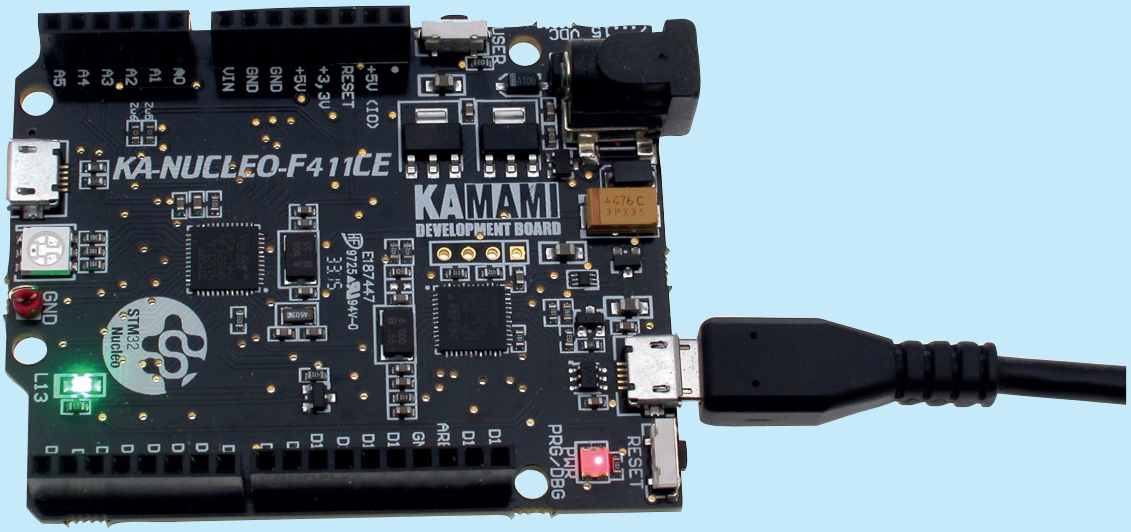


STM-owa układanka: Nucleo, AC6, HAL

Od dłuższego czasu sprzedaż mikrokontrolerów STM32 jest powiązana ze wsparciem dla projektów bazujących na tych układach. Jest to wsparcie polegające zarówno na dystrybucji tanich płytek ewaluacyjnych, jak i na opracowywaniu darmowych narzędzi programistycznych. Takie działanie ma na celu przybliżenie świata STM-ów potencjalnym użytkownikom, ułatwienie i przyśpieszenie pisania oprogramowania. W cyklu kilku artykułów pokażemy jak opanować tytułowe elementy układanki i użyć je do swoich celów, mając przy tym dobrą zabawę.

Rodzina kontrolerów STM32 rozrasta się zgodnie z zasadami: więcej, szybciej, oszczędniej. Pojawiają się nowe warianty z coraz bogatszymi peryferiami, szybsze a jednocześnie o mniejszym zapotrzebowaniu na energię. Płytki startowe mają za zadanie ułatwić testowanie nowych mikrokontrolerów a nawet mogą być wykorzystane do budowy unikatowych, gotowych urządzeń. W przypadku płytek typu NUCLEO, dodatkową zaletą jest rozłożenie złącz i sygnałów zgodne z popularnym standardem ARDUINO. Jako poligon do eksperymentów z STM32, użyta zostanie płytka rozwojowa *KA-NUCLEO-F411CE* z mikrokontrolerem STM32F411CE.

Darmowe oprogramowanie narzędziowe

Ceną za zwiększanie możliwości mikrokontrolerów, jest postępująca komplikacja kodu na etapie konfiguracji. Mnoży się liczba rejestrów sterujących i kombinacji ustawień. Żeby jakoś sobie z tym problemem poradzić firma ST dostarcza darmowe narzędzie programistyczne o nazwie *STM32CubeMX*. To graficzny konfigurator pozwalający przy pomocy myszki i kilku kliknięć dobrać funkcje i ustawienia poszczególnych portów, timerów, sprzętowych interfejsów komunikacyjnych a nawet wewnętrznych zegarów taktujących kontrolera. W równie prosty sposób można wygenerować pliki źródłowe w formacie odpowiednim dla darmowego kompilatora *AC6 System Workbench for STM32*. To kolejny element układanki.

