Programowanie układów STM32F4 (2)

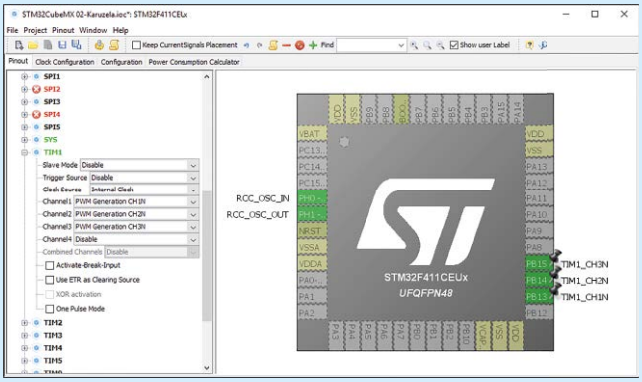
W tej części kursu zapoznamy się z licznikami oraz generatorem sygnału PWM. Dowiemy się jak płynnie zmieniać jasność świecenia diody, czym jest korekcja gamma oraz jak mieszać kolory, korzystając z różnych przestrzeni barw. Wszystkie te rzeczy omówione zostaną w trakcie tworzenia projektu, którego efektem będzie program płynnie zmieniający kolor diody RGB, przechodząc przy tym przez wszystkie możliwe do uzyskania barwy.

Liczniki to układy – rejestry zawierające liczby, najczęściej 16–bitowe, rzadziej 8–bitowe lub 32–bitowe. Sygnał prostokątny wchodzący na wejście licznika, powoduje każdorazowe zwiększenie (inkremntacje) jego zawartości na zboczu narastającym lub opadającym. Sygnał ten może pochodzić z wejścia GPIO, gdy np. zliczamy liczbę wejść do obiektu lub wciśnięć przycisku, lub być odpowiednio podzielonym sygnałem taktującym układu. Po doliczeniu do wartości maksymalnej, licznik zeruje się. Po osiągnięciu zadanej wartości, licznik może wywoływać ustaloną akcję. Na przykład, w momencie wyzerowania może wygenerować przerwanie licznika.

Innym przykładem takiej akcji może być generowanie sygnału PWM. Generator sygnału PWM ustawia wyjście (3,3 V) w momencie wyzerowania się licznika i zeruje je (0 V) po doliczeniu do zadanej wartości z zakresu od zera do wartości, po której osiągnięciu licznik się zeruje (tę również możemy zdefiniować na niższą niż maksymalna). Jeśli na przykład wartość, po której osiągnięciu licznik wyzeruje się ustawimy na 999, co da nam (licząc od zera) 1000 stanów, przez które każdorazowo przejdzie licznik oraz wartość, po której osiągnięciu generator wyzeruje pin na 499, poziom wysoki oraz niski będą panowały na pinie przez 50% czasu, zmieniając się bardzo szybko. Jeśli do pinu zostanie dołączona dioda LED, będzie ona bardzo szybko migała pozostając zaświeconą przez 50% czasu i zgaszoną przez drugą połowę. Da nam to złudzenie możliwości sterowania jasnością świecenia diody. W identyczny sposób możemy sterować silnikami.

Do dzieła!

Podobnie jak w pierwszej części, zacznijmy od utworzenia nowego projektu w programie STM32CubeMX. Wybieramy z listy nasz mikrokontroler. Dla przypomnienia, na używanej podczas tworzenia tego kursu, płytce rozwojowej Kamami KA–NUCLEO–F411CE, znajduje się układ STM32F411CEU6. Podobnie jak poprzednio, z listy po lewej stronie rozwijamy kolejno *Peripherals* oraz *RCC*. Następnie, w polu *High Speed Clock (HSE)* wybieramy opcję *Cristal/Ceramic Resonator*. Jeśli korzystamy z płytki KA–NUCLEO–F411CE, odszukujemy piny: PB13, PB14 i PB15. Są to piny dołączone do diody LED RGB. Pin PB13 odpowiada

