Systemy Wbudowane

Laboratorium 1:

Zapoznanie się ze zintegrowanym środowiskiem programistycznym (IDE) na przykładzie obsługi GPIO

1.1 Rozpoczęcie pracy z IDE

1.1.1 IDE

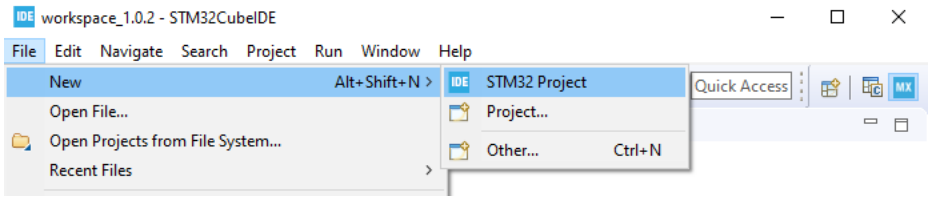
Do programowania mikrokontrolerów STM32 bazujących na procesorach Arm® Cortex®-M można wykorzystać wiele narzędzi oraz gotowych IDE (zintegrowanych środowisk programistycznych - *ang. Integrated Development Environment*). Do przykładowych z nich należą:

- Elicpse™ C++ wraz z nakładką System Workbench for STM32
- Atollic TrueSTUDIO for STM32
- Keil μVision® IDE wraz z dodatkiem MDK Microcontroller Development Kit
- STM32CubeIDE bazujące na frameworku ECLIPSE™/CDT

W ramach laboratoriów będzie wykorzystywane środowisko STM32CubeIDE

1.1.2 Tworzenie projektu

Po uruchomieniu środowiska STM32CubeIDE należy utworzyć nowy projekt typu STM32 Project:

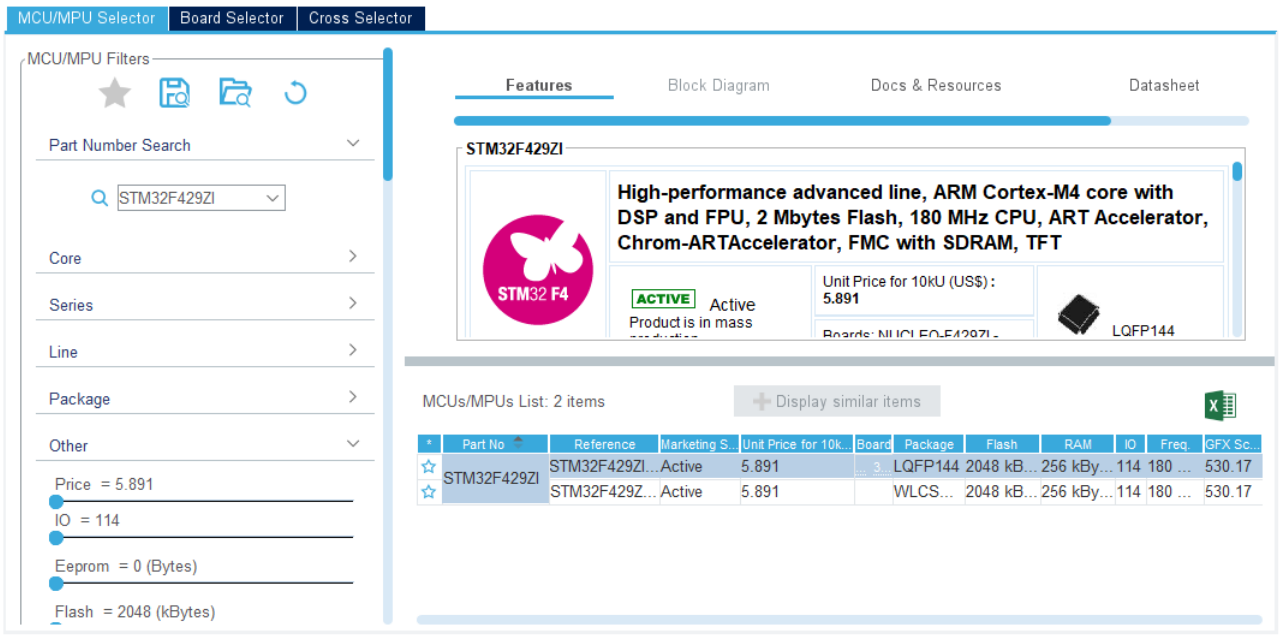


Rysunek 1.1: Tworzenie projektu STM32

Następnie należy odnaleźć na liście posiadaną płytkę rozwoją. Płytka rozwojowa to płytka drukowana (PCB - *ang. printed circuit board*) zawierająca mikroprocesor oraz podstawowe elementy pozwalające na zapoznanie się z nim (np. elementy zasilania, oscylatory, diody, przyciski, programatory). W trakcie laboratoriów będą używane m.in. płytki STM32F429ZI (aka: STM32F429I-DISC1, STM32F429IDISCOVERY, 32F429IDISCOVERY). Płytkę można wyszukać na kilka sposobów:

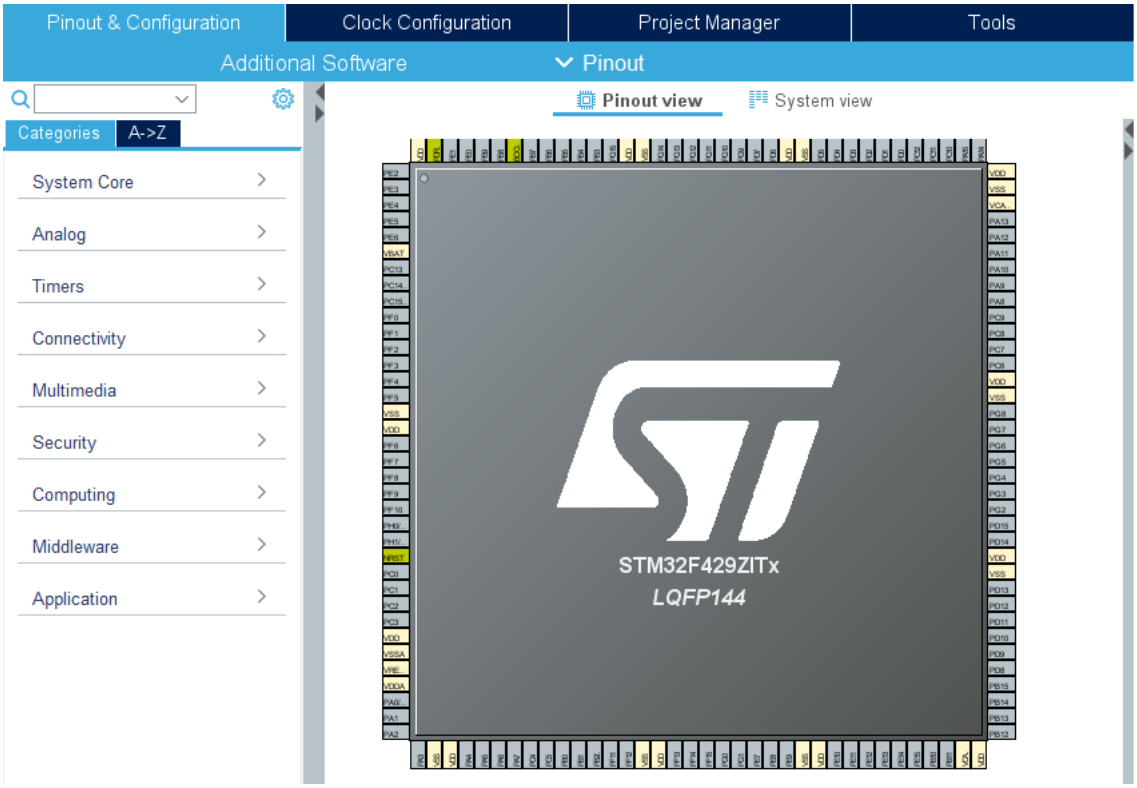
Rysunek 1.2: Dostępne sposoby wyboru płytki rozwojowej

Wyszukanie płytki przez Board Selector skutkuje domyślną konfiguracją jej peryferiów, nawet tych z których się nie korzysta (co początkowo utrudnia analizę kodu programu i jest często uznawane za nadmiarowe). Należy zatem wyszukać płytkę przez MCU/MPU Selector:



Rysunek 1.3: Odnalezienie płytki STM32F429ZI przez wyszukanie jej nazwy

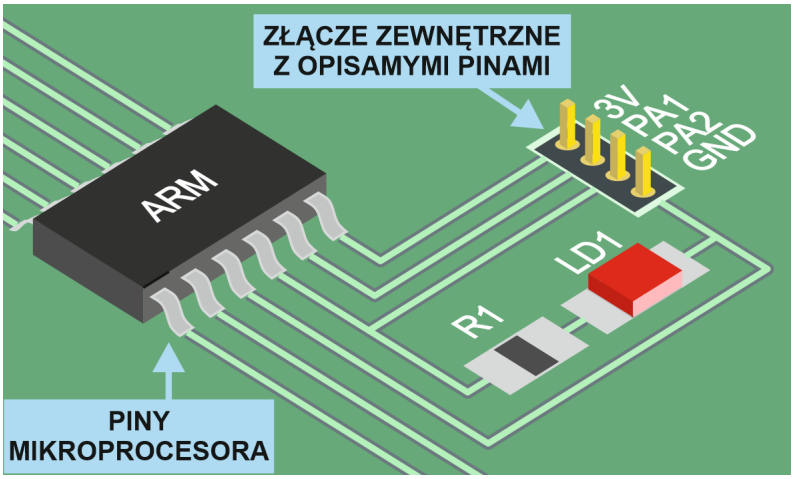
Następnie należy wpisać nazwę projektu (np. LAB1) i można przejść do konfiguracji płytki (przycisk Finish):



Rysunek 1.4: Konfiguracja płytki rozwojowej - STM32CubeMX

1.1.3 Konfiguracja Pinów

Piny są wyprowadzeniami elementów elektronicznych służącymi do wykonywania połączeń elektrycznych. W płytkach rozwojowych piny mikroprocesora są podprowadzone do elementów wbudowanych w płytkę oraz wyprowadzone w złącza zewnętrzne. Na poniższym rysunku można zauważyć że jeden pin procesora jest podpięty zarówno do diody LED (oznaczenie LD1), jak i wyprowadzony do złącza zewnętrznego (PA2):

