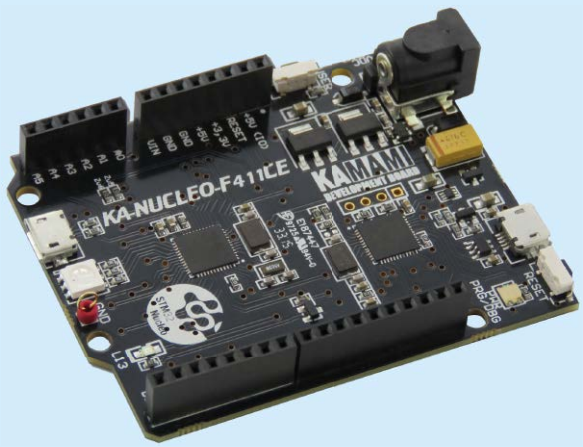
Programowanie układów STM32F4 (1)

W tym kursie, bazując na nieskomplikowanych projektach, zaprezentuję w praktyczny sposób programowanie układów z rodziny STM32F4. Jest to pierwszy artykuł z serii. Przedstawione tutaj zostało narzędzie STM32CubeMX oraz środowisko programistyczne System Workbench for STM32. Podczas czytania artykułu oraz wykonywania przykładów utworzymy pierwszy projekt – mikrokontrolerowe „Hello World!". W kolejnych częściach omówione zostaną liczniki, przerwania, generowanie sygnału PWM, odczyt stanów pinów wejściowych, komunikacja z komputerem oraz innymi urządzeniami za pośrednictwem interfejsu UART, sterowanie adresowalnymi paskami diod LED, bazujących na chipie WS2812b. Dodamy także do naszego mikrokontrolera obsługę Wi-Fi, dzięki zastosowaniu układu ESP8266, obsłużymy prosty wyświetlacz LCD i odbierzemy dane z różnych czujników.

STM32 to rodzina 32-bitowych mikrokontrolerów produkcji STMicroelectronics. Układy te bazują na rdzeniu z serii ARM Cortex-M. W zależności od wersji, charakteryzują się małym poborem mocy lub bardzo dużą prędkością i bogatym wyposażeniem. Układy te mogą być taktowane przebiegiem o maksymalnej częstotliwości do 32 MHz (STM32L0) lub do 216 MHz (STM32F7). Wersje STM32F7, STM32F4 oraz STM32F3 bazujące na rdzeniach Cortex-M4F mają wsparcie dla obliczeń zmiennoprzecinkowych (FPU) oraz cyfrowego przetwarzania sygnałów (DSP). Mikrokontrolery te mają również wiele interfejsów: SPI, UART, I²C, I²S, a w niektórych wersjach również wbudowane interfejsy USB 1.1, USB 2.0 oraz Ethernet.

Ich producent, firma STMicroelectronics, opracował zestaw bibliotek Hardware Abstraction Layer (HAL), umożliwiający programowanie tych mikrokontrolerów w nieskomplikowany sposób, z użyciem języka wysokiego poziomu. Dzięki temu nie ma konieczności zagłębiania się ani w obszerną specyfikację układu, ani rdzenia ARM. Wraz z bibliotekami HAL, producent dostarcza również program STM32CubeMX. Jest to graficzny generator konfiguracji mikrokontrolera, pozwalający na skonfigurowanie wszystkich wyprowadzeń, interfejsów, liczników oraz taktowania całego układu, a następnie na wygenerowanie gotowego projektu dla różnych środowisk IDE.



Fotografia 1. Płytka rozwojowa KA-NUCLEO-F411CE

Wszystko to sprawia, że programowanie tak zaawansowanego mikrokontrolera jest niemal równie łatwe i intuicyjne, jak np. programowanie platformy Arduino.

