Sterowniki robotów

Laboratorium – Wprowadzenie

Wykorzystanie narzędzi STM32CubeIDE oraz STM32CubeMx do budowy programu mrugającej diody z obsługą przycisku

Wojciech Domski

Dokument `sr_lab01.pdf`, Wersja 2.1.0

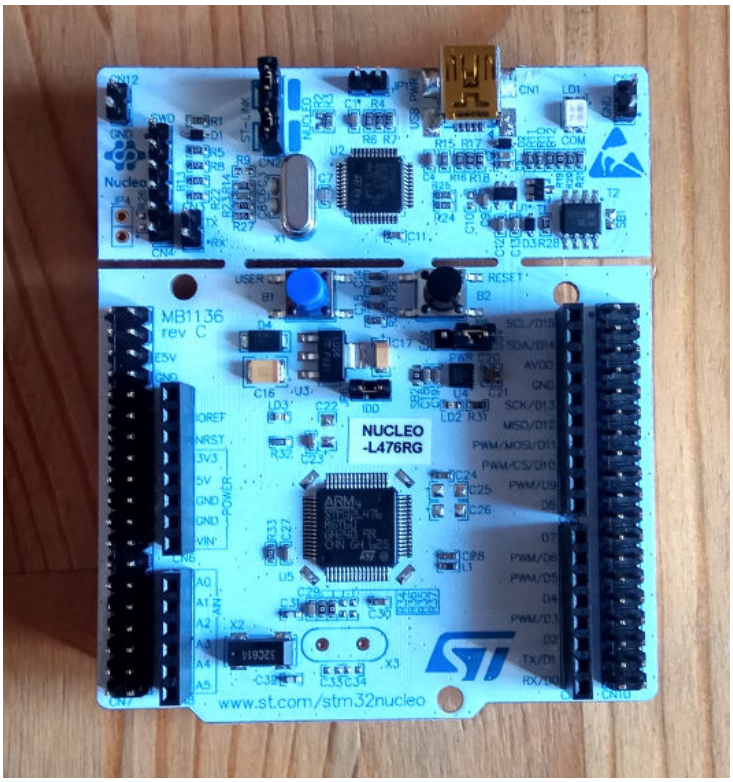
Spis treści

1 Wprowadzenie

Ćwiczenie prezentuje podstawowe narzędzia do pracy z mikrokontrolerem firmy ST. Jest to `STM32CubeMX` [3], [8] – aplikacja pozwalająca w wygodny sposób na przygotowanie podstawowej konfiguracji peryferiów mikrokontrolera poczynając od GPIO, a kończąc na middleware firm trzecich, jak FreeRTOS. Jest ono integralną częścią programu `STM32CubeIDE` lub występuje osobno jako samodzielna aplikacja. Ponadto, oprogramowanie pozwala na konfigurację częstotliwości taktowania rdzenia w trybie ręcznym lub półautomatycznym. Do rozwoju oprogramowania dla mikrokontrolera wykorzystywane będzie środowisko `STM32CubeIDE`. Środowisko to bazuje na zmodyfikowanej wersji oprogramowania `Atollic TrueStudio for STM32`. Jest to środowisko oparte o popularne IDE – Eclipse, jednakże zostało ono odpowiednio skonfigurowane do pracy z układami firmy ST.

Jako zestaw ewaluacyjny wykorzystana zostanie płytka deweloperska NUCLEO-L476RG [4], [5]. Posiada ona mikrokontroler STM32L476RGT6 [1], szerszy opis tej rodziny mikrokontrolerów można znaleźć w [9].

Zdjęcia płytki ewaluacyjnej zostały przedstawione poniżej.



Rysunek 1: Zestaw ewaluacyjny NUCLEO-L476RG

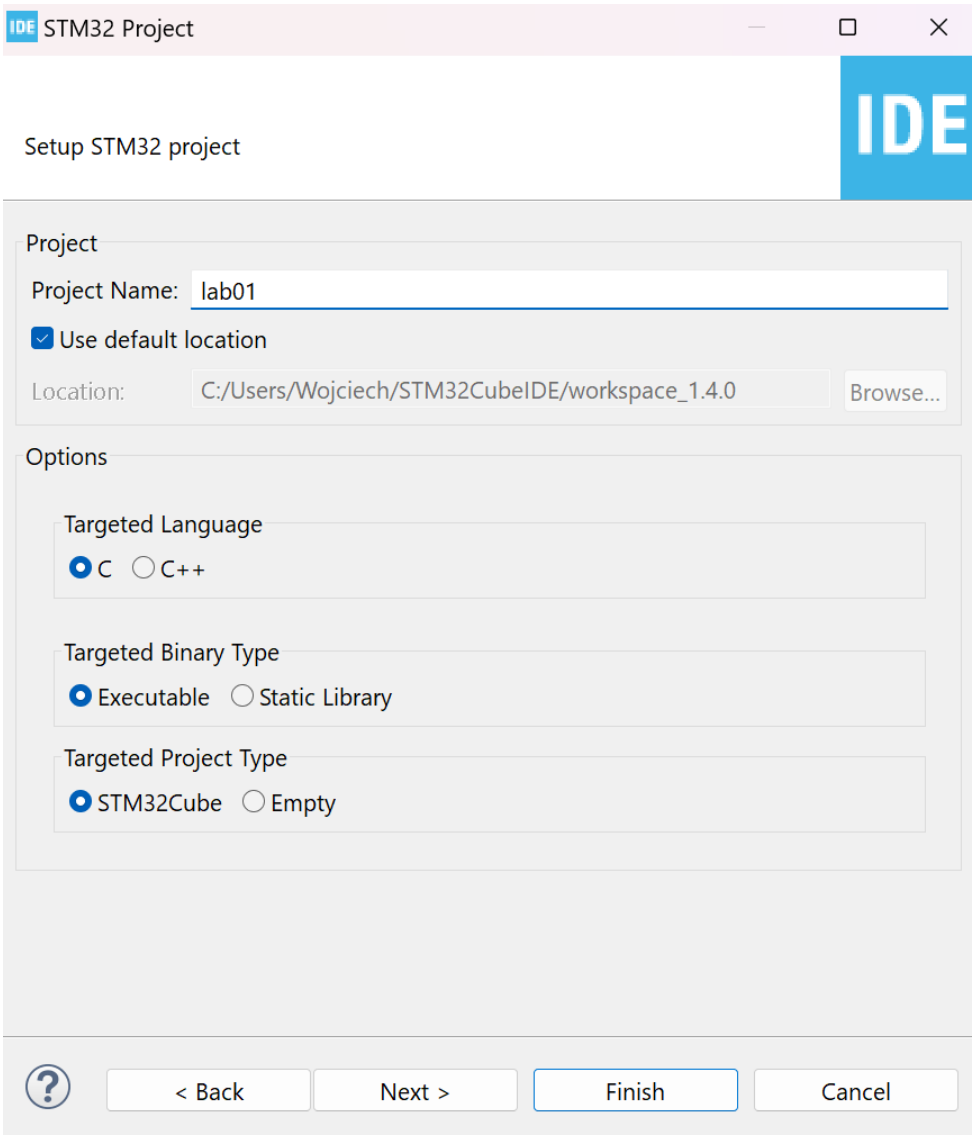
Rysunek 2: Zestaw ewaluacyjny NUCLEO-L476RG w opakowaniu

2 Opis ćwiczenia

Ćwiczenie ma za zadanie zaznajomić studenta z obsługą oraz wykorzystaniem przedstawionego oprogramowania: konfiguracja peryferiów, dobór częstotliwości pracy układu, automatyczne generowanie kodu. Ponadto ćwiczenie, obejmuje zakres związany z obsługą GPIO na przykładzie diody LED (cyfrowe wyjście) oraz przycisku (cyfrowe wejście).