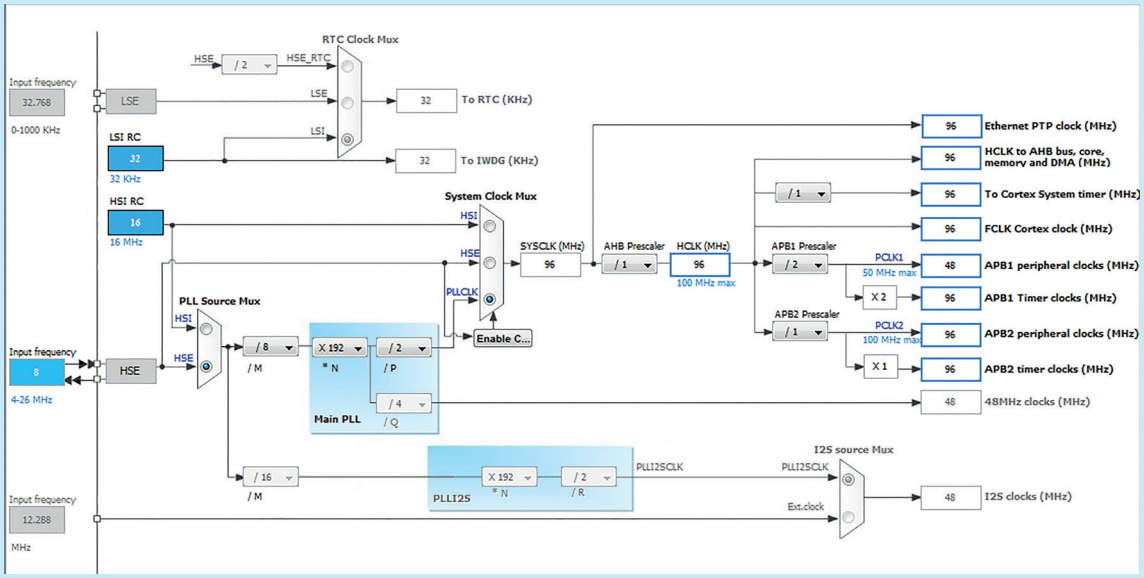
Ponieważ rodzina STM32 wciąż się powiększa, pożądaną cechą pisanego oprogramowania powinna być przenośność kodu. Zastosowanie nowego kontrolera o większych możliwościach, nie powinno wymuszać pisania oprogramowania od nowa, ale musi istnieć możliwość adaptacji starego oprogramowania do nowego sprzętu. W przypadku mikrokontrolerów STM jest to możliwe, jeśli pisząc oprogramowanie korzystamy ze wspólnych bibliotek. Konfigurator *STM32CubeMX* generując szkielet oprogramowania, korzysta z bibliotek HAL, które zarządzają nisko poziomowymi operacjami związanymi ze sprzętową budową kontrolera. Dla nowych typów STM-ów producent dostarcza uaktualnione biblioteki HAL, więc z reguły wystarczy przekompilować stare oprogramowanie z nowymi bibliotekami a otrzymany kod wynikowy będzie się nadawał do zaprogramowania nowego typu kontrolera.

KA-NUCLEO-F411CE

W użytej do testów płytce rozwojowej *KA-NUCLEO-F411CE* zamontowano 32-bitowy mikrokontroler STM32F411CE, z rdzeniem Cortex-M4. Jego najważniejsze parametry są następujące:

- 512 kB pamięci Flash do zapisu programu.
- 128 kB pamięci statycznej SRAM.
- Maksymalna częstotliwość taktowania 100 MHz.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hhcxxtt**



Rysunek 1. Wygląd zakładki „Clock Configuration”

- 7 timerów.
- Interfejsy komunikacyjne: SPI, I²C, UART, USB OTG.
- 12-bitowy przetwornik A/C.

Płytkę zaprojektowano w standardzie Arduino, więc użytkownik może skorzystać z 16 linii D0…15 i 6 linii A0…5, do których doprowadzone są linie portów mikrokontrolera. Z większością tych wyprowadzeń skojarzone są wymienione wcześniej interfejsy i timery.