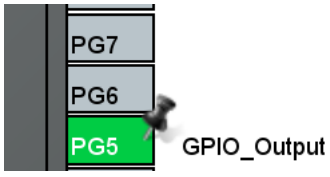


Rysunek 1.5: Mikroprocesor i wyprowadzenia pinów

W płytkach rozwojowych piny mogą pełnić różnorakie funkcje, które zależą od ich konfiguracji (modyfikacji odpowiednich rejestrów mikroprocesora). Interfejs STM32CubeMX pozwala na bardzo prostą konfigurację poszczególnych pinów - po kliknięciu na dany pin pojawia się lista wyboru jego konfiguracji:

Rysunek 1.6: Konfiguracja pinu PG5 - STM32CubeMX

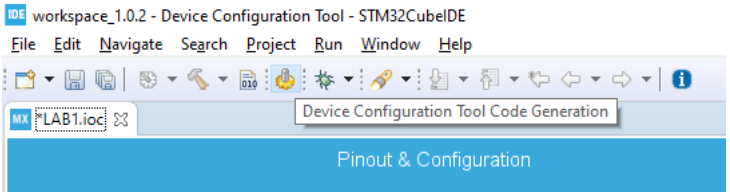
Poprawnie skonfigurowany pin zostaje oznaczony kolorem zielonym (kolor żółty oznacza piny związane z układem zasilania, kolor jasnozielony piny specjalne):



Rysunek 1.7: Skonfigurowany pin PG5

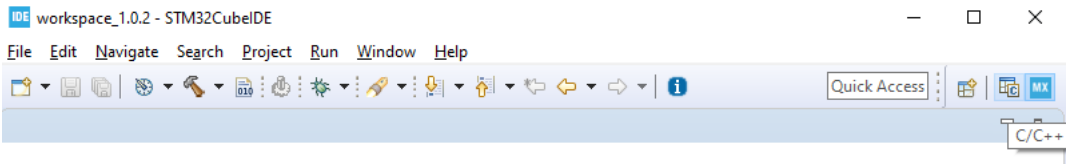
1.1.4 Struktura projektu

Po konfiguracji pinów oraz innych elementów mikrokontrolera wbudowany interfejs narzędzia STM32CubeMX wygeneruje kod źródłowy projektu. Aby tego dokonać należy z konfiguratora wywołać skrót Ctrl+S, Alt+K lub kliknąć następującą ikonę:



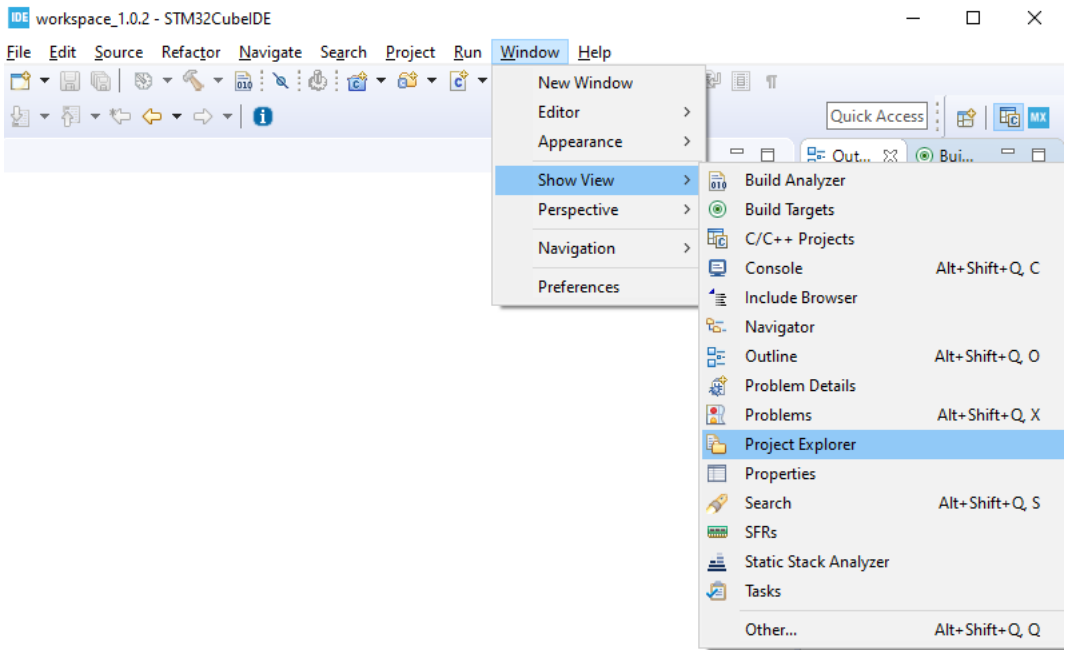
Rysunek 1.8: Generowanie kodu źródłowego

Po tej czynności można powinien otworzyć się automatycznie wygenerowany projekt. Jeżeli to się nie stanie, należy przełączyć się na perspektywę C/C++ klikając odpowiednią ikonę w prawym górnym rogu programu:



Rysunek 1.9: Generowanie kodu źródłowego

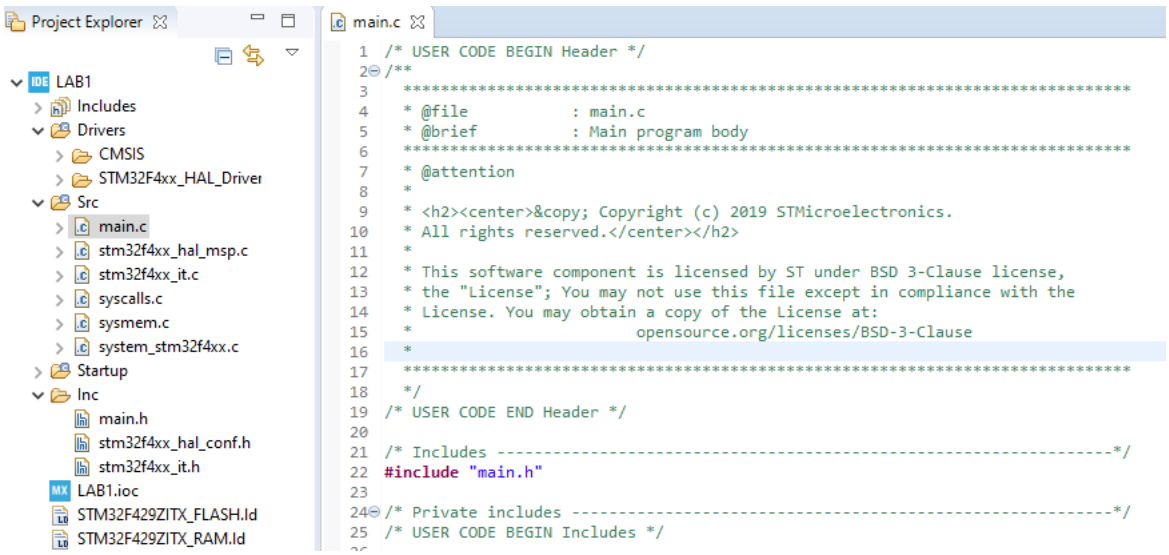
Opcjonalne może być także włączenie widoku struktury projektu:



Rysunek 1.10: Włączenie widoku struktury projektu

W strukturze projektu podstawowe pliki związane z ćwiczeniami z laboratorium to:

- main.c (z podkatalogu Src) - główny kod programu
- main.h (z podkatalogu Inc) - główny kod programu (plik nagłówkowy)
- NAZWA.ioc (np. LAB1.ioc) - powrót do konfiguracji mikrokontrolera



Rysunek 1.11: Wygenerowana struktura projektu

1.1.5 Pisanie kodu źródłowego

Często zdarza się sytuacja w której podczas pisania programu zachodzi potrzeba dodatkowej konfiguracji mikrokontrolera i przez to wygenerowania nowego kodu źródłowego. Na szczęście narzędzie STM32CubeMX nie nadpisuje napisanego już wcześniej przez użytkownika kodu, a jedynie uaktualnia odpowiednie miejsca związane z konfiguracją mikrokontrolera. Możliwe jest to dzięki wyodrębnieniu w kodzie sekcji kodu użytkownika (**UWAGA**: sekcje te są pozostawiane bez zmian - cała reszta zostaje nadpisana przez wygenerowany kod). Sekcje użytkownika rozpoczynają i kończą frazy USER CODE - pokazuje to następujący przykład:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h" // kod automatycznie wygenerowany
#include "moje_funkcje.h" // kod dopisany w złym miejscu - zostanie usunięty po generowaniu kodu
/* Private includes ----------------------------------------------------------*/