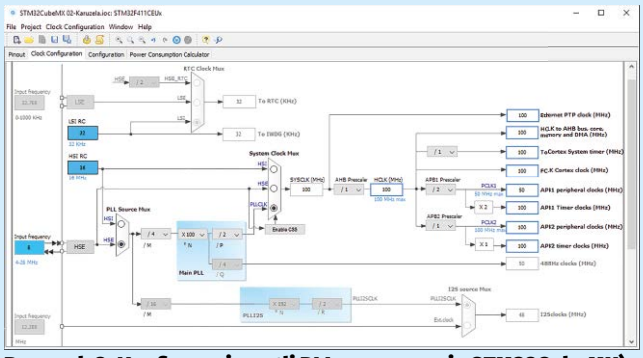


Rysunek 1. Konfiguracja pinów w programie STM32CubeMX

za kolor niebieski, PB14 to kolor czerwony, a PB15 – zielony. Klikamy na każdy z wymienionych pinów lewym przyciskiem myszy i z menu wybieramy jego funkcję alternatywną – *TIM1_CHxN*. Od teraz piny te będą obsługiwane przez układ pierwszego licznika. Musimy jeszcze skonfigurować, w jaki sposób ma się to odbywać. W tym celu przechodzimy do menu po lewej stronie i rozwijamy w nim zakładkę *Peripherals* → *TIM1* przechodzimy do konfiguracji licznika *TIM1*. Z listy rozwijanej, w polach *Channel* od 1 do 3 wybieramy pozycję *PWM Generation CHxN*, powodującą generowanie sygnału PWM na pinach skojarzonych z kanałem licznika. Należy zwrócić szczególną uwagę na literę „N" występującą w funkcjach alternatywnych pinów oraz w polach *Channel*. Trzy pierwsze kanały licznika *TIM1* obsługują jednocześnie po dwa piny. Jeden z tych pinów oznaczono *CHx*, a drugi – *CHxN*. Możemy generować sygnał PWM na dowolnym z nich lub na obu na raz – opcje *PWM Generation CHx*, *PWM Generation CHxN* oraz *PWM Generation CHx CHxN*.

Jeśli korzystamy z innej płytki rozwojowej, musimy zorientować się, do których pinów jest przyłączona dioda RGB lub podłączyć ją samemu. Wybieramy wtedy takie piny, aby możliwe było na nich generowanie sygnału PWM – muszą one zawierać na liście funkcji alternatywnych pozycję *TIMx_CHx* lub *TIMx_CHxN*, a dany kanał *CHx* licznika *TIMx* musi mieć opcję *PWM Generation CHx* i/lub *PWM Generation CHxN*. W obu wypadkach, w polu *Clock Source* w konfiguracji licznik, wybieramy opcję *Internal Clock* – w ten sposób ustawiamy sygnał wejściowy,

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **66532**, pass: **8nnjjeaa**



Rysunek 2. Konfiguracja pętli PLL w programie STM32CubeMX)

inkrementujący licznik, na wewnętrzny sygnał taktujący (**rysunek 1**).

Następnie, po skonfigurowaniu pinów, przechodzimy do drugiej zakładki głównego okna programu STM32CubeMX i konfigurujemy pętlę PLL. Ustawiamy ją analogicznie jak w poprzedniej części – *PLL Source Mux* przestawiamy na *HSE*. W polu *Input frequency* przed blokiem *HSE* wpisujemy częstotliwość naszego generatora (na płytce KA–NUCLEO–F411CE jest to 8 MHz). W sekcji *System Clock Mux* wybieramy pozycję *PLLCLK*. W pole *HCLK* wpisujemy maksymalną częstotliwość obsługiwaną przez nasz układ (tutaj – 100 MHz). Teraz należy się jeszcze upewnić, jaka częstotliwość wchodzi na blok *APB2 Timer clock* (przy korzystaniu z liczników o numerach TIM1, TIM9, TIM10 lub TIM11) albo *APB1 Timer clock* (liczniki TIM2, TIM3, TIM4 oraz TIM5) i zapamiętać tą wartość lub ją zapisać (**rysunek 2**).