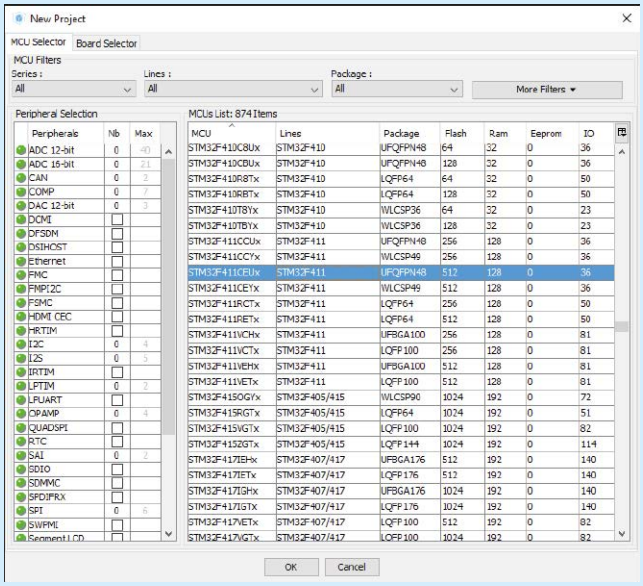
Podczas tworzenia tego kursu korzystałem z płytki rozwojowej Kamami KA-NUCLEO-F411CE (**fotografia 1**) mającej zamontowany układ mikrokontrolera STM32F411CEU6. Ma on interfejs USB 2.0 (do dyspozycji użytkownika), pięć interfejsów SPI, pięć I²S, trzy I²C, trzy USART, sześć liczników 16-bitowych oraz dwa 32-bitowe i może być taktowany przebiegiem o częstotliwości maksymalnej 100 MHz. Na płytce, oprócz właściwego mikrokontrolera, zamontowano również prostszy układ STM32F103C8U6, pełniący funkcję programatora ST-LINK z wyjściem USB. Płytka ma wyprowadzenia zgodne z Arduino, wbudowany przycisk do dyspozycji użytkownika oraz trójkolorową diodę LED RGB. Większość omawianych przykładów będzie możliwa do wykonania na dowolnej innej płytce rozwojowej z układem STM32F4, jednak zachęcam do zakupu tej konkretnej, mającej zamontowane wymienione elementy.

W trakcie wykonywania przykładów korzystać będziemy z omówionego wcześniej programu STM32CubeMX. Jest on dostępny dla Windowsa, Linuksa oraz macOS. Możemy go pobrać ze strony producenta – jest dostępny pod adresem ***https://goo.gl/tI9FrN***. Drugim potrzebnym programem jest bazujące na Eclipse środowisko programistyczne System Workbench for STM32. Również i ono jest dostępne w wersjach dla różnych systemów operacyjnych. Możemy je pobrać ze strony projektu po uprzedniej rejestracji lub dodać jako plug-in do już zainstalowanego programu Eclipse. Oprogramowanie jest dostępne pod adresem ***https://goo.gl/GbI0yT***. Przy instalowaniu jako plug-in do Eclipse będziemy jeszcze potrzebowali sterowników dla programatora ST-LINK, które w pierwszym przypadku instalują się wraz z System Workbench for STM32. Oba omawiane programy wymagają do swojego działania wirtualnej maszyny Java, którą możemy pobrać spod adresu ***https://goo.gl/kimB9s***.

Witaj migająca diodo!

Pierwszym projektem będzie mikrokontrolerowy odpowiednik programu „Hello World!", znanego z nauki różnych języków

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **66838**, pass: **643ttoo**



Rysunek 2. Kreator wyboru mikrokontrolera w programie STM32CubeMX

programowania, czyli program, którego zadaniem będzie naprzemienne zapalanie i gaszenie diody znajdującej się na płytce rozwojowej. Podczas tworzenia tego projektu przyjrzymy się podstawom korzystania z programu STM32CubeMX – konfiguracji funkcji wyprowadzeń, sygnału taktującego rozchodzącego się po mikrokontrolerze oraz środowisku System Workbench for STM32, w którym napiszemy kod naszej aplikacji, skompilujemy i załadujemy do mikrokontrolera.

Generowanie projektu w STM32CubeMX

Po uruchomieniu programu STM32CubeMX i utworzeniu w nim nowego projektu, klikamy w przycisk *New project*. Pierwszym oknem, które ujrzymy, będzie pokazane na **rysunku 2** okno kreatora wyboru mikrokontrolera. Możemy tutaj od razu wybrać z list rozwijanych model posiadanego przez nas układu oraz przejść dalej, klikając *OK*. Na wykorzystywanej przeze mnie płytce Kamami KA-NUCLEO-F411CE zamontowano układ z serii STM32F411CEUx w obudowie UFQFPN48, więc mój wybór był oczywisty.

