Systemy mikroprocesorowe

Instrukcja laboratoryjna

Spotkanie 1

Wprowadzenie do programowania mikrokontrolera

Autorzy: dr inż. Paweł Dąbal

Ostatnia aktualizacja: 08.11.2022 r.

Wersja: 2.0.0

Warszawa 2022

Spis treści

1. Wprowadzenie

Instrukcja ta stanowi wstęp wprowadzenie do pierwszych zajęć laboratorium, które mają na celu wprowadzenie studentów w elementarz niezbędny do programowania mikrokontrolera z rdzeniem *ARM Cortex-M4*, z użyciem języka C. W trakcie zajęć student zapozna się z środowiskiem programistycznym *STM32CubeIDE*, sposobem tworzenia projektu w oparciu o konfigurator, płytą uruchomieniową *KAmeLeon* oraz obsługą portów wejścia/wyjścia mikrokontrolera *STM32L496ZGT6*. Ponadto będzie potrafił obsługiwać narzędzia wspomagające kontrolę wersji oprogramowania w celu dokumentowania własnych postępów.

1.1. Cel ćwiczenia

Ćwiczenie laboratoryjne ma na celu:

- nabycie umiejętności konfiguracji stanowiska pracy w oparciu o oprogramowanie STM32CubeIDE 1.7.0;
- poznanie płyty uruchomieniowej *KAmeLeon* z mikrokontrolerem *STM32L496ZGT6*;
- przypomnienie i usystematyzowanie wiedzy i umiejętności z zakresu posługiwania się językiem C,
- praktyczne korzystanie z systemu kontroli wersji *Git* i serwisu *GitHub*.

1.2. Wymagania wstępne

Przed przystąpieniem do wykonywania ćwiczenia laboratoryjnego należy :

- zapoznać się z schematem płyty uruchomieniowej *KAmeLeon* oraz dokumentacją mikrokontrolera *STM32L496ZGT6* (*Product Specifications*, *Reference Manuals*);
- zapoznać się z składnią języka C – pojęcie zmiennej, stałej, funkcji, prototypu funkcji, parametru funkcji, wyrażenia warunkowego, pętli, wskaźnik, struktury i tablicy;
- utworzyć konto na platformie *GitHub*;
- zapoznać się z dokumentacją środowiska *STM32CubeIDE* i biblioteki *STM32CubeL4*.

1.3. Opis stanowiska laboratoryjnego

Stanowisko do przeprowadzenia zajęć składa się z komputera PC z zainstalowanym oprogramowaniem koniecznym do realizacji zajęć: *STM32CubeIDE*, dedykowaną do układu biblioteką *STM32CubeL4* i systemem *Git* oraz płyty *KAmeleon* podłączonej do komputera za pomocą przewodu USB do złącza programatora *SWD PRG/DBG/vCOM* płyty uruchomieniowej. Połączenie to również odpowiada za zasilanie płyty.

1.3.1. Środowisko programistyczne

Środowisko projektowe *STM32CubeIDE* (IDE) firmy *STMicroelectronics* (STM) powstało na bazie oprogramowania *TrueStudio* firmy *Atolic*. Bazuje ono na szablonie powszechnie stosowanego oprogramowania *Eclipse*. Łączy w sobie edytor kodu, menadżer projektów, zarządzanie kontrolą wersji, kompilator GCC, debugger oraz narzędzie do generowania projektów dla mikrokontrolerów STM (*Device Configuration Tool*), które jest wbudowaną wersją samodzielnego programu *STM32CubeMX*. Cały proces tworzenia oprogramowania dla mikrokontrolerów STM może być realizowany z użyciem jednego środowiska. Tworzenie oprogramowania może być realizowane również w takich narzędziach komercyjnych jak *Keil µVision IDE*, *IAR Embedded Workbench* jednak ich ograniczenia dla wersji bezpłatnych w rozmiarze wynikowego kodu i ubogie rozbudowanie w zakresie wsparcia edycji kodu nie są dostatecznie rekompensowane przez bardzo dobre kompilatory.

1.3.2. Płyta uruchomieniowa *KAmeLeon*

W sprzedaży dostępnych jest kilkaset płyt uruchomieniowych z mikrokontrolerami z rdzeniem ARM Cortex-M4. Od rozwiązań miniaturowych o ograniczonej funkcjonalności do płyt wyposażonych w wiele komponentów niekiedy wymagających dla obsługi dobrej znajomości platformy. Odpowiednikiem płyt uruchomieniowych takich jak *Arduino* w portfolio firmy STM jest grupa produktów serii *Nucleo* dostępna z wieloma różnymi modelami mikrokontrolerów. Co więcej powstała również stosowna implementacja szkieletu (ang. *framework*) biblioteki *Arduino* dla tych płytek umożliwiająca łatwą migrację na zdecydowanie wydajniejszą platformę. Jednakże prawdziwy potencjał w używaniu mikrokontrolerów z rdzeniem *ARM Cortex-M4* tkwi w wbudowanych w nie peryferiach, a zademonstrowanie bogatej funkcjonalności wymaga zewnętrznych komponentów. Przykładem mogą być dostępne płytki rodziny *Discovery*, które posiadają liczne komponenty uzupełniające takie jak wyświetlacze LCD, pamięci *flash* i RAM, czytniki kart, kodeki audio, liczne diody LED i przyciski oraz sensory. Z racji na rozbudowanie charakteryzują się znacznie wyższą ceną.

Płyta *KAmeLeon* przygotowana została przez firmę *Kamami* i jest dedykowana do celów edukacyjnych. Wyposażona jest w nowoczesny mikrokontroler STM32L496ZGT6 z 32 bitowym rdzeniem ARM Coretx-M4. Może ona pracować z maksymalną częstotliwością 80 MHz. Produkt ten zastępuje starsze płyty ZL27ARM z mikrokontrolerem STM32F103VBT6. Do dyspozycji jest 320 KB pamięci RAM i 1MB pamięci *flash*, 16 różnych liczników, 20 różnych kontrolerów interfejsów (SPI, I2C, UART, SAI, CAN, USB, SDMMC, SWPMI), wsparcie dla obsługi kamery, wyświetlaczy, kontroler zewnętrznych pamięci RAM/ROM. Ponadto dostępne są trzy 12-bitowe przetworniki ADC które mogą monitorować 24 kanały, dwa 12-bitowe przetworniki DAC oraz po dwa komparatory i konfigurowane wzmacniacze operacyjne. Zestaw ma wbudowany programator/debuger ST-Link z gniazdem micro-USB co minimalizuje wymagane wyposażenie użytkownika. Zasilanie może być dostarczone przez złącze programatora lub dedykowane do maksymalnie 12 V. Na płycie znajdują się takie komponenty jak diody LED, potencjometr, joystick, wyświetlacz 7 segmentowy, analogowy mikrofon oraz wejścia/wyjścia audio ze złączami jack i dedykowanymi wzmacniaczami, 1MB pamięć QSPI Flash, układ akceleratora/magnetometru (LSM303C), czujnik temperatury (LM75) oraz kontroler silnika DC. Dodatkowym atutem *KAmeLeon-a* jest złącze przeznaczone do dołączenia analizatora/oscyloskopu Analog Discovery 2, dzięki czemu można "podglądać" stany linii mikrokontrolera. Za pomocą zamontowanych złączy do płytki można dołączyć kamerę DCMI, wyświetlacz LCD (złącze FMC), wyświetlacz LCD alfanumeryczny ze sterownikiem HD44780, a także dowolne inne rozszerzenie za pomocą złącz *Arduino*, *Pmod* i *KAmod* (I2C oraz SPI). Szczegółowy schemat płyty można znaleźć w [1].

1.3.3. System kontroli wersji *Git*

Kontrola wersji w programowaniu odgrywa bardzo istotną rolę. W odróżnieniu od wykonywania kopi zapasowych dostarcza szereg narzędzi umożliwiających porównywanie plików, ale również porównywanie plików od różnych edytorów (programistów). Jednym z najpopularniejszych systemów kontroli wersji jest *Git*, który jest dostępny na wielu platformach . System bazuje na rozproszonej architekturze, tzn. każdy pracujący nad wspólnym projektem posiada lokalną kopię. Wprowadzane zmiany są scalane, a ewentualne różnice rozstrzygane i zatwierdzane. *Git* jest narzędziem pracującym w linii poleceń. Obecnie zdecydowana większość środowisk programistycznych posiada wbudowaną obsługę systemu *Git*. Powstało wiele darmowych serwisów internetowych udostępniających użytkownikom dzielenie się efektami własnej pracy z innymi, np. *GitHub*, *GitLab*, *Bitbucket*.

W trakcie zajęć użyty zostanie system kontroli wersji *Git*, który pozwoli wykonać dokumentację z realizowanych zajęć i zapisać wyniki prac studentów. Do tego celu posłuży serwer kontroli wersji *GitHub* na

którym należy posiadać imienne konto. Student będzie korzystał z wstępnie przygotowanego szablonu projekt udostępnionego przez prowadzącego, który to uzupełni o wymaganą na zajęciach treść – będzie to rodzaj sprawozdania z zajęć i pozwoli dodatkowo na skomentowanie efektów pracy przez prowadzącego zajęcia. Do realizacji zajęć użyty zostanie moduł *Classroom* serwisu *GitHub*, który usprawnia proces zarządzania repozytoriami w ramach grupy studenckiej.

1.4. Sposób realizacji ćwiczenia laboratoryjnego

Każde zajęcia składać będą się z następujących elementów:

- zadań obowiązkowych - do wykonania krok po kroku na podstawie instrukcji – do uzyskania od 0 do 12 pkt.;
- zadań uzupełniających – do samodzielnego zbudowania i napisania programu – do uzyskania od 0 do 14 pkt;
- pytań sprawdzających rozumienie działania programu – zadawane przez prowadzącego przyjmującego wykonanie zadania – odpowiedź wpływa na punktację z zadań obowiązkowych i uzupełniających, tzn. czy przyznać maksymalną możliwą liczbę punktów za zadanie czy tylko część przy ewidentnym braku zrozumienia problemu.

Po zrealizowaniu każdego z zadań należy poprosić prowadzącego o sprawdzenie i przyznanie punktów. Na koniec zajęć wystawiana jest ocena na podstawie sumy uzyskanych punktów: **2,0** <0; 10), **3,0** <10; 13), **3,5** <13; 16), **4,0** <16; 19), **4,5** <19; 23), **5,0** <23; 26>. Przy każdym zadaniu określona jest liczba punktów jakie można uzyskać. W przypadku zadań uzupełniających premiowana jest *jakość* i *czas realizacji* zaprezentowanego rozwiązania. Ocena końcowa z laboratorium to średnia arytmetyczna ocen z każdego spotkania.

W trakcie wykonywania ćwiczenia należy zwrócić szczególną uwagę na wyróżnione w treści instrukcji fragmenty tekstu wyróżnione kolorem:

- ***zielonym*** – etykiety wartości, nazwy pola, nazwy funkcji;
- ***niebieskim*** – wartości jakie należy użyć we skazanym miejscu;
- ***purpurowy*** – odwołania do kodu programu, nazw własnych.

