Code::Blocks + GCC ARM Embedded Toolchain + OpenOCD + STM32CubeMX + STM32F4 + Discovery + Linux

Tutorial - Code::Blocks, Rev. 0.4

Przemysław Fornalkiewicz

e-mail: przemyslaw.fornalkiewicz@gmail.com

Spis treści

1. Wstęp

Poniższy tutorial dotyczy głównie konfiguracji `Code::Blocks`, umożliwiającej pisanie i debugowanie kodu w języku C/C++, m.in. dla mikrokontrolerów z rodziny `STM32`. Opis zrealizowany został na przykładzie płytki `STM32F4DISCOVERY`. Jednak przedstawioną metodę można wykorzystać do konfiguracji innych układów z tej rodziny. Prezentowany opis zawiera informacje dotyczące konfiguracji: ***Code::Blocks* + *GCC ARM Embedded Toolchain* + *OpenOCD* + *STM32CubeMX* + *STM32F4Discovery* + *Linux***.

Aby możliwe było stworzenie takiego połączenia narzędzi, konieczne jest zainstalowanie kilku pakietów. Instalacja kompilatora i debuggera została dość dobrze opisana w sieci internet, więc ich instalacja nie została szczegółowo przedstawiona. Zamiast tego, wskazane zostały odniesienia do odpowiednich stron internetowych. Podobnie sprawa wygląda z obsługą programu `STM32CubeMX`. Opis konfiguracji `Code::Blocks` bazuje na anglojęzycznej wersji tutoriala http://www.hackvandedam.nl/blog/?p=707. Został rozszerzony o dodatkowe informacje oraz wskazówki, jak go skonfigurować do współpracy z niemal każdym układem z rodziny `STM32`.

Dla układów STM32 jest już dostępne gotowe, zintegrowane środowisko programistyczne: `System Workbench for STM32 (SW4STM32)`, który bazuje na środowisku `Eclipse`. Jaki jest więc sens samodzielnego składania środowiska? Autor nie lubi środowiska `Eclipse`, nie odpowiada mu jego rozmieszczenie okien, obsługa, po prostu źle mu się w tym środowisku pracuje. Jest to wystarczający powód. Jednak, żeby było ciekawiej, samo środowisko `SW4STM32` zostanie wykorzystane do „ściągnięcia” konfiguracji dla kompilatora `GCC`. Dzięki temu konfigurację można przeprowadzić szybko, dla wielu różnych mikrokontrolerów, nie zagłębiając się w tajniki wymaganych przez kompilator parametrów.

Ponieważ strony w internecie lubią czasami „znikać”, wszystkie wykorzystane odnośniki do konfiguracji poszczególnych pakietów, zostały umieszczone w załączniku, aby tutorial nie stał się bezużyteczny na wypadek „zniknięcia”, któreś ze stron. Wszystkie wykorzystane narzędzia są dostępne również dla systemu `Windows`, możliwe jest zatem zastosowanie opisanej konfiguracji również w tym systemie, uwzględniając różnice w sposobie działania systemów obu systemów.

2. Narzędzia wykorzystane do budowanego środowiska programistycznego

Konfiguracja systemu została zrealizowana w systemie operacyjnym ***Linux Mint Cinnamon 17.3 Rosa 64-bit***. Do budowy środowiska programistycznego, wykorzystano najnowsze dostępne w tym czasie wersje oprogramowania, które w czasie prac (kwiecień 2016) oznaczały się następującymi wersjami:

a) `Linux Mint Cinnamon – v17.3 Rosa 64-bit`
b) `GCC ARM Embedded Toolchain – v5_3-2016q1`
c) `OpenOCD – v0.9.0`
d) `STM32CubeMX – v4.14.0`
e) `System Workbench STM32 (SW4STM32) – v1.3`
f) `Code::Blocks – v16.01`

3. Instalacja kompilatora - GCC ARM Embedded Toolchain

Pakiet kompilatora dla mikrokontrolerów z rodziny ARM (w tym STM32) nazywa się ***gcc-arm-none-eabi***. Można wykorzystać pakiet dostępny w repozytorium Linuxa lub zainstalować najnowszą dostępną wersję, ściągniętą ze strony opiekunów pakietu. W poniższym opisie wykorzystano drugi sposób, wybierając najnowszą dostępną wersję pakietu. W chwili tworzenia poniższej konfiguracji była to wersja `gcc-arm-none-eabi-5_3-2016q1`. Niezbędne pliki można pobrać ze strony https://launchpad.net/gcc-arm-embedded/+download. Potrzebna jest wersja oznaczona jako ****-linux.tar.bz2***. Szczegóły instalacji pakietu dostępne są pod adresem http://gnuarmeclipse.github.io/toolchain/install/.

4. Instalacja debuggera sprzętowego - OpenOCD (Open On-Chip Debugger)

Pakiet `OpenOCD` tworzy interfejs pomiędzy sprzętowym debuggerem/programatorem mikrokontrolera, a standardowym debuggerem `GDB`, dostępnym w systemie Linux. Aby rozróżnić debugger `OpenOCD` od debuggera `GDB`, pierwszy nazywany będzie ***sprzętowym***, a drugi ***programowym***. Również w tym przypadku możliwe jest wykorzystanie pakietu dostępnego w systemie, jednak podobnie jak dla kompilatora, zainstalowana została najnowsza dostępna wersja `OpenOCD 0.9.0`, dostępna pod adresem: https://sourceforge.net/projects/openocd/files/openocd/0.9.0/. Potrzebna wersja ma rozszerzenie ****.tar.bz2***. Szczegóły instalacji dostępne są pod adresem http://hertaville.com/stm32f0discovery-part-1-linux.html w sekcji „***Downloading and Installing OpenOCD***”

Podczas instalacji pakietu zgodnie ze wskazanym źródłem, konfigurację pakietu można zakończyć na poleceniu ***sudo make install***. Pozostałe komendy nie są konieczne do poprawnego działania budowanego środowiska. Aby zachować porządek w systemie instalacja pakietu została zmieniona z lokalizacji *`$HOME/EmbeddedArm/openocd-bin`* na lokalizację `/usr/local/` (taka sama jak kompilatora). Należy o tym pamiętać podczas wykonywania polecenia ***`./configure`***.

W wyniku zainstalowania powyższych pakietów według wskazanych źródeł, kompilator `GCC` został zainstalowany w lokalizacji: ***`/usr/local/gcc-arm-none-eabi-5_3-2016q1`*** a debugger sprzętowy `OpenOCD` w lokalizacji: ***`/usr/local/openocd-bin`***. W dalszej części tutoriala, właśnie do tych lokalizacji będzie się odwoływać pakiet `Code::Blocks`.

5. Instalacja STM32CubeMX

Narzędzie `STM32CubeMX` dla Linuxa można pobrać ze strony producenta, pod adresem: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242.

Po ściągnięciu oprogramowania (plik *.zip), znajdując się w katalogu ze ściągniętym plikiem, należy rozpakować archiwum poleceniem:

unzip stm32cubef4.zip

Powyższe polecenie spowoduje rozpakowanie archiwum do folderu, w którym się znajduje. W przypadku gdyby pakiet `unzip` był niedostępny w systemie, można go doinstalować, w tym celu należy wydać kolejno polecenia:

sudo aptitude update
sudo aptitude install unzip

Po rozpakowaniu archiwum można rozpocząć instalację poleceniem (będąc w katalogu z rozpakowanymi plikami):

./SetupSTM32CubeMX-4.14.0.linux

Pojawi się graficzny kreator, który prowadzi użytkownika przez kolejne etapy instalacji pakietu. Jeżeli pakiet ma być zainstalowany w miejscu dostępnym dla wszystkich użytkowników (np. `/usr/local/`) należy instalację zainicjować poleceniem:

sudo ./SetupSTM32CubeMX-4.14.0.linux

Dla pakietu `STM32CubeMX` dostępne są dodatkowe biblioteki dla każdej rodziny mikrokontrolerów STM32. Dla rodziny STM32F4 jest to dodatek `STM32CubeF4`. Instalację dodatkowych bibliotek można przeprowadzić w dwojaki sposób:

a) Należy najpierw zainstalować `STM32CubeMX`, a następnie po jego uruchomieniu, w menu: ***Help -> Install New Libraries,*** wybrać ***Firmware Package for Family STM32F4*** oraz wybrać przycisk ***Install Now***.

b) Można też wcześniej ściągnąć paczkę `STM32F4Cube` (plik *.zip) samodzielnie, ze strony http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1743/PF259243 i w oknie wyboru bibliotek do instalacji wybrać opcję: ***From Local …*** oraz wybrać przycisk ***Open***.

