

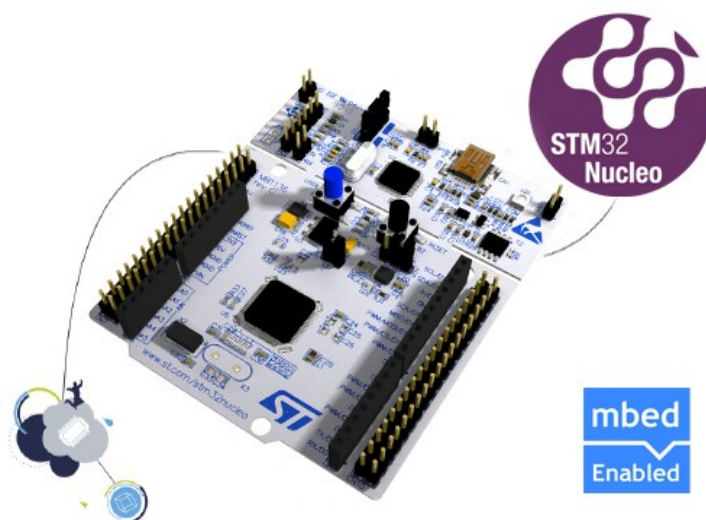
# Nucleo-F334R8 – Właściwości i praca z zestawem

**Autor:** Jan Szemiet, [john.s.22x@gmail.com](mailto:john.s.22x@gmail.com)

**Słowa kluczowe:** zestaw, nucleo, F334R8, stm32, właściwości, arduino, morpho, dioda, przycisk, mbed, projekt, szablon, c++, moduł, KAmoLED8

## Zawartość artykułu:

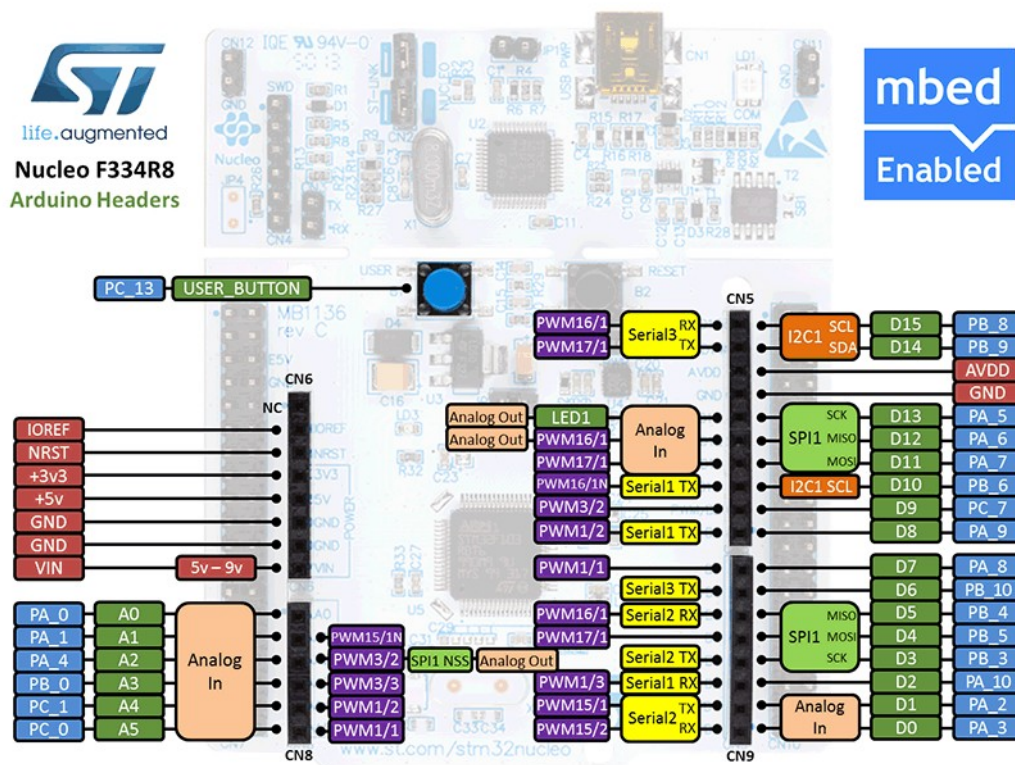
- Charakterystyka zestawu Nucleo-F334R8
- Szablon projektu w środowisku programistycznym mbed
- Projekt aplikacji sterującej diodami LED



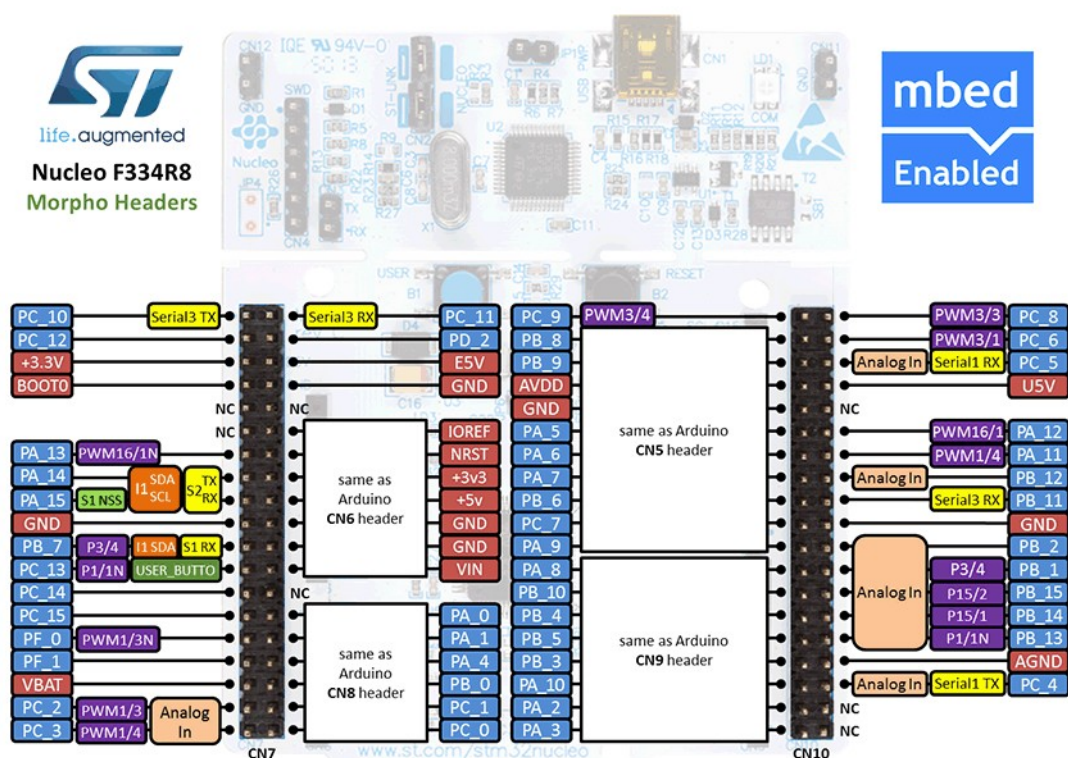
## Charakterystyka zestawu Nucleo-F334R8

Płytką Nucleo-F334R8 jest jedną z dziewięciu aktualnie dostępnych modeli płytek rodziny Nucleo, identycznych pod względem budowy i różniących się jedynie zamontowanym mikrokontrolerem STM32. W porównaniu do istniejących od dłuższego czasu popularnych zestawów Discovery rodzinę płytek Nucleo cechuje mechaniczna i elektryczna zgodność z płytkami systemu Arduino™ (**rysunek 1**), obecność złącz nowego systemu połączeniowego ST Morpho (**rysunek 2**) oraz możliwość współpracy z internetowym środowiskiem mbed.

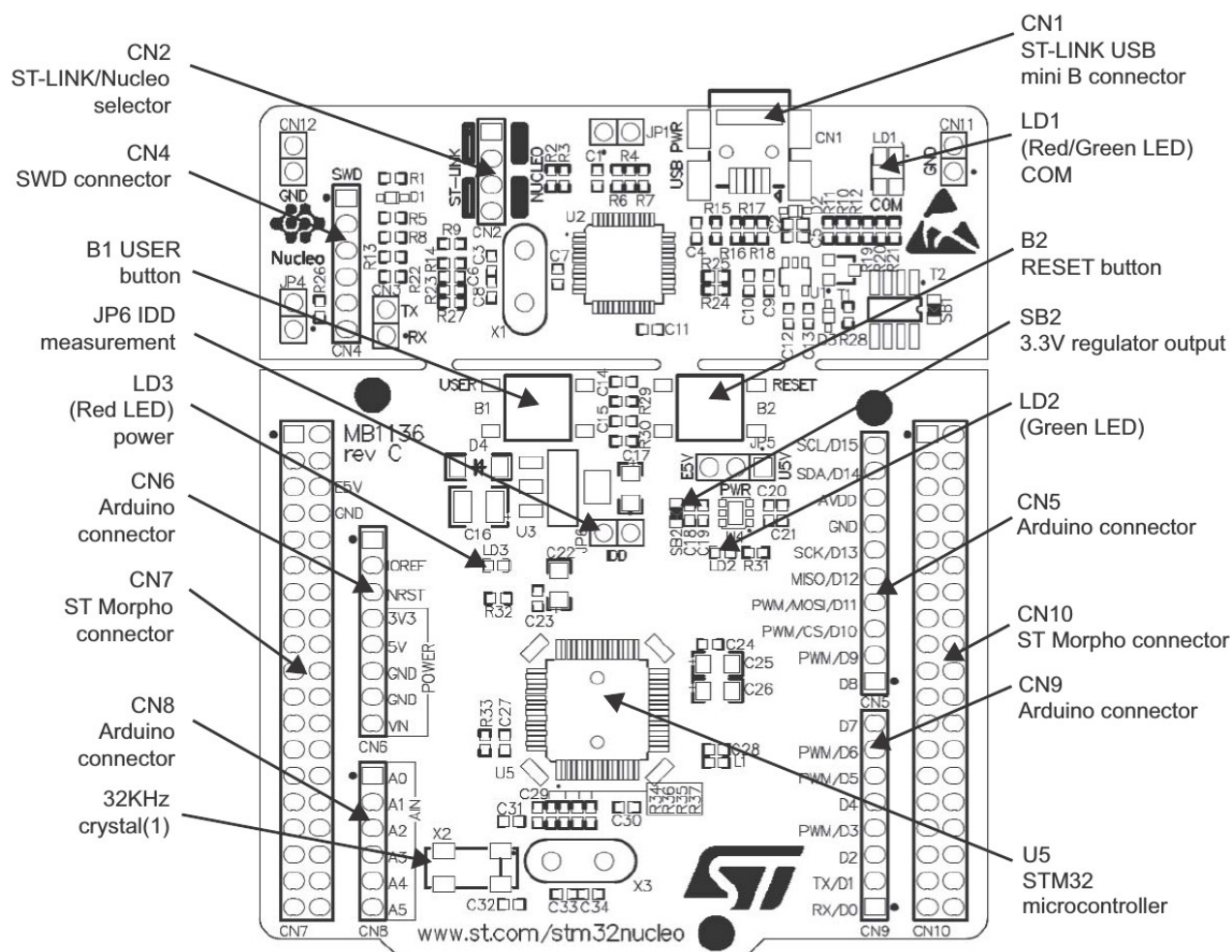
Część użytkowa zestawu z mikrokontrolerem STM32F334R8T6 posiada dość skromne wyposażenie (**rysunek 3**) w postaci dwóch diod LED (zielona LD2 do wykorzystania w aplikacji użytkownika oraz czerwona LD3 informująca o zasilaniu mikrokontrolera), dwóch przycisków (USER – ogólnego przeznaczenia oraz RESET – resetujący mikrokontroler), zworki JP6 umożliwiającej pomiar poboru prądu przez mikrokontroler oraz zworki JP5 odpowiedzialnej za wybór źródła zasilania mikrokontrolera – zasilanie może być doprowadzone ze złącza USB umieszczonego w części programatora zestawu lub z odpowiednich wyprowadzeń w części użytkowej, do których podłączone jest zewnętrzne źródło zasilania (VIN (7V-12V), E5V (5V), +3V3 (3.3V)). Dodatkowo, w dolnej części płytki znajduje się oscylator 32 kHz (wykorzystywany przez RTC mikrokontrolera) oraz miejsce przeznaczone do zamontowania zewnętrznego rezonatora kwarcowego dużej częstotliwości (domyślnie mikrokontroler pracuje z wewnętrznym generatorem taktującym).