2. Zadania podstawowe do realizacji (12 pkt.)

W tej części instrukcji zamieszczone są treści, z którymi obowiązkowo należy się zapoznać i praktycznie przećwiczyć. Ważne jest aby zapamiętać wykonywane przedstawione czynności aby móc na kolejnych zajęciach wykonywać je na kolejnych zajęciach bez potrzeby sięgania do niniejszej instrukcji.

Uwaga: Załączone wycinki z ekranu są poglądowe i pomagają jedynie w wskazaniu lokalizacji elementów interfejsu. Należy używać wartości podanych w tekście.

2.1. Zagadnienia wstępne

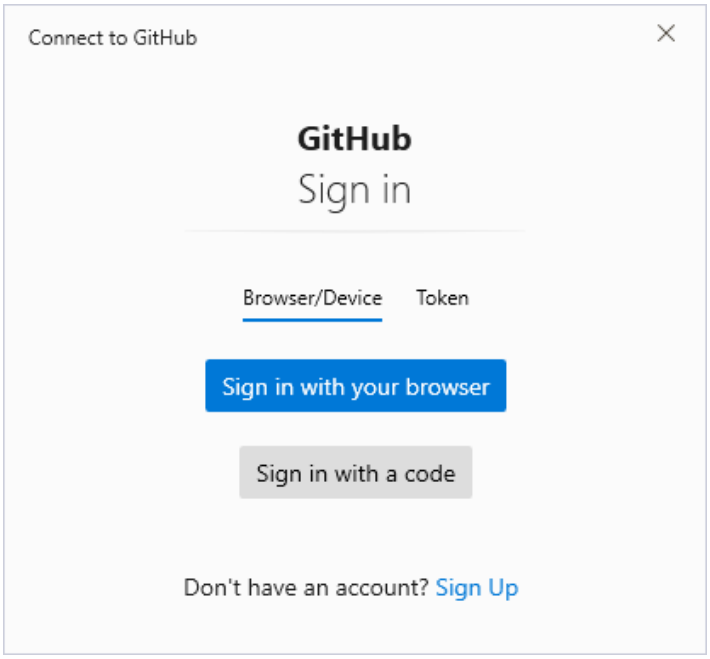
Przed przystąpieniem do pracy należy skonfigurować zainstalowane oprogramowanie i uzyskać dostęp do repozytorium, tzn. system kontroli wersji *Git*, instalacji biblioteki do obsługi mikrokontrolerów rodziny STM32L4 w środowisku *STM32CubeIDE* oraz sklonować repozytorium dla zajęć.

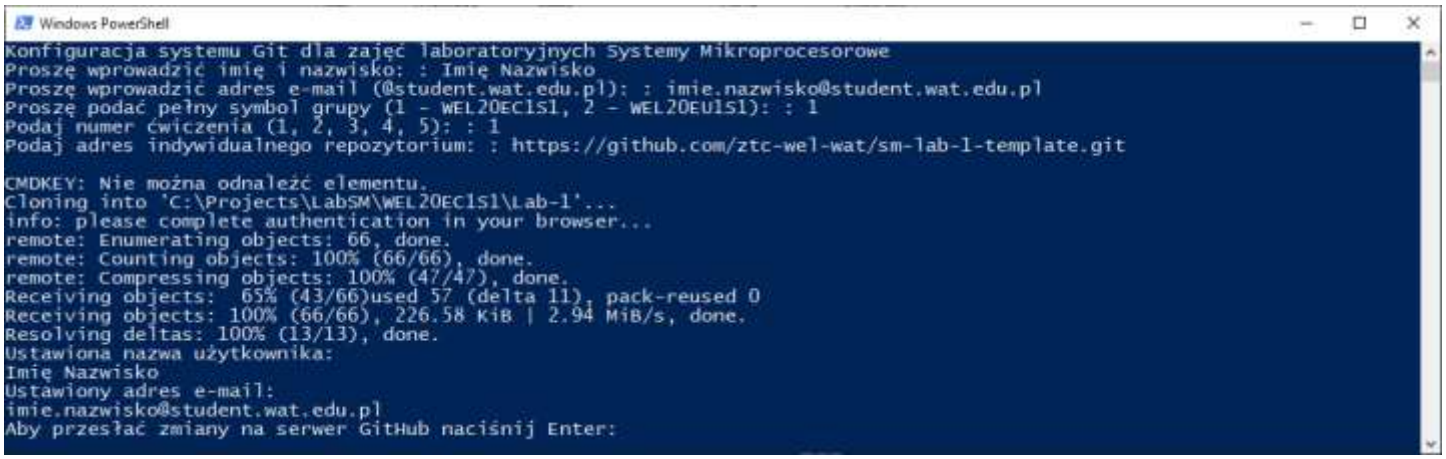
2.1.1. Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania

Na zajęciach należy dołączyć do wirtualnej grupy w ramach *Classrom GitHub* *za pomocą udostępnionego odnośnika prowadzącego do zadania*. Odnośnik prowadzi do kreatora w którym tworzone jest zadanie – indywidualne repozytorium studenta. Z listy należy wybrać swój adres e-mail i potwierdzić przyjęcie zadania (ang. *assignment*). Automatycznie zostanie utworzone prywatne repozytorium indywidualnie dla każdego studenta na podstawie przygotowanego repozytorium-szablonu. Na pierwszych zajęciach jest to wersja minimalna, a na kolejnych będzie zawierała już wstępnie skonfigurowane projekty. W sytuacji jeżeli student nie może odszukać się na liście (np. ktoś inny podłączył się pod daną osobę) proszę zgłosić to prowadzącemu zajęcia. W celu skorygowania nieprawidłowości. Zaakceptowanie zadania wiąże się również z dołączeniem do organizacji – wirtualnego konta organizacji w ramach którego tworzone są indywidualne repozytoria do zadań, z którego prowadzący zajęcia mają dostęp do wszystkich repozytoriów tworzonych w ramach zajęć.

2.1.2. Pobranie repozytorium i konfiguracja lokalna

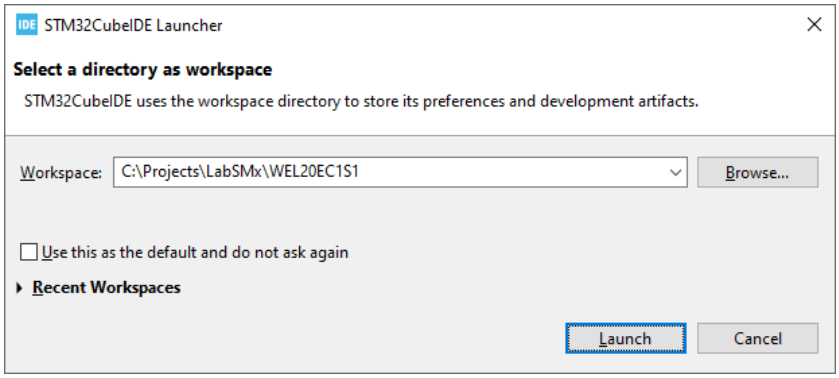
W celu pobrania repozytorium należy odszukać na pulpicie skrót o nazwie ***LabGitConfig***, który uruchomi skrypt wiersza poleceń, w którym należy podać: 1) *swoje imię i nazwisko*, 2) *adres e-mail*, 3) *symbol grupy*, 4) *numer ćwiczenia*, 5) *adres indywidualnego repozytorium* uzyskany w wcześniejszym punkcie. Po wykonaniu punktu 5) pojawi się okno logowania do serwisu *GitHub* podobne do zamieszczonego obok. W celu połączenia należy wybrać przycisk ***Sign in with your browser*** co spowoduje otworzenie nowej karty przeglądarki w którym należy udzielić dostępu do konta. Po uzyskaniu zgody nastąpi sklonowanie projektu z serwera do lokalizacji wynikającej z symbolu grupy. Na ten moment należy zminimalizować okno wiersza poleceń. Na zakończenie zajęć pozwoli ono na wypchnięcie zmian (ang. *push*) do repozytorium zdalnego. Przykładowy przebieg wykonania skryptu zamieszczony został poniżej.



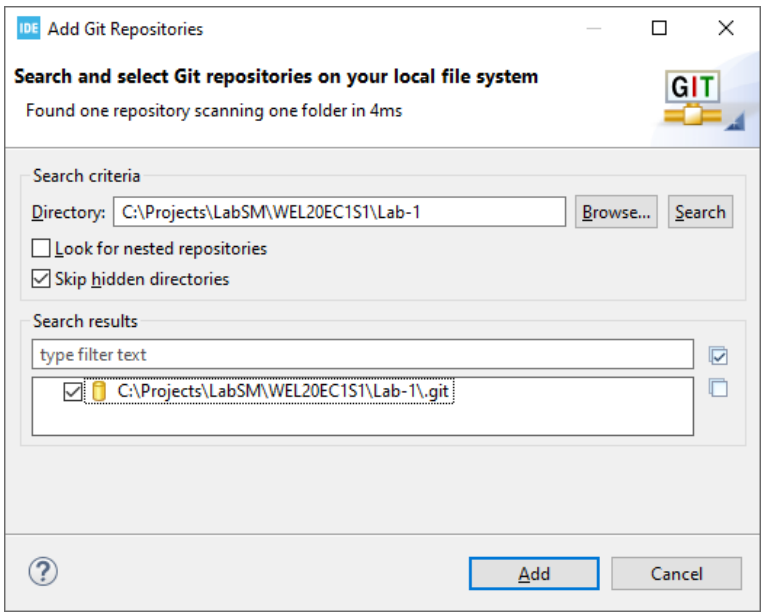


2.1.3. Konfiguracja środowiska STM32CubeIDE

Korzystając ze skrótu na pulpicie lub z menu *Start* należ uruchomić środowisko projektowe *STM32CubeIDE*. Po jego uruchomieniu pojawi się pytanie o wskazanie katalogu, który będzie pełnił rolę przestrzeni roboczej. Należy wskazać na katalog *swojej grupy studenckiej* gdzie umieszczone zostało sklonowane repozytorium (w przykładzie: *C:\Projects\LabSM\WEL20EC1S1*). Wybór katalogu zatwierdzamy przyciskiem ***Launch***. Można używać wielu różnych przestrzeni roboczych w przypadku pracy z różnymi projektami. Po załadowaniu środowiska należy zamknąć zakładkę *Information Center*.



W celu zarządzania repozytorium dla zadania należy w środowisku *STM32CubeIDE* należy otworzyć widok (ang. *perspective*) zarządzania repozytorium. W tym celu z menu wybieramy: ***Window → Perspective → Open Perspective → Other…,*** gdzie z listy należy wybrać ***Git***. Środowisko przełączy się do widoku *Git* i po lewej stronie pojawi się zakładka *Git Repositories*. Następnie korzystając z odnośnika ***Add an existing local Git repository*** (domyślnie po lewej stronie programu) otworzy się okno dodawania repozytorium (*Add Git Repositories*). Za pomocą przycisku ***Browse…*** należy wskazać położenie sklonowanego repozytorium (np. *C:\Projects\LabSM\WEL20EC1S1\Lab-1*), a następnie zaznaczyć pozycję dla ćwiczenia pierwszego.