6. Instalacja System Workbench STM32 (opcjonalnie)

Aby ułatwić sobie pracę można skorzystać z narzędzia `System Workbench STM32`, z którego w łatwy i szybki sposób można „skopiować” konfigurację dla kompilatora `GCC`. Należy ściągnąć pakiet w wersji dla Linuxa; bezpośredni link: install_sw4stm32_linux_64bits-latest.run oraz install_sw4stm32_linux_32bits-latest.run. Lub po uprzedniej darmowej rejestracji, ze strony http://www.openstm32.org/Downloading+the+System+Workbench+for+STM32+installer, a następnie w katalogu ze ściągniętym plikiem uruchomić skrypt instalacyjny poleceniem (dla wersji 64-bit):

```
./install_sw4stm32_linux_64bits-latest.run
```

Pojawi się graficzny kreator, który prowadzi użytkownika przez etap instalacji pakietu. Warto tutaj zaznaczyć, aby instalować pakiet w miejscu, do którego nie są wymagane prawa ***root***, gdyż w przeciwnym wypadku program może nie działać prawidłowo dla „standardowego” użytkownika. Jako miejsce instalacji można wybrać np. katalog domowy użytkownika.

7. Instalacja Code::Blocks

Konfigurację Code::Blocks przeprowadzono dla wersji 16.01 pakietu. System operacyjny wykorzystywany podczas konfiguracji środowiska to: Linux Mint Cinnamon 17.3 Rosa 64-bit. Ponieważ jest to system bazujący na Ubuntu 14.04, dla którego twórcy Code::Blocks przewidzieli możliwość instalacji najnowszej wersji oprogramowania wprost z repozytorium, możliwe jest zainstalowanie pakietu za pomocą standardowych poleceń dostępnych z terminala. W tym celu należy dodać do systemu prywatne repozytorium z najnowszą wersją Code::Blocks (standardowo instalowana jest wersja 13.12):

```
sudo add-apt-repository ppa:damien-moore/codeblocks-stable
```

Następnie wykonać standardową procedurę instalacji pakietu, zaczynając od aktualizacji systemu:

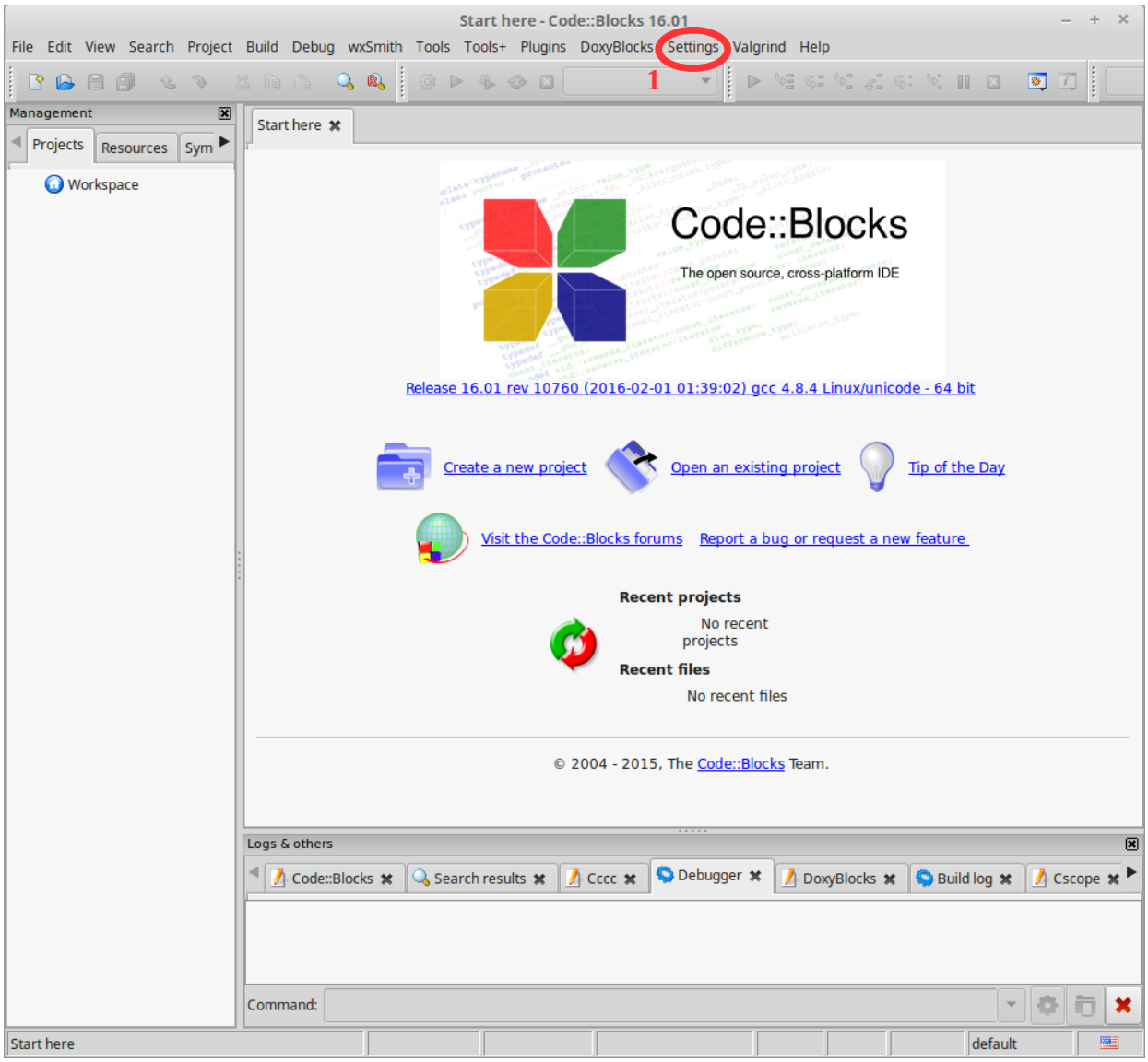
```
sudo aptitude update
sudo aptitude dist-upgrade
sudo aptitude install codeblocks
```

Jeżeli w systemie jest zainstalowana starsza wersja pakietu, wystarczy po dodaniu repozytorium wykonać polecenie update oraz dist-upgrade. Dla innych wersji systemu, dla których nie ma możliwości instalacji pakietu z repozytorium, można ściągnąć wersję instalacyjną, kody źródłowe lub repozytorium SVN ze strony opiekunów pakietu: http://www.codeblocks.org/downloads.

8. Konfiguracja środowiska programistycznego oraz utworzenie nowego projektu

Przepływ informacji w konfigurowanym środowisku jest następujący: ***STM32CubeMX → Code::Blocks → Kompilator GCC → Debugger GDB (programowy debugger)→ OpenOCD (sprzętowy debugger) → STM32F4DISCOVERY (dowolny mikrokontroler z rodziny STM32 z rdzeniem ARM)***.