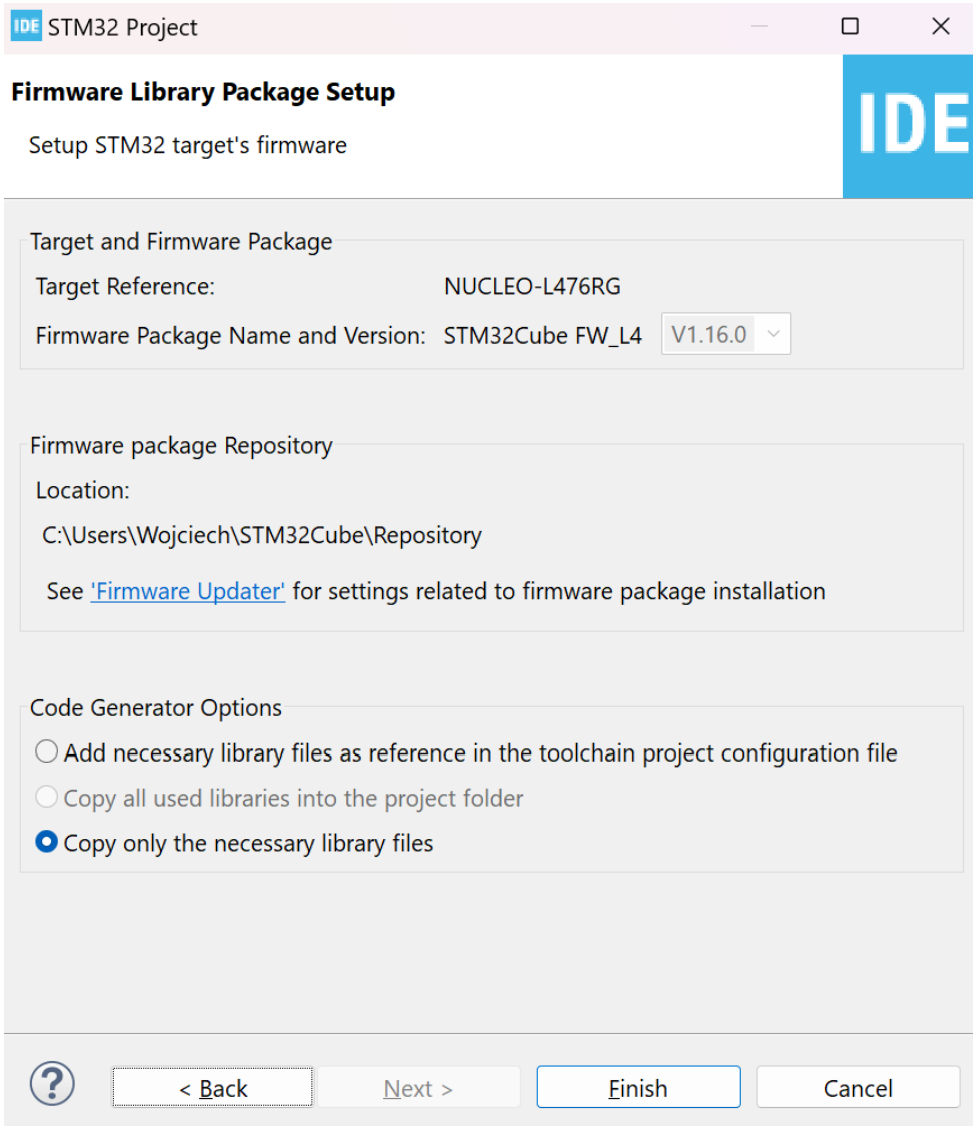
3 Tworzenie projektu w STM32CubeIDE

Nowy projekt można utworzyć na kilka sposobów. Jednym z nich jest wybranie odpowiedniej pozycji z głównego menu (*File* → *New* → *STM32 Project*). Wówczas pojawi się okno, które pozwala na wybór jednostki, czy wybór płytki testowej. Konfiguracja projektu od tego momentu jest bardzo podobna do konfiguracji projektu w `STM32CubeMX` z kilkoma różnicami. Po wybraniu układu, bądź płytki deweloperskiej należy nadać nazwę projektu, tak jak to pokazano na rysunku 3. Pozostałe opcje powinny być bez zmian, to jest wybrany język ustawiony na C, docelowy plik binarny ustawiony an wykonywalny, a typ projektu to STM32Cube.



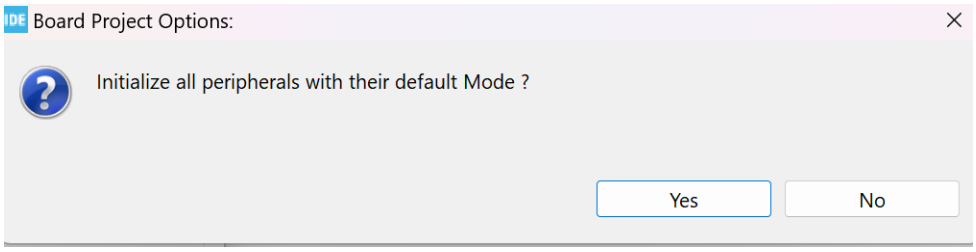
Rysunek 3: Wybór nazwy projektu

Po konfiguracji podstawowych ustawień projektu należy przejść do następnego okna (Rys. 4). Na tym etapie należy zaakceptować domyślną konfigurację.



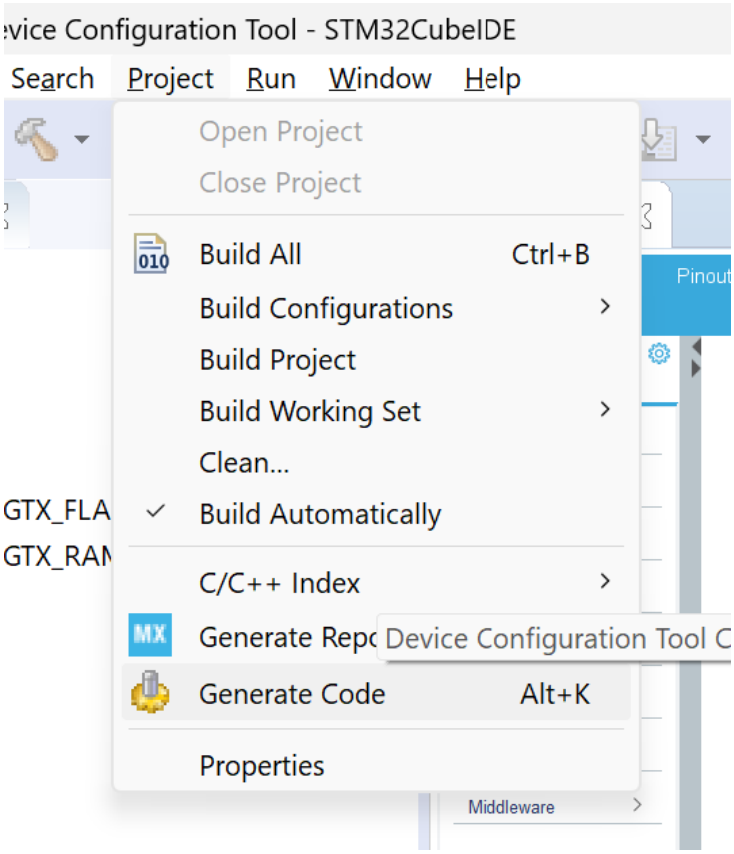
Rysunek 4: Konfiguracja bibliotek projektu

Po przejściu dalej, pojawi się komunikat z pytaniem, czy zainicjalizować dostępne peryferia (Rys. 5). Należy zaakceptować akcję i oczekiwać na zakończenie działań przez środowisko. W tym momencie program może wymagać pobrania dodatkowych bibliotek z Internetu, których rozmiar to około 700MB.



Rysunek 5: Inicjalizacja wybranych peryferiów na płytce deweloperskiej

W odróżnieniu do aplikacji `STM32CubeMX` generowanie kodu odbywa się z poziomu programu `STM32CubeIDE`. W tym celu należy wybrać *Project* → *Generate Code*, tak jak to zostało pokazane na rysunku 6.



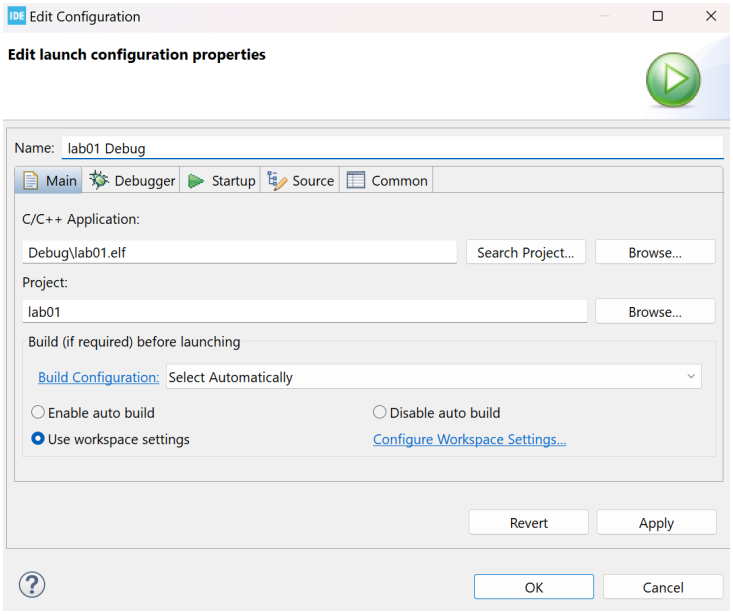
Rysunek 6: Generowanie kodu w aplikacji STM32CubeIDE

Budowanie projektu odbywa się poprzez wybranie z menu opcji *Project* → *Build All*. Po zbudowaniu projektu, jeśli skompiluje się on bez ostrzeżeń, w katalogu projektu pojawi się folder Debug. To w nim zostaną umieszczone zbudowane obrazy firmware'u (*.bin, *.hex).

Uwaga! Podczas regeneracji kodu może zajść konieczność wyczyszczenia oraz przebudowania projektu. Aby wyczyścić projekt należy z menu wybrać projekt, a następnie wyczyść (*Project* → *Clean*)

3.1 Wgrywanie programu

Wgranie programu do mikrokontrolera jest możliwe bezpośrednio z poziomu środowiska programistycznego. W tym celu należy wybrać z menu *Run* → *Run*. Pierwsze wywołanie akcji uruchomienia spowoduje otwarcie okna z konfiguracją (rys. 7). Domyślnie wybrane ustawienia są wystarczające do prawidłowego zaprogramowania układu.



Rysunek 7: Wgrywanie aplikacji w programie STM32CubeIDE

Każde kolejne wywołanie tej akcji spowoduje wybranie ostatniej konfiguracji. Dostęp do listy dostępnych konfiguracji jest również dostępny z poziomu menu *Run* → *Run Configurations* .... Uruchamianie sesji debugowania jest niemal identyczne, różny się ono jedynie wyborem odpowiedniej akcji (*Run* → *Debug*).

Uwaga! Aby możliwe było poprawne uruchomienie aplikacji muszą być spełnione następujące warunki.