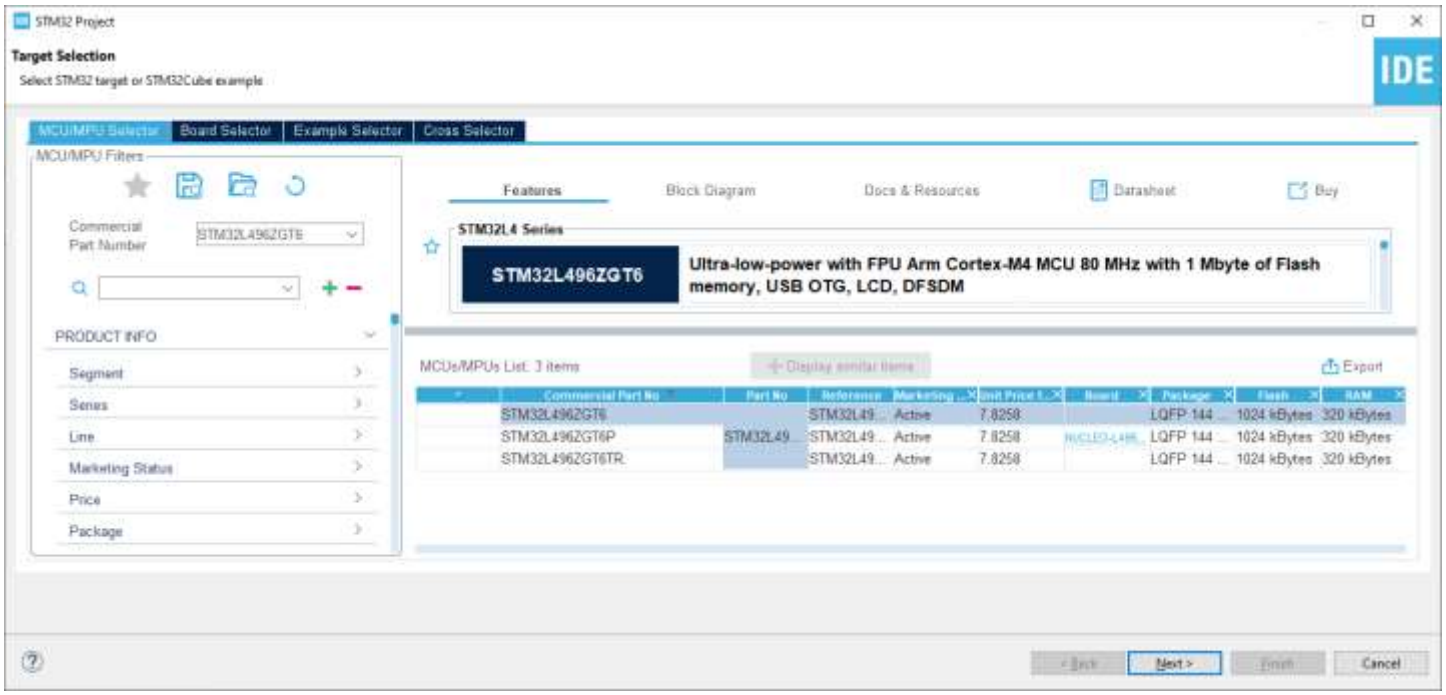


Mechanizm migawek pozwala na zapisywanie chwilowego stanu projektu. Można cofnąć się do wcześniejszej fazy projektu, tworzyć rozgałęzienia, łączyć rozgałęzienia. W trakcie zajęć ograniczymy się do prostej liniowej struktury migawek wykonywanych po zakończeniu każdego z zadań instrukcji opatrzonych stosownym

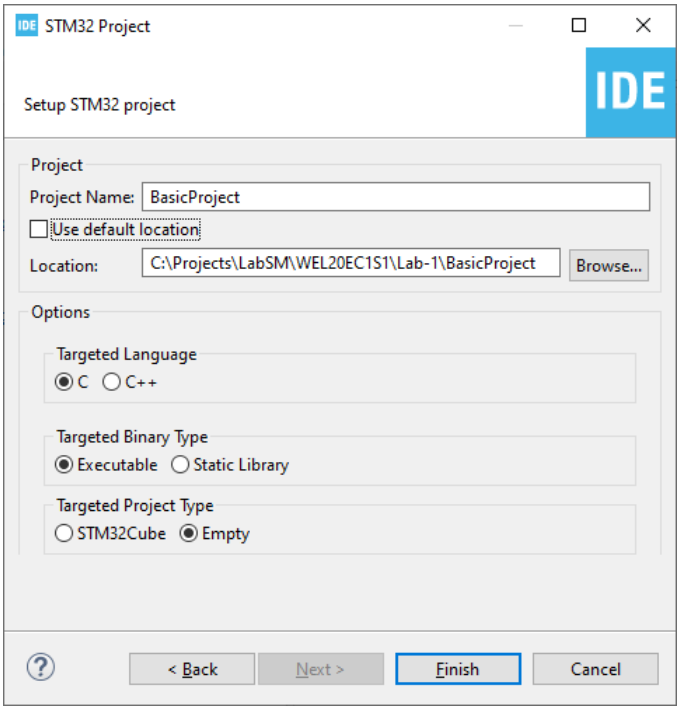
komentarzem. W dalszej części instrukcji będą podane treści komentarzy jakimi należy opatrzyć realizowane migawki. Powstanie zatem coś w rodzaju sprawozdania z zajęć, które będzie *przechowywane* w serwisie *GitHub*.

2.1.4. Utworzenie projektu bazowego i programowanie

Po przeładowaniu widoku środowiska, jeżeli to nie nastąpiło automatycznie, należy przełączyć się na widok w perspektywie ***C/C++*** (***Window → Perspective → Open Perspective → Other → C/C++***). W sklonowanym repozytorium nie został umieszczony jeszcze żaden projekt w związku z czym należy go utworzyć. W tym celu należy z wybrać w menu ***File → New → STM32 Project*** co otworzy kreator nowego projektu. W pierwszym oknie *Target selector* w zakładce *MCU/MPU Selector* w polu *Commercial Part Numer* należy podać oznaczenia mikrokontrolera ***STM32L496ZGT6***, a następnie z listy pasujących MCU wybrać pierwszą pozycję, a następnie przejść dalej przyciskiem *Next*.

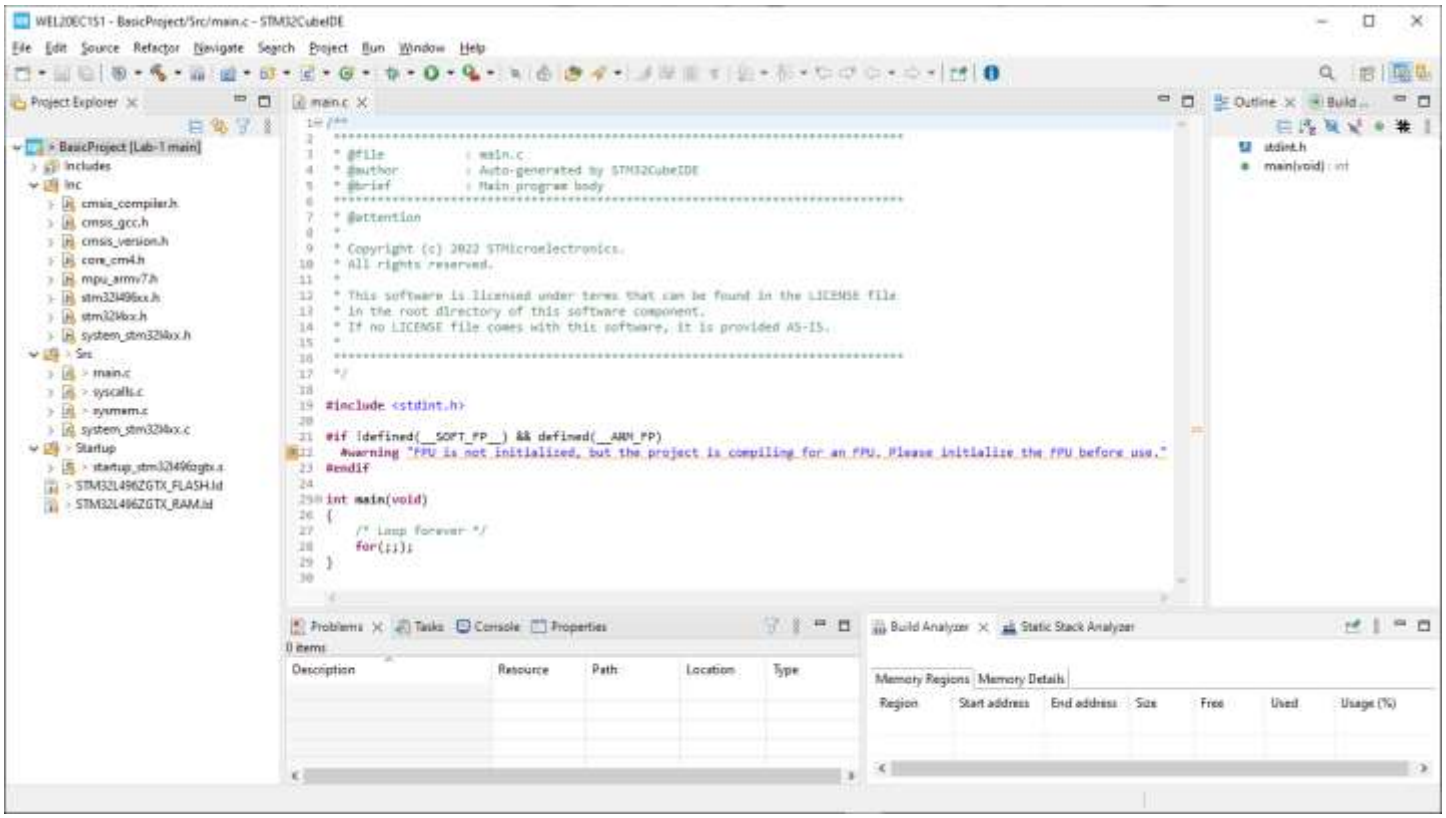


W kolejnym oknie (*Setup STM32 project*) należy podać nazwę projektu ***BasicProject***, w części *Target Project Type* należy zmienić typ projektu na pusty (***Empty***). Następnie należy odznaczyć pole *Use default location* i w polu *Location* podać lokalizację katalogu repozytorium (np.: ***C:\Projects\LabSM\WEL20EC1S1\Lab-1***).



Utworzony zostanie podstawowy projekt z pustą funkcją *main* w pliku *main.c* oraz źródłowymi plikami pomocniczymi (*syscall.c*, *sysmem.c*, *startup_stm32l496zgtx.s*) oraz skryptami dla linkera (*STM32L496ZGTX_FLASH.ld*, *STM32L496ZGTX_RAM.ld*). Konfiguracja ta umożliwia kompilowanie i uruchomienie na mikrokontrolerze jednak nie jest wystarczająca do korzystania z jego peryferii. W sklonowanym repozytorium znajduje się katalog ***BasicLib***, z którego

należy skopiować zawartość (katalogi *Inc* oraz *Src*) do katalogu głównego projektu (*BasicProject*). W ten sposób dodany zostanie plik źródłowy *system_stm32l4xx.c* konfigurujący podstawowe nastawy rdzenia mikrokontrolera, oraz pliki nagłówkowe biblioteki *CMSIS* (dla rdzenia) i definicji rejestrów mikrokontrolera (*stm32l4xx.h*, *stm32l496xx.h*, *system_stm32l4xx.h*). Finalnie struktura projektu powinna wyglądać jak przedstawiona poniżej.