Rys. 1. Złącza mechanicznie i elektrycznie zgodne z systemem Arduino™



Rys. 2. Złącza nowego systemu połączeniowego ST Morpho umożliwiające dostęp do wszystkich wyprowadzeń GPIO mikrokontrolera STM32



Rys. 3. Rozmieszczenie elementów w zestawie Nucleo-F334R8

Wspomniany powyżej mikrokontroler STM32F334R8T6, wyposażony w 32-bitowy rdzeń ARM Cortex-M4 (mogący pracować z częstotliwością 72MHz) z jednostką zmiennoprzecinkową FPU, posiada 64 kB pamięci Flash, 16 kB pamięci ulotnej SRAM oraz 51 wyprowadzeń GPIO (w tym 25 mogących pracować z poziomami napięć do 5V). Jest on przeznaczony głównie do zastosowań w cyfrowych układach zasilających, jak np. zasilacze impulsowe sterowane cyfrowo, układy sterowania oświetleniem oraz układy bezprzewodowego ładowania. Z tego względu do najważniejszych podsystemów mikrokontrolera należą:

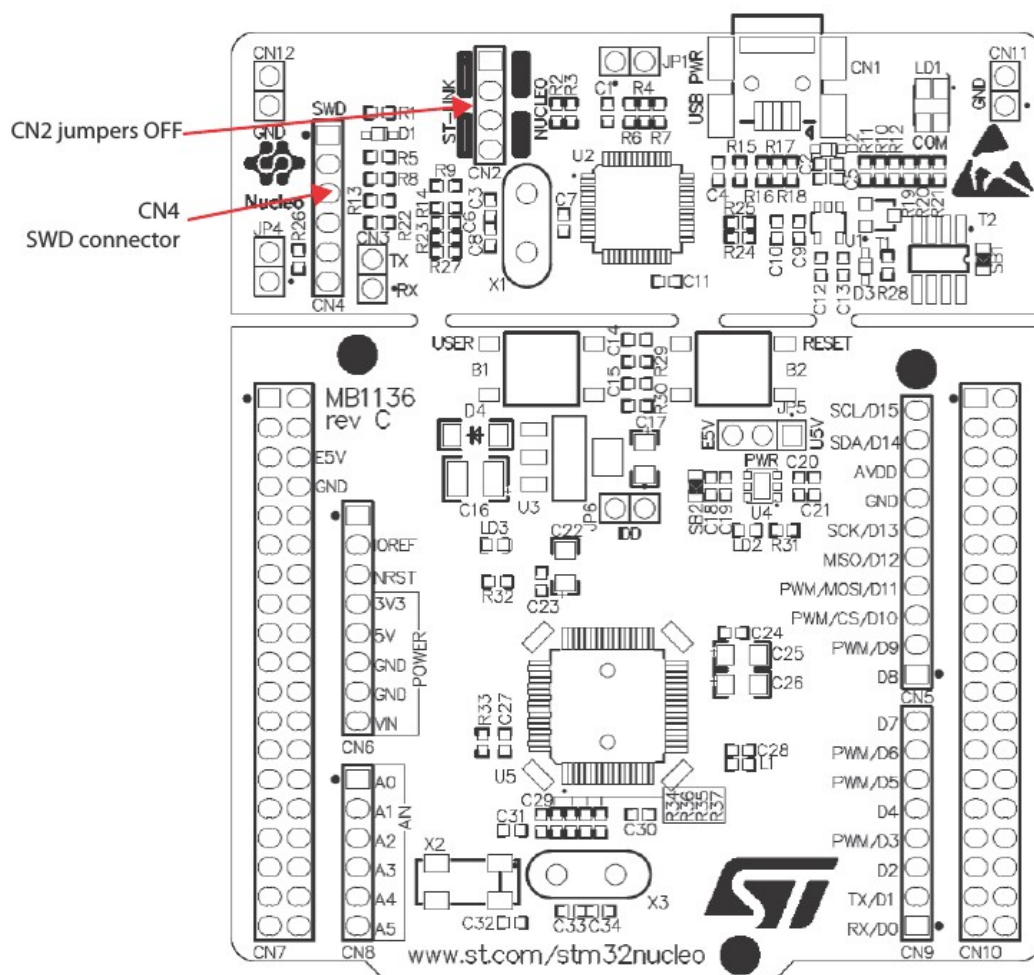
- 2 przetworniki A/C z minimalnym okresem próbkowania wynoszącym 0.2 us (5Mps) i wybieralną 12/10/8/6-bitową rozdzielczością oraz 21 kanałami, mogące pracować zarówno w trybie jednokońcówkowym (pomiar względem analogowej masy) jak również w trybie różnicowym (pomiar różnicy napięć między dwoma liniami),
- 12-bitowy przetwornik C/A z 3 kanałami,
- 3 ultraszybkie (50 ns) komparatory analogowe typu rail-to-rail,
- 1 wzmacniacz operacyjny mogący pracować w trybie PGA z programowalnym wzmocnieniem równym 2, 4, 8 lub 16,
- 16-bitowy licznik HRTIM wysokiej rozdzielczości (217 ps) z 10 kanałami,
- 16-bitowy licznik rozszerzony,
- 4x 16-bitowe oraz 1x 32-bitowy liczniki ogólnego przeznaczenia.

Oprócz powyższych w układzie można znaleźć również:

- interfejsy komunikacyjne: 1x CAN 2.0B, 1x I<sup>2</sup>C, 1x SPI, 3x USART (w tym LIN oraz IrDA),
- kontroler DMA z 7 kanałami,

- układ licznika RTC z możliwością wybudzania układu z trybu stop,
- unikalny 96-bitowy numer ID,
- wbudowana jednostka do obliczania sumy kontrolnej CRC.

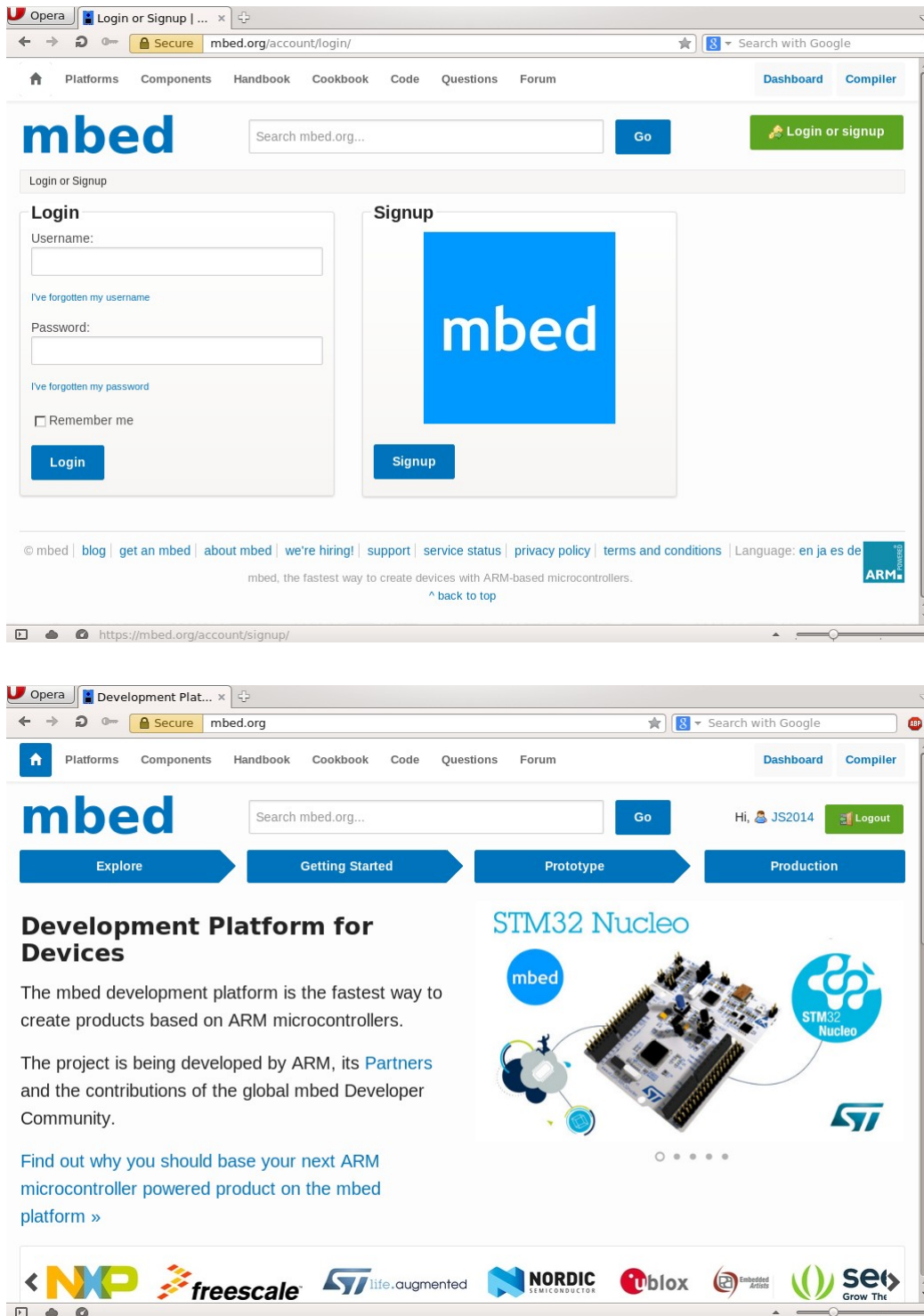
W omawianym zestawie Nucleo znajduje się zintegrowany programator/debuger ST-LINK/V2-1, współpracujący ze środowiskiem mbed i będący rozszerzeniem popularnego ST-LINK/V2 obsługiwanego w popularnych środowiskach, takich jak np. Atollic TrueStudio, Keil uVision i IAR Embedded Workbench. Złącze miniUSB, jakie znajduje się na płytce, oprócz doprowadzania zasilania i umożliwienia komunikacji z programatorem jest wykorzystywane również przy emulacji pamięci masowej urządzenia USB, a także w komunikacji pomiędzy głównym mikrokontrolerem zestawu a komputerem (dzięki zaimplementowanej funkcji wirtualnego portu w układzie programatora). W tym ostatnim przypadku istnieje również możliwość skomunikowania się z zewnętrznym układem, wyposażonym w interfejs USART, dzięki zamontowanemu na płytce złączu CN3 w postaci gold-pinów z liniami Rx i Tx. Podobnie wygląda sytuacja z programowaniem zewnętrznych mikrokontrolerów, przy czym wykorzystywane jest do tego celu 6-sygnałowe złącze SWD (należy pamiętać o odpowiedniej konfiguracji zworek na złączu CN2, **rysunek 4**). Dwukolorowa dioda LD1 znajdująca się w części programatora służy do sygnalizacji stanu pracy programatora.