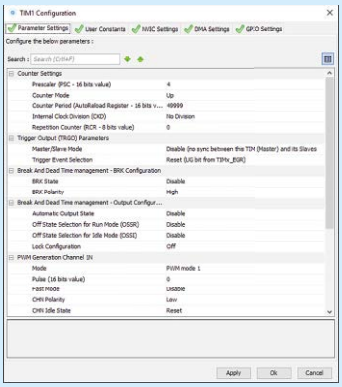
Konfigurowanie licznika

Teraz, po raz pierwszy, przechodzimy do zakładki *Configuration* – kolejnej karty okna programu STM32CubeMX. Pozwala ona na dokładną konfigurację peryferiali. To właśnie tutaj ustawimy szybkość pracy interfejsów UART, a także wartość, po doliczeniu do której, mają się zerować liczniki. Z sekcji *Control* wybieramy przycisk oznaczający licznik wybrany do generowania sygnału PWM (TIM1, jeśli korzystamy z płytki rozwojowej KA–NUCLEO–F411CE).

Jak już wspominałem, generator sygnału PWM, w normalnym trybie pracy, ustawia pin w momencie wyzerowania licznika. Stan niski ustawiany jest w momencie osiągnięcia pewnej innej zdefiniowanej wartości. Powoduje to bardzo szybkie miganie diody oraz złudzenie świecenia ze zmienną jasnością.

Jak więc powinniśmy skonfigurować licznik? Na początku musimy zdecydować, z jaką częstotliwością chcemy, aby nasza dioda migała. Żarówki zasilane z domowej sieci elektrycznej, migają z częstotliwości 100 Hz (zapalają się i gasną dwukrotnie w trakcie jednego okresu fali). Jest to już wystarczająca, niezauważalna dla ludzkiego oka częstotliwość. Załóżmy jednak, że chcemy, aby nasza dioda migała z częstotliwością 400 Hz. Musimy doprowadzić do sytuacji, w której licznik będzie się zerował z taką częstotliwością.

Odpowiadają za to pola/rejestry *Prescaler (PSC – 16 bit value)* oraz *Counter Period (AutoReload Register – 16 bit value)*. W pierwszym z nich ustalamy, z jaką częstotliwość licznik zwiększa swoją wartość (lub zmniejszał, bo i tak można go ustawić). W drugim, po osiągnięciu jakiej wartości, licznik ma się wyzerować. Ta druga wartość powinna być jak największa, aby pozwolić nam na płynne ustalanie jasności świecenia diody. Tę będziemy wybierać z zakresu od zera do wartości *Counter Period (AutoReload Register – 16 bit value)*, gdzie zero odpowiada jasności 0%, a wartość rejestru/pola AAR – 100%.



Rysunek 3. Konfiguracja licznika TIM1 w CubeMX

Jeśli zatem chcemy uzyskać częstotliwość migania 400 Hz, możemy ustawić częstotliwość, z jaką licznik inkrementuje swoją wartość na 20 MHz, dzieląc wejściową częstotliwość 100 MHz na 5, oraz wartość po doliczeniu do której, licznik zresetuje się (*Counter Period*) na 49999 (aby przyjął w trakcie zliczania 50000 różnych wartości, z przedziału od 0 do 49999). Do pola „Prescaler" wpisujemy wartość dzielnika pomniejszoną o jeden – 4 (zamiast 5), a do *Counter Period* – 49999.