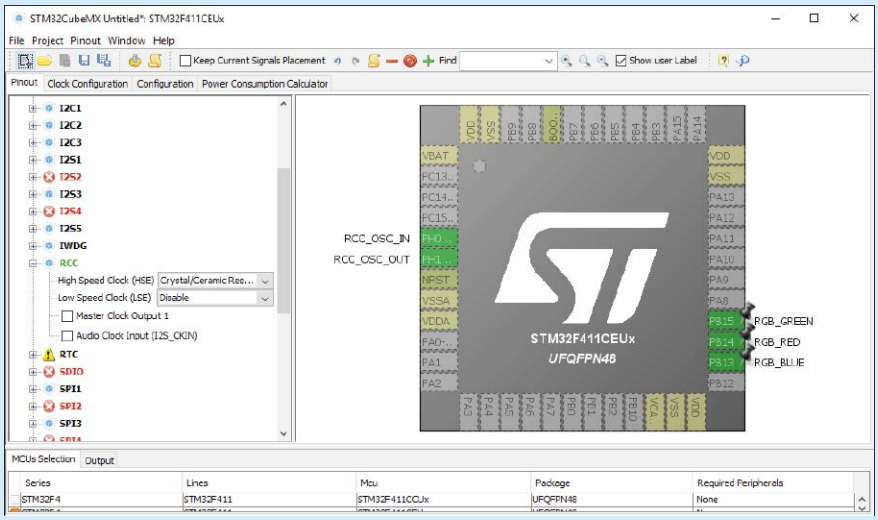
Kreatora możemy jednak również używać „odwrotnie", to znaczy może on sugerować wybór układu do projektowanego przez nas zastosowania. Z listy po lewej stronie wybieramy wszystkie funkcje i interfejsy mikrokontrolera, których potrzebuje do działania nasze urządzenie, a po prawej stronie okna otrzymujemy listę wszystkich mikrokontrolerów z rodziny STM32, które spełniają nasze wymagania. Możemy w ten sposób wybrać układ spełniający nasze oczekiwania, kierując się różnymi kryteriami. W podobny sposób, w drugiej zakładce, jesteśmy też w stanie wybrać płytkę rozwojową. Na liście są jednak tylko płytki wyprodukowane przez producenta układów STM32 – firmę STMicroelectronics.

Po wyborze mikrokontrolera przechodzimy do właściwego okna programu STM32CubeMX. W pierwszym jest widoczna obudowa mikrokontrolera wraz ze wszystkimi jego wyprowadzeniami oraz (na liście po lewej stronie) lista komponentów, które możemy skonfigurować. Należą do nich wszystkie liczniki i interfejsy, ale nie tylko. Zazwyczaj są to też funkcje alternatywne poszczególnych wyprowadzeń. Z tego względu, większość peryferii, które możemy skonfigurować w tej zakładce, możemy ustawić na dwa sposoby – albo klikając prawym przyciskiem myszy na wyprowadzenie i wybierając jedną z jego funkcji, tj. wejście/wyjście GPIO, wejście konwertera A/C, czy wyjście interfejsu UART, albo ustawiając tę funkcję na liście po lewej stronie i przenosząc metodą przeciągnij – upuść odpowiednio skonfigurowane wyprowadzenie w inne miejsce.

To, co powinniśmy ustawić w tej zakładce, to wejście przebiegu zegarowego dostarczającego sygnał taktujący do mikrokontrolera, o ile taki znajduje się na naszej płytce. Mikrokontrolery STM32 mają wbudowany generator RC, jednak do niektórych zastosowań jego stabilność jest niewystarczająca. Na płytce KA-NUCLEO-F411CE zamontowano dwa oscylatory kwarcowe wspomagające pracę generatorów HSE (o wysokiej częstotliwości) i LSE (o niskiej częstotliwości). W tej chwili potrzebujemy ustawić jedynie pierwszy z nich. Będzie on źródłem sygnału taktującego dla całego układu. Drugi – LSE, jest wykorzystywany jako źródło częstotliwości dla bloku odmierzającego czas rzeczywisty (RTC). Nie będziemy go jednak w tej chwili konfigurować.