Do niewątpliwych zalet płytki *KA-NUCLEO-F411CE*, należy zaliczyć zintegrowany programator umożliwiający programowanie i uruchamianie (debugowanie) softu zapisanego w pamięci Flash mikrokontrolera. Oprócz tego na płytce zamontowano: przycisk i diodę LED ogólnego przeznaczenia, trójkolorową diodę LED do wykorzystania przez użytkownika, gniazdo micro USB dla interfejsu USB OTG, przycisk Reset. Płytkę można jednocześnie zasilać z portu USB komputera poprzez gniazdo programatora jak i z zewnętrznego zasilacza 7…15 V o obciążalności co najmniej 400 mA.

Pierwszy program: sterowanie liniami GPIO

Poznawanie nowego kontrolera zwykle rozpoczyna się od opanowania sposobów sterowania jego portami wejść/wyjść. W STM32 nie jest to trudna umiejętność, a przy tym bardzo przydatna. W końcu linie portów są najczęściej używanymi interfejsami do komunikacji kontrolera ze światem zewnętrznym. Napisanie prostego programu do sterowania i odczytywania stanu linii portów, skrótowo nazywanych GPIO, będzie doskonałym sposobem na poznanie działania narzędzia konfigurującego *STM32CubeMX*. Obsługiwanymi liniami niech będą te, podłączone do zamontowanych na płytce diod LED i przycisku.

Na płytce *KA-NUCLEO-F411CE*, do sterowania diodami LED używa się 4 linii portów skonfigurowanych jako wyjściowe i 1 linii wejściowej do odczytu stanu przycisku. Funkcje tych linii i ich powiązanie z portami kontrolera zostały zebrane w **tabeli 1**.

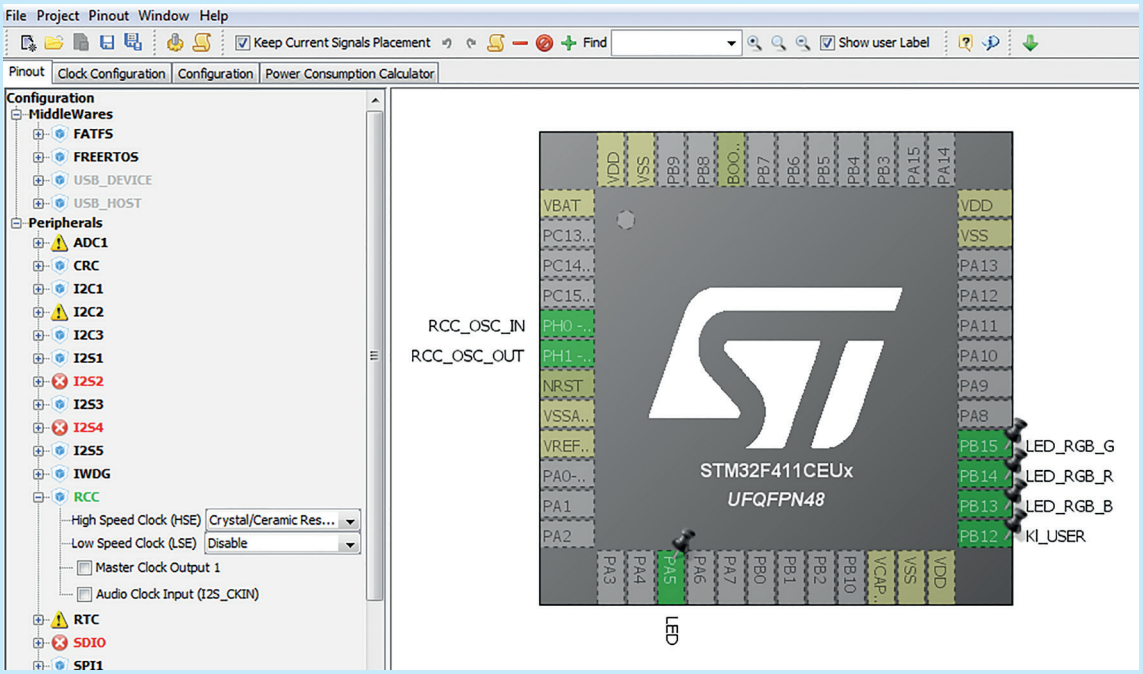
Generowanie szkieletu programu przez konfigurator STM32CubeMX

Zastosowanie konfiguratora *STM32CubeMX*, pozwala programiście na zaoszczędzenie czasu potrzebnego na własnoręczne pisanie procedur inicjacji samego kontrolera jak i poszczególnych interfejsów. Ponadto konfigurator ostrzega przed potencjalnymi konfliktami, gdy używane przez program interfejsy chcą wykorzystywać jednocześnie tą samą linię portu. Najnowszą wersję *STM32CubeMX*, można pobrać ze strony producenta *http://www.st.com/web/en/catalog/tools/PF259242#*. Po typowej instalacji program gotów jest do działania bez żadnych dodatkowych ustawień.