Pozwala to wyprowadzić wzór:

Częstotliwość migania × (Counter Period + 1) × (Prescaler + 1) = Częstotliwość wchodząca na licznik

Oba pola – *Counter Period* oraz *Prescaler* przyjmują wartości z zakresu od 0 do 65535 dla liczników 16-bitowych.

W dalszej części okna, znajdują się sekcje odpowiadające za ustawienia każdego z wykorzystywanych kanałów licznika. Dla przypomnienia, każdy z nich odpowiada za generowanie sygnału PWM dla osobnego koloru (niebieskiego, czerwonego oraz zielonego). Ponieważ nasza dioda świeci, gdy pin jest wyzerowany i gaśnie, gdy ustawiony, musimy odwrócić (zanegować) sygnał PWM. Robimy to osobno dla każdego kanału, zmieniając parametr *CH Polarity* na *LOW*. Przykładową konfigurację układu czasowo – licznikowego TIM1 pokazano na **rysunku 3**.

Sterowanie jasnością z kodu programu

Następnie generujemy projekt dla środowiska System Workbench, w sposób opisany w poprzedniej części – klikamy na zębatkę znajdującą się na pasku narzędziowym CubeMX, w polu *Toolchain/IDE* ustawiamy *SW4STM32* i klikamy *OK*. Dalej, importujemy wygenerowany projekt w System Workbench for STM32 – po uruchomieniu i zamknięciu karty powitalnej, klikamy prawym przyciskiem myszy w pole *Project Explorer* i z menu wybieramy kolejno: *Import* → *General* → *Existing Projects into Workspace*.

Do pliku main.c, do sekcji *USER CODE 2* oraz *USER CODE 3* dopisujemy kod z **listingu 1**. Funkcja *HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1)* odpowiada za uruchomienie generatora sygnału PWM na drugim pinie (N) obsługiwanym przez kanał pierwszy (TIM_CHANNEL_1), pierwszego licznika (&htim1). Jeśli potrzebujemy uruchomić generator sygnału PWM na pierwszym pinie (bez

Listing 1. Kod do dopisania w aplikacji przykładowej

```
/* USER CODE BEGIN 2 */
HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
    for (int i = 0; i <= 49999; i += 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, i);
        HAL_Delay(1);
    }
    for (int i = 49999; i >= 0; i -= 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, i);
        HAL_Delay(1);
    }
    for (int i = 0; i <= 49999; i += 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, i);
        HAL_Delay(1);
    }
    for (int i = 49999; i >= 0; i -= 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, i);
        HAL_Delay(1);
    }
    for (int i = 0; i <= 49999; i += 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, i);
        HAL_Delay(1);
    }
    for (int i = 49999; i >= 0; i -= 50) {
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, i);
        HAL_Delay(1);
    }
}
```

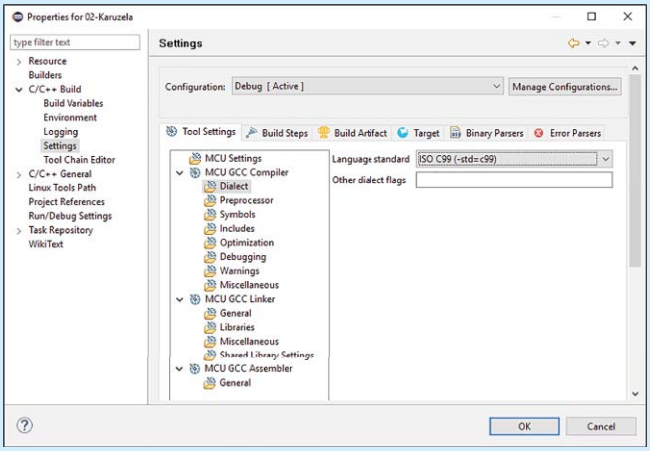
N), korzystamy z funkcji *HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1)*. Funkcja *__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, i)* ustawia wartość, po doliczeniu do której generator ma wyzerować pin (na płytce STM32F411CEU6 – ustawić, ponieważ diody są dołączone od napięcia dodatniego do pinu mikrokontrolera i odwróciliśmy sygnał PWM w ustawieniach). Wartość tą określa ostatni parametr – tutaj zmienna „i", zwiększana, a następnie zmniejszana w pętli w zakresie od 0 do 49999 włącznie, z krokiem równym 50. Efektem działania powyższego kodu jest powolne rozjaśnianie oraz wygaszanie poszczególnych kolorów (niebieskiego, czerwonego oraz zielonego).