Konfiguracja narzędzi rozpoczyna się w głównym oknie programu `Code::Blocks` [1], przedstawionym na *rys. 1*.

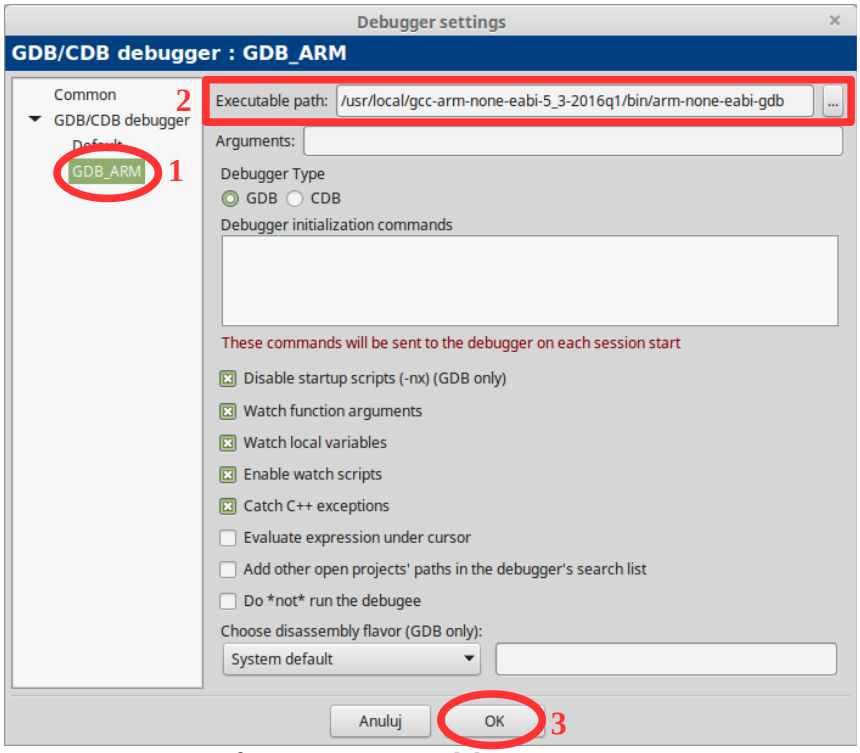


Rys. 1: Okno główne programu Code::Blocks

8.1. Konfiguracja debuggera programowego

Konfiguracja debuggera programowego `GDB`, który wchodzi w skład zainstalowanego kompilatora `GCC` wykonywana jest raz dla całej rodziny mikrokontrolerów z rdzeniem ARM. Zatem jest wspólna dla wszystkich przyszłych projektów z wykorzystaniem rodziny STM32 ARM. Polecenia debugowania generowane przez ten pakiet są następnie przechwytywane przez debugger sprzętowy `OpenOCD` (co zostało opisane w dalszej części tutoriala), który komunikuje się z mikrokontrolerem.

Aby dodać konfigurację debuggera, należy w `Code::Blocks` wybrać menu ***Settings*** → ***Debugger...*** W nowym oknie zaznaczyć ***GDB/CDB debugger*** oraz wybrać ***Create Config***. Następnie w kolejnym oknie należy wpisać nazwę dla debuggera, np. ***GDB_ARM*** i wcisnąć ***OK***. Dla nowo utworzonego debuggera [1] - *rys. 2*, należy podać lokalizację pliku debuggera programowego: ***arm-none-eabi-gdb*** [2]. Plik ten znajduje się w podkatalogu `bin`, w lokalizacji do którego wcześniej został zainstalowany pakiet ***arm-none-eabi***. Pozostałą konfigurację można pozostawić bez zmian. Następnie należy wybrać ***OK*** [3].



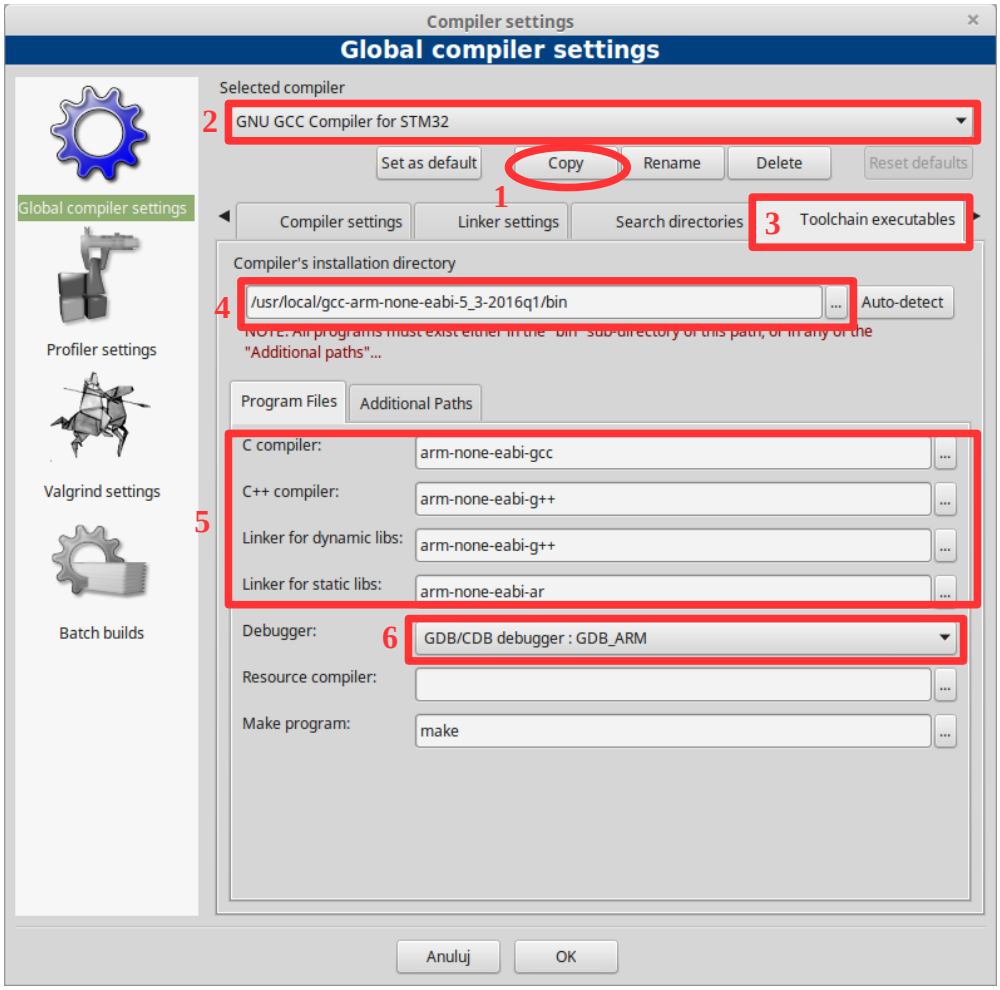
Rys. 2 Konfiguracja nowego debuggera programowego

8.2. Konfiguracja kompilatora GCC

Podobnie jak dla debuggera, konfiguracja kompilatora wykonywana jest tylko raz dla całej rodziny STM32 ARM. Aby rozpocząć konfigurację należy wybrać menu ***Settings → Compiler...*** Pojawi się nowe okno, przedstawione na *rys. 3*. Aby utworzy nową konfigurację dla kompilatora należy wcisnąć przycisk ***Copy*** [1]. W nowo otwartym oknie należy wpisać nazwę dla nowej konfiguracji kompilatora, np. ***GNU GCC Compiler for STM32*** i wcisnąć ***OK***. Następnie z rozwijalnej listy należy wybrać nowo utworzoną konfigurację [2] i przejść do zakładki ***Toolchain executables*** [3]. W oknie ***Compiler's installation directory*** należy wskazać lokalizację zainstalowanego kompilatora [4]. Kolejnym krokiem jest uzupełnienie pól z informacjami jakich kompilatorów program ma używać podczas kompilacji poszczególnych typów plików. Są to odpowiednio [5]:

C compiler: ***arm-none-eabi-gcc***
C++ compiler: ***arm-none-eabi-g++***
Linker for dynamics libs: ***arm-none-eabi-g++*** lub ***arm-none-eabi-gcc***, jeżeli ktoś nie potrzebuje linkowania z plikami C++
Linker for static libs: ***arm-none-eabi-ar***

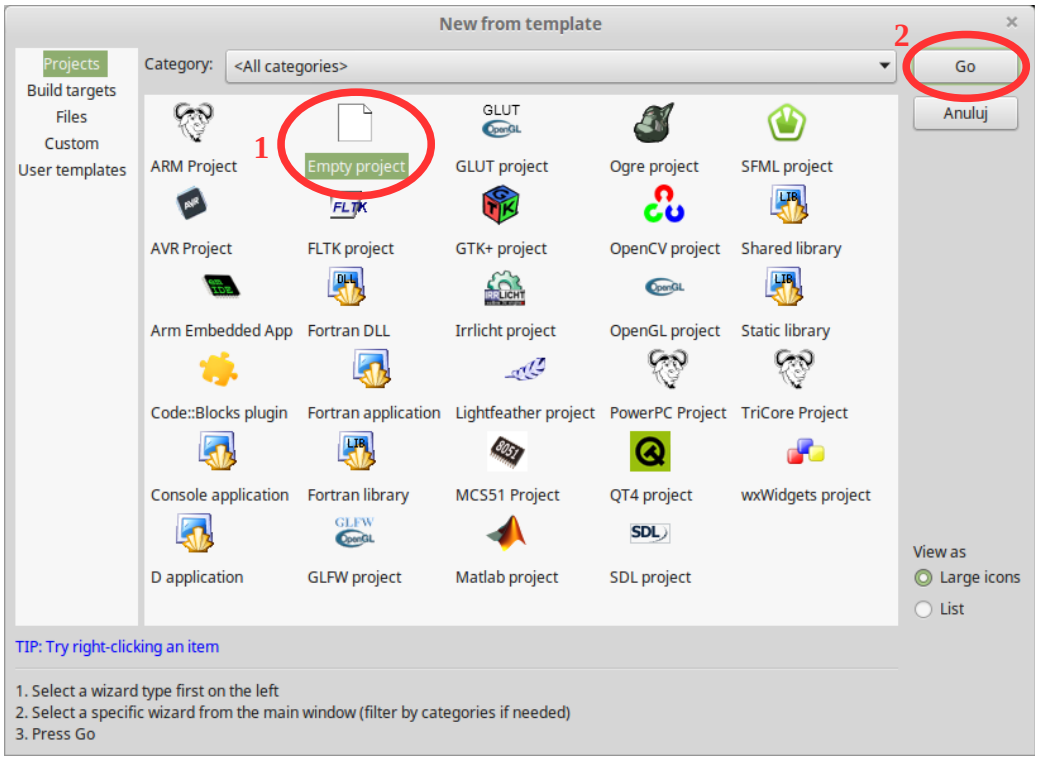
Ostatnim krokiem jest wybór debuggera programowego, współpracującego z zestawem wybranych kompilatorów. W tym celu należy z rozwijalnej listy ***Debugger*** wybrać dodany wcześniej kompilator ***GDB_ARM*** [6].



Rys. 3: Konfiguracja kompilatora GCC

8.3. Tworzenie nowego projektu

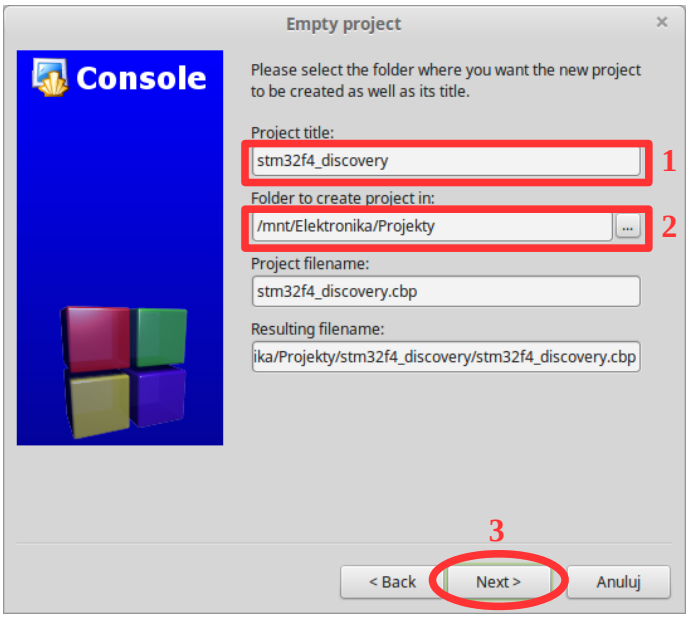
Po podstawowym skonfigurowaniu kompilatora oraz debuggera, można rozpocząć tworzenie nowego projektu. W tym celu w głównym oknie programu należy wybrać: ***`File`*** → ***`New`*** → ***`Project…`***, a nowo otwartym oknie ***`Empty project`*** [1] → ***`GO`*** [2] - *rys. 4*.



Rys. 4: Tworzenie nowego projektu

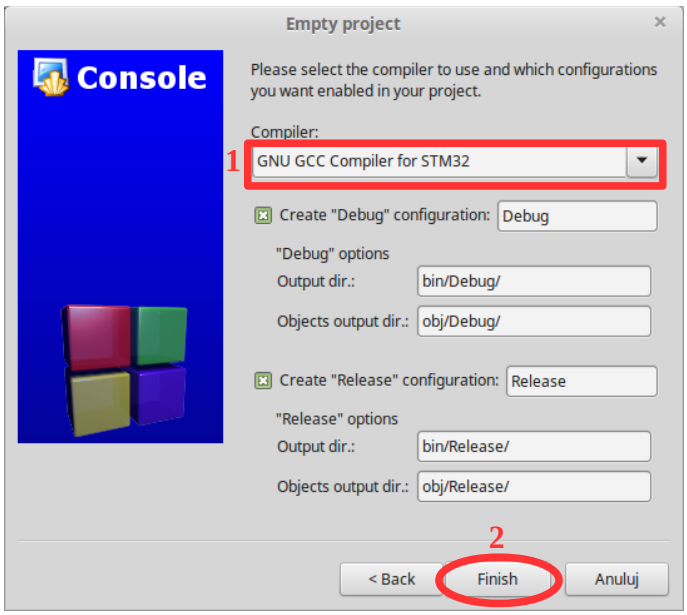
Pojawi się nowe okno kreatora, w którym należy wcisnąć przycisk ***Next***.

W kolejnym oknie kreatora, przedstawionym na *rys. 5* należy wpisać nazwę projektu [1], wybrać ścieżkę zapisu projektu [2] oraz wcisnąć przycisk Next [3].



Rys. 5: Wybór lokalizacji projektu

W ostatnim oknie kreatora - *rys. 6*, należy z rozwijalnej listy wybrać wcześniej skonfigurowany kompilator [1] oraz wcisnąć przycisk ***`Finish`*** [2].