/* USER CODE BEGIN Includes */
#include <stdio.h> // to zostało dopisane i nie zostanie zmienione po ponownym generowaniu kodu
/* USER CODE END Includes */
```

Rysunek 1.12: Sekcje kodu źródłowego

Najważniejsza funkcja pliku main.c (*int main(void)*) w najprostszej wygenerowanej postaci wykonuje:

- Resetowanie urządzeń peryferyjnych i inicjalizację interfejsu Flash oraz zegara systemowego - *HAL_Init();*
- Konfigurację zegara systemowego na podsatwie wygenerowanej procedury - *SystemClock_Config();*
- Inicjalizacji skonfigurowanych przez użytkownika peryferiów - *MX_GPIO_Init();*
- Przechodzi do pętli nieskończonej w której zazwyczaj pisze się główny kod programu - *while(1) ...*

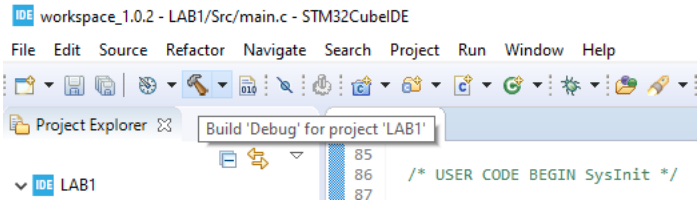
```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  uint8_t var1 = 0;             // utwórz zmienną całkowitą nieujemną 8 bitów
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      HAL_Delay(500);           // zaczekaj 500 ms
      var1++;                   // zwiększ wartość zmiennej o 1
  }
  /* USER CODE END 3 */
}
```

Rysunek 1.13: Przykładowy kod programu

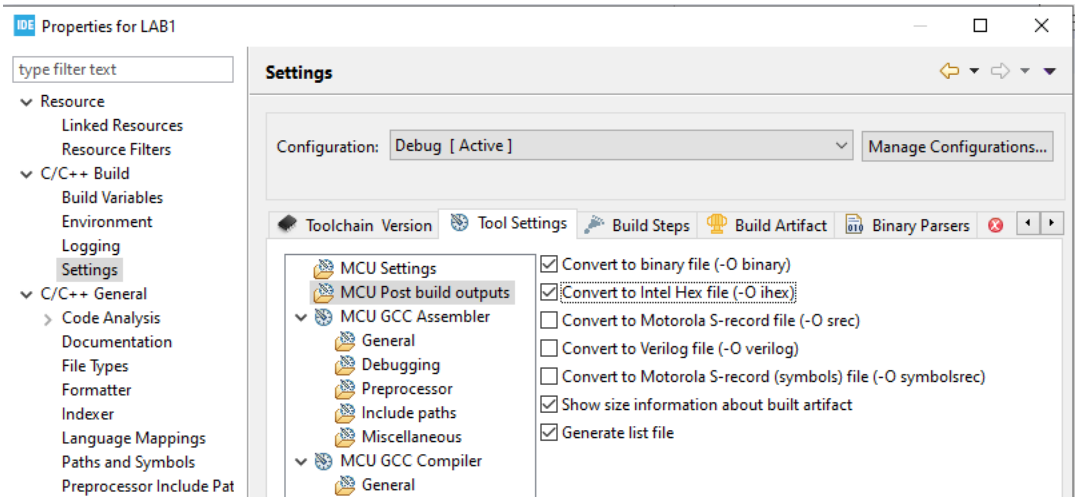
1.1.6 Kompilowanie i wgrywanie kodu

Przygotowany kod można skompilować za pomocą skrótu Ctrl+B (pliki źródłowe należy wcześniej zapisać - Ctrl+S) lub odpowiedniego przycisku:



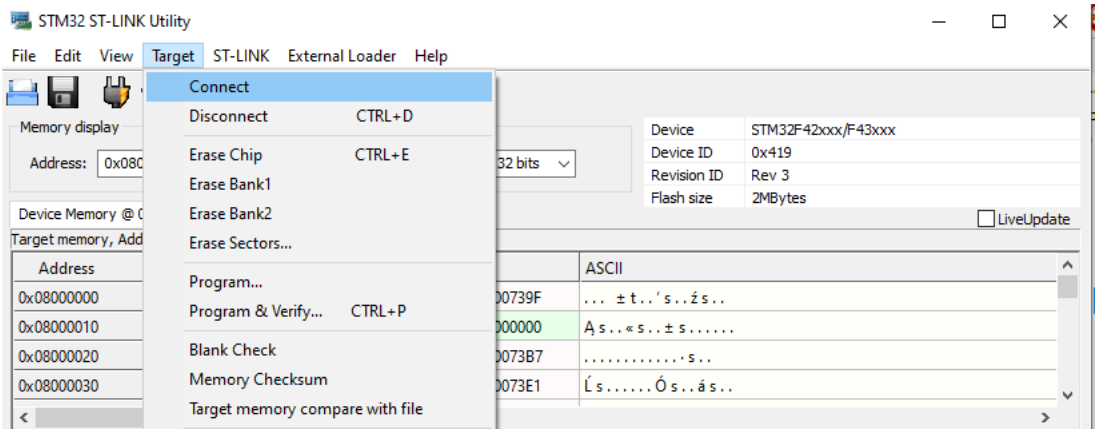
Rysunek 1.14: Kompilacja programu

Po kompilacji zostanie wygenerowany plik .elf (np. LAB1.elf) - plik wykonywalny (*ang. Executable and Linkable Format*). Do wgrania kodu na płytkę potrzebny jest jednak plik binarny (.hex lub .bin), aby uzyskać takie pliki należy odpowiednio skonfigurować projekt zaznaczając w jego właściowościach opcje *Convert to binary file* lub *Convert to Intel Hex file*:



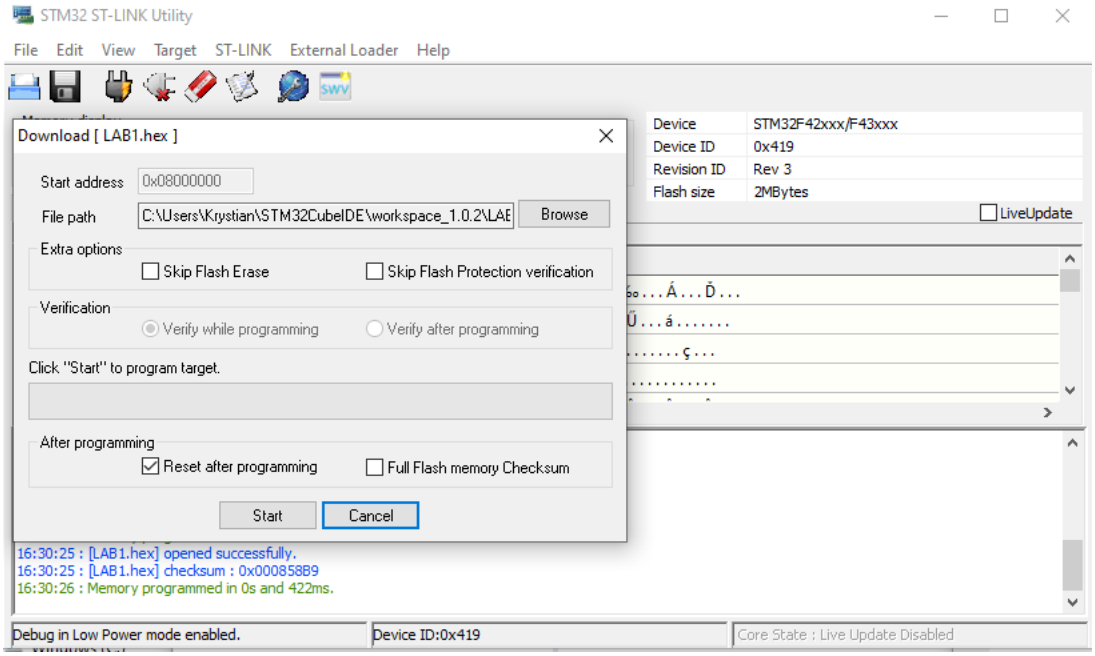
Rysunek 1.15: Konfgiracja umożliwiająca uzyskanie plików binarnych

Po ponownej kompilacji zostaną wygenerowane pliki .hex oraz .bin. Aby wgrać je na płytkę należy najpierw uruchomić narzędzie STM32 ST-LINK Utility i połączyć się z urządzeniem (Target → Connect lub klikając ikonkę czarnej wtyczki):



Rysunek 1.16: Połączenie się z urządzeniem za pomocą STM32 ST-LINK Utility

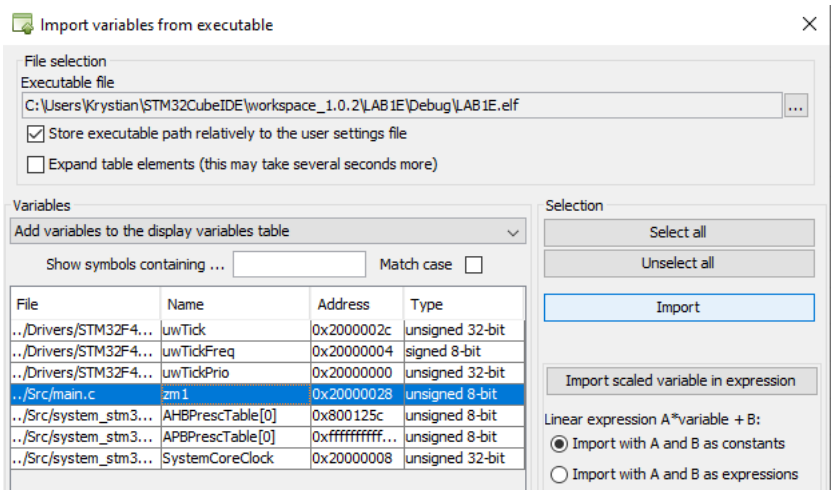
Następnie należy wybrać z opcji Target → Program..., odnaleźć wygenerowany plik .hex (domyślnie znajduje się w katalogu Debug) i kliknąć Start (**UWAGA**: czynność wyboru pliku przed jego wgraniem należy powtórzyć zawsze po kompilacji programu):