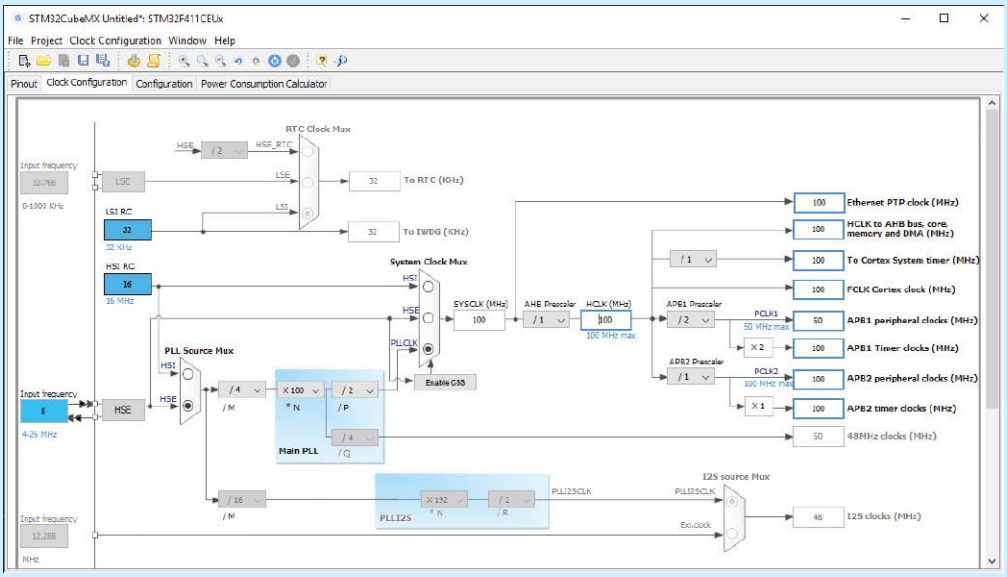
Z listy po lewej stronie rozwijamy kolejno *Peripherials* → *RCC* i z listy rozwijanej w polu *High Speed Clock (HSE)* wybieramy pozycję *Crystal/Ceramic Resonator*. Spowoduje to załączenie funkcji alternatywnych dwóch wyprowadzeń mikrokontrolera – do nich jest fizycznie przyłączony oscylator na używanej przez nas płytce rozwojowej.

Kolejną rzeczą, którą zrobimy, będzie skonfigurowanie wyprowadzenia, do którego jest dołączona dioda LED, jako *GPIO_Output*, tak abyśmy mogli ją włączać i wyłączać. Wykorzystamy tutaj diodę RGB znajdującą na płytce KA-NUCLEO-F411CE, a konkretnie jej jedną część odpowiadającą za świecenie w kolorze niebieskim. Dioda ta jest przyłączona do wyprowadzenia PB13 (kolor czerwony to PB14, a zielony to PB15). Należy zwrócić uwagę, że dioda będzie świeciła, gdy odpowiednie wyjście



Rysunek 2. Skonfigurowanie funkcji wyprowadzeń w programie STM32CubeMX

zostanie wyzerowane, oraz gasnąć, gdy zostanie ustawione. Tryb pracy wyprowadzeń *GPIO_Output* ustawiamy, klikając lewym przyciskiem myszy na pin i wybierając z listy funkcję, którą ma pełnić. Dodatkowo, możemy też kliknąć na symbolu wyprowadzenia prawym przyciskiem myszy, wybrać z menu *Enter User Label* i nadać mu nazwę symboliczną – tutaj *RGB_BLUE*. Będziemy z niej później korzystali w programie, odwołując się zarówno do portu, do którego jest przyłączona dioda niebieska, jak i do konkretnego wyprowadzenia tego portu (**rysunek 3**).



Rysunek 4. Konfigurowanie pętli PLL w STM32CubeMX

Konfigurowanie pętli PLL

Druga zakładka okna programu STM32CubeMX pozwala na skonfigurowanie systemu rozdzielającego przebiegi taktujące dostarczane do poszczególnych bloków funkcjonalnych mikrokontrolera (**rysunek 4**). Na pierwszy rzut oka ten schemat może wyglądać na dość skomplikowany, jednak jest on bardzo łatwy do skonfigurowania, gdy tylko zrozumie się podstawy działania bloków zaznaczonych na schemacie. Jego konfigurowanie najczęściej sprowadza się do zmiany pozycji dwóch przełączników i wpisania oczekiwanej wartości częstotliwości przebiegu taktującego. Wszystkie parametry potrzebne do uzyskania zadanej wartości taktowania zostaną wyliczone przez program w pełni automatycznie.