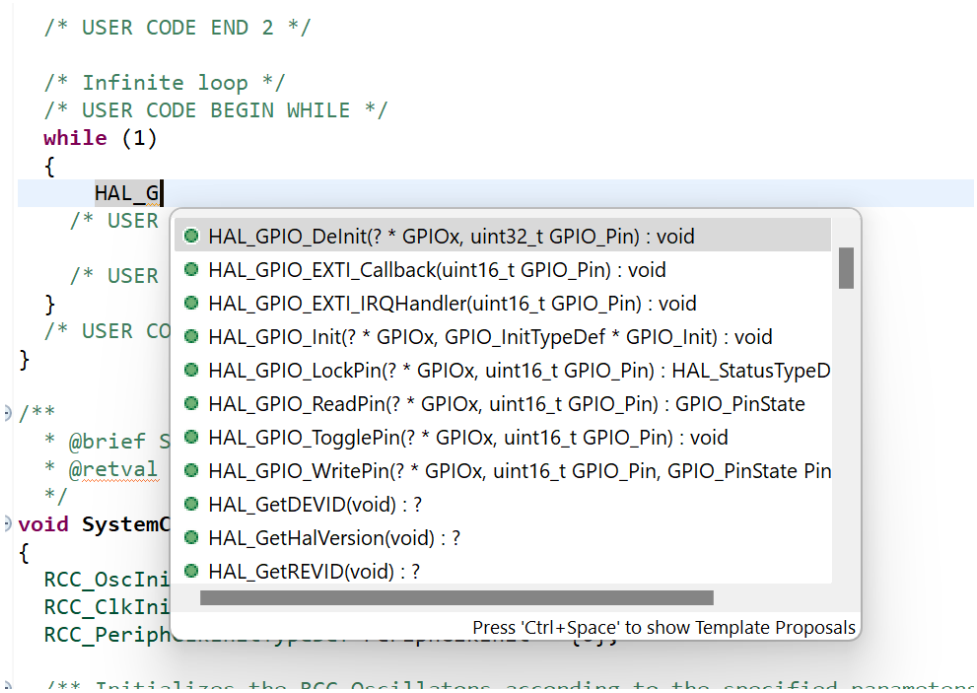
1. Aplikacja została zbudowana bez błędów.
2. Do komputera został podłączony programator (Płytka testowa posiada już wbudowany programator).
3. Wszystkie zworki na płytce testowej muszą być we właściwej pozycji.

W przeciwnym wypadku mogą pojawić się błędy.

1. Konfiguracja uruchomieniowa nie może odnaleźć zbudowanego programu.
2. Programator nie jest podłączony do komputera.
3. Mikrokontroler docelowy nie jest odnaleziony najprawdopodobniej przez niewłaściwą konfigurację zworek na płytce testowej.

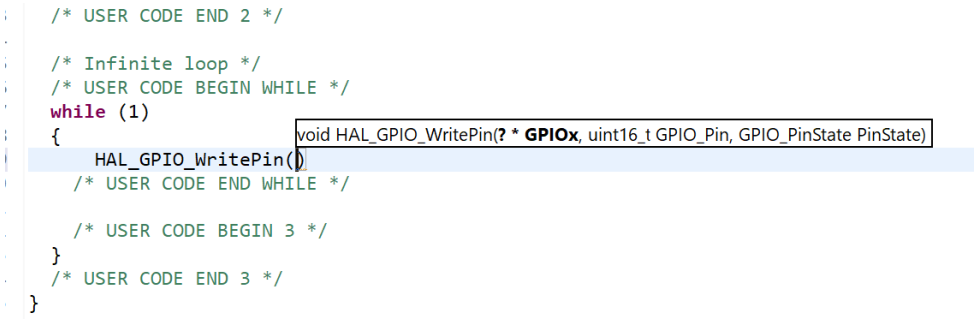
3.2 Przydatne informacje

`STM32CubeMX` dostarcza kilku przydatnych opcji podczas pisania oprogramowania. Pierwszą z nich jest automatyczne podpowiadanie nazw funkcji, zmiennych, itp. W tym celu należy wpisać początek nazwy funkcji **Ctrl** + **Spacja**. Wówczas wyświetlą się dopasowania, które odpowiadają aktualnie wprowadzonemu fragmentowi funkcji, zmiennej (Rys. 8).



Rysunek 8: Podpowiadanie nazw

Innym przydatnym narzędziem jest podpowiadanie listy argumentów funkcji. Przykład okazano na rysunku 9). Aby wyświetlić podpowiedź należy znaleźć się w liście argumentów funkcji (pomiędzy okrągłymi nawiasami) i wcisnąć kombinację klawiszy **Ctrl** + **Shift** + **Spacja**. Aktualnie aktywny argument jest podświetlany.



Rysunek 9: Podpowiadanie nazw

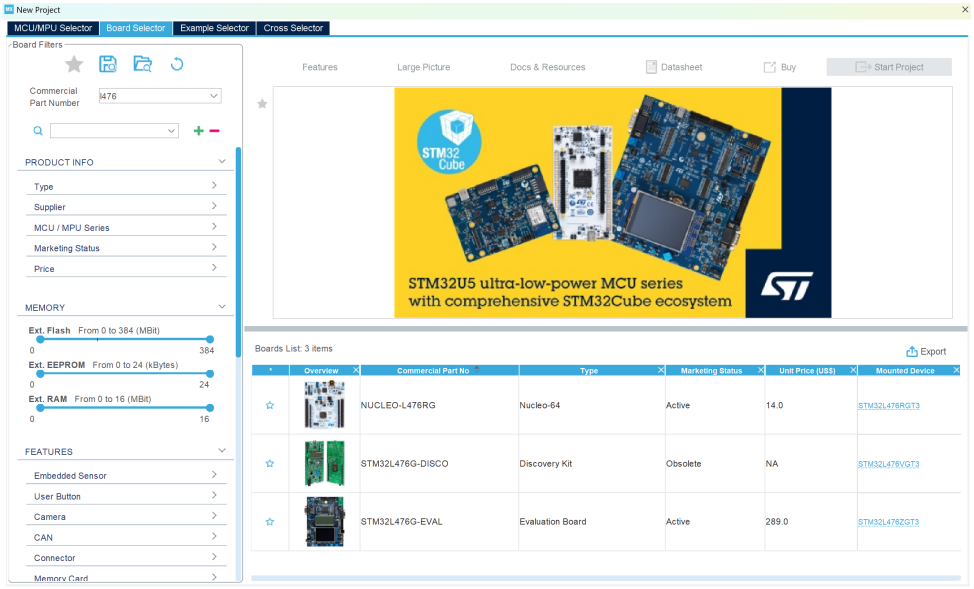
Podczas gdy chcemy przełączyć się z pliku źródłowego na odpowiadający plik nagłówkowy wystarczy wykorzystać kombinację klawiszy **Ctrl** + **Tab**. Skrót ten umożliwia naprzemienne przełączanie się pomiędzy powiązanymi plikiem nagłówkowym oraz plikiem źródłowym.

Jeśli zajdzie potrzeba przejścia do definicji funkcji można wykorzystać przycisk funkcyjny **F3**. Po najechaniu kursorem na funkcję, której definicja jest interesująca wystarczy wcisnąć klawisz **F3**, a środowisko programistyczne automatycznie odnajdzie definicję. Możliwy jest powrót do poprzedniego miejsca w programie przez wciśnięcie kombinacji klawiszy **Alt** + ←

4 Praca z narzędziem STM32CubeMX

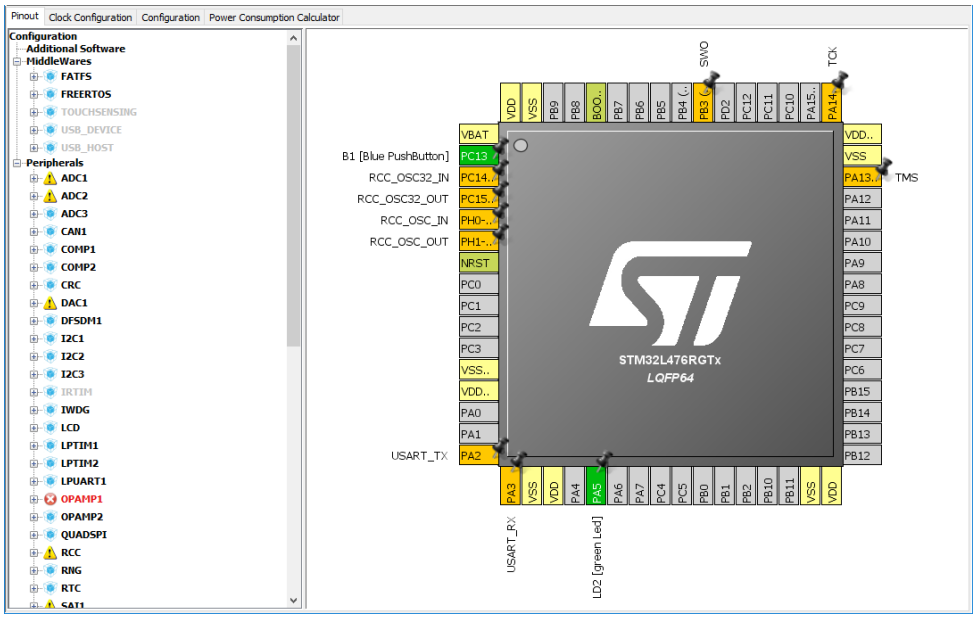
4.1 Przygotowanie projektu w programie STM32CubeMX

Należy uruchomić program `STM32CubeMX`. Następnie wybieramy nowy projekt (*New Project*). Operacja ta może chwilę potrwać, gdyż aplikacja sprawdza dostępność nowych produktów. W oknie nowego projektu wybieramy odpowiednią płytkę ewaluacyjną, aby to zrobić należy wybrać zakładkę wyboru płytki (Board Selector), a następnie wybrać typ płytki (*Type*) oraz serię mikrokontrolera (*MCU Series*), Nucleo64 oraz STM32L4, odpowiednio. Po zredukowaniu liczby dostępnych zestawów deweloperskich wybieramy NUCLEO-L476RG z listy (Rys. 10).