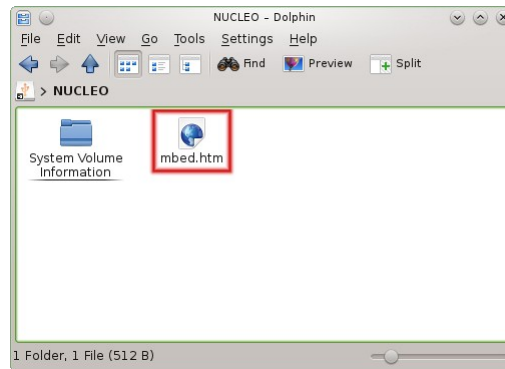
Rys. 4. Konfiguracja zworek na złączu ST-LINK/NUCLEO

## Szablon projektu w środowisku programistycznym mbed

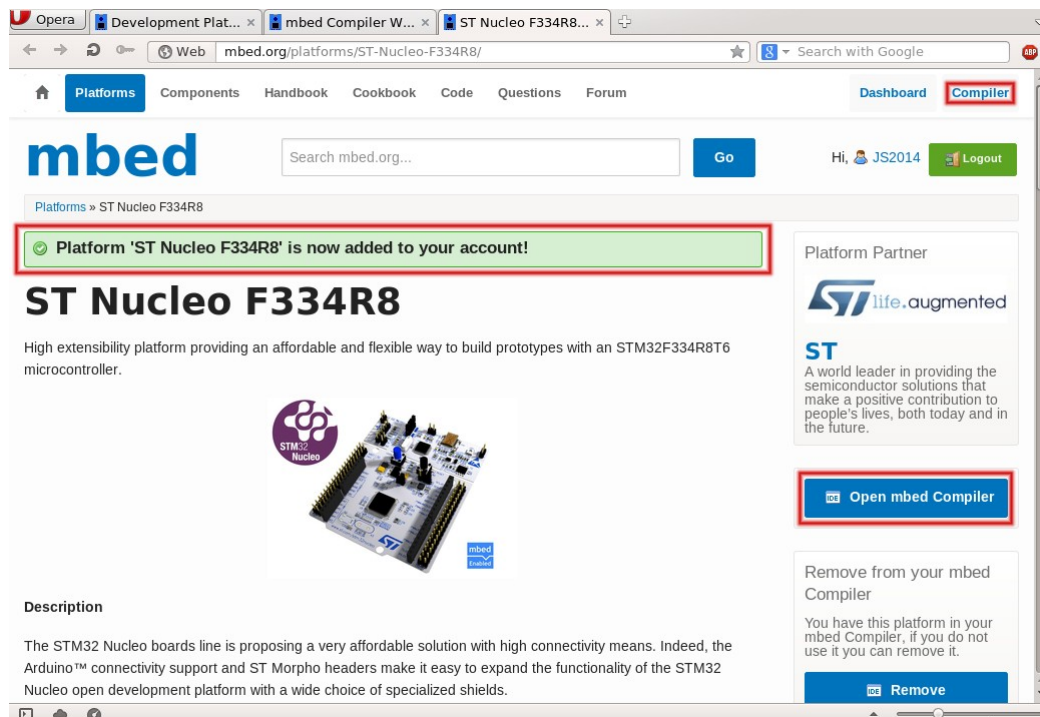
Uruchomienie środowiska programistycznego **mbed** rozpoczynamy od zalogowania/rejestracji na stronie <https://mbed.org/account/login/> oraz kliknięcia przycisku *Compiler* w prawym górnym rogu strony głównej.

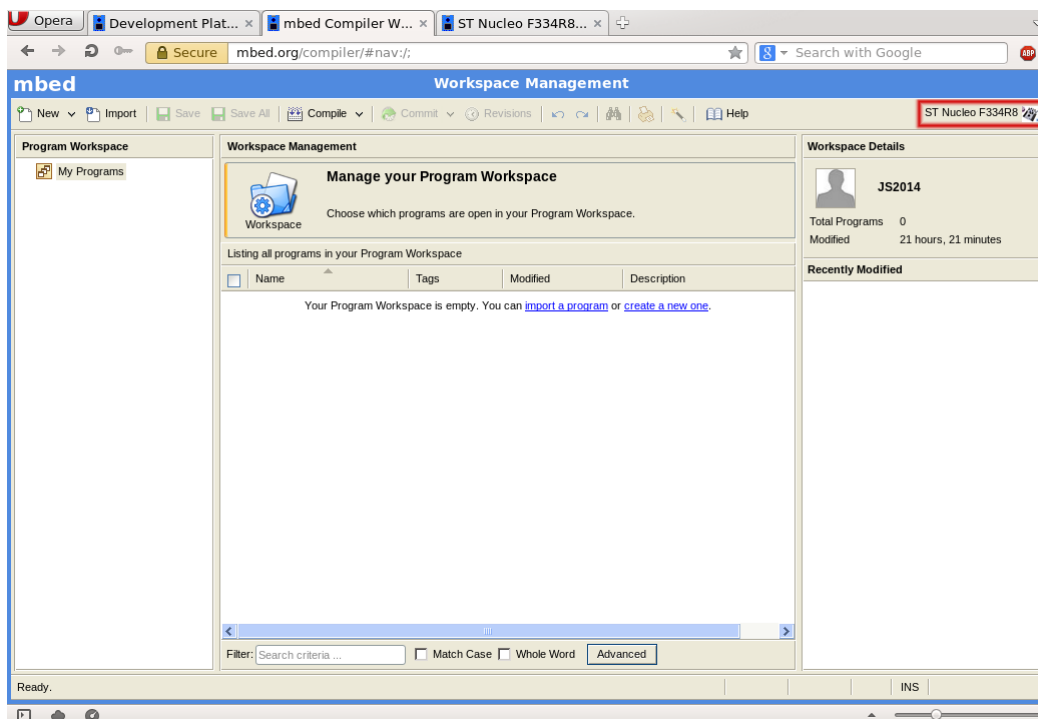


Następnie dodajemy zestaw Nucleo-F334R8 do listy docelowych platform środowiska, przy czym można to zrobić na dwa sposoby. Pierwszy sposób polega na podłączeniu płytki Nucleo do komputera poprzez przewód USB i otwarciu pliku `mbed.htm` znajdującego się na dysku NUCLEO, jaki zaimplementowano w zestawie.



W rezultacie powinna załadować się strona z informacją o pomyślnym dodaniu platformy, po czym można przejść do okna projektu.

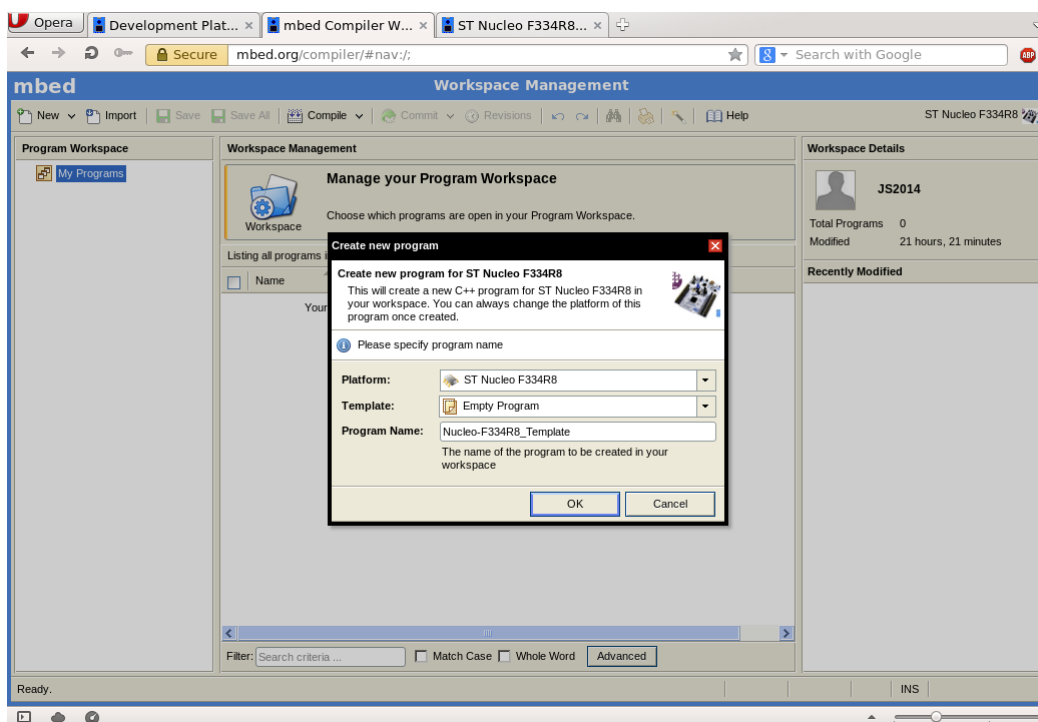
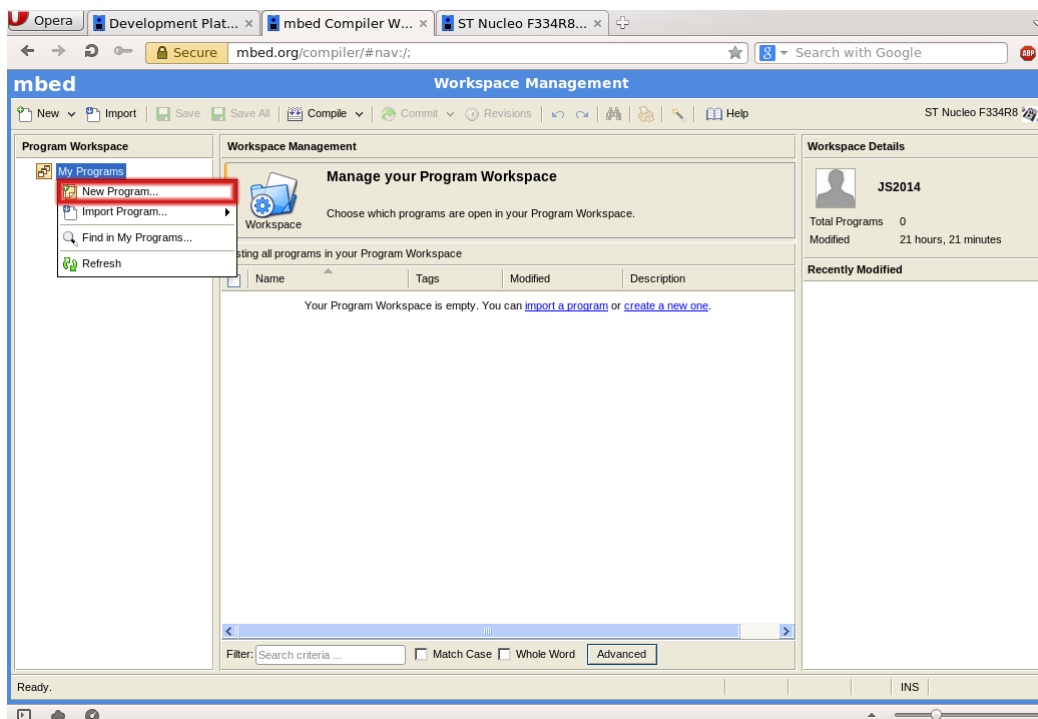