Rys. 6: Wybór kompilatora do projektu

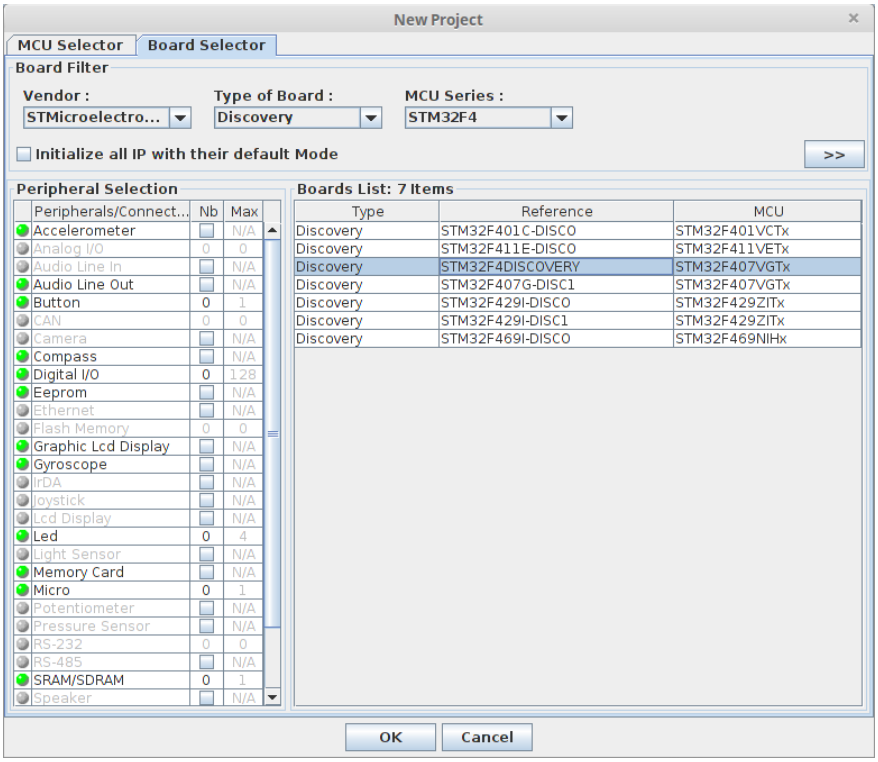
W głównym oknie programu, w lewym panelu (***`Management`***) w przestrzeni roboczej ***`Workspace`*** (zakładka ***`Projects`***) pojawi się nowo utworzony projekt o nazwie ***`stm32f4_discovery`***. Na tym etapie konfiguracji, do projektu można dodawać pliki z kodem źródłowym. Na potrzeby projektu, szkielet kodu źródłowego zostanie wygenerowany w programie `STM32CubeMX`. Należy uruchomić wskazany program i wygenerować odpowiedni kod źródłowy dla nowo powstałego projektu.

9. Generowanie kodu źródłowego w programie STM32CubeMX

Krótki kurs pozwalający zapoznać się z programem `STM32CubeMX` został opisany na stronie http://www.stm32.eu/stm32cube-1 oraz http://www.stm32.eu/stm32cube-2. W ostatniej części kursu http://www.stm32.eu/stm32cube-3 zaprezentowany został prosty przykład pozwalający naprzemiennie gasić i zapalać diodę na płytce `STM32F4DISCOVERY`. Szkielet kodu źródłowego dla przykładowego projektu, został skonfigurowany zgodnie ze wskazanym opisem (cz. 2 oraz cz. 3 kursu) i wykorzystano go w dalszym etapie konfiguracji środowiska `Code::Blocks`.

9.1. Konfiguracja ustawień podczas generowania kodu

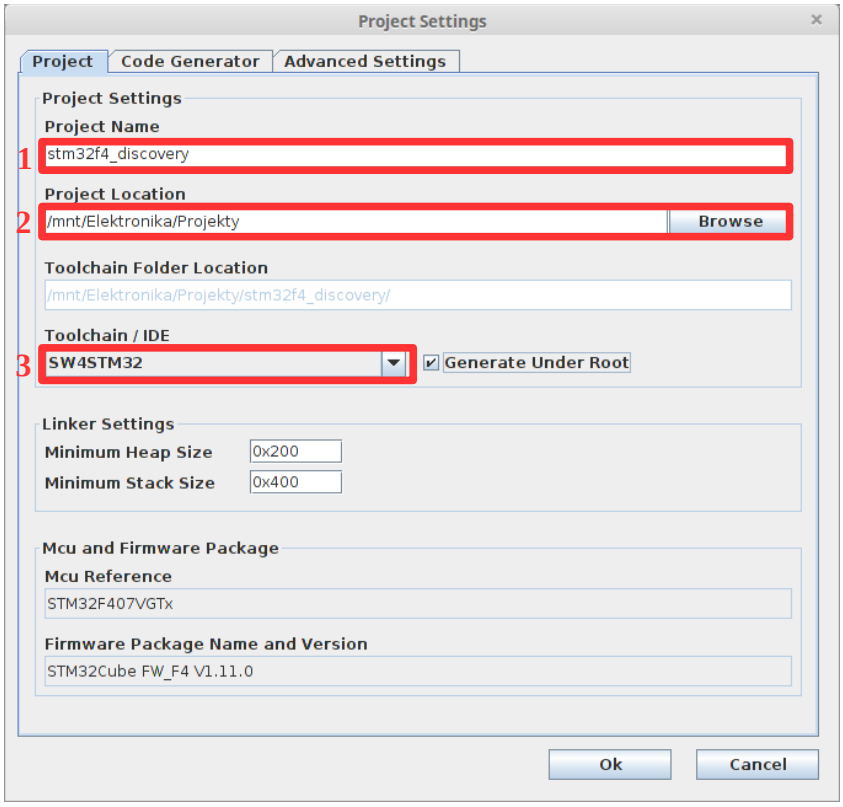
Podczas tworzenia nowego projektu w programie `STM32CubeMX` wybrano wsparcie dla zestawu `STM32F4DISCOVERY`, dzięki czemu automatycznie skonfigurowane zostały linie do których podłączone są dostępne na płytce peryferia. Wybór odpowiedniej opcji przedstawiono na *rys. 7*.



Rys. 7: STM32CubeMX - Tworzenie nowego projektu

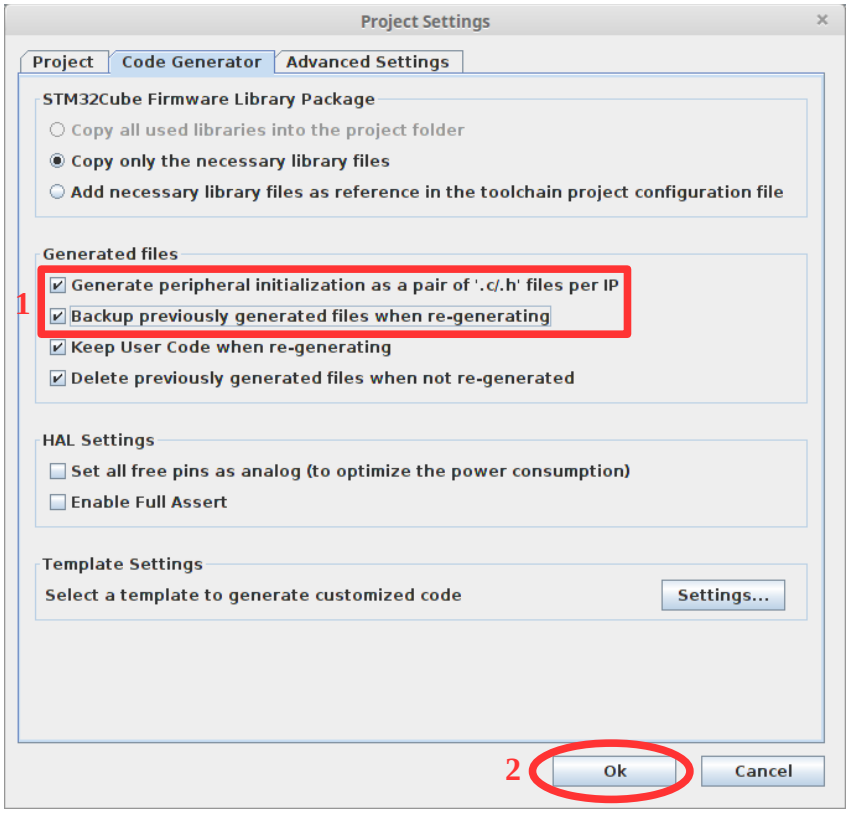
Pierwszym krokiem po wybraniu płytki `STM32F4DISCOVERY` była konfiguracja ustawień dla generowanego kodu. Ustawienia są dostępne w zakładce ***Project → Settings…*** W nowym oknie w zakładce ***Project*** należy nadać nazwę projektowi [1], wskazać