Rysunek 10: Okno z wyborem zestawu ewaluacyjnego NUCLEO-L476RG

Po kliknięciu przycisku *Start Project* zostaniemy zapytani, czy chcemy zainicjalizować peryferia domyślnymi ustawieniami. Wybór przycisku Tak *Yes* spowoduje, że peryferia takie, jak zegar *LSI*, czy USART zostaną skonfigurowane domyślnie. Wybranie opcji Nie *No* nie spowoduje przeprowadzenia domyślnej konfiguracji. W obu przypadkach wyjścia dla diody LED oraz dla przycisku będą odpowiednio skonfigurowane. Po tym etapie powinien pojawić się czysty projekt z wstępnie skonfigurowanymi wyjściami mikrokontrolera, jak na rysunku 11.



Rysunek 11: Okno przedstawiające początkową konfigurację mikrokontrolera dla płyty ewaluacyjnej NUCLEO-L476RG

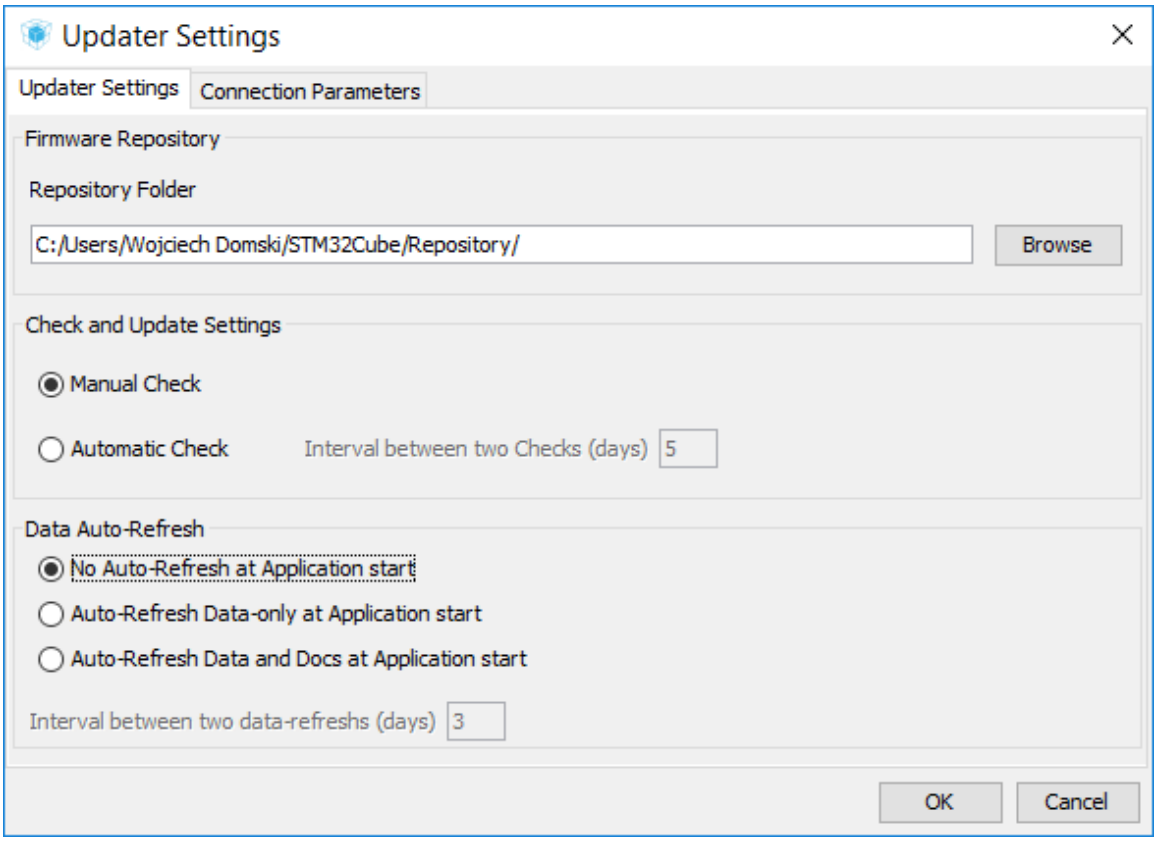
Piny podświetlone na kolor zielony (PC13 i PA5) oznaczają, że te porty mikrokontrolera zostały skonfigurowane poprawnie. Piny podświetlone na kolor pomarańczowy oznaczają, że funkcjonalność portu została wybrana lecz nie został on jeszcze w pełni skonfigurowany, jak np. PA2 oraz PA3, które wykorzystywane są do komunikacji za pomocą portu szeregowego, jednakże sam port szeregowy (USART) nie został jeszcze skonfigurowany (chyba, że wybrano automatyczną inicjalizację peryferiów domyślnymi ustawieniami).

Uwaga! Warto zwrócić uwagę na nazwy poszczególnych pinów (ich etykiety), jakie są widoczne w widoku *Pinout*. Przykładowo dla pinu PA5, do którego podłączona jest dioda LED została ustawiona etykieta *LD2 [green Led]*. Podczas automatycznego generowania kodu opisanego w rozdziale 4.2 zostaną wykorzystane etykiety, które zostały nadane przez użytkownika do stworzenia przyjaznych definicji, którymi można się później posługiwać. Definicje te znajdują się zwyczajowo w pliku *Inc/main.h*. Dla wyżej wspomnianego pinu PA5 zostaną utworzone poniższe definicje:

```
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
```

Mogą one później zostać wykorzystane do wywołań funkcji z biblioteki HAL, jak np. przełączanie stanu cyfrowego wyjścia. Wówczas programista nie musi pamiętać, do którego portu i numeru pinu przynależy dana nóżka mikrokontrolera, a jedynie wystarczy wiedza na temat etykiety, jaka została do niej przypisana. Jak widać na przykładzie napis znajdujący się w nawiasach kwadratowych jest pomijany w czasie generacji definicji.

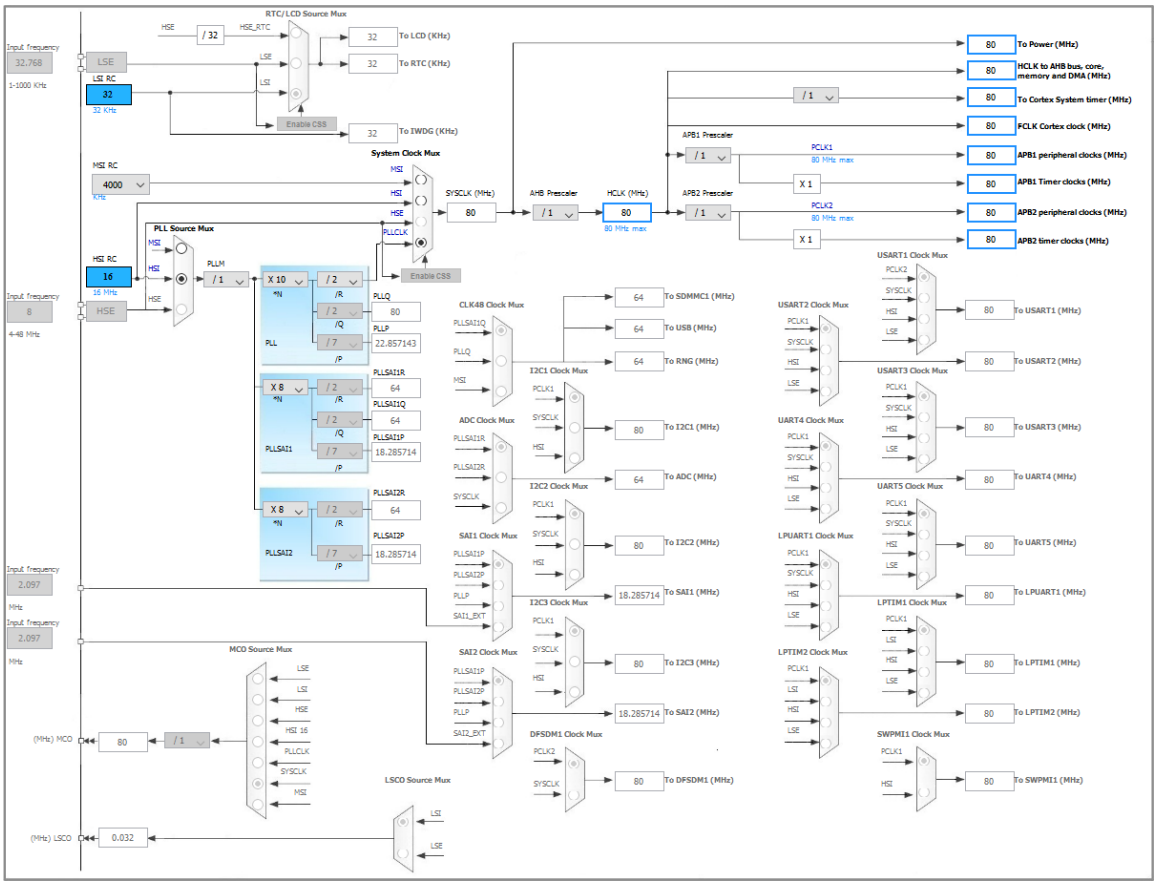
Do poprawnego działania oprogramowanie `STM32CubeMX` wymaga połączenia z Internetem. Jest to spowodowane tym, że pobierane są aktualizacje produktów dostępnych w aplikacji. Jeśli uruchamianie narzędzia trwa zbyt długo można wyłączyć sprawdzanie aktualizacji w menu programu. Wystarczy wybrać *Help* → *Updater settings ...*, a następnie w zakładce *Updater Settings* zaznaczyć opcję *Manual checks* oraz *No Auto-Refresh at Application start*. Zostało to zaprezentowane na Rys. 12.