Pracę nad nowym projektem, należy rozpocząć od wybrania typu kontrolera zamontowanego na płytce *New Project* ➞ *STM32F411CEUx*. Na zakładce „Pinout” zostanie wyświetlony rysunek obudowy wybranego układu a z lewej wszystkie dostępne interfejsy. Na początku należy zadeklarować, że wewnętrzny zegar kontroler będzie

Tabela 1. Funkcje wyprowadzeń I/O mikrokontrolera

Funkcja	Port STM32	Opis
Sterowanie pojedynczym LED-em L13 Wyjście	PA5	Sterowanie pojedynczym LED-em poprzez tranzystor, wymagana konfiguracja portu z podciąganiem do +3,3 V, poziom aktywny – wysoki
Sterowanie D2-B Wyjście	PB13	Sterowanie katodą niebieskiej części potrójnego LED-a, podciąganie niewymagane, poziom aktywny – niski
Sterowanie D2-R Wyjście	PB14	Sterowanie katodą czerwonej części potrójnego LED-a, podciąganie niewymagane, poziom aktywny – niski
Sterowanie D2-G Wyjście	PB15	Sterowanie katodą zielonej części potrójnego LED-a, podciąganie niewymagane, poziom aktywny – niski
Odczyt stanu przycisku „USER” Wejście	PB12	Klawisz podciągnięty na płytce do +3,3V, podciąganie nie wymagane po naciśnięciu zwiera do masy



Rysunek 2. Wygląd zakładki „Pinout”

taktowany impulsami z generatora (HSE) z dołączonym rezonatorem kwarcowym *Pinout ➞ RCC ➞ High Speed Clock (HSE) ➞ Crystal*. Wyprowadzenia PH0, PH1 powinny zostać zaznaczone kolorem zielonym. Oznaczać to będzie, że ich wybraną funkcją będzie współpraca z dołączonym zewnętrznym kwarcem. Kolejnym krokiem będzie przejście do zakładki „Clock Configuration” i ustawienie konfiguracji generatora HSE. Ponieważ na płytce zamontowany kwarc pracuje z częstotliwością 8 MHz, taką wartość należy wpisać w polu *Input Frequency Clock Configuration ➞ Input Frequency*=8. Następnie należy wskazać generator HSE jako źródło wewnętrznego zegara taktującego kontroler *Clock Configuration ➞ PLL Source MUX ➞ HSE*. Jeżeli zależy nam, żeby częstotliwość zegara taktującego była wyższa niż częstotliwość kwarcu generatora HSE, należy skorzystać z pętli fazowej PLL jako źródła impulsów taktujących *Clock Configuration ➞ System Clock MUX ➞ PLLCLK*. Dokładną częstotliwość pętli ustawia się korzystając z rejestrów mnożnikowych /M, *N, /P. Ich ustawienia wpływają na generowaną przez pętlę PLL częstotliwość wewnętrznych impulsów zegarowych HCLK. Częstotliwość nie może przekraczać największej dopuszczalnej, w przypadku STM32F411CE jest to 100 MHz. Jeżeli na zakładce *Clock Configuration* zostaną wyświetlone czerwone komunikaty, będzie to sygnał przekroczenia maksymalnej dopuszczalnej częstotliwości i należy skorygować ustawienia rejestrów mnożnikowych /M, *N, /P. Dla częstotliwości HCLK=96 MHz ustawienia rejestrów mnożnikowych będą następujące: /M=8, *N=192, /P=2. Po tych zmianach wygląd zakładki „Clock Configuration” powinien przypominać ten z **rysunku 1**.

Następnym krokiem jest ustawienie trybu pracy dla wybranych linii portów. W tym celu na zakładce „Pinout” należy wskazać kursorem na wybrane wyprowadzenie i kliknąć lewym przyciskiem myszy. Z wyświetlonej listy należy wybrać tryb pracy linii (numer linii portu i tryb pracy podany został w tab. 1):

- *PB12 ➞ GPIO_Input*.
- *PB13 ➞ GPIO_Output*.
- *PB14 ➞ GPIO_Output*.
- *PB15 ➞ GPIO_Output*.
- *PA5 ➞ GPIO_Output*.