Teraz możemy skompilować oraz uruchomić program – klikamy ikonkę młotka (Build) oraz robaka (Debug), na pasku narzędziowym, a po przejściu do perspektywy debugowania, przycisk strzałki (Resume).

Jeśli, z powodu błędów, nasz program się nie kompiluje, w panelu „Project Explorer", klikamy prawym przyciskiem myszy na nazwę projektu i z menu kontekstowego wybieramy polecenie „Properties". W nowootwartym oknie, otwieramy zakładkę „C/C++ Build" → „Settings". W nowootwartej karcie, otwieramy zakładkę „MCU GCC Compiler" → „Dialect" i w polu „Language standard" ustawiamy pozycję „ISO C99". Standard ISO C90, który może być ustawiony jako domyślny, nie pozwala na deklarację zmiennych w instrukcjach pętli („...for (int i...") – **rysunek 4.**

Czy coś jest nie tak?

Jeśli teraz przyjrzymy się temu, jak zwiększa się jasność świecenia diody, z pewnością zauważymy, że dioda najpierw bardzo szybko się rozjaśnia, a po chwili, zmiany te nie są w ogóle zauważalne. Dzieje się tak dlatego, że ludzkie oko



Rysunek 4. Wybór standardu języka C w System Workbench for STM32

nie postrzega jasności w skali liniowej, w jakiej rozjaśniamy diodę. Aby temu zaradzić, stosuje się korekcję gamma. Jasność z zakresu od 0.0 do 1.0 (wartość zmiennoprzecinkową) podnosimy do potęgi 2.2, a następnie rozszerzamy wynik na zakres wartości od zera do wartości rejestru ARR (tutaj 49999), mnożąc wynik przez zawartość tego rejestru. Aby nie kopiować kodu tej operacji, utworzymy funkcję ustawiającą jasność diody po przesłaniu jej takich samych parametrów jak dla funkcji *__HAL_TIM_SET_COMPARE()*, z tą różnicą, że jasność będzie argumentem zmiennoprzecinkowym, z zakresu od 0.0 do 1.0 (**listing 2**).

Kod z list. 2 umieszczamy w sekcji „USER CODE 0", jeszcze przed kodem funkcji main(). Od teraz, aby ustawić jasność świecenia diody, zamiast wywoływać bezpośrednio funkcję *__HAL_TIM_SET_COMPARE()*, korzystamy z funkcji *set_led_brightness()*. Musimy też przystosować kod naszej pętli, aby parametr „i" był zmienną typu zmiennoprzecinkowego (double) i przyjmował wartości z zakresu od 0.0 do 1.0.

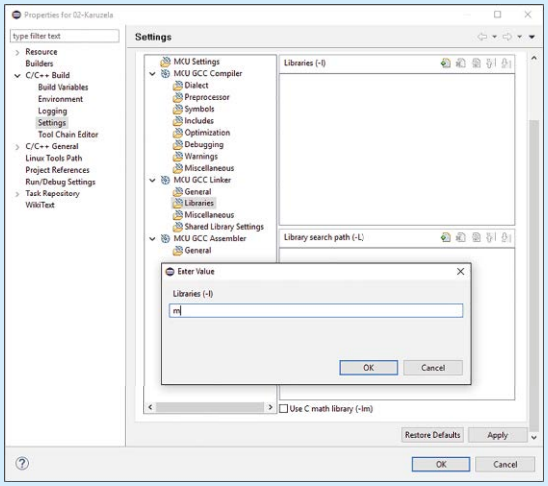
Przykładowe wywołania funkcji w pętli:

```
for (double i = 0; i <= 1; i += 0.001) {
    set_led_brightness(&htim1, TIM_CHANNEL_1, i);
    HAL_Delay(1);
}
```