Rysunek 12: Wyłaczenie automatycznego sprawdzania aktualizacji

4.1.1 Konfiguracja zegara

Ważnym aspektem pracy mikrokontrolera jest konfiguracja częstotliwości pracy zegara. W ramach tego ćwiczenia należy zapoznać się z informacjami, jakie są dostępne w zakładce konfiguracji zegara (Clock Configuration) (Rys. 13).

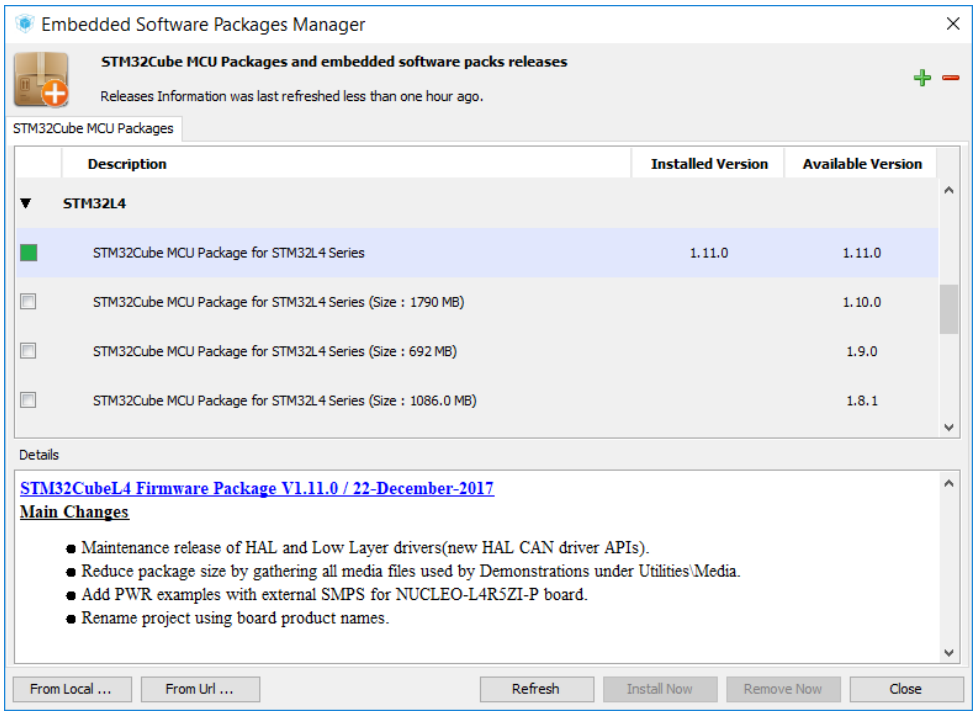


Rysunek 13: Zakładka z konfiguracją zegara dla całego układu

Z rysunku 13 można wywnioskować, jaka jest częstotliwość pracy rdzenia oraz jakie jest jego źródło – zidentyfikuj je i prześledź ścieżkę generowania zegara. Czy wykorzystywany jest do tego jakiś układ, jeśli tak, to jaka jest jego rola.

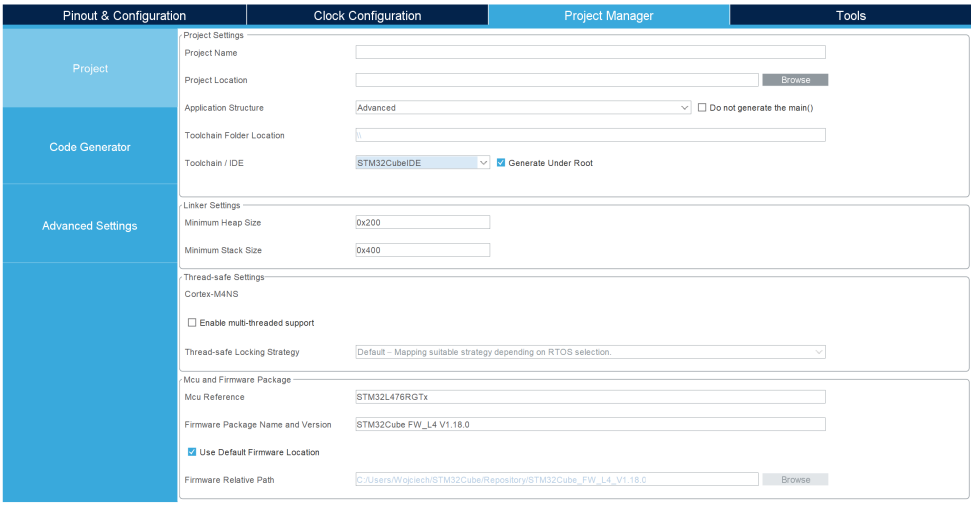
4.2 Automatyczne generowanie kodu

`STM32CubeMX` nie tylko pozwala na łatwą konfigurację peryferiów mikrokontrolera, ale przede wszystkim jest narzędziem do automatycznego generowania kodu. Przed przystąpieniem do generowania kodu z projektu należy upewnić się, że zostałą zainstalowana biblioteka sterowników HAL dla rodziny mikrokontrolerów STM32L4. W tym celu należy wybrać z menu *Help* → *Manage embedded software packages* oraz upewnić się, że odpowiedni pakiet został zainstalowany (będzie posiadał ikonę małego zielonego kwadratu). Na potrzeby laboratoriów nie jest wymagana instalacja najnowszej wersji biblioteki, jeśli jest taka dostępna (Rys. 14).



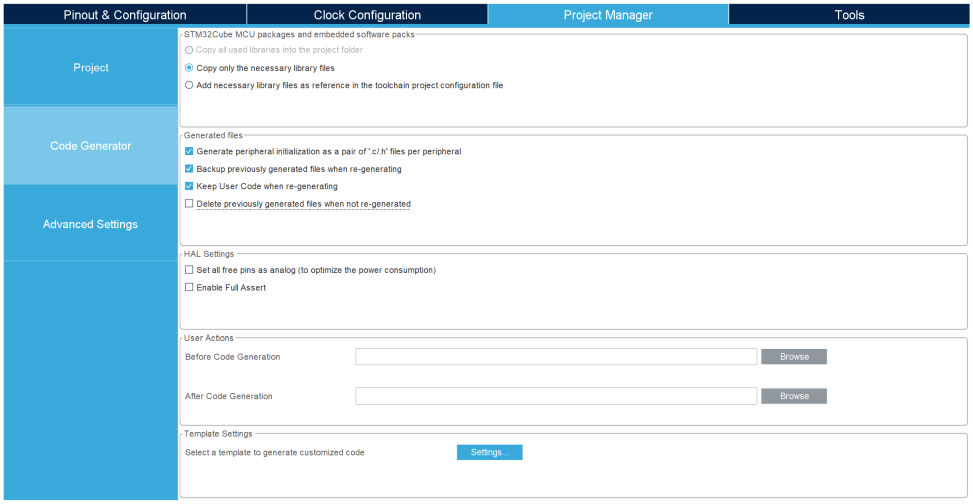
Rysunek 14: Zainstalowane biblioteki

Aby skonfigurować sposób w jaki kod zostanie wygenerowany należy użyć kombinacji kombinacji klawiszy **Alt**+**P** lub wybrać *Project* → *Settings*. W nowym oknie należy wpisać potrzebne dane 15.



Rysunek 15: Konfiguracja projektu – ustawienia ogólne

Konfigurację należy rozpocząć od ustawienia zestawu narzędzi, IDE (Toolchain / IDE). Należy wybrać `STM32CubeIDE`. Dodatkowo opcja generowania projektu w głównym drzewie katalogu (Generate Under Root) powinna być zaznaczona. Po wyborze IDE ustalamy nazwę projektu (Project Name), w tym przypadku jest to lab01 oraz wybieramy lokalizację. Kolejnym etapem jest ustalenie, w jaki sposób będzie generowany sam kod. Przykładowa konfiguracja została pokazana na rysunku 16.



Rysunek 16: Konfiguracja projektu – ustawienia generowania plików

Szczególną uwagę należy zwrócić na grupę generowanych plików (Generated files). Znajdują się tam następujące opcje:

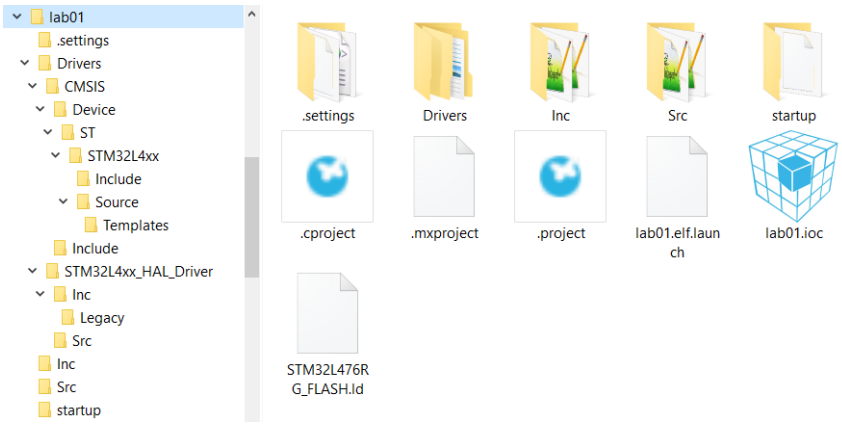
1. Generate peripheral initialization as a pair of '.c/.h' files per peripheral
2. Backup previously generated files when re-generating
3. Keep User Code when re-generating
4. Delete previously generated files when not re-gnerated