Po wybraniu zakładki „Configuration–>System–> GPIO” można dokończyć ustawiania zaznaczonych linii, ustalić czy kontroler będzie je wewnętrznie podciągał, wybrać w przypadku linii wyjściowych szybkość, z którą będą pracowały oraz nadać im etykietę – własną nazwę, pod którą będą występowały w utworzonym kodzie programu:

PA5, Output Push Pull, Pull-Up, maximum output speed Low, user label: LED

PB12, No pull-up and no pull-down, user label: KI_USER

PB13: Output Open Drain, No pull-up and no pull-down, user label: LED_RGB_B

PB14: Output Open Drain, No pull-up and no pull-down, user label: LED_RGB_R

PB15: Output Open Drain, No pull-up and no pull-down, user label: LED_RGB_G

Po zakończeniu ustawień wygląd zakładki „Pinout” powinien przypominać ten z **rysunku 2**.

Żeby można było wygenerować szkielet oprogramowania trzeba jeszcze zadeklarować nazwę projektu np. „KA_NUC_Test_IO”, wybrać katalog docelowy i pakiet kompilatora, dla którego projekt ma być wygenerowany:

Project ➞ Settings ➞ Project Name: KA_NUC_Test_IO

Project ➞ Settings ➞ Project Location (wybrany katalog na dysku użytkownika).

Project ➞ Settings ➞ Tolchain/IDE: SW4STM32

Wreszcie należy wybrać opcję generowania projektu *Project ➞ Generate Code*.

Import wygenerowanego kodu do kompilatora AC6

Pakiet darmowego kompilatora dla kontrolerów i płytek STM32 o nazwie *AC6 (SW4STM32)*, można pobrać spod adresu: *http://www.openstm32.org/HomePage*. Pakiet jest darmowy, ale przed pobraniem należy się zarejestrować. Po rozpakowaniu i instalacji pakiet jest od razu gotowy do pracy. Można przystąpić do importu plików projektu wygenerowanych przez konfigurator *STM32CubeMX*.

Uruchamiając kompilator *AC6* z uprawnieniami administratora, użytkownik zostanie zapytany o miejsce lokalizacji pliku ustawień kompilatora „*Select a workspace*”. Należy podać ścieżkę dostępu zaznaczoną w *STM32CubeMX Project → Settings → Project Location*. Następnie, po otwarciu obszaru roboczego kompilatora należy wybrać opcję: *File → Import… → General → Existing Projects into Workspace*. W opcji Browse… zaznaczyć ścieżkę dostępu podaną w *STM32CubeMX Project → Settings → Project Location*. Ostatnim krokiem jest ustawienie typu programatora zamontowanego na płytce *KA-NUCLEO-F411CE*. Z menu należy wybrać *Run → Debug Configuration → Debugger* i zaznaczyć opcję *Manual spec*. W tym momencie wyświetlą się dodatkowe pola, za których pomocą należy wybrać typ programatora i jego interfejs: *Debug device: ST-LinkV2-1* oraz *Debug interface: SWD*.

Po zaakceptowaniu ustawień przyciskiem *Apply* IDE jest gotowe do generowania kodu wynikowego, jego zapisu do pamięci Flash kontrolera i do debugowania. Widok obszaru roboczego po zakończeniu importu plików do edytora pakietu kompilatora pokazano na **rysunku 3**.

Biblioteka sterowników HAL

Efektem działania konfiguratora *STM32CubeMX*, jest wygenerowanie szkieletu oprogramowania zawierającego procedury inicjujące kontroler oraz użyte w projekcie interfejsy np. porty GPIO. Procedury inicjujące umieszczone zostały w pliku *main.c*.