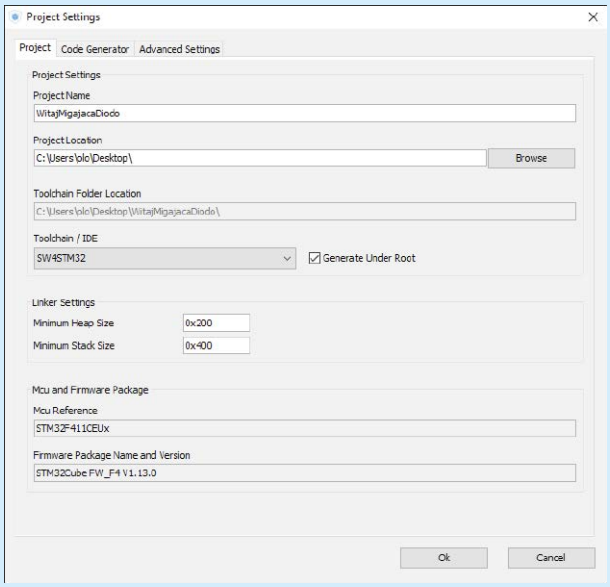
Na środku schematu, w niebieskiej sekcji oznaczonej podpisem *Main PLL*, znajduje się generator z fazową pętlą synchronizacji PLL. Bez wdawania się w szczegóły techniczne jej działania można powiedzieć, że jest to układ, który na bazie przebiegu o ustalonej częstotliwości generuje przebieg o innej (wyższej) częstotliwości. Jako sygnał odniesienia dla generatora PLL możemy wybrać HSI, czyli wewnętrzny oscylator RC lub HSE – zewnętrzne źródło taktowania, które skonfigurowaliśmy w poprzedniej zakładce. Klikamy więc na przełącznik HSE znajdujący się przed pętlą PLL. Potrzebujemy jeszcze zdefiniować jego częstotliwość. Na omawianej płytce rozwojowej wynosi ona 8 MHz. Tę liczbę wpisujemy w niebieskie pole podpisane *Input frequency* znajdujące się zaraz przed blokiem HSE.

W sekcji *System Clock Mux* wybieramy źródło sygnału taktowania dla całego mikrokontrolera. Domyślnie jest to ponownie HSI, czyli wewnętrzny oscylator RC. Przełączamy źródło na wyjście dopiero co skonfigurowanej pętli PLL. Możemy tutaj również wybrać jako źródło zewnętrzny oscylator HSE z pominięciem generatora PLL. W polu HCLK wpisujemy częstotliwość, w MHz, którą chcemy uzyskać. Maksymalna jej wartość, obsługiwana przez nasz układ, jest wyświetlana pod tym polem. Po wpisaniu wartości i kliknięciu *Enter* lub kursorem myszy poza pole, program STM32CubeMX przeliczy pozostałe parametry i ustawi wartości dzielników i mnożników, tak aby uzyskać zadaną wartość taktowania. Jeśli wybierzemy częstotliwość, której nie możemy uzyskać za pomocą PLL lub zbyt dużą, spoza zakresu obsługiwanego przez mikrokontroler, program poinformuje nas o błędzie.

Mikrokontroler znajdujący się na płytce rozwojowej KA-NUCLEO-F411CE może być taktowany przebiegiem o częstotliwości maksymalnej 100 MHz – spróbujmy ustawić tę wartość. Ze względu na oszczędność energii, w rzeczywistych aplikacjach częstotliwość pracy mikrokontrolera powinna być jak najniższa.

Przebieg taktujący trafia na preskalery, czyli dzielniki częstotliwości oraz na magistrale doprowadzające przebiegi taktujące do poszczególnych bloków funkcjonalnych. *FCLK Cortex Clock* to częstotliwość taktowania rdzenia procesora ARM. W wypadku mikrokontrolerów z rodziny STM32F4, *APB1* to magistrala doprowadzająca taktowanie do interfejsów SPI2, SPI3, USART2, I2C1, I2C2, I2C3, I2S2, I2S3 oraz liczników TIM2, TIM3, TIM4 i TIM5. *APB2* to kolejna magistrala, dostarczająca sygnał taktowania do interfejsów SPI1, SPI4, SPI5, USART1, USART6, ADC1 oraz liczników TIM1, TIM9, TIM10, TIM11. Peryferiale znajdujące się na magistrali *APB1*, w przypadku omawianego układu, nie mogą być taktowane z częstotliwością wyższą niż 50 MHz. Dlatego program CubeMX podzielił tę częstotliwość za pomocą preskalera, a następnie pomnożył ją, aby liczniki były taktowane z częstotliwością 100 MHz.

Na schemacie znajdują są jeszcze inne źródła i układy korzystające z przebiegu taktującego. Zaznaczono tutaj wejście drugiego oscylatora zewnętrznego – LSE, generującego przebieg o niskiej częstotliwości, używany przez zegar czasu rzeczywistego. Z głównej pętli PLL mamy też dodatkowe wyjście na kolejny generator PLL – PLLI2S. Jest on używany tylko wtedy, gdy korzystamy z interfejsu I²S. Jej implementacja jest spowodowana koniecznością stosowania nietypowych częstotliwości podczas transmisji danych za pomocą tego interfejsu. Powstał do przesyłania cyfrowego sygnału audio o częstotliwości



Rysunek 5. Generowanie projektu dla System Workbench w STM32CubeMX

próbkowania 44,1 kHz. Ten generator nie znajdzie zastosowania w naszym projekcie i nie potrzebujemy go konfigurować.