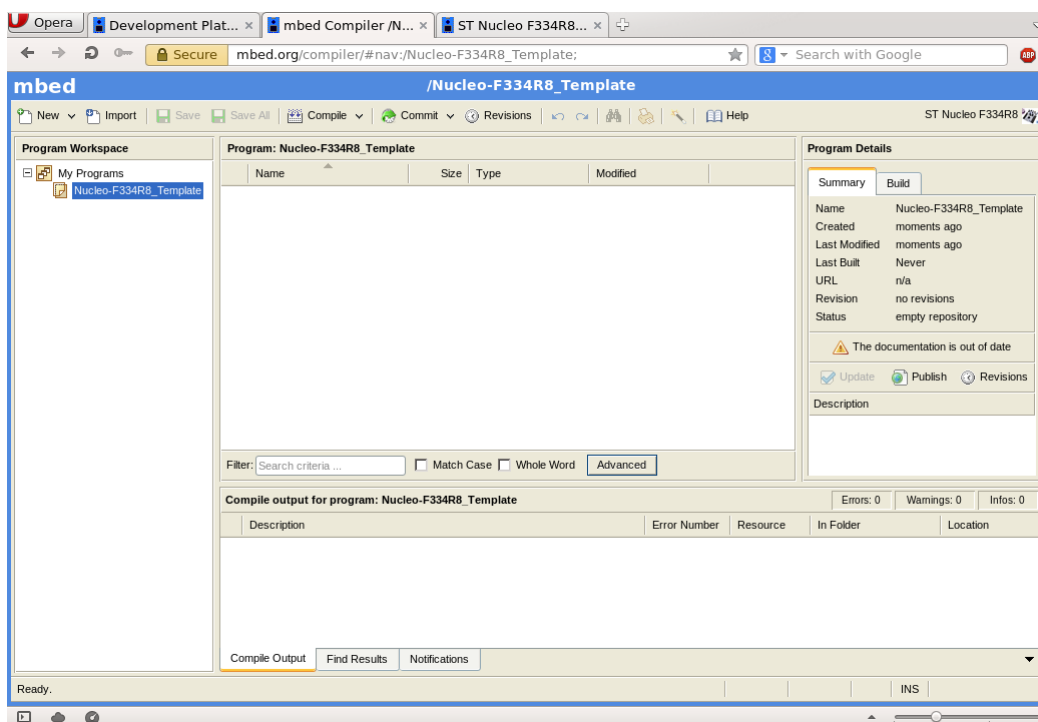
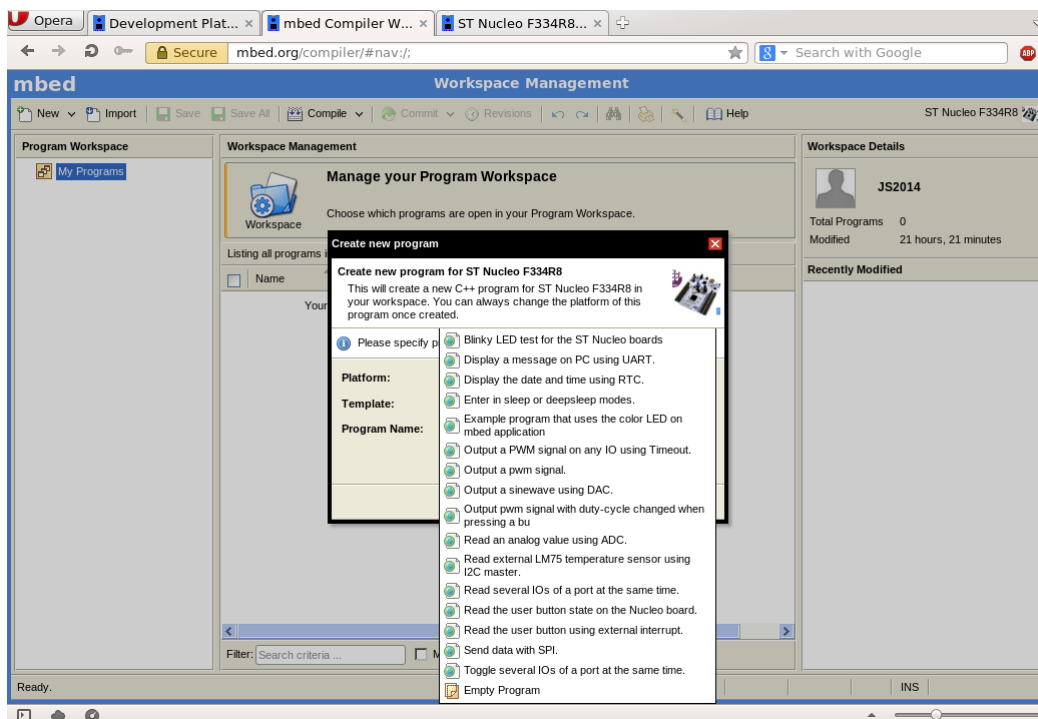
W przypadku drugiego sposobu należy kliknąć ikonkę w prawym górnym rogu okna projektu i w okienku *Select a Platform* wcisnąć przycisk *Add Platform*. Na stronie, jaka się załaduje, znajdujemy interesujący nas zestaw, wybieramy go oraz klikamy *Add to your mbed Compiler* (jeśli zestaw został już dodany to w tym miejscu można go usunąć z listy przyciskiem *Remove*).



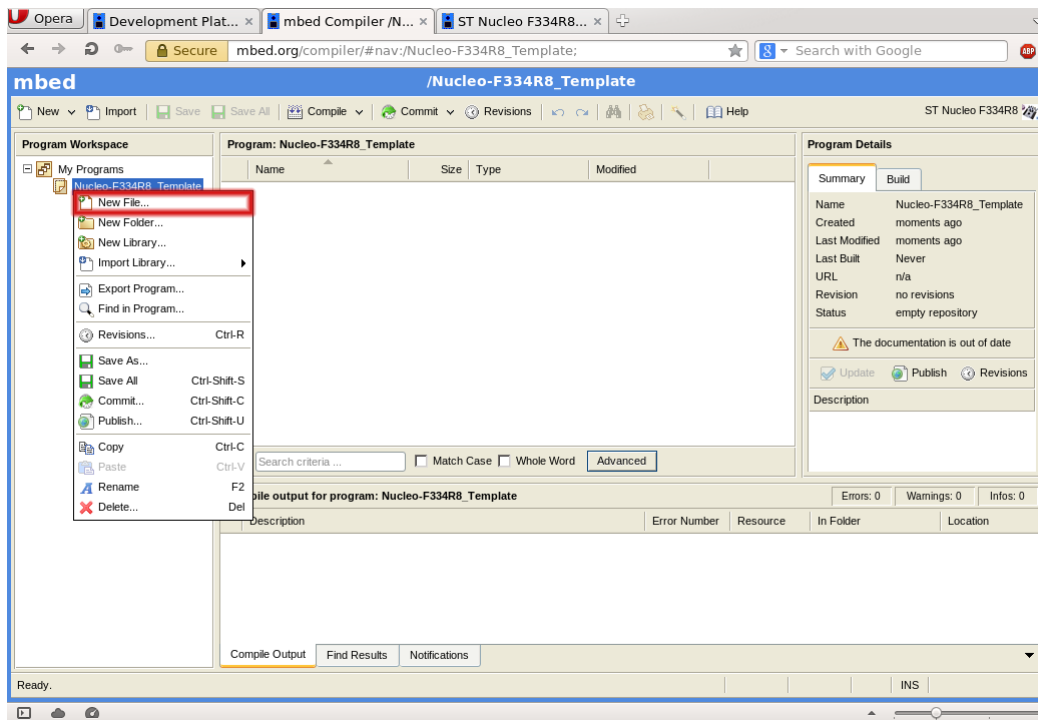
Aby utworzyć nowy projekt należy kliknąć PPM na głównym katalogu *My Programs* w sekcji *Program Workspace* i wybrać *New Program...* W otwartym okienku *Create New Program* wybieramy platformę *ST Nucleo F334R8*, szablon *Empty Program* i wprowadzamy nazwę projektu, np. *Nucleo-F334R8\_Template*.