W przykładowym programie do odczytywania stanu przycisku i sterowania zapaleniem i gaszeniem diod LED, najważniejszymi procedurami inicjującymi są:

- Konfigurowanie sposobu taktowania mikrokontrolera (**listing 1**). Zgodnie z dokonanym w programie konfiguratora wyborem, kontroler będzie pracował z generatorem HSE i zewnętrznym kwarcem o częstotliwości 8 MHz. Pętla fazowa PLL będzie generowała wewnętrzny zegar taktujący kontrolera HCLK, którego częstotliwość będzie wynosiła 96 MHz.
- Konfigurowanie funkcji wyprowadzeń portów mikrokontrolera (**listing 2**). Do użytych portów podłączone zostaną wewnętrzne impulsy zegara taktującego. Zgodnie z ustawieniami w konfiguratorze, linie portów zadeklarowane jako używane zostaną zainicjowane w odpowiednim trybie pracy: jako wejściowe lub wyjściowe, z podciąganiem lub bez itd. Podczas inicjowania użyte zostaną nazwy portów podane w procesie konfigurowania.

W automatycznie wygenerowanym pliku *mxconstans.h* znajduje się blok deklaracji nazw i przyporządkowania do nich stałych rozpoznawanych przez kompilator jako numery linii i portów. W przypadku deklaracji odnoszących się do stałych oznaczających porty (np. PA, PB itd.), do zadeklarowanych nazw automatycznie dodane zostały sufiksy „_GPIO_Port”, natomiast w wypadku linii sufiksy „_Pin”.

W wygenerowanych procedurach inicjacji wykorzystane zostały polecenia biblioteki HAL. Dokumentacja biblioteki HAL (***UM1725:*** *Description of STM32F4xx HAL drivers*) jest do pobrania tutaj: *http://goo.gl/zFUDeY*.

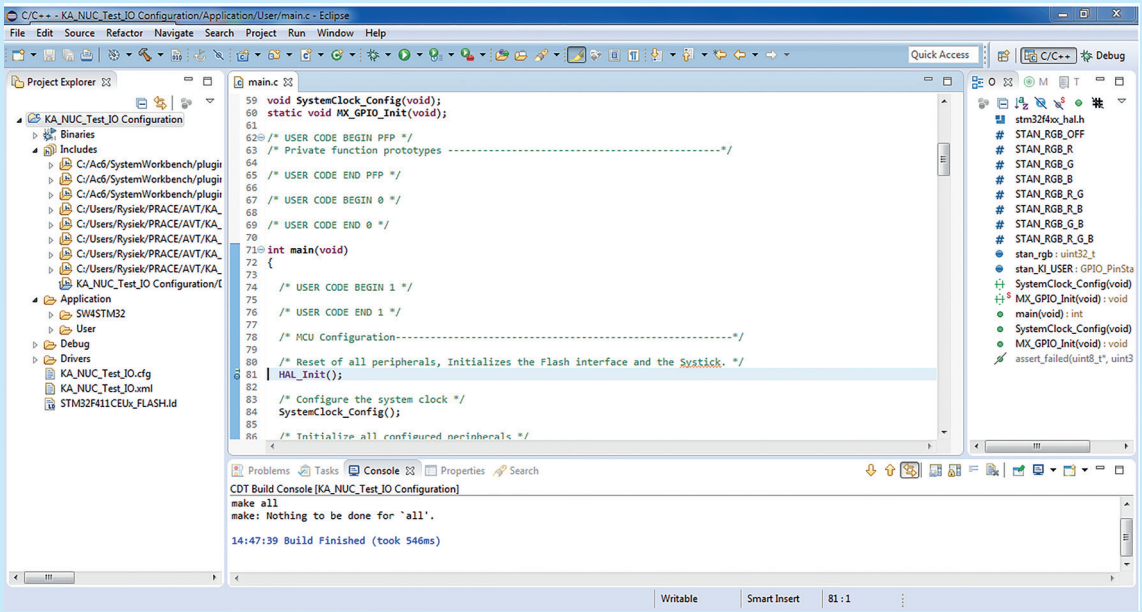
Operacje związane z portami GPIO

Wygenerowane automatycznie pliki z procedurami inicjującymi ułatwiają programiście pracę zwalniając z obowiązku pisania tej części kodu. Jednak program poza „kręceniem się” w nieskończonej pętli while*(1)*{} nie robi niczego użytecznego. Resztę kodu programista musi napisać samodzielnie. W naszym programie przykładowym do sterowania diodami LED na płytce *KA-NUCLEO-F411CE* potrzebne będą komendy odwołujące się do procedur związanych z portami GPIO. Opis procedur można znaleźć w dokumentacji biblioteki HAL w sekcji *HAL GPIO Generic Driver*. Niektóre z nich np.: *HAL_GPIO_Init* były już wykorzystywane podczas inicjacji portów sterujących diodami LED. W tej chwili będą nas szczególnie interesowały procedury do odczytu stanu portów oraz w przypadku wyjść procedury zmiany ich poziomu.