Aby móc używać biblioteki w pliku *main.c* należy zamieścić dołączenie pliku nagłówkowego w postaci: ***#include*** *"stm32l4xx.h"*. W celu skompilowania projektu należy wybrać z menu ***Project → Build All*** (*Ctrl + B*). Poprawnie zakończony proces kompilacji kończy się podsumowaniem w postaci wyświetlenia rozmiaru podstawowych sekcji programu w zakładce *Consola*:

```
arm-none-eabi-size   BasicProject.elf
arm-none-eabi-objdump -h -S  BasicProject.elf  > "BasicProject.list"
   text    data     bss     dec     hex filename
    788       8    1568    2364     93c BasicProject.elf
Finished building: default.size.stdout

Finished building: BasicProject.list
```

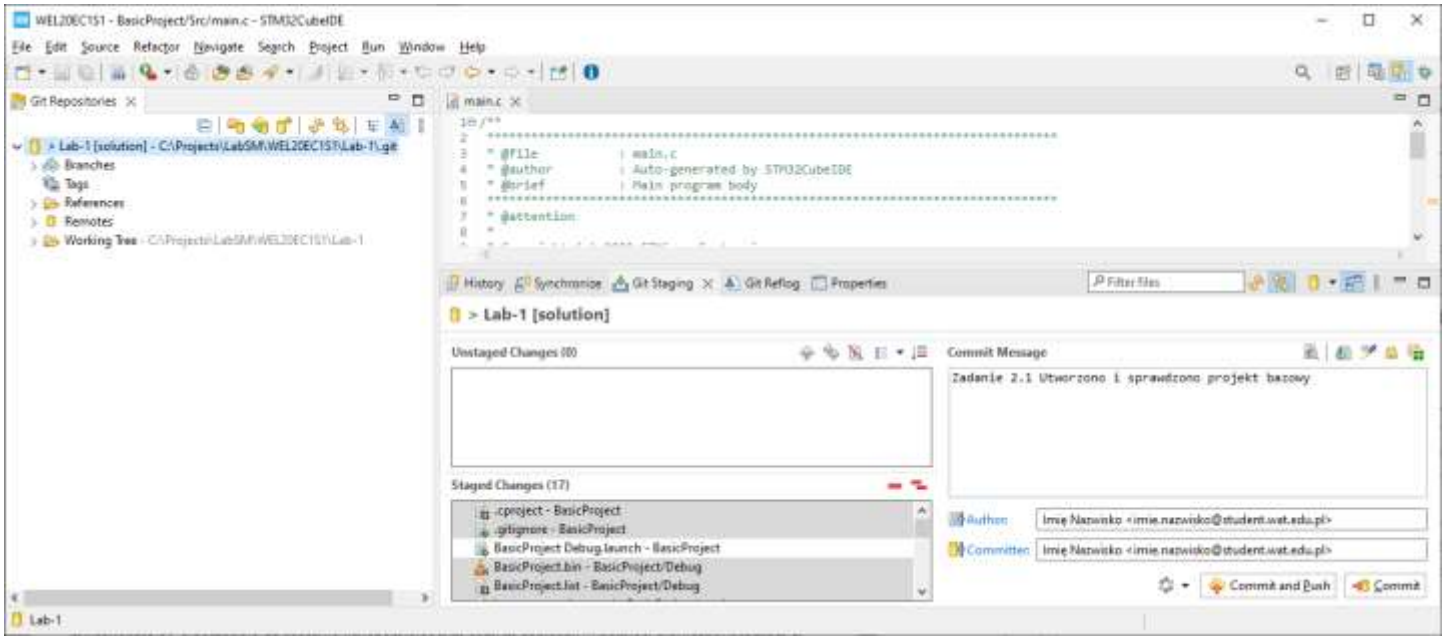
Jeżeli wszystko zostało wykonane prawidłowo to skompilowany projekt można włączyć płytkę i wgrać program wywołując polecenie w menu ***Run → Debug As → STM Cortex-M C/C++ Application***. Pojawi się okno *Edit launch configuration properties* gdzie można zmienić ewentualnie: plik programujący, parametry połączenia programatorem, dodać opcjonalne elementy do umieszczenia w pamięci (np. grafikę). Naciśnięcie przycisku ***Ok*** powoduje inicjalizację połączenia z programatorem. Jeżeli podłączona i włączona jest płyta uruchomieniowa to po uzyskaniu komunikacji z programatorem następuje wgranie programu do pamięci i przełączenie do widoku perspektywy *Debug*. Może pojawić się pytanie czy wyrażamy zgodę na zmianę perspektywy. Wskazane jest aby zaznaczyć aby środowisko nie pytało więcej i wyrazić zgodę na zmianę widoku.

Program może być wgrany i uruchamiany również za pomocą polecenia *Run→ Run*, które programuje mikrokontroler, a następnie go resetuje uruchamiając tym samym zapisany w pamięci nieulotnej program.

Pojawi się po kompilacji ostrzeżenie o braku konfiguracji bloku jednostki zmiennoprzecinkowej (FPU), który na tym etapie ćwiczeń nie będzie używany i może być usunięty (skasować linie 20--22) z wygenerowanego kodu lub w konfiguracji projektu wyłączony. Jednakże w pliku *system_stm32l4xx.c* znajduje się funkcja *SystemInit*, która konfiguruje jednostkę FPU. Ponadto funkcja ta ustawia domyślne wartości dla podsystemu zarządzania zegarami mikrokontrolera i ustawia położenie początku wektora przerwań.

2.1.5. Wykonanie pierwszej migawki z przygotowanym projektem

Jeżeli udało się uruchomić przygotowany szablon projektu należy wykonać pierwszą migawkę projektu. W tym celu należy przełączyć się do widoku perspektywy ***Git***. W dolnej części ekranu znajduje się szereg zakładek spośród których należy wybrać ***Git Staging***. W części *Unstagged Changes* znajdują się pliki oczekujące na dołączenie do migawki. Aby dodać je do plików oczekujących (*Staged Changes*) należy nacisnąć ikonę podwójnego znaku **+** lub zaznaczyć wszystkie pliki i z menu kontekstowego wybrać ***Add to Index***. Po prawej stronie znajduje się okno pozwalające na wpisanie komentarza do migawki (*Commit Message*). Komentarz powinien być zwięzły i wskazywać na wprowadzone zmiany. Z racji, że wszystkie pliki, dla których była wykonana migawka są opatrzone tym samym komentarzem nie ma potrzeby wskazywać gdzie były wykonane zmiany. W tym miejscu należy wpisać w polu komentarza „***Zadanie 2.1 Utworzono i sprawdzono projekt bazowy***”, a następnie zatwierdzić przyciskiem ***Commit***, który zapisze migawkę. Pamiętać należy aby na zakończenie zajęć wypchnąć wszystkie zmiany na serwer za pomocą skryptu wiersza poleceń (*Push HEAD…*).



2.2. Tworzenie programów dla mikrokontrolera z użyciem rejestrów (6 pkt.)

Jedną z podstawowych funkcjonalności mikrokontrolera jest możliwość sterowania wyprowadzeniami lub odczytanie stanu logicznego na nich. Mikrokontrolery STM32 w zależności od rodziny, a przede wszystkim od obudowy i liczby fizycznie dostępnych wyprowadzeń posiadają różną liczbę *portów wejścia/wyjścia ogólnego przeznaczenia* (ang. *general purpose I/O*). Użyty w konstrukcji płyty uruchomieniowej mikrokontroler

STM32L496ZGT6 posiada obudowę LQFP144, która udostępnia 115 wejść/wyjść (I/O), które są pogrupowane po 16 I/O w bloki od *GPIOA* do *GPIOH*. Zdecydowana większość z nich może pełnić funkcję alternatywną, tzn. przekierowane są do dostępnych w mikrokontrolerze licznych peryferii. Domyślenie wszystkie wyprowadzenia mikrokontrolera ustawione są jako niepodłączone pływające (ang. *floating*) wejścia analogowe. Analogicznie jak dla platformy *Arduino* należy zmienić tryb pracy wyprowadzenia przez modyfikację odpowiedniego rejestru mikrokontrolera. W przykładu użytego układu rejestr ten ma nazwę ***MODER***, gdzie na dwóch bitach wybierany jest tryb pracy. Aby ustawić tryb cyfrowego wyjścia należy wpisać wartość ***1***. Stan logiczny jaki ma być ustawiony podać można na kilka sposobów. Podstawowym jest zapis do rejestru ***ODR*** na odpowiedniej pozycji bitowej. Alternatywnie można użyć rejestru ***BSRR***, w którym 16 młodszych bitów służy do ustawienie stanu wysokiego, a 16 bitów bardziej znaczących do ustawienia zera logicznego. Odczyt stanu na wejściu możliwy jest przez rejestr ***IDR***.

W odróżnieniu od tworzenia programów w języku C/C++ dla komputerów w przypadku mikrokontrolerów należy zwrócić uwagę na kwestię rozmiaru poszczególnych typów, np. typ *int* ma typowo rozmiar 32 bitów, jednakże w przypadku np. mikrokontrolera *AVR* zamontowanego stosowanego na płytach rodziny *Arduino Uno* rozmiar ograniczony jest do 16 bitów. W celu uniknięcia nieporozumień i poprawienia przenośności kodu aplikacji stosuje się typy niestandardowe zdefiniowane w pliku nagłówkowym *stdint.h*: *uint8_t*, *uint16_t, uint32_t* (całkowitoliczbowe bez znaku), *int8_t*, *int16_t, int32_t* (całkowitoliczbowe ze znakiem). Nie ma potrzeby jawnego przywoływania tego pliku nagłówkowego dyrektywą preprocesora *#include*, jest to już zrobione w pliku nagłówkowym *stm32l4xx.h*, który powinien być przywołany na początku pliku *main.c*. Używając zmiennych standardowych trzeba jednakże również zachować ostrożność przy przenoszeniu kodu pomiędzy platformami. Ten sam program na różnych mikrokontrolerach może zachowywać się inaczej. W trakcie zajęć preferowane będzie używanie przede wszystkim typów zdefiniowane w pliku nagłówkowym *stdint.h*.