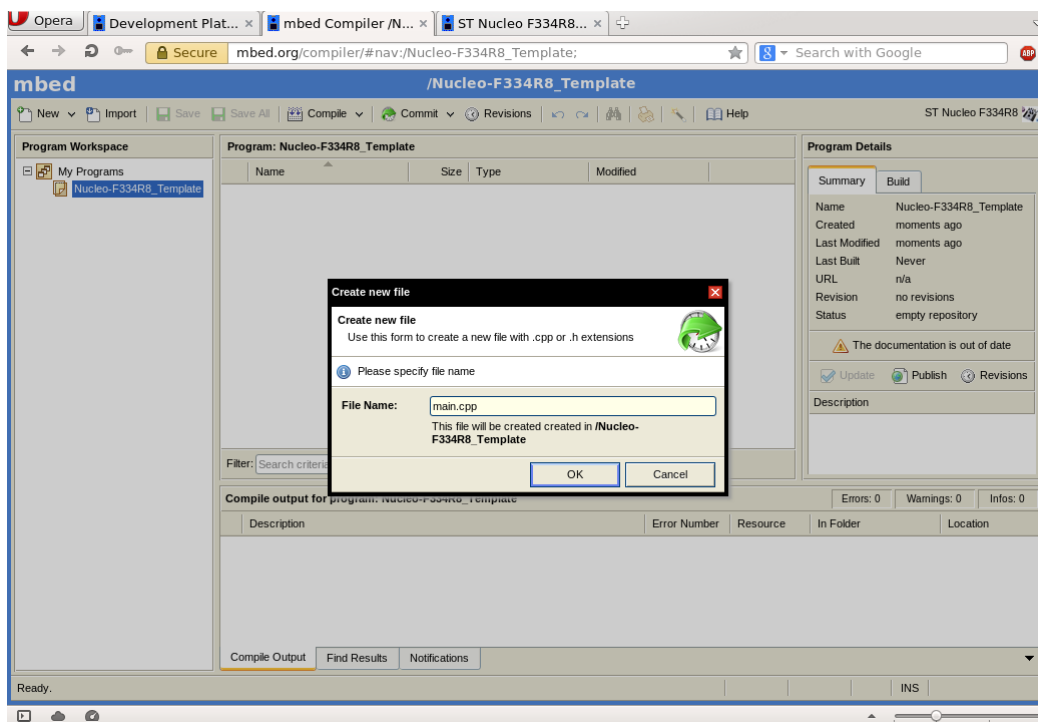
W tym miejscu można wybrać jeden z dostępnych szablonów i zapoznać się z przykładowymi projektami, ilustrującymi pracę z różnego rodzaju peryferiami zestawu oraz mikrokontrolera.

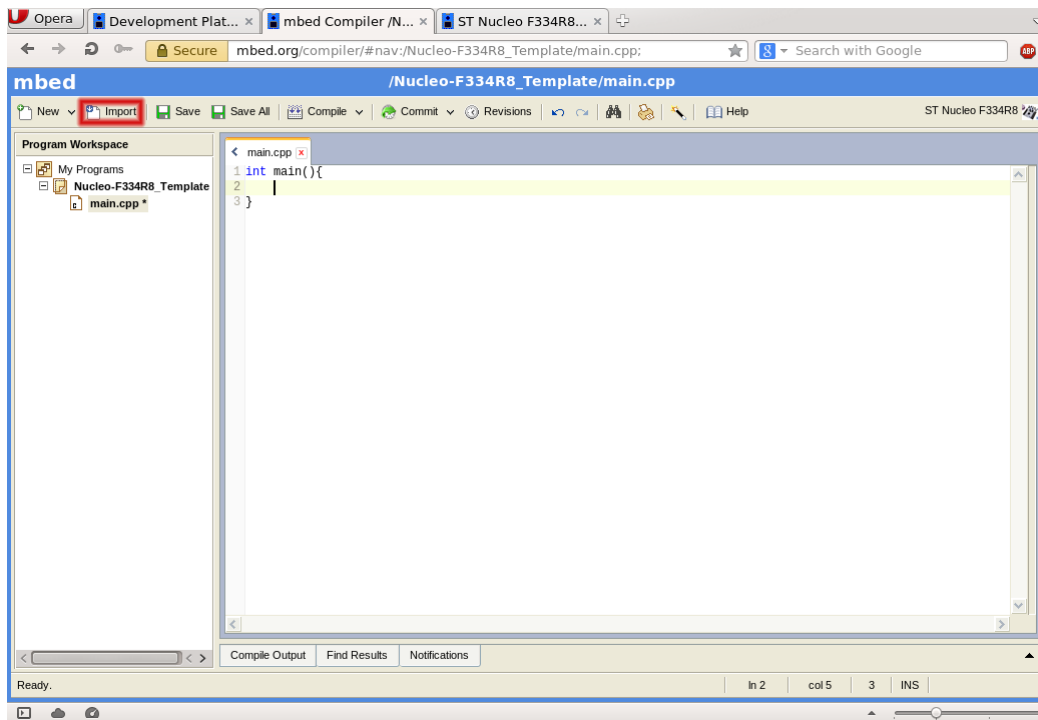


Po zatwierdzeniu nowego projektu w sekcji *Program Workspace* powinien ukazać się nowy katalog, na którym klikamy PPM i wybieramy *New File...*

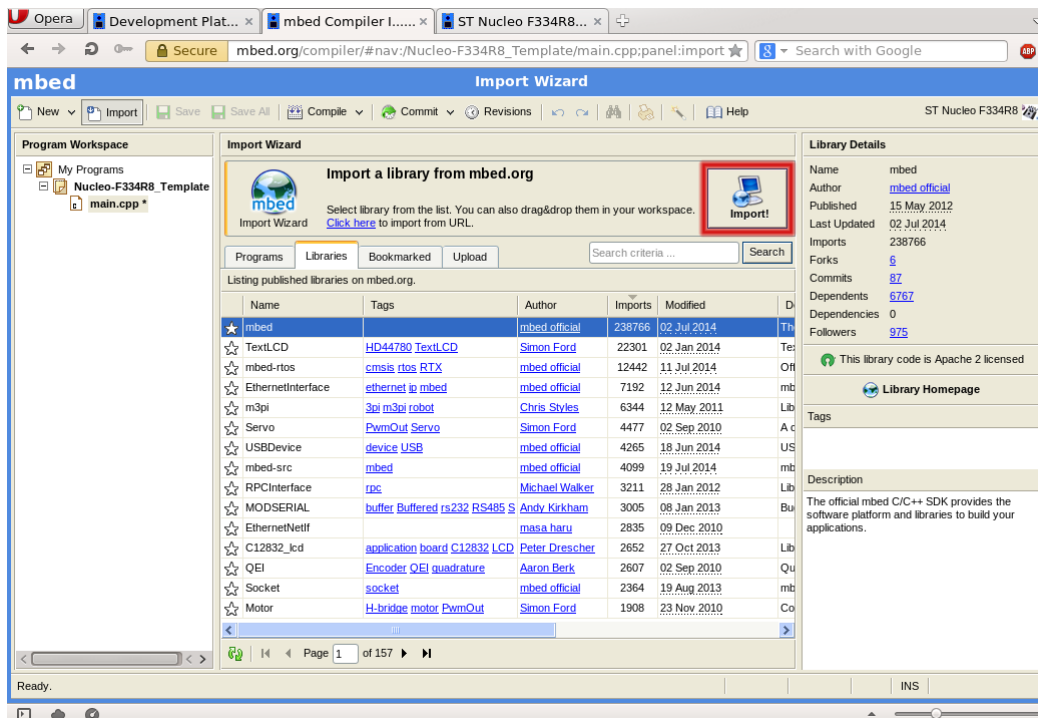


Tworzymy plik `main.cpp` i dodajemy pustą definicję funkcji `main()`.

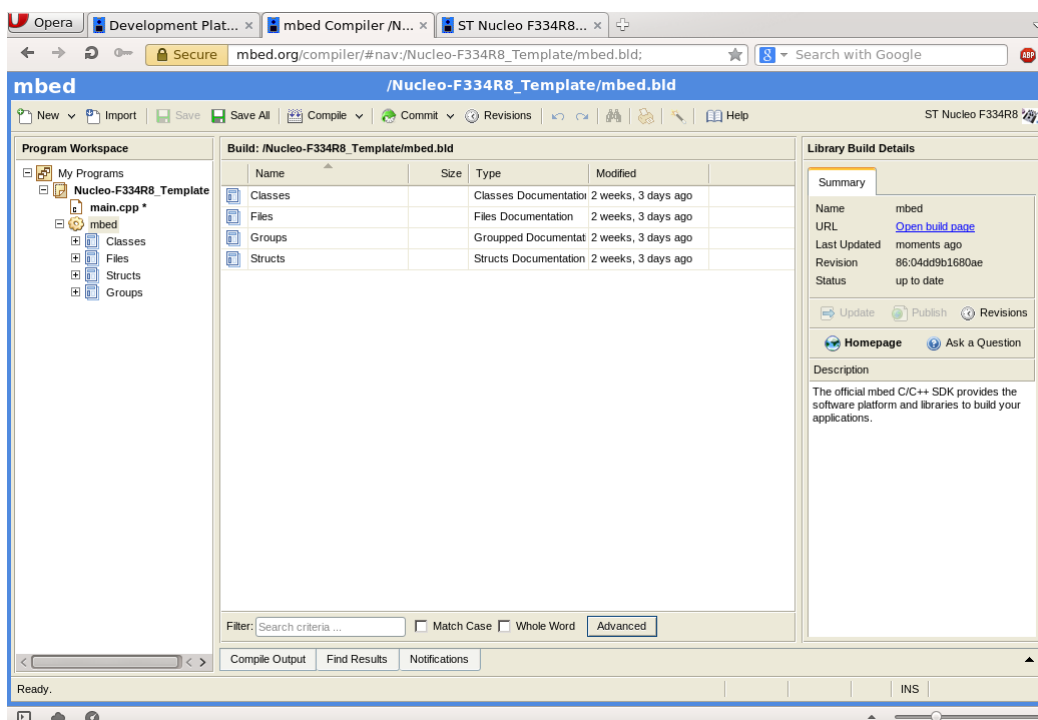
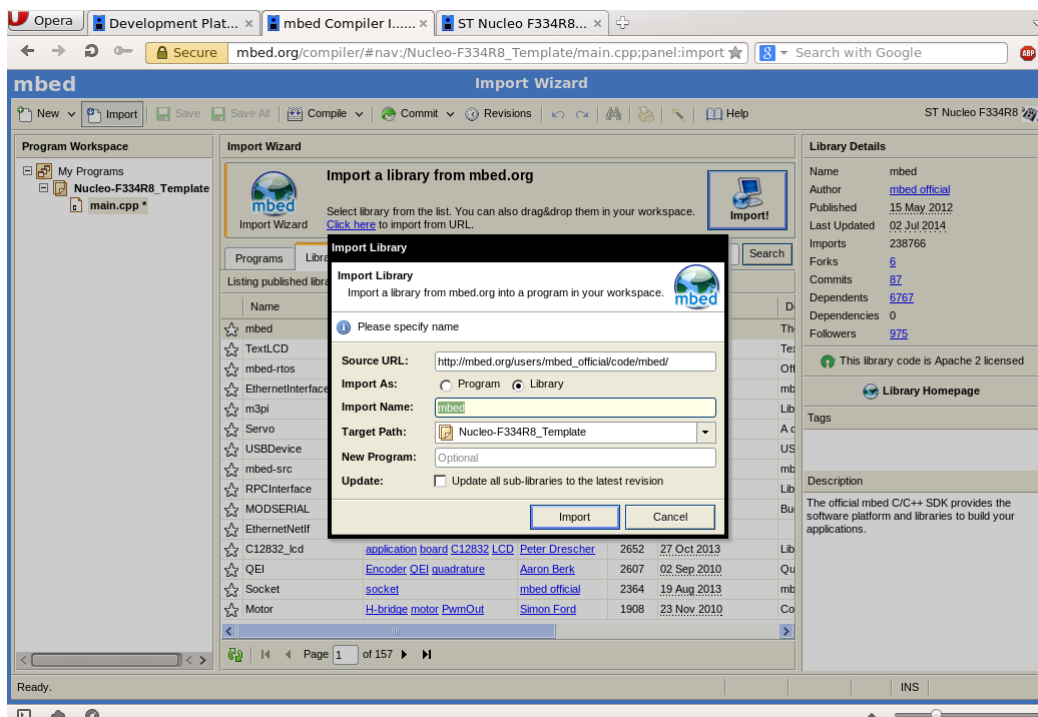