Rysunek 1.17: Proces wgrywania programu na płytkę rozwojową

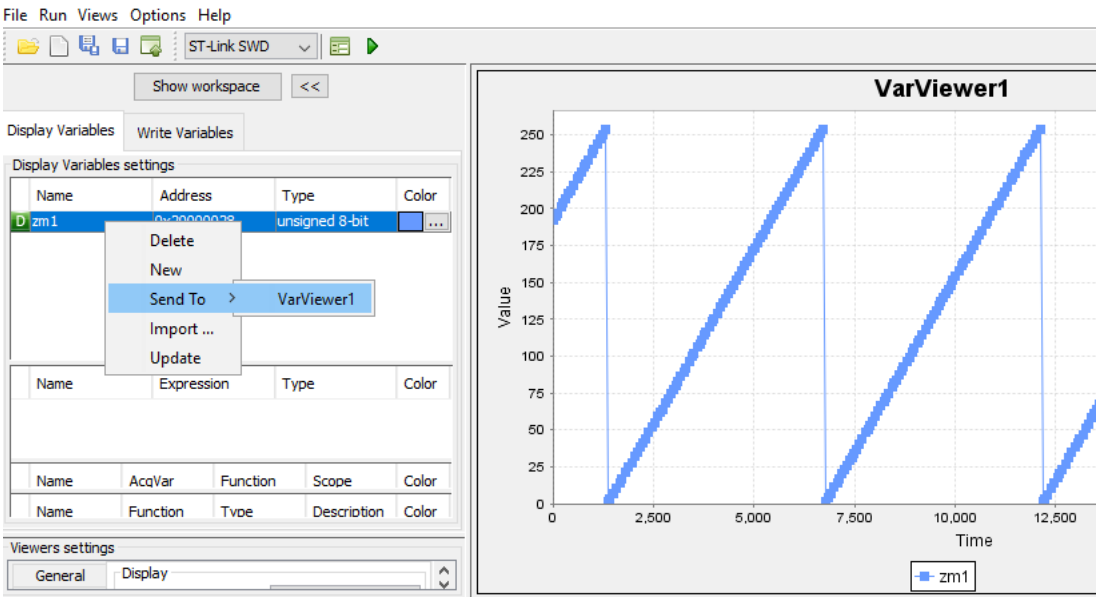
1.1.7 Współpraca z STMStudio

STMStudio pozwala między innymi na podgląd działania programu w czasie rzeczywistym (np. podgląd zmieniających się wartości zmiennych). Po uruchomieniu programu należy zaimportować plik .elf projektu (File → Import Variables) i wybrać interesujące nas zmienne (dostępne są jedynie zmienne o zasięgu globalnym):



Rysunek 1.18: Importowanie zmiennych w STMStudio

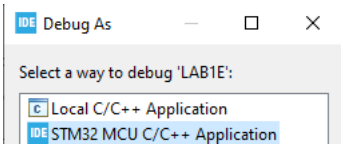
Następnie zmienne te można umieścić na dowolnym wykresie i po uruchomieniu programu (Run → start) rozpocznie się podgląd wartości zmiennych (**UWAGA**: kod programu musi być wcześniej wgrany na płytkę, a narzędzie STM32 ST-LINK Utility powinno być rozłączone):



Rysunek 1.19: Podgląd zmiennej w programie STMStudio

1.1.8 Debugowanie

Środowisko STM32CubeIDE posiada także możliwość debugowania. Aby je uruchomić należy wybrać z menu Run → Debug, lub wcisnąć klawisz F11:

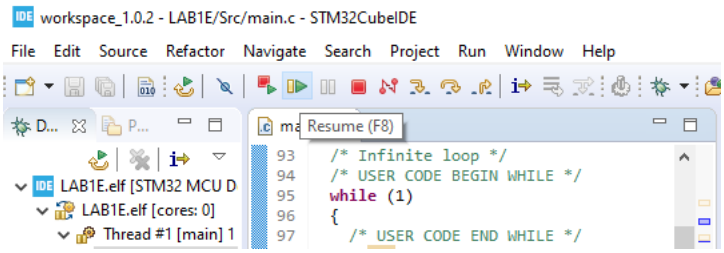


Rysunek 1.20: Rozpoczęcie debugowania

Następnie można dokonać odpowiedniej konfiguracji (domyśla konfiguracja jest poprawna) i uruchomić proces debugowania (program zostanie automatycznie wgrany na płytkę - w tym wypadku nie trzeba korzystać z STM32 ST-LINK Utility). Rozpoczęcie debugowania automatycznie dodaje nową perspektywę do programu:

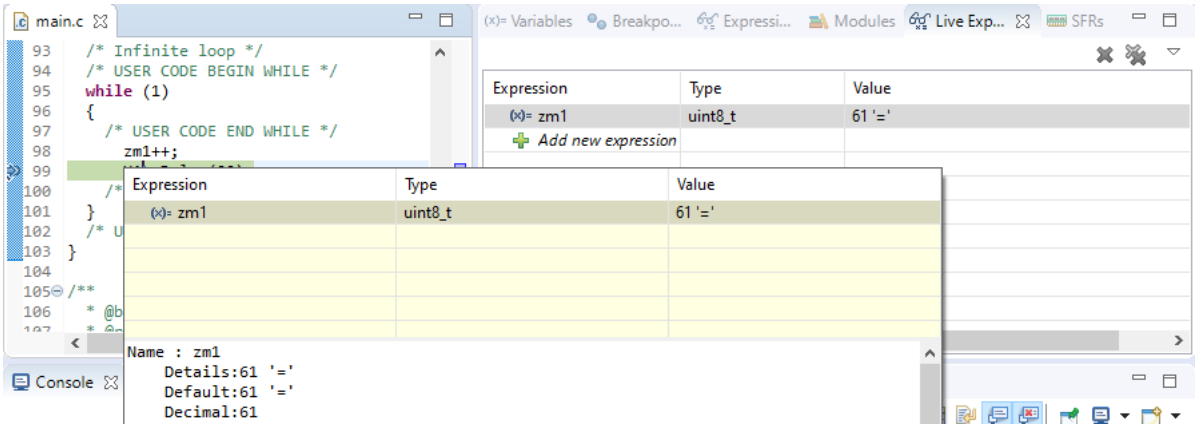
Rysunek 1.21: Perspektywy środowiska z widoczną perspektywą debugowania

Możliwe jest także sterowanie debugowanym programem umożliwiające jego zatrzymanie i ponowne uruchomienie:



Rysunek 1.22: Sterowanie debugowaniem

Debugowanie umożliwia także wstawianie pułapek (punktów przerwania - *ang. breakpoints*) w których automatycznie zatrzymuje się program (dwukrotne kliknięcie po lewej stronie od numeru linii kodu), podglądu zawartości zmiennych (najechanie kursorem na zmienną w momencie zatrzymania programu) i podglądu zmiennych w czasie rzeczywistym (zakładka live expression w trakcie uruchomionego programu):



Rysunek 1.23: Obsługa trybu debugowania

1.2 Ćwiczenie 1

Celem ćwiczenia jest napisanie programu, który będzie naprzemiennie (z odstępem 0.5 sekundy) zapalał i zgaszał diody umieszczone na płytce rozwojowej. W celu realizacji ćwiczenia należy:

- Stworzyć nowy projekt o nazwie LAB1A dla otrzymanej płytki rozwojowej. Tworząc projekt należy wybrać odpowiedni folder roboczy (*ang. Workspace*). Każdy student lub para studentów powinna posiadać własny folder np: C:/Users/student/STM32CubeIDE/StudGr7K). Należy także pamiętać o zapisywaniu utworzonych plików na pendrive lub przesyłanie ich na swój adres e-mail (**UWAGA**: pliki mogą zostać usunięte)
- Odnaleźć na płytce dwie diody oznaczone LD3 i LD4 i odczytać znajdujące się przy nich oznaczenia pinów. Przykładowe oznaczenia pinów na płytkach STM32 to np. PB2, gdzie B oznacza port (piny zazwyczaj są pogrupowane w porty oznaczane kolejnymi literami alfabetu), a wartość 2 oznacza numer pinu (od 0 do 15)
- Skonfigurować odnalezione piny jako GPIO_Output (GPIO - wejście-wyjście ogólnego przeznaczenia *ang. general-purpose input/output*). Taka konfiguracja pozwoli na zmianę stanu wybranych pinów wyjściowych (na niski lub wysoki), a tym samym zapalanie i zgaszanie diod
- Wygenerować kod projektu i zmodyfikować pętle w funkcji main tak, aby zapalała i zgaszała diody.

  Do zmiany stanu pinu służy metoda:
  HAL_GPIO_WritePin(PORT, PIN, STAN);
  Port oznaczane są jako GPIOX (za literę X należy podać oznaczenie portu, np. dla PB2 będzie to GPIOB),
  Piny oznaczane są jako GPIO_PIN_X (za literę X należy podać numer pinu, np. GPIO_PIN_2),
  Stan może przyjąć wartości GPIO_PIN_SET (napięcie) i GPIO_PIN_RESET (brak napięcia),
  Przykład: *HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);*

  Do odczekania określonej długości czasu służy metoda:
  HAL_Delay(LICZBA_MILISEKUND_DO_ODCZEKANIA);
- Skompilować kod programu, wgrać na płytkę i przetestować jego działanie. Należy pamiętać o zaznaczeniu odpowiednich opcji przed kompilacją kodu (Rysunek 1.15)
- Przeanalizować wygenerowany kod programu, a w szczególności metodę *MX_GPIO_Init* (oznaleźć w metodzie konfigurowane piny i zastanowić się nad właściowościami *GPIO_InitStruct.Pull* oraz *GPIO_InitStruct.Speed*)
- Opcjonalnie do zmiany stanu pinów wykorzystać metodę odwracającą ustalony stan:
  HAL_GPIO_TogglePin(PORT, PIN);

1.3 Ćwiczenie 2

Celem ćwiczenia jest napisanie programu, który będzie odczytywał stan przycisku znajdującego się na płytce rozwojowej i zależnie od jego zmiany zapalał lub zgaszał podpiętą do układu diodę. W celu realizacji ćwiczenia należy:

- Zgłosić się do prowadzącego po diodę
