w analogiczny sposób, dodajemy 2 kolejne pliki – „lighting.h" i „lighting.c", których kod przedstawiono w **listingach 4** i **5** oraz modyfikujemy kod pliku „main.c", zgodnie z **listingiem 6**. Teraz, nasz program będzie na przemian ustawiał na wszystkich 30 diodach na pasku, gradient złożony z kolorów od niebieskiego do czerwonego oraz po wygaszeniu i ponownym zapaleniu, od czerwonego do niebieskiego. W plikach „lighting.h" oraz „lighting.c", znalazły się funkcje odpowiedzialne za konwersję kolorów z przestrzeni barw HSV do RGB, korekcję gamma oraz obliczanie wartości kolorów tworzących gradient. Działania te, wraz z przestrzeniami barw i korekcją gamma, opisane zostały dokładniej w drugiej części niniejszego kursu.

Funkcje z plików „lighting.h/.c", podobnie jak te z „ws2812b.h/.c" korzystają ze struktury konfiguracyjnej, zawierającej informacje o liczbie diod oraz ich kolorach, tym razem jednak, zapisanych w postaci zmiennoprzecinkowej. Strukturę konfiguracyjną generujemy przy pomocy funkcji lighting_init(), przesyłając, jako jej parametr jedynie liczbę diod. W sytuacji, gdy chcemy, aby wprowadzone zmiany kolorów, zostały naniesione na pasku, korzystamy z funkcji lighting_update_ws2812b(), przesyłając do niej wskaźnik na struktury konfiguracyjne obu bibliotek. Funkcja lighting_update_ws2812b() wywoła, uprzednio zdefiniowaną, funkcję ws2812b_refresh(), obliczając przedtem rzeczywiste wartości kolorów po korekcji gamma (lighting_gamma_correction()).

Efekty świetlne możemy tworzyć, korzystając z funkcji lighting_draw_gradient_rgb() i lighting_draw_gradient_hsv(). Obie funkcje generują gradient kolorów i zapisują jego wartości, wypadające poszczególnych na diodach, w strukturze konfiguracyjnej. Gradient jest płynnym, „płaskim", przejściem między dwoma zadanymi kolorami. „Płaskim", ponieważ jest on obliczany w przestrzeni kolorów RGB, jako prosta średnia arytmetyczna, wyliczana osobno dla każdego kanału – czerwonego, zielonego i niebieskiego, pomiędzy wartościami tych kanałów dla koloru początkowego i końcowego. Podobne działanie, w przestrzeni kolorów HSV (Hue Saturation Value – Odcień Nasycenie Jasność), spowodowałoby wygenerowanie tęczy, od zadanego koloru poprzez wszystkie pośrednie, do końcowego. Parametrami przyjmowanymi przez obie funkcje są kolejno: wskaźnik na strukturę konfiguracyjną, numer diody początkowej (licząc od zera), numer diody, na której kończyć ma się gradient oraz dwa kolory – początkowy i końcowy, zadane w postaci struktur lighting_rgb oraz lighting_hsv, w zależności od „wersji" funkcji. Struktura lighting_rgb składa się z trzech wartości zmiennoprzecinkowych, kodujących natężenie każdej z barw składowych koloru – czerwonego, zielonego i niebieskiego. Struktura lighting_hsv przechowuje zmiennoprzecinkowe wartości kolejno: odcienia (hue – od 0 do 360 stopni), nasycenia (saturation – od 0 do 1) oraz jasności (value – ponownie, od 0 do 1) danej barwy.

Aleksander Kurczyk

Listing 6. Modyfikacja pliku *main.c*

```
/* USER CODE BEGIN Includes */
#include „ws2812b.h"
#include „lighting.h"
/* USER CODE END Includes */
/* USER CODE BEGIN 2 */
ws2812b_config ws2812b = ws2812b_init(&hspi1, 30);
lighting_config lighting = lighting_init(30);
lighting_hsv from_color, to_color;
from_color.hue = 240;
from_color.saturation = 1;
to_color.hue = 360;
to_color.saturation = 1;
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    for (double i = 0; i < 2; i += 0.01) {
        double j;
        if (i > 1) j = 2 - i; else j = i;
        from_color.value = j;
        to_color.value = j;
        lighting_draw_gradient_hsv(&lighting, 0, 30, from_color,
to_color);
        lighting_update_ws2812b(&lighting, &ws2812b);
        ws2812b_refresh(&ws2812b);
        HAL_Delay(20);
    }
    double tmp;
    tmp = from_color.hue;
    from_color.hue = to_color.hue;
    to_color.hue = tmp;
}
/* USER CODE END 3 */
```


Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77322, pass: 8qxonzsb