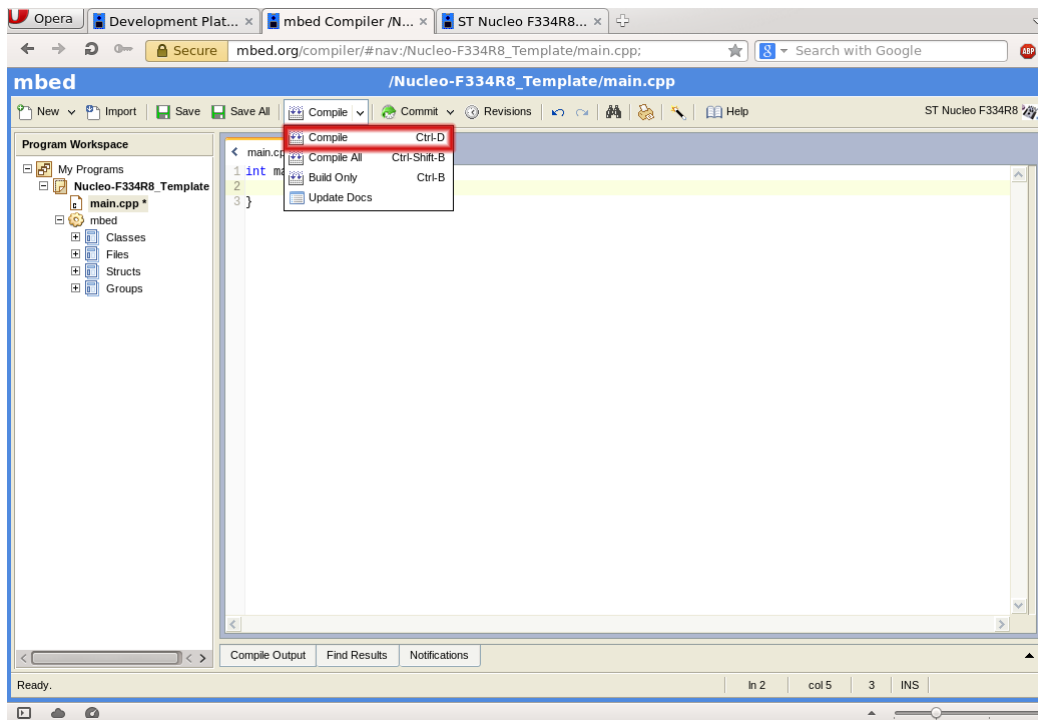
Aby skorzystać z gotowych funkcji do obsługi peryferii, należy do projektu dodać bibliotekę *mbed*. W tym celu klikamy przycisk *Import* na panelu głównym i w zakładce *Libraries* sekcji *Import Wizard* zaznaczamy *mbed*, a następnie klikamy *Import!*



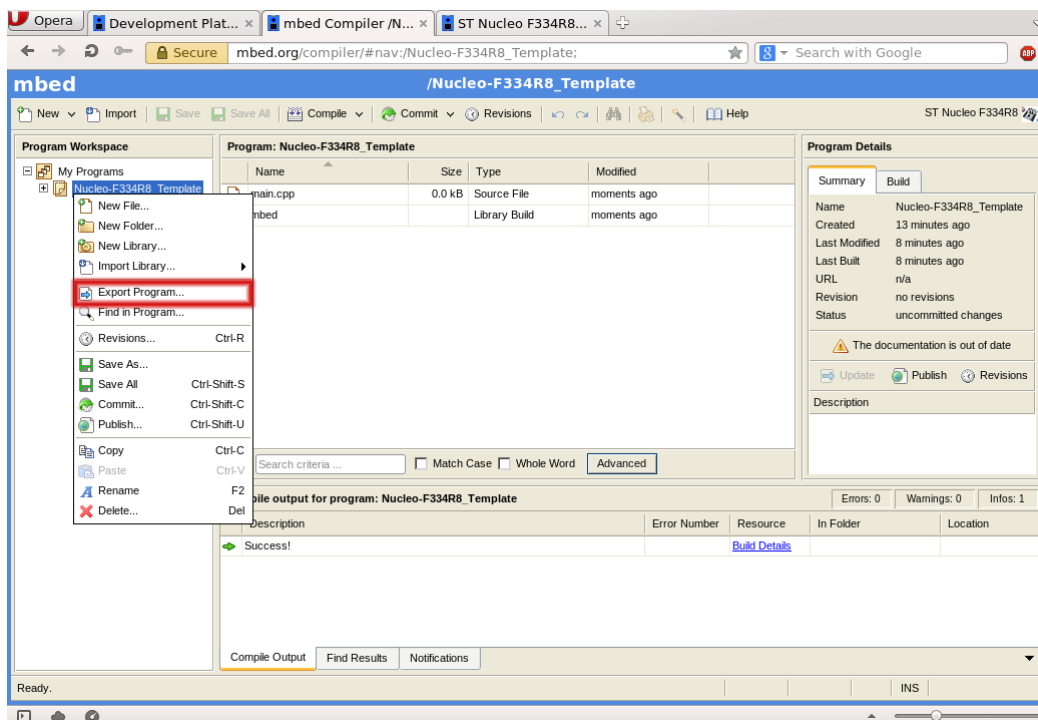
W okienku *Import Library* zostawiamy domyślne parametry, zaznaczając tylko *Import As Library*.

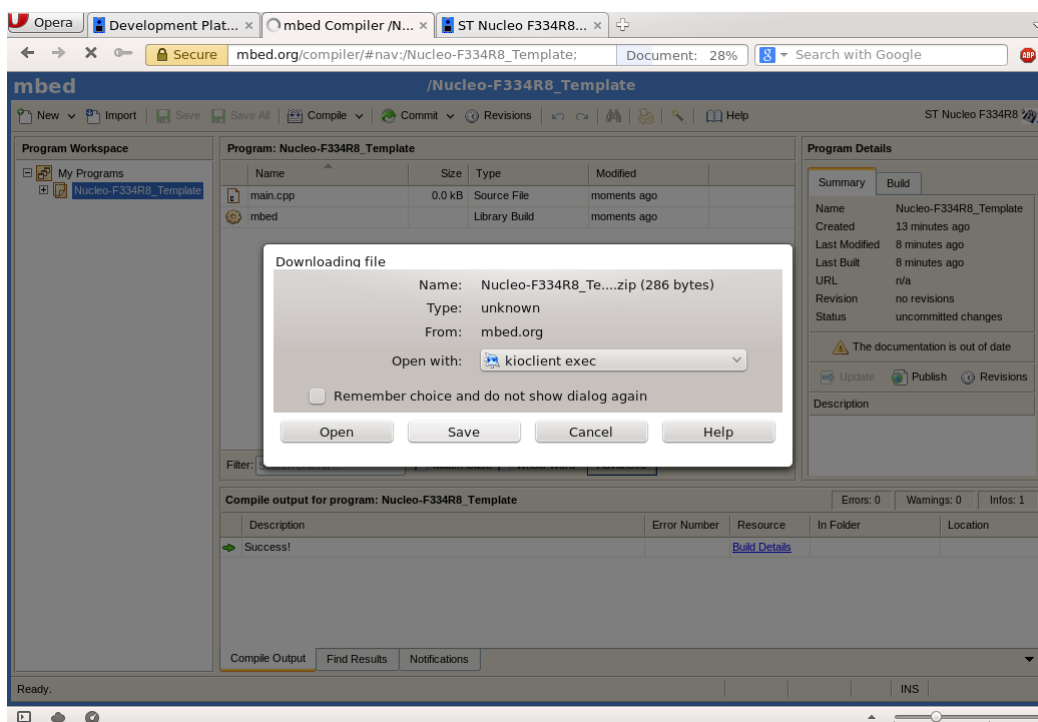
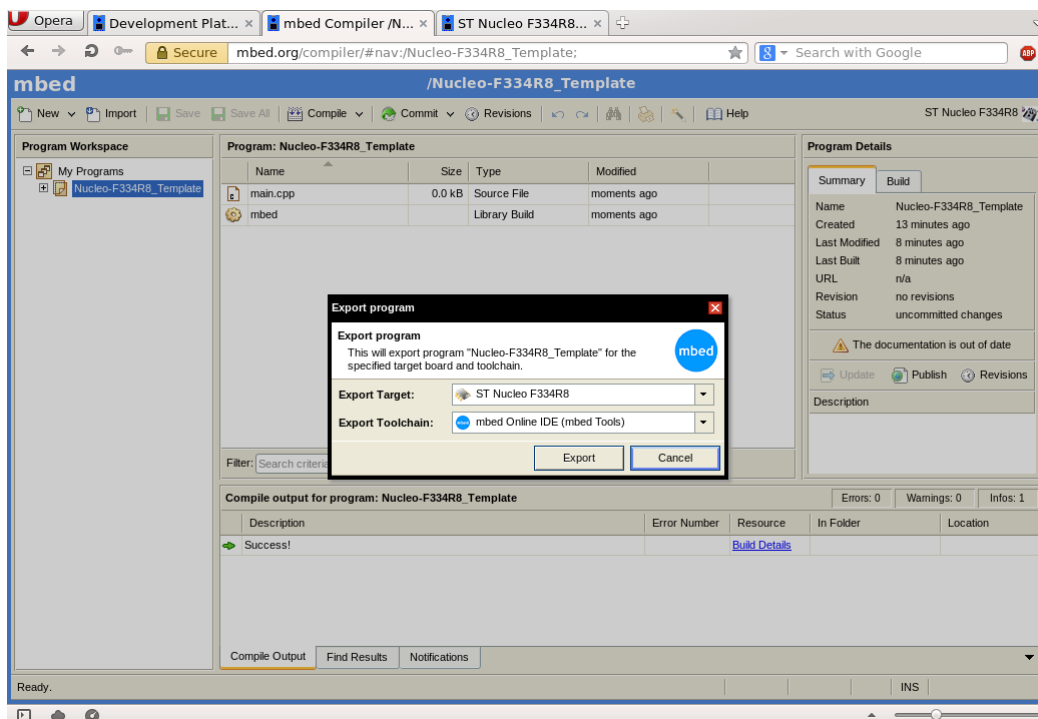


Przygotowany w powyższy sposób projekt-szablon kompilujemy, a wygenerowany plik z rozszerzeniem bin informuje o bezbłędnej operacji budowy projektu. Jakikolwiek błąd znalezione przez kompilator są prezentowane w sekcji widocznej po wybraniu zakładki *Compile Output* w dolnej części okna projektu.