2.2.1. Sterowanie wyjściem cyfrowym - wbudowana dioda LED (2 pkt.)

Pierwsze zadanie do realizacji to napisanie programu do sterowania wyjściem cyfrowym, do którego podłączona jest dioda LED. Jest to jedna podstawowych funkcjonalności jaką może wykonywać mikrokontroler. Na płycie *KAmeLeon* umieszczonych jest 8 diod LED czerwonych i jedna dioda LED RGB (dioda potrójna). Analizując schemat ideowy płyty można odczytać, że diody *LED0* do *LED7* podłączone są odpowiednio do wyprowadzeń portów: ***PC6, PC7, PC8, PC9, PE4, PD3, PE5, PE6***. Natomiast diody LED RGB odpowiednio do wyprowadzeń: ***PD12, PD13, PB8***. W związku z powyższym aby zapalić diodę ***LED0*** należy włączyć taktowanie bloku portu *GPIOC*, zmienić konfigurację portu PC6 do pracy jako cyfrowe wyjście (ang. *push-pull*) i ustawić stan wysoki na wyjściu. Należy zmodyfikować wygenerowany we wcześniejszym etapie kod następująco:

```
20 #include    "stm32l4xx.h"
21
22 int main(void) {
23     // Włączenie zegara dla bloku GPIOC
24     RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
25     // Konfiguracja portu PC6 do pracy jako cyfrowe wyjście
26     GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODE6_Msk) | (1 << GPIO_MODER_MODE6_Pos);
27
28     // Nieskończona petla główna
29     while (1) {
30         GPIOC->ODR |= GPIO_ODR_OD6;
31         GPIOC->ODR &= ~GPIO_ODR_OD6;
32     }
33 }
```

W funkcji ***main*** w linii 24 konfigurowany jest rejestr *AHB2ENR* (*ang. AHB2 peripheral clock enable register*) znajdujący się w bloku *RCC* (*ang. Reset and Clock Control*) za pomocą symbolu *RCC_AHB2ENR_GPIOCEN* ustawia 2 bit rejestru. Powoduje to podanie sygnału zegarowego do bloku sterującego portami wejścia/wyjścia portu C. Przed pierwszym użyciem bloku peryferii mikrokontrolera, tzn. modyfikacji wartości rejestrów do nich przypisanych należy włączyć taktowanie danego bloku. Określanie indywidualne, które komponenty mikrokontrolera są taktowane (ang. *clock gating*) pozwala na dodatkowe obniżenie zużycia energii przez mikrokontroler. Sterowanie zegarami odbywa się przez odpowiednie rejestry grupy *RCC*.

Następnie w celu ustawienia trybu pracy portu ***PC6*** jako wyjścia należy skonfigurować rejestr ***MODER*** znajdującego się w bloku ***GPIOC*** (ang. g*eneral-purpose I/Os*) tak jak podano to w linii 26. W pierwszej kolejności kasowane są bity związane z konfiguracją wyprowadzenia ***PC6*** za pomocą maski w postaci iloczynu logicznego, a następnie za pomocą sumy logicznej ustawiony jest bit zmieniający domyślny tryb pracy z analogowego wejścia (ang. *analog mode*) na wyjście ogólnego przeznaczenia (ang. *general purpose output mode*). W pętli ***while*(1)**, znajduje się polecenie ustawiające stan wysoki na wyjściu ***PC6*** (linia 30) i ustawiające stan niski (linia 31). Odbywa się to przez wykonanie operacji na rejestrze ***ODR*** (ang. *output data register*) ustawienia bitu 6 w celu ustawienia stanu wysokiego wyprowadzeniu ***PC6*** lub skasowaniu w celu ustawienia stanu niskiego. Należy zwrócić uwagę na to, że modyfikacja zawartości rejestrów nie odbywa się przez bezpośrednie przypisanie nowej wartości ale za pomocą operacji *sumy* (**|**) lub *iloczynu* (**&**) nowej wartości z bieżącą. Szczegółowy opis rejestrów można znaleźć w Reference Manuals na stronach 223 (*RCC*) oraz 305 (*GPIO*). Należy skompilować i uruchomić program. Jeżeli wszystko jest w porządku należy wykonać migawkę z komentarzem „***Zadanie 2.2.1a Sterowanie diodą LED0***”.

Jeżeli program jest uruchomiony nie jest widoczny efekt zmiany stanu diody LED. Aby to zmienić potrzebny jest fragment programu, który „zajmie pracą” rdzeń mikrokontrolera. Takim przykładowym kodem może być pętla, która ma się wykonać określoną liczbę razy. W związku z powyższym należy zmodyfikować dotychczasowy kod do postaci jak poniżej, a następnie zweryfikować jego działanie.

```
20 #include    "stm32l4xx.h"
21
22 int main(void) {
23     // Włączenie zegara dla bloku GPIOC
24     RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
25     // Konfiguracja portu PC6 do pracy jako cyfrowe wyjście "push-pull"
26     GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODE6_Msk) | (1 << GPIO_MODER_MODE6_Pos);
27
28     // Nieskończona petla główna
29     while (1) {
30         GPIOC->ODR |= GPIO_ODR_OD6;
31         for (int i = 100000; i > 0; i--);
32         GPIOC->ODR &= ~GPIO_ODR_OD6;
33         for (int i = 100000; i > 0; i--);
34     }
35 }
```

W odróżnieniu od biblioteki *Arduino* na tym etapie nie ma dostępnych funkcji blokujących wykonywanie programu przez określony czas (np. *delay*). Zaproponowane rozwiązanie wprowadza *opóźnienie*, które przy braku optymalizacji w procesie kompilacji wynosi 9 instrukcji na każdą iterację pętli, czyli około 900 000 instrukcji. Należy skompilować i uruchomić program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.2.1b Sterowanie diodą LED0 – opóźnienie programowe***”.

2.2.2. Odczyt wejścia cyfrowego – obsługa przycisku (2 pkt.)

Na płycie *KAmeLeon* umieszczony jest joystick (*SW*), który podłączony jest do wyprowadzeń mikrokontrolera ***PE0*** (*SW_RIGHT*), ***PE1*** (*SW_LEFT*), ***PE2*** (*SW_DOWN*), ***PE3*** (*SW_UP*) oraz ***PE15*** (*SW_OK*). Skierowanie joysticka w określoną stronę lub jego wciśnięcie powoduje przyłączenie odpowiedniego wyprowadzenia mikrokontrolera do masy. W związku z tym mikrokontroler *„widzi”* stan niski na odpowiednich wyprowadzeniach. W celu odczytania stanu danego wejścia należy odczytać rejestr ***IDR*** (ang. *input data register*) znajdującego się w grupie *GPIOE*.

W dotychczasowym programie należy wpierw włączyć zegar dla modułu ***GPIOE*** przez dodanie w linii 24 za pomocą sumy logicznej ***RCC_AHB2_ENR_GPIOEEN*** oraz dodać konfigurację wyprowadzenia do pracy jako wejście przez wyzerowanie odpowiednich bitów rejestru *MODER* (linia 28). Odczyt stanu ***PE15*** odbywa się w wyrażeniu warunkowym (linia 33). Jeżeli iloczyn logiczny jest równy **0** to nastąpi ustawienie stanu wysokiego na wyprowadzeniu ***PC6***. W przeciwnym razie ustawia stan niski na tym wyprowadzeniu.

```
20 #include    "stm32l4xx.h"
21
22 int main(void) {
23     // Właczenie zegara dla bloku GPIOC i GPIOE
24     RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN | RCC_AHB2ENR_GPIOEEN;
25     // Konfiguracja portu PC6 do pracy jako cyfrowe wyjście "push-pull" dla LED0
26     GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODE6_Msk) | (1 << GPIO_MODER_MODE6_Pos);
27     // Konfiguracja portu PE15 do pracy jako cyfrowe wejście dla SW1_OK
28     GPIOE->MODER = (GPIOE->MODER & ~GPIO_MODER_MODE15_Msk);
29
30     // Nieskończona petla główna
31     while (1) {
32         // Odczytanie stanu na PE15
33         if ((GPIOE->IDR & GPIO_IDR_ID15) == 0)
34             GPIOC->ODR |= GPIO_ODR_OD6; // Jeżeli joystick wciśniety -> zapal LED0
35         else
36             GPIOC->ODR &= ~GPIO_ODR_OD6;// Jeżeli joystick puszczony -> zgaś LED0
37     }
38 }
```

Należy skompilować i uruchomić program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.2.2 Odczyt SW1_OK – odczyt stanu wejścia***”.

2.2.3. Odczyt wejścia cyfrowego – obsługa zewnętrznego przerwania (2 pkt.)

Sprawdzanie rejestrów ***IDR*** dla portów *GPIO* przez cykliczny ich odczyt nie jest optymalną metodą obsługi przycisków. Częściej spotykane rozwiązanie bazuje na użyciu mechanizmu przerwań. W tym celu należy skonfigurować moduł ***EXTI*** (ang. *extended interrupts and events controller*), który umożliwia wykrywanie do 41 źródeł przerwań z czego pierwsze 16 podłączone są do modułów *GPIO*. Jego użycie wymaga konfiguracji dodatkowych rejestrów. W pierwszej kolejności należy włączyć zegar dla modułu ***SYSCFG*** (ang. *system configuration controller*) w rejestrze ***APB2ENR*** (ang. *APB2 peripheral clock enable register*) modułu *RCC* (linia 32). Następnie należy wskazać, który port i pin mają posłużyć jako źródło zewnętrznego przerwania przez konfigurację rejestru ***EXTICR*** (ang. *external interrupt configuration register*) modułu ***SYSCFG*** za pomocą symbolu ***SYSCFG_EXTICR4_EXTI15_PE*** (linia 34). W dalszej kolejności konfigurowane są rejestry modułu ***EXTI***. Pierwszy rejestr ***IMR1*** (ang. *interrupt mask register*) włącza źródło przerwania na wejściu 15, natomiast drugi rejestr ***FTSR1*** (ang. *falling trigger selection register*) konfiguruje wykrywanie zbocza opadającego na wejściu 15 (odpowiednio w liniach 36 i 38). Ogólne włączenie przerwań następuję za pomocą makra ***NVIC_EnableIRQ***,

udostępnionego w bibliotece *CMSIS* (ang. *Common Microcontroller Software Interface Standard*), które jako parametr przyjmuje numer przerwania powiązanego ze zdarzeniem (linia 40).

```
20 #include    "stm32l4xx.h"
21 volatile uint8_t set = 0;
22
23 int main(void) {
24     // Włączenie zegara dla bloku GPIOC i GPIOE
25     RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN | RCC_AHB2ENR_GPIOEEN;
26     // Konfiguracja portu PC6 do pracy jako cyfrowe wyjście "push-pull" dla LED0
27     GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODE6_Msk) | (1 << GPIO_MODER_MODE6_Pos);
28     // Konfiguracja portu PE15 do pracy jako cyfrowe wejście dla SW1_OK
29     GPIOE->MODER = (GPIOE->MODER & ~GPIO_MODER_MODE15_Msk);
30
31     // Włączenie zegara dla bloku SysCfg
32     RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
33     // Wybranie wyprowadzenia z portu PE
34     SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI15_PE;
35     // Włączenie źródła przerwania z EXTI15
36     EXTI->IMR1 |= EXTI_IMR1_IM15;
37     // Wykrywanie opadającego zdarzenia z wyprowadzenia PE15
38     EXTI->FTSR1 |= EXTI_FTSR1_FT15;
39     // Włączenie przerwania w rdzeniu ARM Cortex-M4 mikrokontrolera
40     NVIC_EnableIRQ(EXTI15_10_IRQn);
41
42     // Nieskończona petla główna
43     while (1) {
44         if(set == 1){
45             for(int i = 100000; i > 0; i--);
46             GPIOC->ODR &= ~GPIO_ODR_OD6;// Gaszenie LED0 co opóźnienie
47             set = 0;
48         }
49     }
50 }
51
52 void EXTI15_10_IRQHandler(void){
53     if((EXTI->PR1 & EXTI_PR1_PIF15) != 0){
54         EXTI->PR1 |= EXTI_PR1_PIF15;
55         GPIOC->ODR |= GPIO_ODR_OD6; // Zapal LED0
56         set = 1;
57     }
58     NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
59 }
```