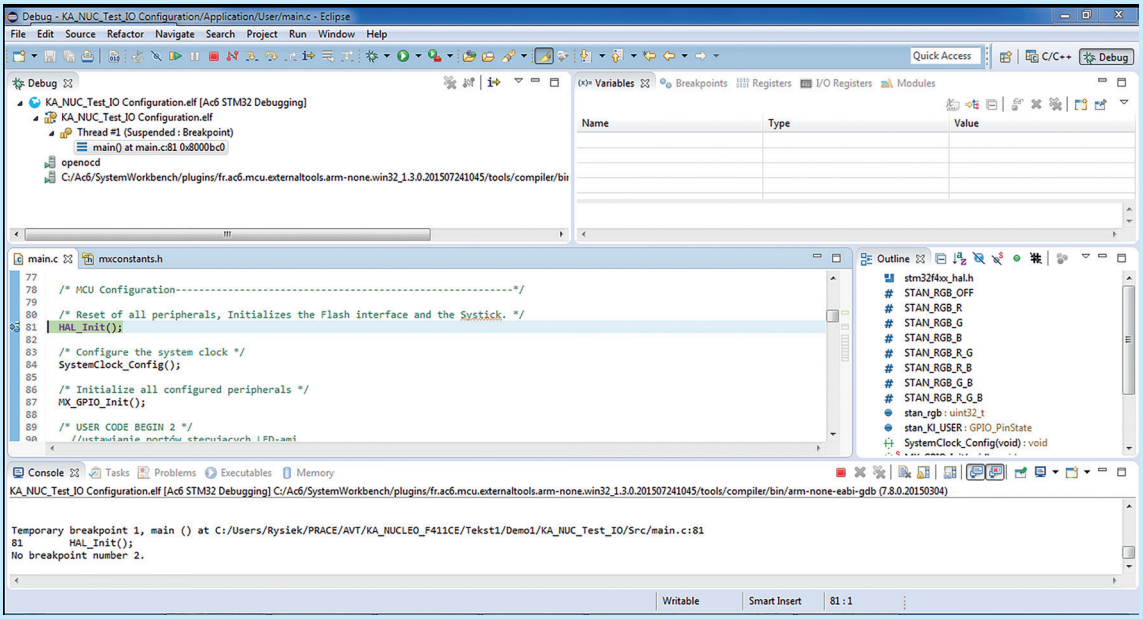
Polecenie *stan_Kl_USER = HAL_GPIO_ReadPin(Kl_USER_GPIO_Port, Kl_USER_Pin);* odczytuje stan przycisku i zapisuje go w zadeklarowanej wcześniej zmiennej *stan_KL_USER*. Port dołączony do przycisku może przyjąć jeden z dwóch stanów:

1. *GPIO_PIN_RESET* – poziom niski,
2. *GPIO_PIN_SET* – poziom wysoki.

W ten sposób program może dowiedzieć się czy przycisk jest naciskany.



Rysunek 3. Widok obszaru roboczego po zakończeniu importowania plików do edytora



Rysunek 4. Perspektywa „Debug”

Do sterowania liniami portów połączonych z katodami diod LED zamontowanych na płytce *KA-NUCLEO-F411CE*, można wykorzystać następujące rozkazy:

- *HAL_GPIO_WritePin(LED_RGB_B_GPIO_Port, LED_RGB_B_Pin, GPIO_PIN_SET)* – zgaszenie diody B.
- *HAL_GPIO_WritePin(LED_RGB_R_GPIO_Port, LED_RGB_R_Pin, GPIO_PIN_RESET)* – zaświecenie diody R.
- *HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin)* – przełączenie stanu diody LED na przeciwny.

Rozkaz *HAL_GPIO_WritePin()* jako argumentów wymaga nazwy portu, nazwy linii i poziomu jaki linia ma przyjąć. Mogą to być *GPIO_PIN_RESET* (poziom niski) lub *GPIO_PIN_SET* (poziom wysoki). W wypadku diod LED, których anody podłączone są do napięcia zasilania, podanie na port połączony z katodą poziomu niskiego spowoduje zaświecenie się diody.