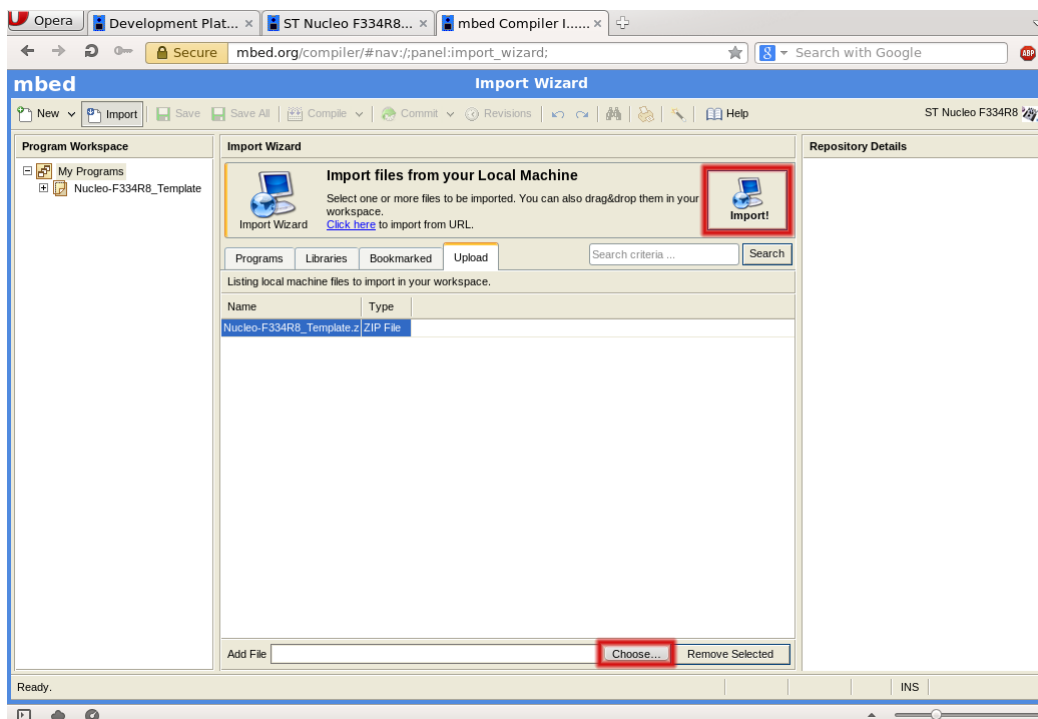


Jeśli chcemy stworzyć nowy projekt na bazie już istniejącego to należy wyeksportować wybrany projekt i zaimportować go ponownie, ale już pod inną nazwą. Klikając PPM na katalogu z projektem wybieramy *Export Program...* i w otwartym okienku wybieramy projekt oraz docelowe środowisko, do którego ma być on przekonwertowany. W ramach środowiska mbed wystarczy wybrać *mbed Online IDE (mbed Tools)*. Po zatwierdzeniu przyciskiem *Export* zostanie wygenerowany plik archiwalny, który możemy zapisać na dysk.

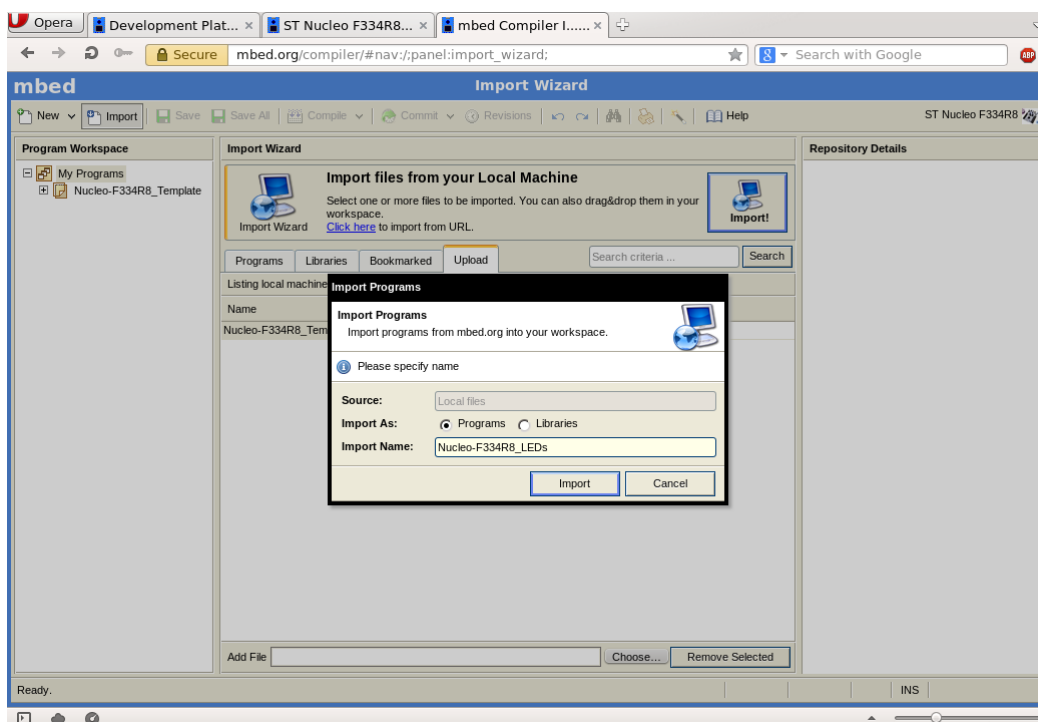




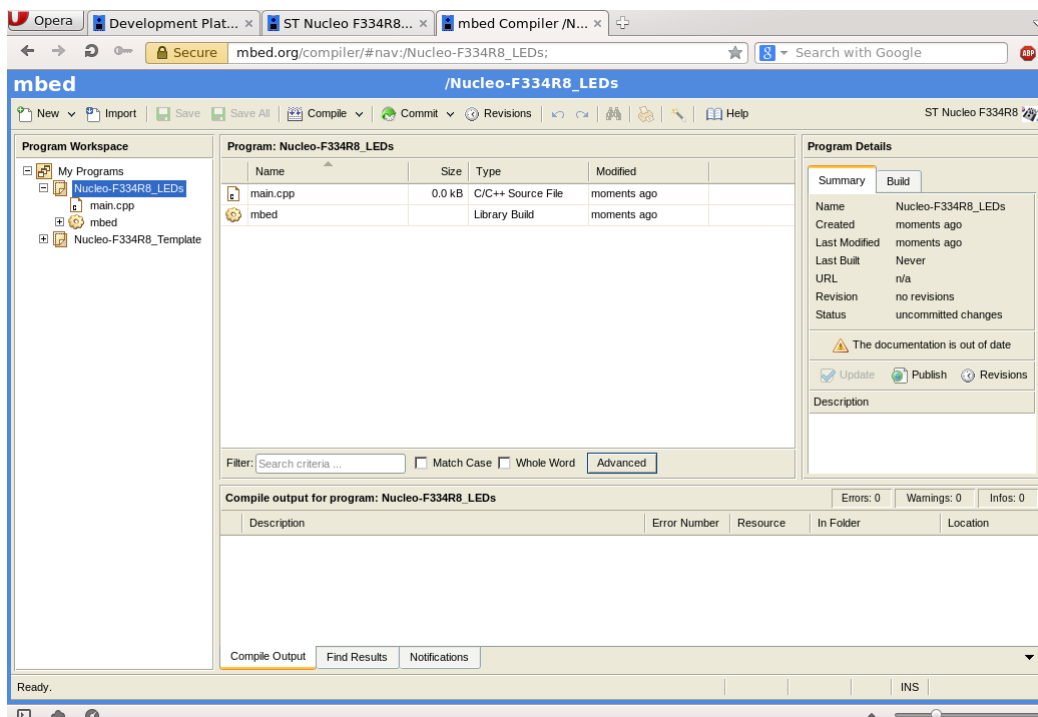
Import projektu dokonujemy w zakładce *Upload*, dostępnej po naciśnięciu przycisku *Import* w lewym górnym rogu, wybierając wygenerowany przed chwilą projekt-szablon.



W okienku, jakie pojawi się po naciśnięciu przycisku *Import!*, wybieramy *Import As Programs* i wpisujemy nazwę, np. *Nucleo-F334R8\_LEDs*.

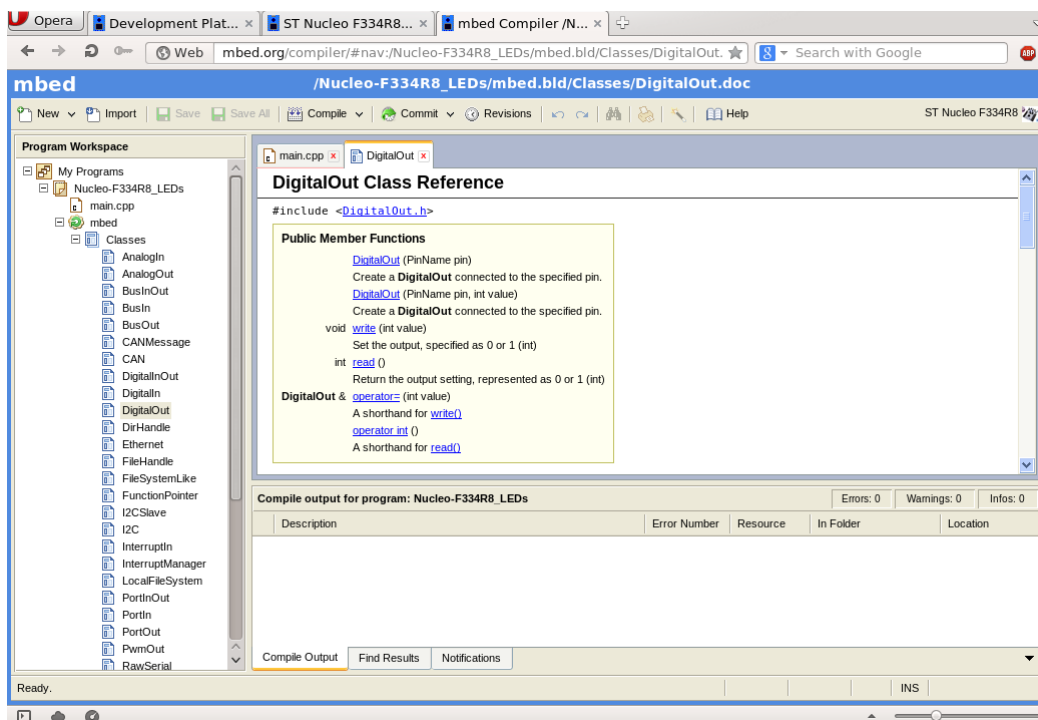


Na tym etapie projekt jest już skonfigurowany i można przystąpić do uzupełniania głównego pliku.