Obsługa przerwania realizowana jest w funkcji ***EXTI15_10_IRQHandler***, która w powyższym programie będzie wykonywana po każdorazowym wykryciu zmiany stanu wejścia ***PE15*** z wysokiego na niski. Z racji, że dla wyprowadzeń o numerach od 10 do 15 jest jeden wektor przerwań wpierw należy zidentyfikować źródło przerwania za pomocą wyrażenia warunkowego w linii 53 za pomocą rejestru ***PR1*** (ang. *pending register*). Jeżeli bit na pozycji pod symbolem ***EXTI_PR1_PIF15*** jest w stanie wysokim to należy wykonać operację jego kasowania przez wpisanie do rejestru (linia 54). W kolejnej linii następuje ustawienie stanu wysokiego na ***PC6*** w celu zapalenia diody ***LED0*** (linia 55). Na zakończenie funkcji obsługi przerwania wywołane jest makro ***NVIC_ClearPendingIRQ***, które kasuje zgłoszenie przerwania do rdzenia mikrokontrolera (linia 58). Dodanie zmiennej globalnej *set* umożliwia „komunikację" pomiędzy funkcją ***main***, a funkcją obsługi przerwania. Dzięki czemu zgaszenie diody LED0 następuje po czasie niezbędnym do realizacji pętli *for* (linia 45).

Należy skompilować i uruchomić program w trybie debugera, a następnie poprzez wybranie lewym klawiszem myszy linię 56 ustawić pułapkę przez wybranie pozycji ***Toggle Breakpoint*** (*Ctrl + Shift + B*). W celu ponownego uruchomienia programu należy wybrać ***Run → Resume*** (*F8*). Wciśniecie joysticka powoduje

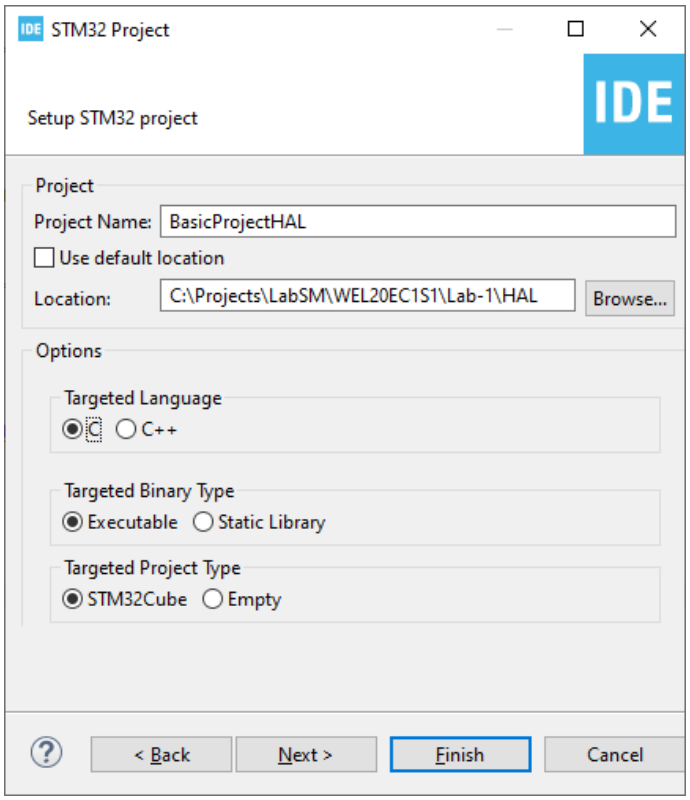
zatrzymanie programu w miejscu postawienia pułapki. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.2.3 Odczyt SW1_OK – obsługa przerwania***”.

2.3. Tworzenie programów dla mikrokontrolera z użyciem biblioteki HAL (6 pkt.)

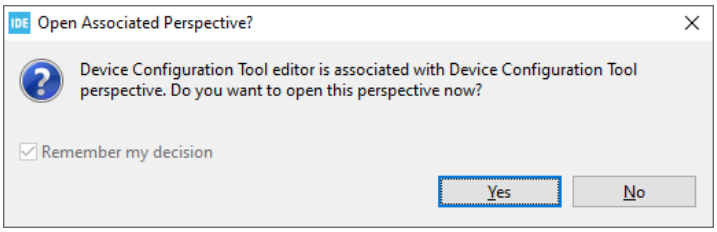
Tworzenie programów w oparciu o bezpośrednie manipulowanie rejestrami mikrokontrolera wymaga od programisty szczególnej uwagi i bardzo dobrej znajomości architektury oraz dokumentacji mikrokontrolera. Programy tak tworzone umożliwiają uzyskanie wydajnych programów zoptymalizowanych pod kątem wydajności i zużywanych zasobów. Jednakże taki sposób programowania jest mocno podatny na błędy, które mogą być trudne w identyfikacji oraz utrudnia przenoszenia programu pomiędzy różnymi mikrokontrolerami. Na potrzeby unifikacji i uproszczenia procesu wytwarzania oprogramowania firma STM przygotowała dla swoich układów szereg bibliotek programistycznych. Pierwszą z nich jest ***SPL*** (ang. *Standard Peripherial Library*), która upraszczała proces korzystania z poszczególnych peryferii i w znaczny sposób odciążała od żmudnego studiowania dokumentacji, jednakże kod programu nie był zbyt przenośny. Kolejnym krokiem było udostępnienie biblioteki ***HAL*** (ang. *Hardware Abstraction Layer*), która w znacznym stopniu poprawiła przenośność kodu oraz umożliwiła konfigurowanie i używanie peryferii przez zbiór funkcji. Jedną z niedogodności biblioteki ***HAL*** jest niekiedy wydajność ograniczona liczbą realizowanych operacji nadmiarowych, które zapewniają jej uniwersalność oraz kontrolę wartości parametrów. W odpowiedzi na potrzebę zmniejszenia narzutu operacji powstała biblioteka ***LL*** (ang. *Low Level*), która jest podzbiorem biblioteki HAL i pozwala na niskopoziomowe operowanie na rejestrach przy zapewnianiu części zalet pełnej biblioteki HAL. Ponadto firma STM udostępniła dedykowane narzędzie *STM32CubeMX* pozwalające na tworzenie projektów wraz ze wstępną konfiguracją peryferii oraz biblioteki sprowadzając tym samym proces tworzenia programu do implementacji logiki działania.

2.3.1. Tworzenie projektu w oparciu o bibliotekę HAL

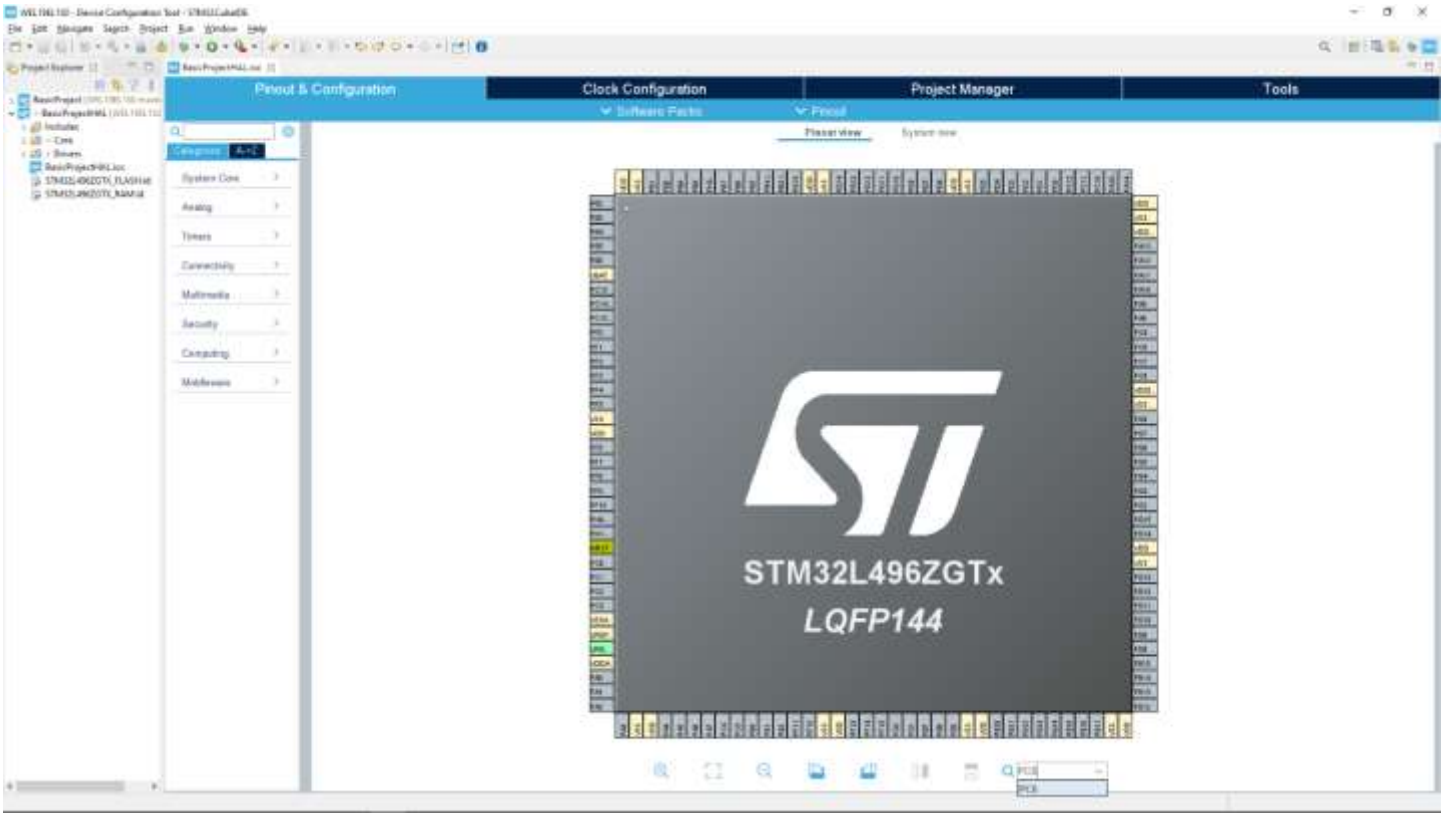
W celu utworzenia nowego pr ojektu należy z wybrać w menu ***File → New → STM32 Project*** co otworzy kreator nowego projektu. W pierwszym oknie *Target selector* w zakładce *MCU/MPU Selector* w polu part numer należy podać oznaczenia mikrokontrolera ***STM32L496ZGT6***, a następnie z listy pasujących MCU wybrać pierwszą pozycję (***STM32L496ZGT6***), a następnie przejść dalej przyciskiem *Next*. W kolejnym oknie (*Setup STM32 project*) należy podać nazwę projektu ***BasicProjectHAL***, w części *Target Project Type* należy upewnić się czy typ projektu to ***STM32Cube***. Następnie należy odznaczyć pole *Use default location* i w polu *Location* podać lokalizację katalogu repozytorium (np.: ***C:\Projects\LabSM\WEL20EC1S1\Lab-1\HAL***). Aby wygenerować projekt należy nacinać przycisk *Finish*. Pojawi się okno *Open Associated Perspective* z



informacją o przełączeniu widoku perspektywy na *Device Configuration Tool*, na co należy wyrazić zgodę przyciskiem *Yes*.