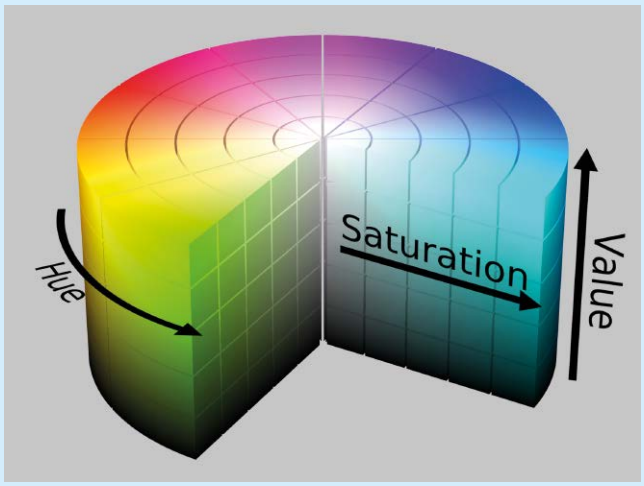
Z powodu błędu w oprogramowaniu System Workbench, możemy mieć problem z kompilacją powyższego kodu. Aby temu zaradzić, należy wejść w ustawienia projektu (klikamy prawym przyciskiem na jego nazwę i z menu wybieramy polecenie *Properties*). Następnie, otwieramy kartę *C/C++ Build* → *Settings* i w kolejnym panelu, wewnątrz nowej karty, rozwijamy *MCU GCC Linker* → Libraries, przewijamy kartę w dół, odznaczamy opcję *Use C math library* oraz dodajemy (przycisk z plusem) w polu *Libraries* bibliotekę o nazwie „m" (**rysunek 5**). Dalej, klikamy przyciski *Apply* i *OK*. Funkcja *powf()*, używana do podniesienia wartości

Listing 2. Jasność jako argument zmiennoprzecinkowy

```
/* USER CODE BEGIN 0 */
void set_led_brightness(TIM_HandleTypeDef * timer, uint32_t channel, double brightness) {
    int32_t value = powf(brightness, 2.2) * 49999;
    __HAL_TIM_SET_COMPARE(timer, channel, value);
}
```



Rysunek 5. Podpis: Dodawanie biblioteki math w ustawieniach linkera



Rysunek 6. Podpis: Cylinder barw HSV (źródło: https://goo.gl/eyGzrs)

zmiennoprzecinkowej jasności do potęgi 2.2, znajduje się w bibliotece math.

Mieszanie barw

Poszczególne kolory możemy zapalać również w jednym czasie, ustawiając osobno jasność każdej z diod wchodzących w skład diody RGB. W ten sposób uzyskujemy różne kolory i ich odcienie. Przykładowo, zapalenie czerwonej oraz niebieskiej diody da nam kolor fioletowy. Jeśli będziemy zmniejszać jasność koloru czerwonego, stopniowo kolor ten zacznie przechodzić w niebieski. Przestrzeń barw RGB (czerwony, zielony, niebieski) pozwala uzyskać każdy kolor i jest bardzo wygodna do implementacji w sprzęcie – potrzebujemy tylko trzech diod, nie pozwala nam ona jednak na łatwy wybór dowolnej barwy, a następnie ustawienie jej nasycenia albo jasności. Tutaj z pomocą przychodzi przestrzeń kolorów HSV – Hue Saturation Value. Kolor definiujemy w niej jako barwę (Hue – wartość w zakresie od 0 do 360 stopni, patrz ilustracja), jej nasycenie (Saturation – u nas, wartość od 0.0 do 1.0, gdzie 0.0 to kolor biały, a 1.0 to największe możliwe nasycenie) oraz jasność (Value – podobnie, wartość z zakresu od 0.0 do 1.0). Wybrany kolor musimy następnie przekonwertować na przestrzeń RGB, aby możliwe było jego ustawienia na diodach (**rysunek 6**).