Generowanie projektu i import do System Workbench

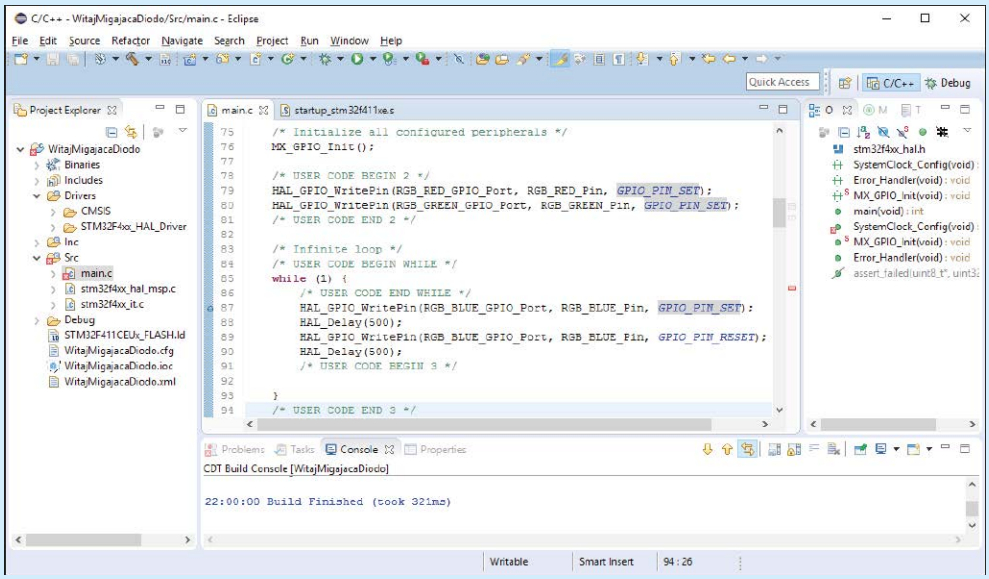
Po przygotowaniu konfiguracji sprzętowej w STM32CubeMX możemy wygenerować projekt dla środowiska IDE, w którym napiszemy program. Robimy to, klikając w „zębatkę" na pasku narzędziowym okna CubeMX. Następnie w nowo otwartym oknie wpisujemy nazwę projektu i wybieramy lokalizację, w której chcemy go zapamiętać. Domyślnie jest to podfolder na pulpicie lub w katalogu domowym użytkownika. W nazwie użytkownika komputera nie powinno być polskich znaków. Jeśli zdecydujemy się wybrać inny folder, w jego ścieżce dostępu również nie mogą występować polskie znaki. W polu *Toolchain/IDE*, z listy rozwijanej wybieramy pozycję *SW4STM32* oznaczającą System Workbench for STM32, z którego zaraz skorzystamy. W kolejnej zakładce możemy zaznaczyć jeszcze opcję *Generate peripherial initialization as pair of .c/.h files per peripherials*, która spowoduje rozdzielenie całej konfiguracji startowej na większą liczbę plików. Podczas pierwszego generowania projektu CubeMX zapyta nas, czy chcemy pobrać firmware oraz bibliotekę HAL – odpowiadamy twierdząco i czekamy na zakończenie pobierania. W folderze projektu, oprócz plików projektu System Workbench, zostanie zapisany dodatkowo plik .ioc przechowujący informację o obecnych ustawieniach w CubeMX. Za jego pomocą możemy zmieniać konfigurację w trakcie rozwoju projektu, nie tracąc przy tym już napisanego kodu, jeśli zastosujemy się do znaczników znajdujących się w plikach.

Uruchamiamy środowisko System Workbench for STM32. Program zapyta nas, gdzie chcemy umieścić naszą przestrzeń roboczą. W praktyce, przechowywana będzie tam tylko konfiguracja środowiska. Polecam tutaj pozostać przy domyślnej wartości, jeśli nie korzystamy z żadnej innej instalacji Eclipse. Po otwarciu programu zamykamy kartę *Welcome* krzyżykiem znajdującym się obok jej nazwy. Teraz w pustym polu w panelu *Project Explorer* klikamy prawym przyciskiem myszy i z menu kontekstowego wybieramy polecenie *Import*. Na pierwszej planszy kreatora wybieramy *General* → *Existing Projects into Workspace*. Następnie klikamy przycisk *Browse…* umieszczony obok napisu *Select root directory*, nawigujemy do folderu projektu wygenerowanego przez STM32CubeMX, wybieramy go i klikamy przycisk *Finish*. Spowoduje to zaimportowanie projektu (**rysunek 6**).