Po chwili załaduje się widok na którym w centralnej części jest przedstawiona obudowa mikrokontrolera z opisanymi wyprowadzeniami. W widoku tym można ustawić konfigurację poszczególnych wyprowadzeń. Za pomocą pola wyszukiwania w prawej dolnej części aplikacji można wyszukać określone wyprowadzenie. Należy znaleźć port ***PC6*** do którego podłączona jest dioda *LED0*. Końcówka zostanie wyróżniona po prawej stronie układu. Po wskazaniu jej należy nacisnąć lewy klawisz myszy i wybrać pozycję ***GPIO_Output***, a następnie naciskając prawy klawisz myszy wybrać *Enter User Label* i w oknie wprowadzić wartość ***LED0***.



Na tym etapie można zapisać wprowadzone zmiany (***File → Save***) co spowoduje pojawienie się pytania czy wygenerować projekt i przełączyć perspektywę. Można zaznaczyć opcję ***Remember my decision*** i zamknąć okno przyciskiem *Yes*. Pojawi się kolejne okno z pytaniem o zmianę okna perspektywy na co należy wyrazić zgodę przyciskiem *Yes*. Nastąpi otworzenie okna umożliwiającego edytowanie kodu znajdującego się w pliku ***main.c***.

Za pomocą komentarzy `/* USER CODE BEGIN … */` oraz `/* USER CODE END … */` wydzielone zostały bloki pomiędzy którymi należy zamieszczać tworzony kod przez programistę. Jest to konieczne jeżeli w przyszłości ponownie będzie uruchamiany konfigurator, który skasuje kod znajdujący się poza wskazanymi sekcjami. W funkcji ***main*** wywoływane są kolejno funkcje: ***HAL_Init*** odpowiedzialna za konfigurację podsystemu przerwań i podstawy czasu; ***SystemClock_Config*** odpowiedzialna za konfigurację systemu zegarowego; ***MX_GPIO_Init*** odpowiedzialna za konfigurację modułu *GPIO* – wyprowadzenia ***PC6***. W dalszej części tego pliku zamieszczony został kod dwóch ostatnich funkcji oraz funkcja ***Error_Handler***, która jest wykonywana jest w przypadku wystąpienia błędów w wyniku działania funkcji biblioteki *HAL*.

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration--------------------------------------------------------*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

Więcej informacji na temat biblioteki można znaleźć w dokumencie *Description of STM32L4/L4+ HAL and low-layer drivers*. W dużym skrócie funkcje biblioteki rozpoczynają się od słowa HAL po czym jest akronim modułu (np. GPIO). Posługując się przytoczoną dokumentacją można zapoznać się API (ang. *application programming interfaces*) dla poszczególnych modułów.

Należy skompilować i uruchomić program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.3.1 Utworzenie projektu z biblioteką HAL***”.

2.3.2. Sterowanie wyjściem cyfrowym - wbudowana dioda LED - HAL (2 pkt.)

Dzięki konfiguratorowi użytemu w procesie tworzenia projektu ustawiony został tryb pracy wyprowadzenia PC6 i wygenerowany stosowny kod zawarty w funkcji ***MX_GPIO_Init***. Zmiana stanu logicznego na wyjściu ***PC6*** może odbyć się za pomocą funkcji ***HAL_GPIO_WritePin*** lub ***HAL_GPIO_TogglePin***. Nadanie etykiety *LED0* w procesie konfigurowania wyprowadzenia umożliwiła generatorowi stworzenie definicji symboli ***LED0_GPIO_Port*** oraz ***LED0_Pin*** zamieszczonych w pliku nagłówkowym ***main.h***.

```
49 int main(void) {
50     /* USER CODE BEGIN 1 */
51     /* USER CODE END 1 */
52     /* MCU Configuration--------------------------------------------------------*/
53     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
54     HAL_Init();
55     /* USER CODE BEGIN Init */
56     /* USER CODE END Init */
57     /* Configure the system clock */
58     SystemClock_Config();
59     /* USER CODE BEGIN SysInit */
60     /* USER CODE END SysInit */
61     /* Initialize all configured peripherals */
62     MX_GPIO_Init();
63     /* USER CODE BEGIN 2 */
64     /* USER CODE END 2 */
65     /* Infinite loop */
66     /* USER CODE BEGIN WHILE */
67     while (1) {
68         HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, GPIO_PIN_SET);
69         HAL_Delay(500);
70         HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
71         HAL_Delay(500);
72         /* USER CODE END WHILE */
73         /* USER CODE BEGIN 3 */
74     }
75     /* USER CODE END 3 */
76 }
```

Do wygenerowanego kodu programu należy wprowadzić fragment kodu zawarty w liniach od 68 do 71 pomiędzy odpowiednie komentarze **`/* USER CODE BEGIN WHILE */`** oraz **`/* USER CODE END WHILE */`**. Wywołanie funkcji ***HAL_GPIO_WritePin*** w linii 68 ustawia wyjście PC6 w stan wysoki, natomiast w linii 70 ustawia wyjście w stan niski. Funkcja ***HAL_Delay*** służy do blokowania wykonywania programu przez określoną liczbę milisekund. Dokładność wprowadzanego opóźnienia wynika z użycia sprzętowego licznika ***SysTick*** do wyznaczania podstawy czasu.

Należy skompilować i uruchomić program. Podczas wykonywania programu można zaobserwować fakt, że część diod LED również się zapaliła bądź zmienia swój stan. Wynika to z faktu, że sterowane są one z jednego układu bufora w postaci układu *74LVC541A*, którego wejścia nie mogą być niepodłączone (tzn. „pływające”). Pomimo faktu, że występuje fizyczne połączenie z wyprowadzeniami mikrokontrolera to domyślna konfiguracji GPIO jest jako analogowe wejście. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.3.2 Sterowanie wyjściem cyfrowym z użyciem biblioteki HAL***”.

2.3.3. Odczyt wejścia cyfrowego – obsługa przycisku (2 pkt.)

W celu dodania obsługi przycisku należy otworzyć konfigurator projektu przez otworzenie pliku ***BasicProjectHAL.ioc***. Nastąpi przełączenie widoku perspektywy gdzie w oknie wyszukiwania należy wpisać ***PE15*** w celu wyszukania wyprowadzenia na widoku obudowy. Końcówka zostanie wyróżniona po prawej stronie układu. Po wskazaniu jej należy nacisnąć lewy klawisz myszy i wybrać pozycję ***GPIO_Input***, a następnie naciskając prawy klawisz myszy wybrać *Enter User Label* i w oknie wprowadzić wartość ***SW1_OK***. Wprowadzone zmiany należy zatwierdzić zapisując wprowadzone zmiany co spowoduje przełączenie widoku do okna edycji kodu.

Do wygenerowanego kodu programu należy wprowadzić fragment kodu zawarty w liniach od 77 do 80 pomiędzy odpowiednie komentarze **/* USER CODE BEGIN WHILE */** oraz **/* USER CODE END WHILE */**. Działanie programu jest analogiczne jak w punkcie 2.2.3, tj. w pętli sprawdzany jest stan wejścia do którego podłączono przycisk. Za pomocą funkcji ***HAL_GPIO_ReadPin*** w wyrażeniu warunkowym odczytywany jest stan wyprowadzenia. Jeżeli jest niski zapalona zostanie dioda *LED0* podłączona do wyprowadzenia ***PC6***. W przeciwnym wypadku zgaszona.

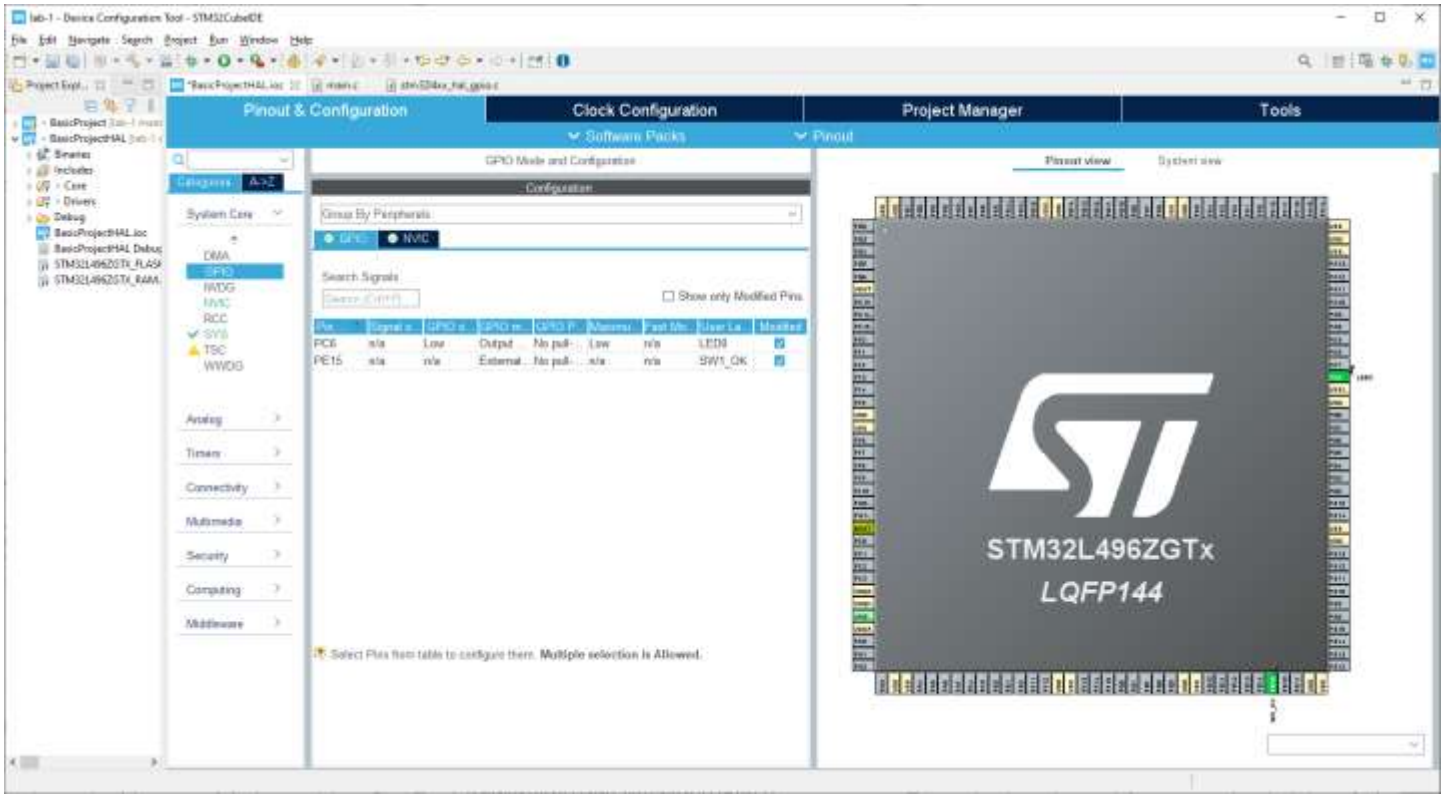
```
58 int main(void) {
59     /* USER CODE BEGIN 1 */
60     /* USER CODE END 1 */
61     /* MCU Configuration--------------------------------------------------------*/
62     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
63     HAL_Init();
64     /* USER CODE BEGIN Init */
65     /* USER CODE END Init */
66     /* Configure the system clock */
67     SystemClock_Config();
68     /* USER CODE BEGIN SysInit */
69     /* USER CODE END SysInit */
70     /* Initialize all configured peripherals */
71     MX_GPIO_Init();
72     /* USER CODE BEGIN 2 */
73     /* USER CODE END 2 */
74     /* Infinite loop */
75     /* USER CODE BEGIN WHILE */
76     while (1) {
77         if (HAL_GPIO_ReadPin(SW1_OK_GPIO_Port, SW1_OK_Pin) == GPIO_PIN_RESET)
78             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, GPIO_PIN_SET);
79         else
80             HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
81         /* USER CODE END WHILE */
82         /* USER CODE BEGIN 3 */
83     }
84     /* USER CODE END 3 */
85 }
```