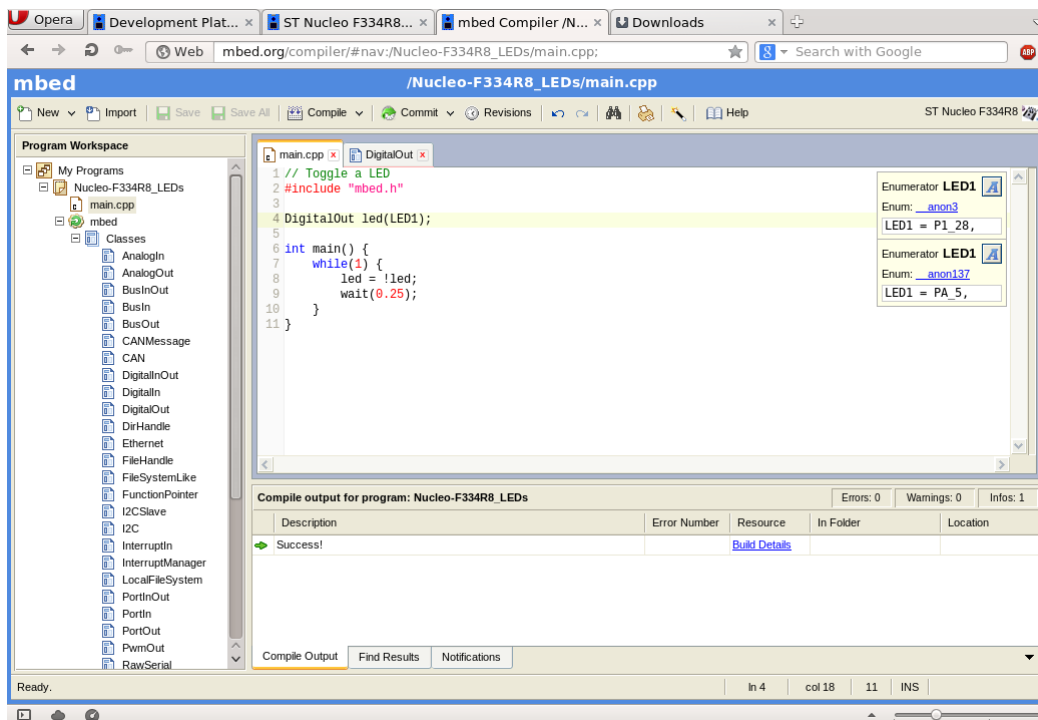


## Projekt aplikacji sterującej diodami LED

Pierwsza aplikacja demonstracyjna będzie miała za zadanie miganie z częstotliwością 2 Hz zieloną diodą LD2, jaka jest dostępna w zestawie. W tym celu wykorzystamy biblioteczną klasę *DigitalOut*, której opis można znaleźć po rozwinięciu elementu *mbed*, a następnie *Classes*.



Kopiując przykład zawarty w dokumentacji danej klasy i kompilując projekt uzyskujemy gotowy plik, który należy zapisać na dysk Nucleo zestawu. Jak można zauważyć, nazwy takie jak LED1 są już zdefiniowane dla konkretnego zestawu, przez co nie musimy przeglądać schematów elektrycznych celem znalezienia nazwy wyprowadzenia mikrokontrolera.



Listing pierwszej aplikacji wygląda następująco:

```
// Toggle a LED
#include "mbed.h"

DigitalOut led(LED1);

int main() {
    while(1) {
        led = !led;
        wait(0.25);
    }
}
```

Jak widać kod aplikacji jest dość prosty i sprowadza się jedynie do utworzenia obiektu klasy *DigitalOut* przy pomocy konstruktora z argumentem zdefiniowanym jako LED1, cyklicznej zmiany stanu obiektu (diody) na przeciwny oraz czekania 250 ms. Zaletą tworzenia aplikacji w języku C++ w porównaniu do zwykłego C jest możliwość wykonywania operacji z wykorzystaniem samych obiektów, bez konieczności wywoływania przy tym funkcji jak np. *read()*/*write()* (mamy tutaj do czynienia z przeciążaniem operatorów).

W kolejnej wersji aplikacji dodamy obsługę przycisku ogólnego przeznaczenia, po wciśnięciu którego dioda zacznie migać, a po odpuszczeniu jej ostatni stan zostanie zachowany. Tym razem potrzebny będzie jeszcze obiekt klasy *DigitalIn* zainicjalizowany na wartość zdefiniowaną jako USER\_BUTTON. Stan przycisku odczytujemy w prosty sposób poprzez pobranie wartości zmiennej *enable* (obiekt klasy *DigitalIn*):

```
// Flash a LED while an USER BUTTON is pressed

#include "mbed.h"

DigitalIn enable(USER_BUTTON);
DigitalOut led(LED1);

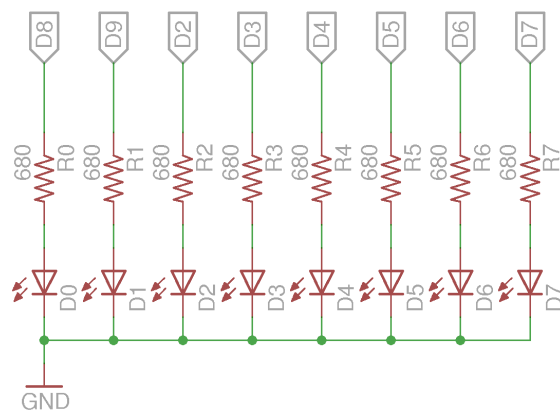
int main() {
    while(1) {
        if(!enable) {
            led = !led;
        }
        wait(0.25);
    }
}
```

```

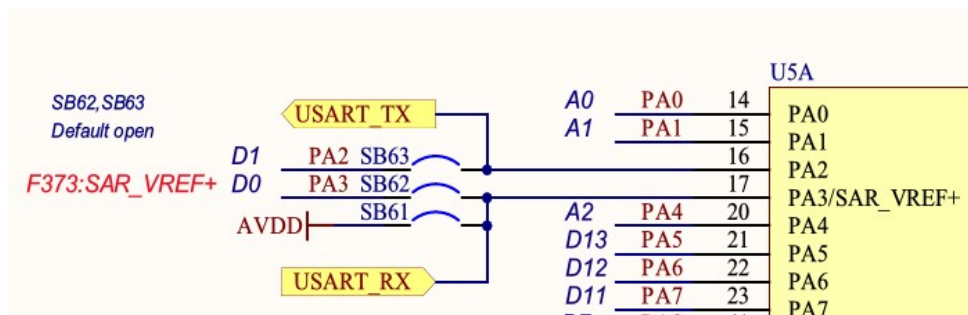
}
}

```

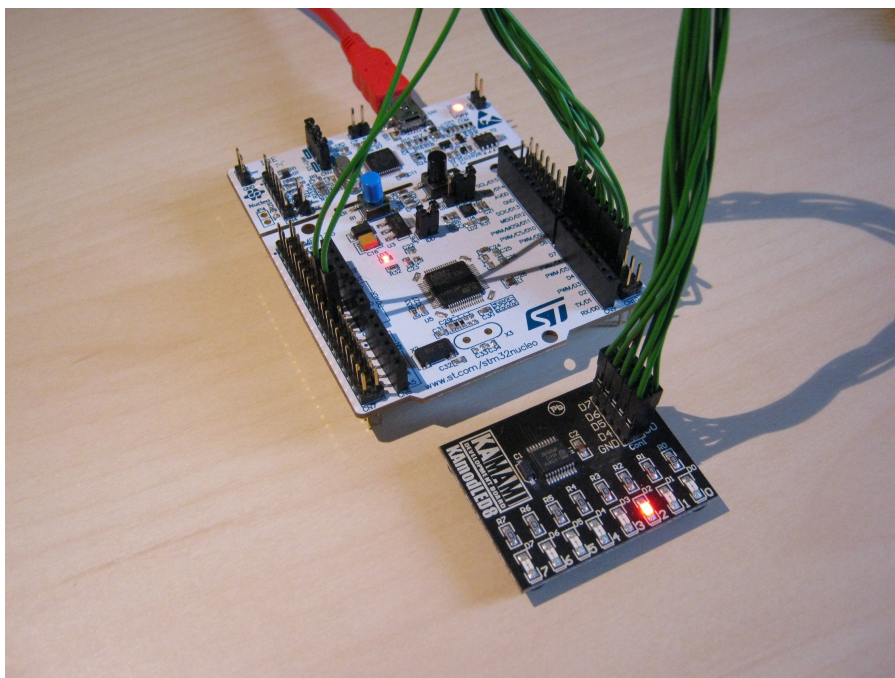
W ostatnim przykładzie zmodyfikujemy nasz program tak, aby obsługiwał zewnętrzne diody LED podłączone do zestawu w sposób pokazany na **rysunku 5** – wykorzystano tutaj moduł z 8 diodami KAmoLED8 (**fotografia 1**). W danym zestawie Nucleo linie D0 oraz D1 są domyślnie odłączone od mikrokontrolera (**rysunek 6**) dlatego w ich miejsce wybrano odpowiednio linie D8 oraz D9.



Rys. 5. Uproszczony schemat podłączenia diod LED do zestawu Nucleo-F334R8



Rys. 6. Domyślnie linie D0 i D1 są odłączone od mikrokontrolera



Fot. 1. Sterowanie zewnętrznymi diodami modułu KAmoLED8

W pierwszej kolejności należy utworzyć obiekty poszczególnych diod i można to zrobić deklarując tablicę obiektów klasy *DigitalOut*. Każdy element tablicy jest inicjalizowany przy pomocy konstruktora z argumentem zdefiniowanym jako D2..D9 (nazwy wyprowadzeń zgodne z systemem Arduino i oznaczone na płytce Nucleo). W głównej pętli programu zamiast operowania na pojedynczym obiekcie będziemy operować na elementach tablicy. Wprowadzając dodatkową zmienną zliczającą od 0 w górę i wykorzystując operację modulo będzie można uzyskać dostęp do elementów z indeksem od 0 do 7. W wyniku działania programu uzyskamy efekt przesuwającej się diody:

```
// Rotate a LED on KAmoLED8 module

#include "mbed.h"

DigitalOut KAmoLED8[] = {DigitalOut(D8), DigitalOut(D9), DigitalOut(D2), DigitalOut(D3),
                        DigitalOut(D4), DigitalOut(D5), DigitalOut(D6), DigitalOut(D7)
};

int main() {
    int i = 0;

    while(1) {
        // Turn on LED
        KAmoLED8[i % 8] = 1;

        // Wait 0.25 s
        wait(0.25);

        // Turn off LED
        KAmoLED8[i++ % 8] = 0;
    }
}
```