lokalizację, w jakiej wygenerowany ma zostać kod [2] oraz wybrać środowisko programistyczne, dla którego generowany będzie kod [3]. Wskazanym środowiskiem był `System Workbench for STM32` (`SW4STM32`). Projekt nazwano tak samo jak projekt `Code::Blocks`. Również jako lokalizację wybrano ten sam katalog, co dla projektu `Code::Blocks`. Dzięki temu pliki zostaną wygenerowane w tym samym katalogu co pliki z projektem `Code::Blocks`. Widok skonfigurowanej zakładki przedstawiono na *rys. 8*.



Rys. 8: Wybór lokalizacji projektu i środowiska programistycznego

W drugiej zakładce konfiguracyjnej ***`Code Generator`***, zaznaczono dodatkowo opcje [1]. Dzięki temu konfiguracja timera oraz linii wejść/wyjść zostanie zapisana w osobnych plikach z kodem źródłowym, zamiast w głównym pliku „`main.c`”. Zdaniem autora tutoriala, przy takim podziale plików, projekt staje się bardziej czytelny. Druga z zaznaczonych opcji powoduje wykonywanie kopii bezpieczeństwa podczas aktualizacji generowanego kodu. Widok zakładki przedstawiono na *rys. 9*. Po zakończeniu konfigurowania ustawień należy wcisnąć przycisk ***`OK`*** [2] oraz zapisać projekt za pomocą polecenia ***`File → Save Project`***.



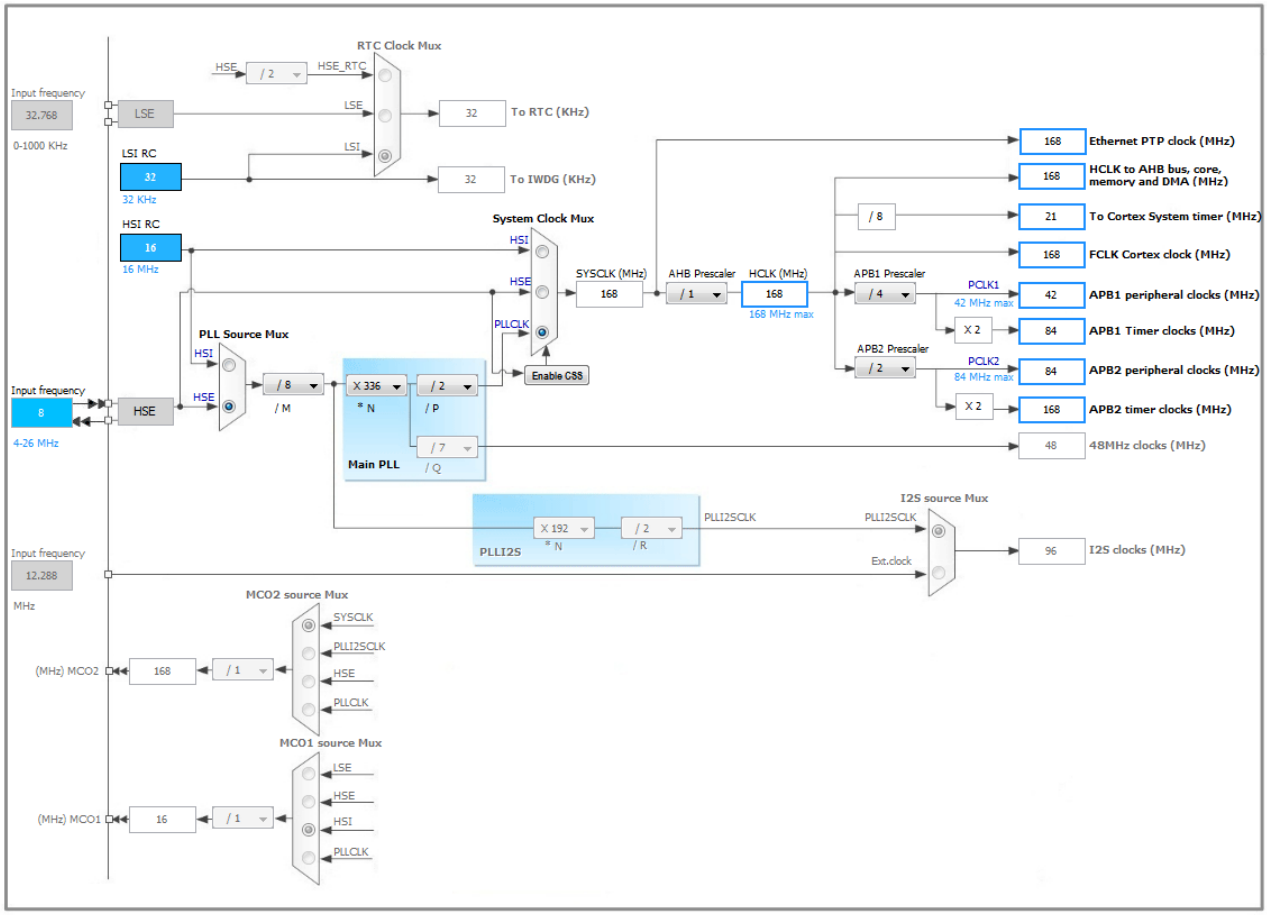
Rys. 9: Wybór sposobu generowania kodu źródłowego

Uwaga!!! Narzędzie `STM32CubeMX` jest bardzo „wrażliwe” na zapisywanie nowych plików z projektami. Aby móc zmienić np. nazwę projektu lub ścieżkę dla generowanego kodu, trzeba zapisać projekt jako nowy plik. Nie ma możliwości zmiany tych ustawień po pierwszym zapisaniu pliku z projektem. Z kolei, jeżeli wykonać polecenie `File → Save Project As`, nie zmieniając nazwy projektu, ani jego lokalizacji, projekt zostanie nadpisany, a pierwsze generowanie kodu wyczyści całkowicie dotychczasową zawartość folderu z projektem. Podczas próby wykonania takiej czynności, pojawia się stosowny komunikat ostrzegający o takim zjawisku.

Powyższy problem można w łatwy sposób ominąć. W tym celu wystarczy zamknąć otwarty w `STM32CubeMX` projekt oraz przenieść plik projektu do nowej, docelowej lokalizacji. Po ponownym otwarciu projektu z nowej lokalizacji, ścieżka zapisu dla generowanego kodu zostanie automatycznie zaktualizowana i ustawiona na bieżący katalog.