Teraz możemy przyjrzeć się zawartości wygenerowanego przez CubeMX projektu. W folderze Dirvers\STM32F4xx_HAL_Driver znajdują się biblioteki, z których będziemy korzystali w trakcie pisania programów. W praktyce to właśnie tam szukać będziemy pomocy na temat działania poszczególnych funkcji. Przed kodem każdej z nich znajduje się komentarz na temat jej działania, przyjmowanych parametrów oraz zwracanej wartości. W folderze Src znajduje się obecnie wygenerowana konfiguracja startowa. Tam też umieszczać będziemy wszystkie pliki zawierające kod źródłowy naszych programów.

Kod programu

Plik, w którym umieścimy nasz program, to *main.c*. Składa się on z wielu sekcji rozdzielonych komentarzami, zawierającymi słowa kluczowe *BEGIN* oraz *END*. Tylko pomiędzy tymi komentarzami możemy umieszczać kod programu. W przeciwnym wypadku, po zmienie konfiguracji w programie STM32CubeMX i wygenerowaniu na nowo projektu, stracimy te fragmenty kodu, które znalazły się pomiędzy blokami.



Rysunek 6. Edycja pliku *main.c* w programie System Workbench

Wykonywanie programu rozpoczyna się od funkcji *main()*. Na jej początku następuje wywołanie funkcji inicjalizującej poszczególne bloki funkcjonalne mikrokontrolera. Następnie jest wykonywany kod użytkownika, przed oraz w pętli głównej. Sekcja *USER CODE 2* jest odpowiednikiem funkcji *setup()* znanej z Arduino. W niej umieścimy kod mający zostać wykonany przed pętlą, po uruchomieniu urządzenia. *USER CODE 3* to odpowiednik funkcji *loop()* – sekcja ta będzie wykonywana w pętli nieskończonej, aż do wyłączenia urządzenia. Ponieważ całą konfigurację wyprowadzeń przeprowadziliśmy wcześniej, w programie CubeMX, do sekcji *USER CODE 3* dopiszemy jedynie pokazane niżej cztery linijki kodu:

```
HAL _ GPIO _ WritePin(RGB _ BLUE _ GPIO _ Port,
RGB _ BLUE _ Pin, GPIO _ PIN _ SET);
HAL _ Delay(500);
HAL _ GPIO _ WritePin(RGB _ BLUE _ GPIO _ Port,
RGB _ BLUE _ Pin, GPIO _ PIN _ RESET);
HAL _ Delay(500);
```

Funkcja *HAL_GPIO_WritePin()* ustawia wybrane wyprowadzenie portu mikrokontrolera (GPIO_PIN_SET) lub zeruje je (GPIO_PIN_RESET). *RGB_BLUE* to nazwa wyprowadzenia nadana przez nas w czasie jego konfiguracji w STM32CubeMX. *RGB_BLUE_GPIO_Port* wskazuje na rejestr portu obsługującego dany pin, a *RGB_BLUE_Pin* oznacza pozycję wyprowadzenia w tym porcie. Funkcja *HAL_Delay(500)* angażuje rdzeń mikrokontrolera na 500 milisekund.

W środowisku IDE Eclipse, po wpisaniu fragmentu nazwy funkcji lub zmiennej oraz wciśnięciu kombinacji klawiszy Ctrl+Spacja, program dopełni wpisywaną nazwę lub zaproponuje kilka możliwości dopełnienia. Kombinacja przycisków Ctrl+Shift+F, z kolei, uporządkuje nam wcięcia w kodzie źródłowym programu.

Kompilowanie, uruchamianie i debugowanie

Na tym etapie możemy już skompilować nasz program. W tym celu klikamy na ikonę „młotka" (*Build*) umieszczoną na pasku narzędziowym. Po zakończeniu kompilacji, w celu uruchomienia programu na mikrokontrolerze, klikamy ikonę „robaka" (*Debug*), również znajdującą się na pasku narzędziowym. Mikrokontroler powinien być przyłączony do komputera kablem USB poprzez programator/debugger. Na płytce KA-NUCLEO-F411CE port microUSB programatora zamontowano obok przycisku *RESET*. Oprogramowanie System Workbench zapyta nas, w jaki sposób chcemy uruchomić aplikację. Dwukrotnie klikamy na pozycję *Ac6 STM32 C/C++ Application*. W nowo otwartym oknie, w polu *Debug Device*, wybieramy *ST-LinkV2-1* i klikamy *OK*. Następnie zgadzamy się jeszcze na utworzenie perspektywy debugowania (**rysunek 7**).