- Stworzyć nowy projekt o nazwie LAB1B dla otrzymanej płytki rozwojowej
- Wybrać pin do którego ma zostać podpięta dioda i skonfigurować go jako GPIO_Output
- Skonfigurować pin PA0 jako GPIO_Input (pin podpięty pod niebieski przycisk użytkownika na płytce rozwojowej). Taka konfiguracja pozwoli na odczytanie stanu wybranych pinów, a tym samym odczytanie stanu wciśnięcia przycisku
- Napisać kod programu tak, aby ciągle odczytywał stan przycisku i odpowiednio zmieniał stan diody.

  Do odczytania stanu pinu służy metoda:
  HAL_GPIO_ReadPin(PORT, PIN);
  Metoda ta zwraca wartość GPIO_PIN_SET lub GPIO_PIN_RESET, można ją zatem umieścić bezpośrednio w odpowiednim warunku lub wynik funkcji przypisać do zmiennej typu *GPIO_PinState*
- Skompilować kod programu, wgrać na płytkę i przetestować jego działanie

1.4 Ćwiczenie 3

Celem ćwiczenia jest przetestowanie STMStudio oraz trybu debugowania. W celu realizacji ćwiczenia należy:

- Stworzyć nowy projekt o nazwie LAB1C dla otrzymanej płytki rozwojowej
- Wygenerować kod źródłowy i umieścić w nim dwie zmienne typu uint16_t: zm1 i zm2
- W pętli głównej zwiększać wartość zmiennej zm1 o wartość 1, i zm2 o wartość 2, a następnie uśpić program na 20 milisekund
- Skompilować kod programu, wgrać na płytkę i przerwać z nią połączenie
- Uruchomić STMStudio i zaimportować wygenerowany plik .elf (znajduje się w katalogu Debug). Następnie dodać do podglądu i wykresu dwie utworzone zmienne.
- Uruchomić STMStudio i obserwować zmiany wartości zmiennych
- Zamknąć STMStudio i przetestować tryb debugowania ustawiając różne punkty przerwania

Dla tego laboratorium należy przygotować sprawozdanie zawierające odpowiednie kody źródłowe oraz zdjęcia przedstawiające działanie programów (dla wszystkich ćwiczeń)

Dodatek - Oprogramowanie i narzędzia

Wykorzystywane w ćwiczeniu STMCube32IDE jest narzędziem darmowym (zbudowanym na otwartej platformie Eclipse i licencji CDT, GCC oraz GDB) dostępnym na wiele systemów operacyjnych i można je pobrać tutaj. Również wykorzystane narzędzia STMStudio i STM32 ST-LINK Utility są dostępne za darmo na stronie producenta.

Osoby zainteresowane mogą samodzielnie zakupić płytki rozwojowe, których ceny zaczynają się od 40 zł (np. STM32L100C-Disco) przez 80 zł (np. STM32F411E-Disco) po 160 zł (np. STM32F429I-DISC1) i więcej (np. STM32F746G-Disco). Oczywiście wraz z ceną zwiększają się możliwości oraz zmieniają się komponenty umieszczone na płytce.