Pierwsza z nich powoduje, że tworzona jest para plików (kod źródłowy i nagłówek) dla każdego z peryferium. Opcja ta powinna być wybrana. Jest ona szczególnie przydatna podczas pracy zespołowej ponieważ pozwala na fragmentację kodu i ułatwia jego utrzymanie oraz zarządzanie. Drugie pole dotyczy tworzenia kopii zapasowej plików, które będą powtórnie generowane. Wówczas tworzony jest katalog BACKUP do wnętrza, którego przenoszone są pliki, do których nazwy dodawany jest człon ".old". Owy katalog jest tworzony zarówno dla kartotek Src oraz Inc, w których przechowywane są pliki źródłowe projektu oraz pliki nagłówkowe, odpowiednio. Wybór tego pola nie jest wymagany. Trzecia opcja pozwala na zachowanie modyfikacji kodu, jakie wprowadził już programista do kodu programu. Zachowywane są one jedynie, gdy zostały one zapisane wewnątrz odpowiednich znaczników w postaci komentarzy (jest to jedynie przykład, gdyż liczba takich znaczników jest większa):

```
1  /* USER CODE BEGIN 2 */
2
3  //tutaj powinien znaleźć się kod programisty
4
5  /* USER CODE END 2 */
```

Ostatnia z pól dotyczy usuwania plików, które w przypadku regeneracji nie ulegną zmianie. Owa opcja powinna być odznaczona, w przypadku, gdy jest aktywna może prowadzić do nieoczekiwanych zachowań podczas kompilacji wynikających z braku plików.

Po zaakceptowaniu ustawień przystępujemy do generowania kodu. Podobnie, jak poprzednim razem można posłużyć się skrótem klawiszowym **Ctrl**+**Shift**+**G** lub wybrać *Project* → *Generate Code*. Operacja ta nie powinna trwać dłużej niż kilkanaście sekund w zależności od konfiguracji sprzętowej stacji roboczej. W efekcie powinien się pojawić komunikat o pomyślnym wygenerowaniu kodu źródłowego. Komunikat ten należy zamknąć wybierając przycisk Zamknij (*Close*). Zawartość katalogu i jego struktura po operacji została pokazana na rysunku 17.



Rysunek 17: Struktura i zawartość drzewa dla przykładowego projektu

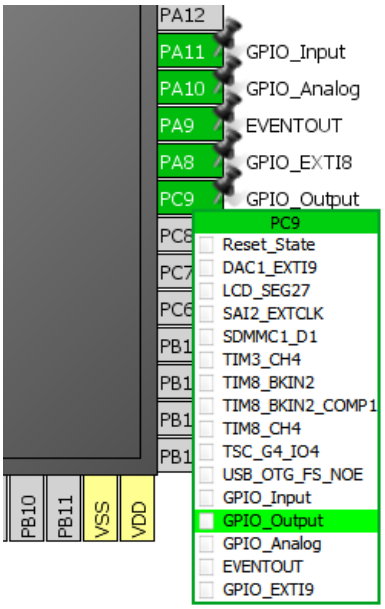
5 GPIO – cyfrowe wejścia/wyjścia ogólnego przeznaczenia

Jednym z podstawowych peryferiów mikrokontrolera są cyfrowe wejścia/wyjścia popularnie nazywane GPIO (*General Purpose Input Output*). Zazwyczaj wykorzystywane są one do sterowania zewnętrznymi układami, które nie wymagają wyrafinowanych szeregowych protokołów komunikacyjnych. Każdy pin może działać w jednym z poniższych trybów:

- wejście pływające,
- wejście podciągnięte do góry (*pull–up*),
- wejście ściągnięte w dół (*pull–down*),
- wejście analogowe,
- wyjście typu otwarty dren (*open–drain*) z możliwości podciągnięcia do góry (*pull–up*) lub ściągnięcia na dół (*pull–down*),
- wyjście typu *push–pull* z możliwości podciągnięcia do góry (*pull–up*) lub ściągnięcia na dół (*pull–down*),
- funkcja alternatywna typu otwarty dren z możliwości podciągnięcia do góry (*pull–up*) lub ściągnięcia na dół (*pull–down*),
- funkcja alternatywna typu *push–pull* z możliwości podciągnięcia do góry (*pull–up*) lub ściągnięcia na dół (*pull–down*).

Na szczególną uwagę zasługuje tryb analogowy. Należ tutaj podkreślić, że nie wszystkie wejścia GPIO mogą być podłączone do przetwornika analogowo–cyfrowego ADC. Jednakże konfiguracja danego pinu mikrokontrolera w trybie analogowym pozwala na ograniczenie pasożytniczych prądów. Osiąga się to poprzez wyłączenie:

- buforów wyjściowych,
- bramki Schitta,
- rezystorów podciągających/ściągających (*pull–up*/*pull–down*).



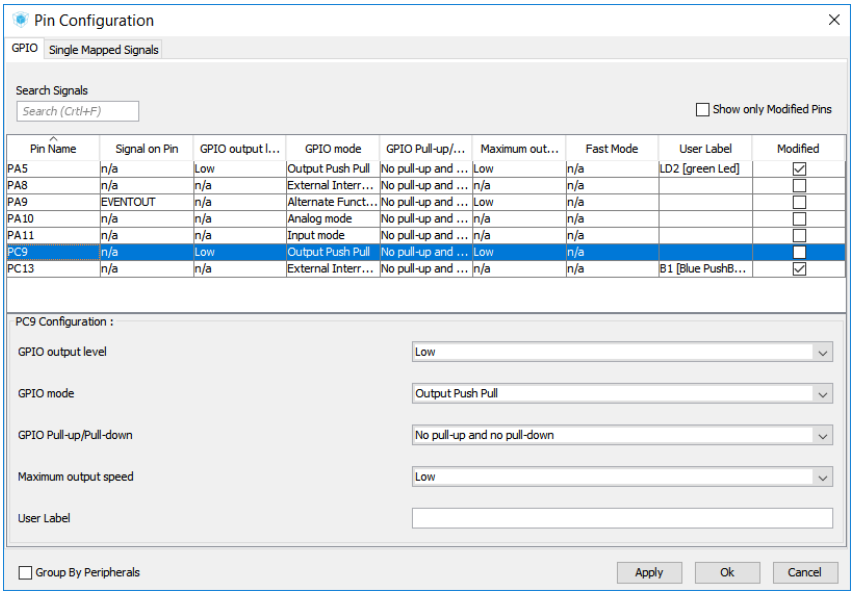
Rysunek 18: Wybór trybu pracy pinu mikrokontrolera

Na rysunku 19 została przedstawiona przykładowa konfiguracja siedmiu wybranych pinów mikrokontrolera (PA5, PA8, PA9, PA10, PA11, PC9, PC13). Aplikacja `STM32CubeMX` pozwala na łatwą i przejrzystą

Nazwa pinu	Funkcja
PA5	Wyjście (dioda LED)
PA8	Wejście przerwania zewnętrznego
PA9	Wyjście zdarzenia
PA10	Tryb analogowy
PA11	Wejście
PC9	Wyjście
PC13	Wejście przerwania zewnętrznego (przycisk)

Tabela 1: Przykładowa konfiguracja pinów mikrokontrolera

konfigurację peryferiów układu. Konfiguracja wybranego peryferium odbywa się na jeden z dwóch sposobów. Pierwszym z nich jest wybranie funkcji pinu mikrokontrolera przez kliknięcie na niego lewym przyciskiem myszy i wybraniu zadanej funkcjonalności z menu kontekstowego (rys. 18). Taka konfiguracja ma sens w przypadku, gdy będzie wykorzystywany pojedynczy pin, tak jak to ma miejsce w wypadku konfiguracji GPIO. Druga ze ścieżek wymaga w pierwszej kolejności wybrania odpowiedniego peryferium np. SPI, a wówczas wymagane piny zostaną automatycznie przypisane do domyślnych portów.



Rysunek 19: Przykładowa konfiguracja wybranych trybów

Jak można zauważyć na rysunku 19 zostało skonfigurowanych aż siedem piny mikrokontrolera. Dwa z nich dostępne są w podstawowej konfiguracji dla płytki deweloperskiej. Są nimi piny PA5, który pełni funkcję wyjścia, do którego podłączona jest dioda LED oraz PC13, który z kolei wykorzystywany jest jako wejście, do którego podłączony jest przycisk. Pozostałe piny zostały skonfigurowane w taki sposób, aby pokazać pełen wachlarz możliwości układu scalonego. Po kliknięciu w jeden z wybranych pinów GPIO rozwija się menu konfiguracyjne, gdzie możliwe jest wybranie dodatkowych ustawień dla aktywnego portu. Są one dostosowywane kontekstowo w zależności, czy nóżka mikrokontrolera pracuje, jako wejście, wyjście, czy w trybie analogowym.

5.1 GPIO w bibliotece HAL

Do podstawowych funkcji, które pozwalają na ingerowanie w pracę portów GPIO należą [2]:

- `GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)`,
- `void HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`,
- `void HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)`,

- `void HAL_GPIO_LockPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)`,
- `void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)`,
- `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`.

`HAL_GPIO_ReadPin()` pozwala na odczytanie aktualnej wartości rejestru wejściowego przypisanego do danego portu. Funkcja przyjmuje dwa parametry. Pierwszym z nich jest port, którego pin ma zostać odczytany natomiast drugim parametrem jest numer pinu, którego wywołanie dotyczy. Wartością zwracaną jest stan portu, która może przyjąć dwa stany `GPIO_PIN_RESET` (0), bądź `GPIO_PIN_SET` (1).