Należy skompilować i uruchomić program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.3.3 Odczyt SW1_OK – odczyt stanu wejścia z użyciem biblioteki HAL***”.

2.3.4. Odczyt wejścia cyfrowego – obsługa zewnętrznego przerwania – HAL (2 pkt.)

W celu zmiany konfiguracji działania programu aby reagował na przerwanie od wciśnięcia przycisku ***SW1_OK*** należy otworzyć konfigurator projektu przez otworzenie pliku ***BasicProjectHAL.ioc***. Należy zmienić sposób funkcjonowania wyprowadzenia ***PE15*** poprzez kliknięcie lewym klawiszem myszy i zmienienie zaznaczenia

z ***GPIO_Input*** na ***GPIO_EXTI15*** po czym należy ponownie nadać etykietę ***SW1_OK***. Po lewej stronie należy wybrać z listy ***Categories-> System Core -> GPIO***, pojawi się okno ***GPIO Mode and Configuration***, w którym należy wybrać wiersz ***PE15***, poniżej pojawi się konfiguracja wyprowadzenia. Dla pozycji ***GPIO Mode*** należy zmienić na ***External Interrupt Mode with Falling Edge trigger detection***. Następnie należy otworzyć zakładkę ***NVIC*** i zaznaczyć opcję ***Enable*** w wierszu ***EXTI line [15:10] interrupts***.



Po wykonaniu zmian należy je zapisać, co spowoduje odświeżenie dotychczasowego kodu programu i uzupełnienie go o konfigurację przerwania funkcji ***MX_GPIO_Init***. Do pętli ***while*** w funkcji ***main*** należy wprowadzić poniższą modyfikację analogiczną w strukturze do przedstawionej w punkcie 2.2.3. W pierwszej kolejności w sekcji zmiennych prywatnych **/* USER CODE BEGIN PV */** oraz **/* USER CODE END PV */** dodana została deklaracja zmiennej *set*, a następnie w pętli *while* dodane jest wyrażenie warunkowe (linie od 69 do 73), które gasi diodę jeżeli wykryto przerwanie i upłynęło 500 ms. Następnie należy dodać funkcję obsługi przerwania ***HAL_GPIO_EXTI_Callback***, którą można umieścić w sekcji pomiędzy znacznikami **/* USER CODE BEGIN 4 */** oraz **/* USER CODE END 4 */**. W funkcji tej następuję identyfikacja źródła przerwania za pomocą wyrażenia warunkowego i jeżeli pochodzi ono z przycisku ***SW1_OK*** to zapalona zostaje dioda ***LED0*** oraz ustawiona zmienna *set*.

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
volatile uint8_t set = 0;
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration--------------------------------------------------------*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        if (set == 1) {
            HAL_Delay(500);
            HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
            set = 0;
        }
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

```
151 /* USER CODE BEGIN 4 */
152 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
153     if (GPIO_Pin == SW1_OK_Pin) {
154         HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_SET);
155         set = 1;
156     }
157 }
158 /* USER CODE END 4 */
```

Należy skompilować i uruchomić program w trybie debugera, a następnie poprzez wybranie lewym klawiszem myszy linię 155 ustawić pułapkę, a następnie uruchomić program. Wciśniecie joysticka powoduje zatrzymanie programu w miejscu postawienia pułapki. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 2.3.4 Odczyt SW1_OK – obsługa przerwania – biblioteka HAL***”.

3. Zadania rozszerzające do realizacji (14 pkt.)

W rozdziale tym przedstawione zostały zadania dodatkowe, za realizację których można podnieść ocenę końcową za wykonane ćwiczenie. Ich realizację należy wykonać w nowym projekcie o nazwie ***ExtendProjectHAL***. Tworzony kod może bazować na

3.1. Sterowanie diodami LED (7 pkt.)

W konfiguracji projektu należy skonfigurować wyprowadzenia do których podłączone są pozostałe diody LED, tzn. od ***LED1*** do ***LED7***. Należy nadać wyprowadzeniom etykiety zgodnie z przypisanymi oznaczeniami na schemacie analogicznie jak zostało to zrobione w punkcie 2.3.1.

3.1.1. Opracowanie funkcji do sterowania diodą LED (3 pkt.)

W zadaniu tym należy przygotować trzy funkcje: ***LED_On***, ***LED_Off*** oraz ***LED_Toggle***, które odpowiednio włączają, wyłączają i przełączają wybraną diodę LED. Każda z funkcji przyjmuje jeden parametr określający dla której diody LED ma być wykonana zmiana stanu logicznego przyłączonego wyjścia mikrokontrolera. Funkcja w wyniku swojego działania nie zwraca żadnej wartości. Należy zweryfikować poprawność parametru, tzn. czy nie wskazuje na diodę której nie ma. Do wykonania zadania można użyć instrukcji warunkowej *if*(…){…} *else if*(…){…}, instrukcji wyboru *switch*(…){case …:{…; *break*;}} lub tablic. Dodać przykładowy kod w pętli *while* demonstrujący działanie funkcji.

```
159 /* Zadanie 3.1.1 */
160 void LED_On(uint8_t led){
161 /* Kod odpowiedzialny za ustawienie stanu wysokiego na wyjściu mikrokontrolera */
162 }
163 void LED_Off(uint8_t led){
164 /* Kod odpowiedzialny za ustawienie stanu niskiego na wyjściu mikrokontrolera */
165 }
166 void LED_Toggle(uint8_t led){
167 /* Kod odpowiedzialny za zmiane stanu na wyjściu mikrokontrolera */
168 }
```

Należy skompilować i przetestować program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 3.1.1 Funkcje do sterowania wybraną diodą LED***”.

3.1.2. Opracowanie funkcji do sterowania grupą diod LED (4 pkt.)

W zadaniu tym należy przygotować funkcje: ***LEDs_SetValue*** oraz ***LEDs_GetValue***. Pierwsza z nich pozwoli na reprezentację za pomocą zestawu diod LED ośmiobitowej wartości w postaci binarnej, tak że dioda LED7 odpowiada najbardziej znaczącemu bitowi, a LED0 najmniej znaczącemu bitowi. Druga funkcja pozwoli na zwrotne odczytanie stanu grupy diod LED do postaci ośmiobitowej wartości, tak że dioda LED7 odpowiada najbardziej znaczącemu bitowi, a LED0 najmniej znaczącemu bitowi. Do wykonania zadania mogą być potrzebne

```
170 /* Zadanie 3.1.2 */
171 void LEDs_SetValue(uint8_t led){
172 /* Kod odpowiedzialny za ustawienie odpowiednich stanów na wyjściach mikrokontrolera */
173 }
174
175 uint8_t LEDs_GetValue(void){
176 /* Kod odpowiedzialny za odczyt stanów odpowiednich wyjść mikrokontrolera  */
177     return 0;
178 }
```

konstrukcje pętli, wyrażenia warunkowego, instrukcja przesunięcia bitowego. Dodać przykładowy kod w pętli *while* demonstrujący działanie funkcji.

Należy skompilować i przetestować program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 3.1.2 Funkcje do sterowania diodami LED***”.

3.2. Obsługa joysticka (7 pkt.)

W konfiguracji projektu należy skonfigurować wyprowadzenia do których podłączone zostały pozostałe wyprowadzenia joystick (tj. PE0, PE1, PE2, PE3). Należy nadać wyprowadzeniom etykiety zgodnie z przypisanymi oznaczeniami na schemacie analogicznie jak zostało to zrobione w punkcie 2.3.4.

3.2.1. Sygnalizacja położeń joysticka z pomocą diod LED (3 pkt.)

W oparciu o kod przedstawiony w podpunkcie 2.3.3 należy wykryć pozostałe stany joysticka i sygnalizować je za pomocą diod LED następująco: *SW1_RIGHT* zapala *LED0, SW1_LEFT* zapala *LED7, SW1_UP* zapala *LED3* i *LED4*, *SW1_DOWN* zapala *LED1* i *LED6*, a *SW1_OK* zapala wszystkie diody LED. Powrót joystick do stanu neutralnego powoduje wygaszenie wszystkich diod LED. Dodać przykładowy kod w pętli *while* demonstrujący działanie funkcji.

Należy skompilować i przetestować program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 3.2.1 Sygnalizacja położeń joysticka z pomocą diod LED***”.

3.2.2. Sygnalizacja położeń joysticka z pomocą diod LED z obsługą przerwania (4 pkt.)

W oparciu o kod przedstawiony w podpunkcie 2.3.4 należy wykryć pozostałe stany joysticka i sygnalizować je za pomocą diod LED następująco: *SW1_RIGHT* zapala *LED0, SW1_LEFT* zapala *LED7, SW1_UP* zapala *LED3* i *LED4*, *SW1_DOWN* zapala *LED1* i *LED6*, a *SW1_OK* zapala wszystkie diody LED. Odpowiednie diody LED zapalane są na czas 500 ms. Pamiętać należy o zmianie konfiguracji wyprowadzeń do pracy w trybie *GPIO_EXTI*. Dodać przykładowy kod w pętli *while* demonstrujący działanie funkcji.

Należy skompilować i przetestować program. Jeżeli działa poprawnie zgłosić prowadzącemu zajęcia do zatwierdzenia. Na zakończenie punktu proszę wykonać migawkę z komentarzem następującej treści: „***Zadanie 3.2.2 Sygnalizacja położeń joysticka z pomocą diod LED z obsługą przerwania***”.