Kod z **listingu 3** to funkcja konwertująca kolor podany w przestrzeni HSV, na kolor w przestrzeni RGB. Przyjmuje on na wejściu strukturę danych typu *hsv*, zawierającą 3 pola typu *double* – barwę, nasycenie i jej jasność oraz zwraca strukturę typu *rgb* (jasność koloru czerwonego, zielonego oraz niebieskiego). Kod ten, wraz z definicjami struktur, należy umieścić w sekcji „USER CODE 0", po funkcji *set_led_brightness()*.

Kolejna funkcja (**listing 4**) uprości nam ustawienie koloru podanego w przestrzeni HSV na diodę RGB. Ją również dodajemy do sekcji *USER CODE 0*.

Karuzela

Teraz możemy już użyć nowych funkcji wywołując je w sekcji *USER CODE 3* – w pętli. Zamieniamy dotychczasową zawartość tej sekcji, na podaną na **listingu 5**.

W ramach zadania domowego, możemy poeksperymentować, zmieniając natężenie oraz nasycenie barw, najlepiej w pętli, tworząc ciekawe efekty. W kolejnej części kursu, zajmiemy się komunikacją z komputerem oraz innymi urządzeniami przez interfejs UART, programator oraz port USB. Omówimy także wykorzystanie przerwań – interfejsu UART oraz zegara.

ALEKSANDER KURCZYK

Oferta dla Czytelników EP
Podczas trwania kursu, co miesiąc wśród Czytelników EP będzie rozlosowywać 3 zestawy KA-NUCLEO-F411, ufundowane przez producenta – firmę KAMAMI.
Osoby zainteresowane proszone są o nadsyłanie zgłoszeń z odpowiedzią na pytanie „Jaką pojemność pamięci Flash i RAM mają mikrokontrolery F411 zastosowane w zestawie" oraz swoim adresem pocztowym na adres e-mail adres F411@kamami.pl.

Listing 3. Konwertowanie argumentu na kolor w przestrzeni HSV
```
typedef struct { double red; double green; double blue; } rgb;
typedef struct { double hue; double saturation; double value; } hsv;
rgb hsv2rgb(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;
    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;
}
```

Listing 4. Konwersja koloru z przestrzeni HSV na ustawienia diody RGB
```
void set_color(hsv color_hsv) {
    rgb color_rgb = hsv2rgb(color_hsv);
    set_led_brightness(&htim1, TIM_CHANNEL_2, color_rgb.red);
    set_led_brightness(&htim1, TIM_CHANNEL_3, color_rgb.green);
    set_led_brightness(&htim1, TIM_CHANNEL_1, color_rgb.blue);
}
```

Listing 5. Modyfikacja kodu z sekcji „USER CODE 3"
```
/* USER CODE BEGIN 3 */
hsv color; // Tworzymy strukturę koloru w przestrzeni HSV
color.saturation = 1; // Ustawiamy maksymalne nasycenie barwy
color.value = 0.5; // Ustawiamy 50% jasności
for (double i = 0; i < 360; i += 0.05) {
    color.hue = i; // W pętli ustawiamy kolejne barwy z zakresu od 0 do 360 stopni
    set_color(color); // Ustawiamy kolor na diodzie RGB
    HAL_Delay(1); // Odczekujemy przez jedną milisekundę, aby można było zobaczyć kolor
}
/* USER CODE END 3 */
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 66532, pass: 8mjjeaa