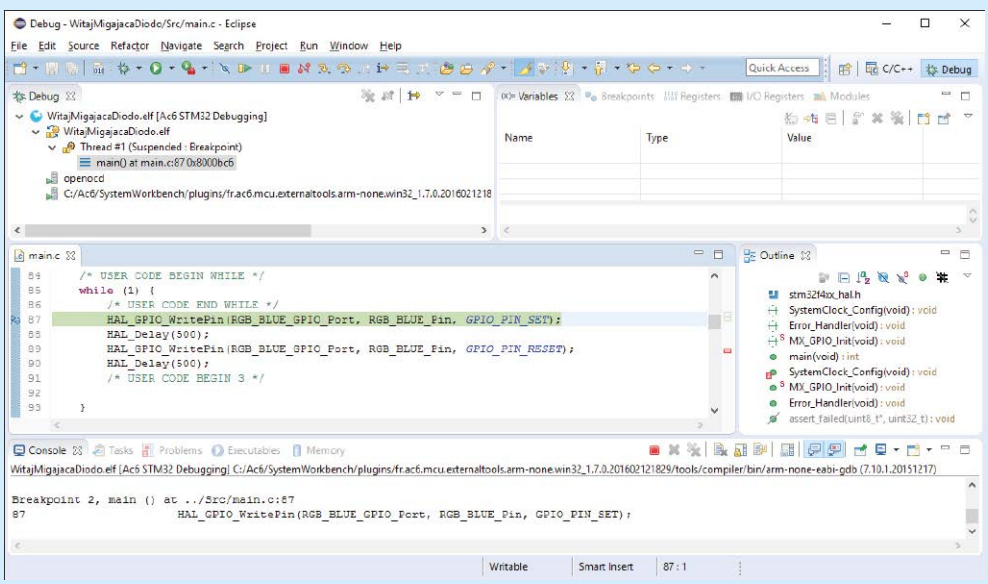
Perspektywy w Eclipse to różne konfiguracje głównego okna programu – każda z nich zawierać może inne paski narzędziowe i panele. Perspektywa debugowania przydatna jest do testowania działania naszej aplikacji, nie jest ona jednak wygodna do pisania kodu – okienko edytora jest w niej dużo mniejsze i nie mamy możliwości kompilowania kodu.

Po przejściu do perspektywy debugowania uruchamiamy nasz program, klikając na ikonie zielonej strzałki z żółtym prostokątem (*Resume*) na pasku narzędziowym. Nasz program jest teraz uruchomiony. Powinniśmy zobaczyć migającą diodę. Jeśli poza pinem PA13, w programie CubeMX, ustawiliśmy w tryb *GPIO_Output* również piny PA14 i PA15, odpowiadające za pozostałe kolory diody (czerwony i zielony), nasza dioda będzie świecić raz na biało, a raz na żółto. Nie będzie natomiast całkiem gasnąć. Aby temu zaradzić, musimy ustawić na wyprowadzeniach PA14 i PA15 poziom wysoki w sekcji *USER CODE 2*. Pamiętajmy, że diody przyłączone są do napięcia dodatniego 3,3 V do wyprowadzenia mikrokontrolera, na którym, po skonfigurowaniu go jako wyjście GPIO, mamy napięcie 0 V.

Aby przerwać debugowanie, należy zaznaczyć sesję debugowania w panelu Debug i usunąć ją przyciskiem *Del*. Następnie możemy powrócić do perspektywy pisania kodu (przycisk *C/C++* w prawym, górnym rogu okna). Jest to o tyle ważne, że jeśli nie usuniemy sesji, nie będziemy mogli ponownie uruchomić programu.

W trybie debugowania, możliwe jest także zatrzymywanie działania programu, po dojściu do określonej linii kodu. Możemy wtedy podglądać zawartość zmiennych i rejestrów, a także ją modyfikować. Aby zatrzymać wykonywanie kodu, odnajdujemy wybraną linię kodu i dwukrotnie klikamy w niebieski pasek po lewej stronie. Powoduje to dodanie breakpointa (pułapki). Usuwamy go w analogiczny sposób, podwójnym kliknięciem. Aby wznowić działanie programu, klikamy przycisk strzałki (*Resume*).

ALEKSANDER KURCZYK



Rysunek 7. Perspektywa debugowania w System Workbench