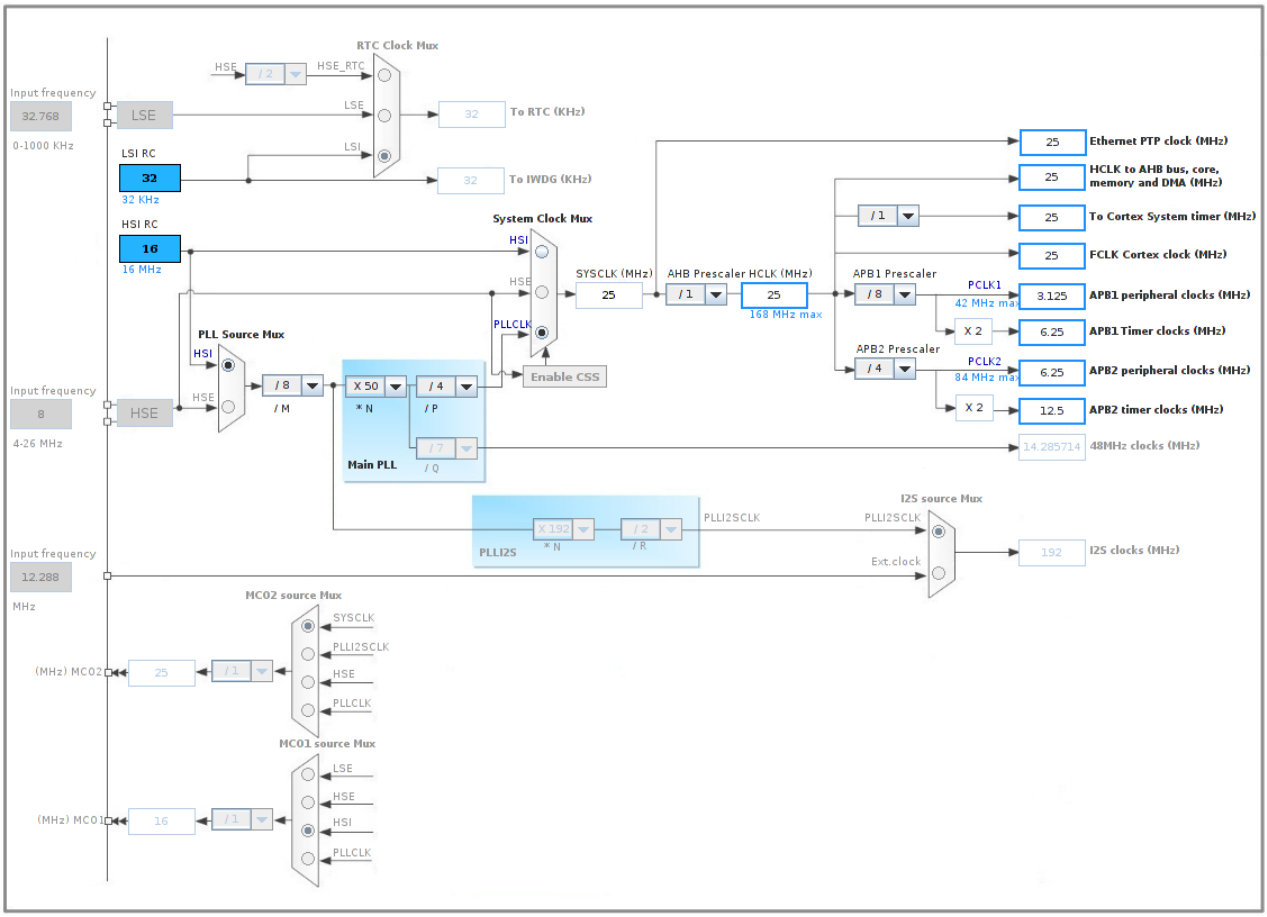
9.2. Konfiguracja peryferiów mikrokontrolera

W drugiej części kursu dotyczącego `STM32CubeMX`, przedstawiony jest sposób konfiguracji zegara. Opis zawiera informację, że domyślna konfiguracja zegara wygląda jak na *rys. 10*.



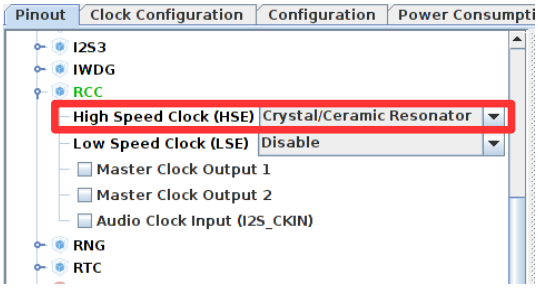
Rys. 10: Docelowa konfiguracja zegara

W rzeczywistości po stworzeniu projektu domyślna konfiguracja zegara wyglądała jak na *rys. 11*.



Rys. 11: Domyślna konfiguracja zegara

Aby możliwa była zmiana konfiguracji zegara, tak by móc skorzystać z zewnętrznego oscylatora należało w zakładce ***Pinout*** wybrać moduł ***RCC*** oraz włączyć zewnętrzny oscylator, co przedstawiono na *rys. 12*.



Rys. 12: Włączenie zewnętrznego oscylatora

WAŻNE!!! Aby przykładowy program działał prawidłowo, ustawienia zegara w zakładce *Clock Configuration* należy skonfigurować zgodnie z RYS. 10!!!

Po skonfigurowaniu zegara i timera `TIM10`, za pomocą polecenia ***Project → Generate Code*** wygenerować należy kod w języku C, zawierający konfigurację mikrokontrolera. W prezentowanym przykładzie kod został zapisany w katalogu z wcześniej utworzonym projektem `Code::Blocks`, czyli ***/mnt/Elektronika/Projekty/stm32f4_discovery***.

10. Konfiguracja kompilacji i debugowania projektu

Oprogramowanie STM32CubeMX korzysta z bibliotek programistycznych HAL (Hardware Abstraction Layer), więc dalsza konfiguracja Code::Blocks została zoptymalizowana pod kątem wykorzystania tych bibliotek. Jednak nic nie stoi na przeszkodzie, aby zrealizować konfigurację z wykorzystaniem bibliotek SPL (Standard Peripheral Library), pomijając wykorzystanie STM32CubeMX.

10.1. Wykorzystanie oprogramowania System Workbench for STM32

Aby móc poprawnie skompilować kod korzystając z biblioteki HAL, konieczne jest wskazanie kompilatorowi, dla jakiego mikrokontrolera należy wygenerować plik finalny zawierający instrukcje kodu maszynowego. Z pomocą przychodzi środowisko SW4STM32, (którego używania autor tutoriala chce uniknąć). Jednak nic nie stoi na przeszkodzie, aby wykorzystać to środowisko do „swoich" celów.

Celem wykorzystania środowiska SW4STM32 jest uzyskanie informacji dla kompilatora GCC, jakich parametrów należy użyć podczas kompilacji kodu pod konkretny mikrokontroler lub zestaw ewaluacyjny z rodziny STM32.

a) Aby tego dokonać należy w programie SW4STM32 utworzyć nowy projekt, poleceniem ***File → New → C Project***. W oknie kreatora, który się pojawi należy uzupełnić pole ***Project name:*** wpisując np. stm32f4_discovery, wybrać lokalizację projektu, która powinna być **INNA(!!!)** niż lokalizacja projektu Code::Blocks. W polu ***Project type:*** wybrać ***Empty Project***, a w polu ***Toolchains:*** wybrać ***Ac6 STM32 MCU GCC***, po czym wcisnąć ***Next***.