Modyfikację stanu wyjścia portu można wykonać poprzez wywołanie funkcji `HAL_GPIO_WritePin()`. Przyjmuje ona trzy parametry, gdzie pierwsze dwa są identyczne, jak w przypadku procedury odczytującej, natomiast trzeci z parametrów określa żądany stan cyfrowego wyjścia.

W przypadku, gdy wymagane jest przełączenie wyjścia na przeciwny stan można wykorzystać funkcję `HAL_GPIO_TogglePin()`. Przełącza ona stan pinu na przeciwny, to jest, z stanu wysokiego na niski i analogicznie z niskiego na wysoki. Parametry wywołania są identyczne, jak w przypadku `HAL_GPIO_ReadPin()`.

Kolejną funkcją, która jest wykorzystywana rzadziej niż poprzednie jest `HAL_GPIO_LockPin()`. Zadaniem funkcji jest zablokowanie rejestrów należących do danego pinu mikrokontrolera. W ten sposób po wywołaniu operacji zapisu, czy przełączenia stanu na zadanym porcie nie przyniesie ona żadnego efektu. Modyfikacja stanu zablokowanego portu będzie dopiero możliwa pod resecie mikrokontrolera.

Kolejne dwa wywołania `HAL_GPIO_EXTI_IRQHandler()` oraz `HAL_GPIO_EXTI_Callback()` dotyczą one obsługi przerwań zewnętrznych, zostaną one opisane szczegółowo później.

6 Zadania do wykonania

6.1 Uzupełnienie programu o kod odpowiedzialny za miganie diodą LED

Biblioteka HAL oferuje szeroką gamę funkcji właściwych dla każdego z peryferium. W przypadku cyfrowych portów wejściowo–wyjściowych ogólnego przeznaczenia (GPIO) można wyróżnić trzy podstawowe funkcje, których prototypy zostały podane poniżej [2].

```
GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
```

Funkcje służą odpowiednio do: odczytu stanu portu, zapisu stanu portu oraz przełączenia aktywnego stanu portu na przeciwny. Szczegółowy ich opis, a także typy wyliczeniowe, jakie są wykorzystywane w owych funkcjach można znaleźć w [2]. Ponadto, `STM32CubeMX` dostarcza definicji nazw portów oraz pinów, jakie zostały nadane przez użytkownika w projekcie `STM32CubeMX`. Mają one postać *XXX_Port* oraz *XXX_Pin*, odpowiednio dla portów i dla pinów. Definicje te znajdują się w pliku nagłówkowym *main.h*, bądź *mxconstants.h*

Wykorzystując podane wyżej API napisz prosty program, który będzie zapalał, a następnie gasił diodę LED podłączoną do pinu PA5.

Funkcją wprowadzającą aktywne opóźnienie (tzw. busy–wait) jest

```
void HAL_Delay (uint32_t Delay);
```

Przyjmuje ona jako parametr wartość wyrażoną w milisekundach [ms].

Uwaga! W przypadku, kiedy funkcja `HAL_Delay()` nie przynosi efektu należy stworzyć własną funkcję o nazwie `delay_ms()`. Jej prototyp został podany poniżej:

```
void delay_ms (uint32_t ms){
  uint32_t i, delay;
  delay = ms * 80000;
  for(i = 0; i < delay;){
    ++i;
  }
}
```

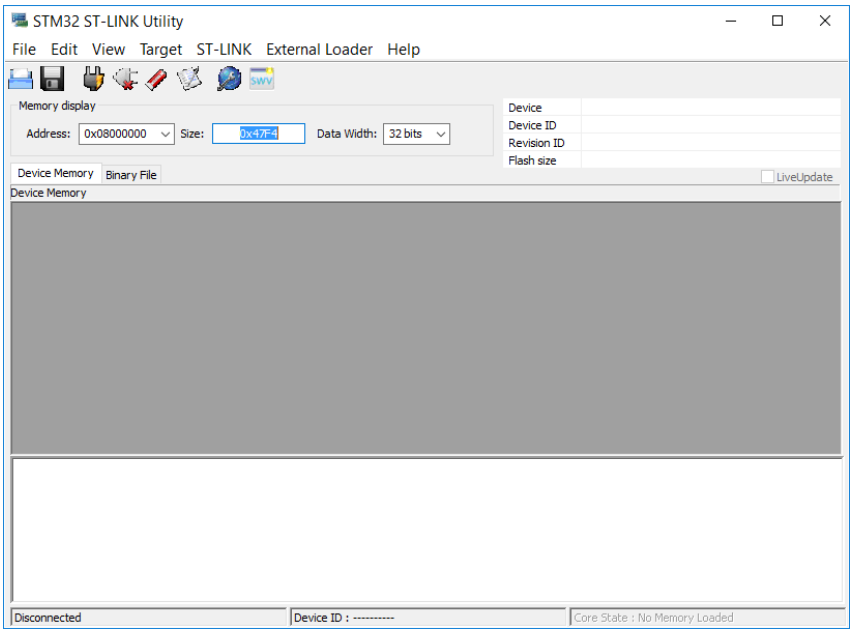
Wartość opóźnienia należy dostosować na zasadzie obserwacji. *Jaki jest powód, że funkcja `HAL_Delay()` nie przynosi oczekiwanego efektu?*

Uwaga! Numer portu, jak i pinu, do którego podłączona jest dioda LED można odczytać z projektu w programie `STM32CubeMX`.

6.2 Wgranie i testowanie programu na mikrokontrolerze

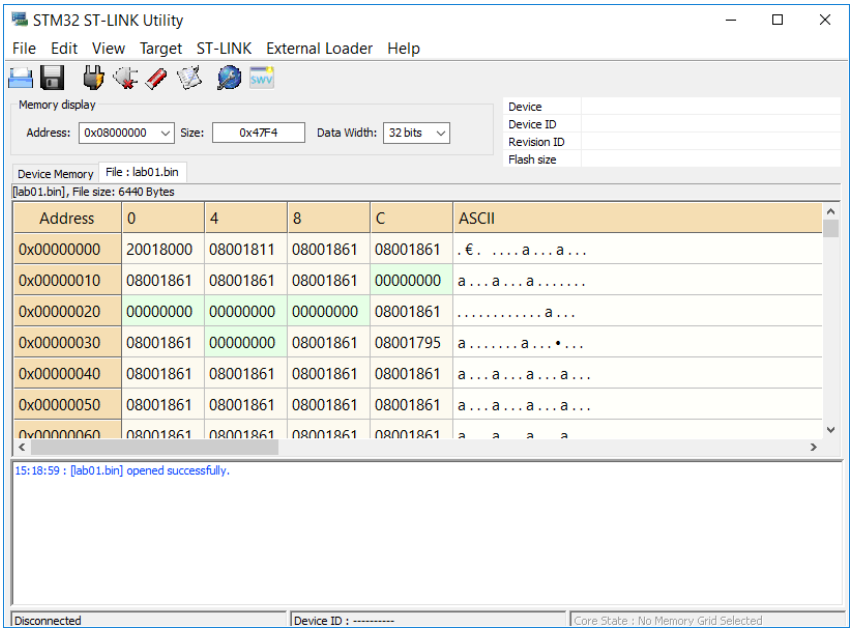
Kolejnym etapem po zaimporotwaniu projektu jest jego zbudowanie. Można to zrobić poprzez wybranie polecenia z menu (*Project* → *Build All*) lub użyć kombinacji klawiszy **Ctrl**+**B**. Budowa projektu jest czasochłonnym i może zająć kilka minut w zależności od konfiguracji projektu, jaki i od parametrów stacji roboczej.

Aby wgrać oprogramowanie na mikrokontroler należy podłączyć zestaw deweloperski do komputera za pomocą kabla USB–mini USB. Kolejnym krokiem jest uruchomienie aplikacji `STM32 ST-Link Utility` [6], [7]. Widok aplikacji `ST-Link` zaraz po uruchomieniu został zaprezentowany na rysunku 20.



Rysunek 20: Aplikacja STM32 ST-Link Utility

Otwórz plik binarny (**.bin* lub **.hex*) za pomocą polecenia z menu (*File* → *Open file ...*). Plik binarny znajduje się wewnątrz katalogu *Debug* w głównej kartotece projektu. Po otwarciu pliku z programem aplikacja powinna zmienić wygląd na podobny do prezentowanego na rysunku 21.