Rozkaz *HAL_GPIO_TogglePin()* jako parametrów potrzebuje nazwy portu i nazwy linii. Efektem jego działania jest przełączenie diody w stan przeciwny. Na przykład, jeśli była zaświecona po wykonaniu rozkazu zostanie zgaszona. Rozkaz zapisu może grupowo ustawiać stan linii tego samego portu. Dla przykładu, linia kodu:

HAL_GPIO_WritePin(LED_RGB_R_GPIO_Port, LED_RGB_R_Pin | LED_RGB_G_Pin, GPIO_PIN_RESET);

powoduje jednoczesne zaświecenie się diod LED_RGB_R i LED_RGB_G.

Istnieje jeszcze inna metoda sterowania portami wyjściowymi. Wymaga ona odwołania się bezpośrednio do rejestru sterującego liniami wybranego portu. Zapis *LED_RGB_R_GPIO_Port->BSRR = (LED_RGB_R_Pin | LED_RGB_G_Pin) <<16;* powoduje ustawienie wybranych starszych bitów w 32-bitowym rejestrze GPIOx_BSRR (gdzie x jest literą A, B, C… określającą wybrany port). Ustawienie starszych bitów powoduje wyzerowanie odpowiadających im linii portu. Z kolei zapis *LED_RGB_R_GPIO_Port->BSRR = (LED_RGB_R_Pin | LED_RGB_G_Pin);* powoduje ustawienie młodszych bitów w 32-bitowym rejestrze GPIOx_BSRR. Będzie to powodowało ustawienie odpowiadających im linii portu. Opis działania rejestru GPIOx_BSRR można znaleźć w dokumentacji technicznej kontrolera STM411 „Reference Manual RM0383” dostępnej na stronach firmy ST.

Debugowanie programu

Po napisaniu programu lub jego części, kolejnym etapem jest zapis kodu wynikowego do pamięci Flash mikrokontrolera zamontowanego na płytce *KA-NUCLEO-F411CE*, a następnie jego debugowanie, jeżeli nie działa prawidłowo. Gniazdo mikro USB programatora zamontowanego na płytce, należy połączyć kablem z dowolnym portem USB komputera PC. Napisany kod źródłowy trzeba skompilować wybierając opcję *Project → Build Project* lub klikając na ikonę z symbolem młotka. Jeżeli kompilacja przebiegnie bez błędów, można uruchomić procedurę automatycznego zapisu do pamięci Flash mikrokontrolera i przejść do trybu debugowania. W tym celu należy wybrać *Run → Debug* lub kliknąć ikonę z zielonym żukiem. Zapis do pamięci Flash, a potem gotowość do rozpoczęcia debugowania będzie sygnalizowana migotaniem diody PWR PRG/DBG zamontowanej na płytce *KA-NUCLEO-F411CE*. Otworzona zostanie nowa zakładka pulpitu tzw. Perspektywa Debug podobna do tej pokazanej na **rysunku 4**. Na zakładce dostępne są typowe opcje procesu debugowania: start i stop wykonywanego programu, praca krokowa, ustawianie pułapek itp. Wykonywanie programu zaczyna się po wybraniu opcji Run-Resume, kończenie po wybraniu *Run → Terminate* lub *Disconect* albo po kliknięciu na odpowiadające im ikony.

Na koniec jedna uwaga praktyczna. Własny kodu, w plikach automatycznie generowanych przez narzędzie konfiguratora *STM32CubeMX*, najbezpieczniej dopisywać w przeznaczonych do tego rejonach. Są one ograniczone komentarzami zaczynającymi się od słów **USER CODE** np.:

/ USER CODE BEGIN 2 */*
miejsce na kod użytkownika
/ USER CODE END 2 */*

Jeżeli przeprowadzimy ponowną generację plików przez *STM32CubeMX*, bo np. chcieliśmy dodać obsługę kolejnej linii portu, stare pliki zostaną nadpisane z wyjątkiem miejsc przeznaczonych na kod użytkownika. Dzięki temu oszczędzimy sobie konieczności ponownego wklepywania dodawanego przez nas kodu. Po procesie automatycznego generowania bez zmian pozostaną także pliki, w całości utworzone przez użytkownika podczas pisania programu.

Ryszard Szymaniak, EP