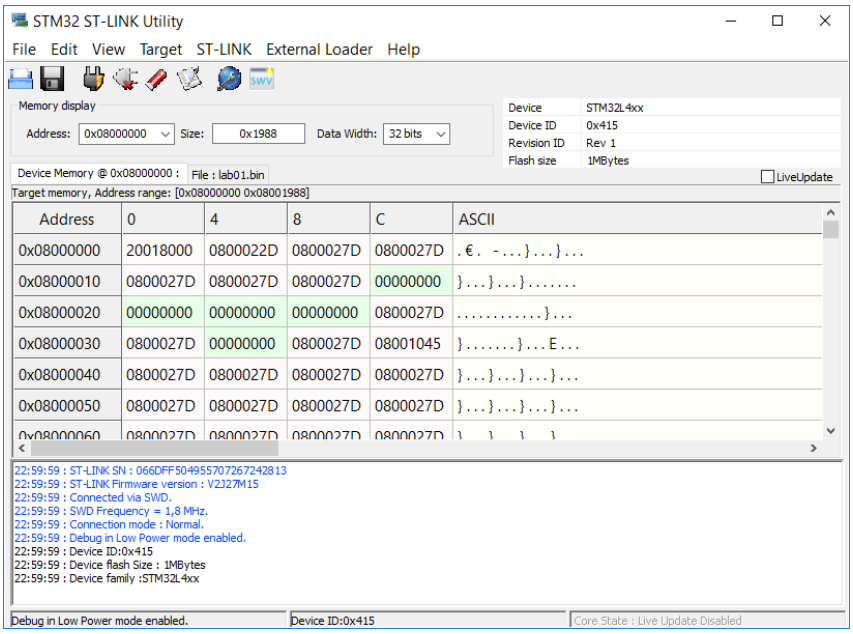


Rysunek 21: ST-Link Utility z otwartym kodem programu

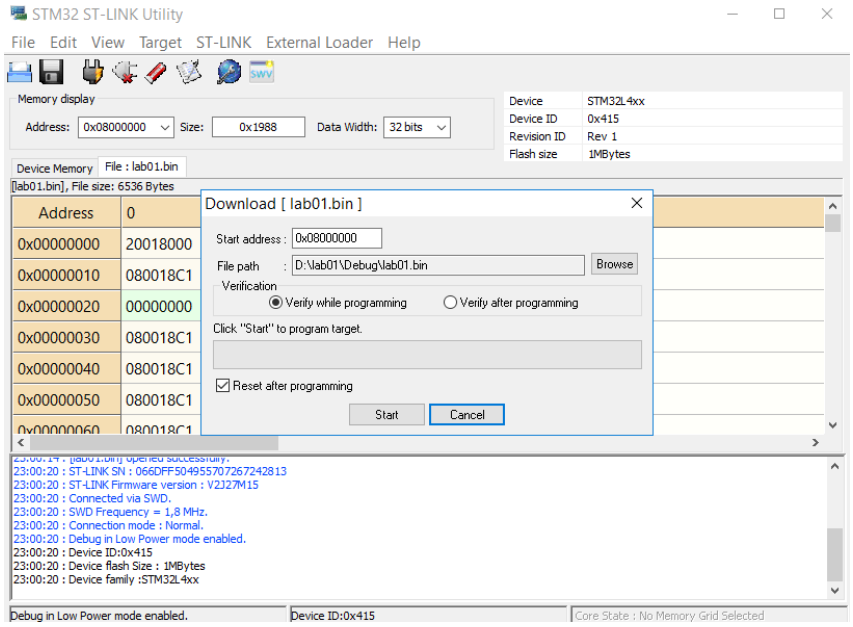
Aby wgrać program należy wykonać trzy operacje:

1. Połączyć się z zestawem edukacyjnym wybierając z menu opcję Połącz (*Target* → *Connect*), Rys. 22.
2. Wgrać program za pomocą polecenia Programowania i weryfikacji (*Target* → *Program & Verify ...*), Rys. 23.
3. Odłączyć zestaw ewaluacyjny wybierając polecenie Rozłącz (*Target* → *Disconnect*), Rys. 24.

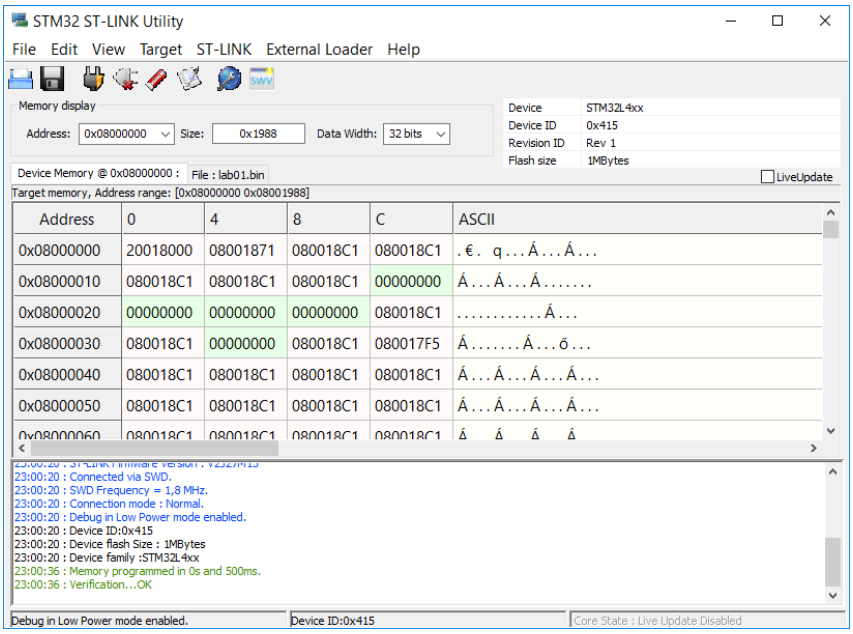
Uwaga! W przypadku problemów z wgraniem programu na płytkę deweloperską należy upewnić się, czy system Windows zainstalował do niej sterowniki. Instalacja oprogramowania może potrwać około minuty. Postęp instalacji można śledzić w zasobniku systemowym.



Rysunek 22: Widok aplikacji ST-Link po udanej próbie połączenia z programatorem



Rysunek 23: Widok aplikacji po wczytaniu pliku oraz wybraniu operacji zapisu programu z weryfikacją



Rysunek 24: Widok aplikacji po pomyślnym wgraniu nowego oprogramowania na mikrokontroler

Po pomyślnym wgraniu programu dioda użytkownika powinna migać.

Uwaga! W przypadku, gdy operacja połączenia się nie powiodła to najprawdopodobniej układ nie został połączony z komputerem za pomocą dostarczonego przewodu USB lub zwory na złączu CN2 nie są zwarte [5]. Innym powodem może być brak zainstalowanych sterowników w systemie. W takim przypadku należy zweryfikować, czy sterowniki te są instalowane, a w tym czasie nie wolno odłączać urządzenia (płytki) od komputera.

6.3 Modyfikacja projektu i programu w celu obsługi przycisku pracującego w roli przełącznika dla diody LED

Jako modyfikację programu należy wykorzystać przycisk znajdujący się na płycie do zmiany stanu diody (włączona/wyłączona). Innymi słowy, pojedyncze naciśnięcie przycisku powinno zmieniać stan diody. Zmiana stanu diody powinna nastąpić tylko podczas wciskania przycisku, a nie jego puszczania. Dodatkowo, w przypadku, gdy przycisk jest trzymany cały czas stan diody nie powinien się zmieniać. Ponadto, w programie nie można wykorzystać opóźnienia większego niż 10 milisekund.

Zaimplementuj procedurę do niwelowania zjawiska drgań styków (tzw. debouncing). Do tego celu można wykorzystać przerwania, ale nie jest to wymagane. Procedura powinna być tak napisana, aby nie zatrzymywać wykonywania programu niepotrzebnie. Do tego celu można wykorzystać mały jednostanowy automat.

Czy modyfikacja projektu w programie `STM32CubeMX` jest konieczna? Jeśli tak to co należy w nim zmienić. Powtórz kroki związane z automatycznym generowaniem kodu jeśli zajdzie taka potrzeba.

Przydatną funkcją może okazać się

```
1 uint32_t HAL_GetTick (void);
```

zwraca ona liczbę tików zegara SysTick. Można przyjąć, że jeden tick to 1 milisekunda. Funkcję tą warto wykorzystać podczas implementowania debouncingu.

Uwaga! Numer portu, jak i pinu, do którego podłączony jest przycisk można odczytać z projektu w programie `STM32CubeMX`.

6.4 Wgranie i testowanie zmodyfikowanego programu na mikrokontrolerze

Postępując podobnie, jak to zostało opisane w sekcji 6.2 nanieś modyfikacje, przebuduj program (*Ctrl+B*) oraz wgraj go na płytkę za pomocą narzędzia `ST-Link Utility`.

6.5 Uporządkowanie stanowiska

Odłóż płytkę i kabel na miejsce. Usuń projekt z Atollic TrueSTUDIO. Można to zrobić przez kliknięcie prawym przyciskiem myszki na projekt i wybranie opcji Usuń (Delete) z menu kontekstowego.

7 Podsumowanie

Poprzez wykonanie ćwiczeń z tej instrukcji zaznajomiłeś(aś) się z podstawowymi narzędziami do generowania i rozwijania oprogramowania dla mikrokontrolera serii STM32L4. Przedstawiono tutaj narzędzie `STM32CubeMX`, które pozwala na wstępne skonfigurowanie peryferiów układu oraz dobór taktowania dla poszczególnych zespołów funkcjonalnych mikrokontrolera. Ponadto, `STM32CubeMX` pozwala na automatyczne generowanie kodu, który może być następnie modyfikowany. Tak wygenerowane pliki źródłowe wraz z całym projektem mogą być bez przeszkód zaimportowane do darmowego środowiska programistycznego `Atollic TrueSTUDIO for STM32`, w którym to wykonuje się edycję kodu źródłowego oraz kompilację. W końcu możliwe jest wgranie pliku binarnego na mikrokontroler umieszczony na płytce deweloperskiej za pośrednictwem `STM32 ST-LINK Utility`.

Literatura

[1] ST. *STM32L476xx, Ultra-low-power ARM® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, analog, audio.*, Grudzień, 2015.

[2] ST. *Description of STM32L4 HAL and Low-layer drivers.*, Luty, 2016.

[3] ST. *STM32 configuration and initialization C code generation.*, Kwiecień, 2016.

[4] ST. *STM32 Nucleo-64 board.*, Listopad, 2016.

[5] ST. *STM32 Nucleo-64 board, User manual.*, Listopad, 2016.

[6] ST. *STM32 ST-LINK Utility for STM32 MCUs.*, Listopad, 2016.

[7] ST. *STM32 ST-LINK utility software description, User manual.*, Sierpień, 2016.

[8] ST. *STM32CubeMX for STM32 configuration and initialization C code generation, User manual.*, Marzec, 2017.

[9] ST. *STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs, Reference Manual.*, Marzec, 2017.