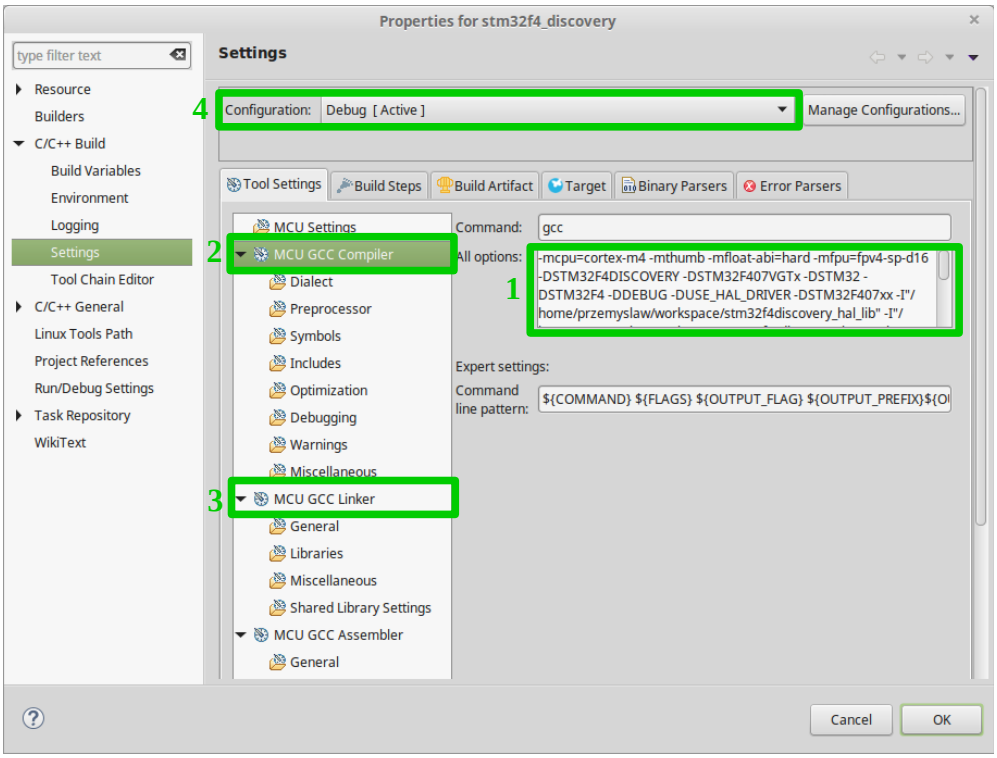
b) W drugim oknie kreatora ponownie wcisnąć ***Next***.

c) W kolejnym oknie w polu ***Series:*** należy wybrać rodzinę ***STM32F4***, a w polu ***Board:*** wybrać ***STM32F4DISCOVERY*** oraz wcisnąć ***Next***. Można w tym miejscu wybrać inny zestaw, lub utworzyć własne rozwiązanie, dzięki czemu dalsza konfiguracja staje się uniwersalna dla wielu różnych mikrokontrolerów.

d) W ostatnim oknie kreatora należy wybrać z jakiej biblioteki chce się skorzystać. Należy zaznaczyć ***Hardware Abstraction Layer (Cube HAL)***. Jeżeli

projekt dla wybranej rodziny `STM32` jest tworzony pierwszy raz, konieczne jest doinstalowanie odpowiednich bibliotek. Pojawi się wtedy komunikat: ***Target firmware has not been found, please download it***. Należy wcisnąć przycisk pod tym komunikatem ***Download target firmware***. Po automatycznym zainstalowaniu biblioteki można zakończyć konfigurację przyciskiem ***Finish***.

Po utworzeniu projektu, wszystkie potrzebne informacje na temat konfiguracji kompilatora dostępne są po otwarciu właściwości projektu, z menu ***Project → Properties***. W nowym oknie z menu po lewej stronie należy wybrać ***C/C++ Build → Settings***. Po prawej stronie okna pojawią się opcje widoczne na *rys. 13*.

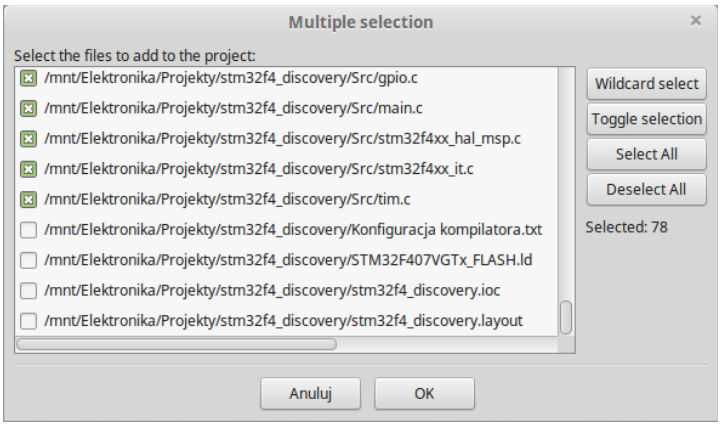


Rys. 13: Konfiguracja kompilatora w programie SW4STM32

Gotowa konfiguracja kompilatora (dla `SW4STM32`) dostępna jest w polu **[1]**, po wybraniu z menu ***MCU GCC Compiler*** **[2]**, lub dla linkera, po wybraniu z menu ***MCU GCC Linker*** **[3]**. Pozostałe elementy podmenu tworzą składową konfigurację widoczną finalnie w oknie **[1]**. Należy zwrócić uwagę, że konfiguracja dotyczy trybu ***Debug*** **[4]**. Po uzyskaniu dostępu do informacji na temat konfiguracji kompilatora, można kontynuować prace w programie `Code::Blocks`, z którego kilkukrotnie będą następowały odniesienia do okna z *rys. 13*.

10.2. Konfiguracja kompilacji projektu w środowisku Code::Blocks

Po otwarciu wcześniej utworzonego w Code::Blocks projektu, należy dodać wygenerowane w STM32CubeMX pliki. W tym celu z menu należy wybrać ***Project → Add files recursively…***,wskazać główny katalog z plikami projektu, np. ***/mnt/Elektronika/Projekty/stm32f4_discovery/*** oraz wcisnąć przycisk ***Otwórz***. Pojawi się okno, jak na *rys. 14* umożliwiające wybór plików, które chce się dodać do projektu. Domyślnie zaznaczone są wszystkie pliki *.c oraz *.h. Inne pliki są domyślnie odznaczone, więc nie trzeba nic zmieniać, wystarczy wcisnąć ***OK***. W kolejnym oknie, które się pojawi również należy wcisnąć ***OK***. Widok dodanych do projektu plików można podejrzeć w drzewie projektu,w lewym panelu programu. Pliki *.c przeznaczone do edycji przez użytkownika znajdują się w katalogu ***./Drivers/Src/***, natomiast pliki *.h w katalogu ***./Headers/Inc/***.



Rys. 14: Dodawanie plików do projektu

Następnie można rozpocząć proces konfiguracji kompilatora, który dostępny jest w menu ***Project → Build options...*** W nowym oknie, w menu po lewej stronie należy zaznaczyć tryb ***Debug***, dla którego konfiguracja zostanie przeprowadzona. Poniżej przedstawiono ustawienia dla kolejnych zakładek, które należy dostosować do każdego nowo tworzonego projektu.

W zakładkach, w których możliwy jest wybór opcji ***Policy:*** pozostawiono domyślne zaznaczenie ***Append target options to project options***. Oznacza to, że opcje kompilacji ustawione dla całego projektu (***stm32f4_discovery***) będą występowały zarówno podczas kompilacji w trybie ***Debug*** jak i ***Relase***.

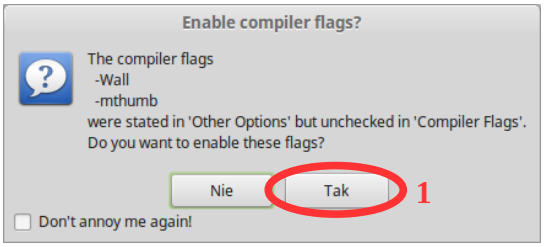
a) Zakładka ***Compiler settings → Compiler Flags***

Umożliwia podstawową konfigurację kompilatora. Ponieważ konfiguracja zostanie dodana „ręcznie”, odznaczone zostały wszystkie opcje w tej sekcji, zarówno w trybie ***Debug*** jak i dla całego projektu ***stm32f4_discovery***. Zapobiegnie to dublowaniu się opcji podczas kompilacji projektu.

b) Zakładka ***Compiler settings → Other compiler options***

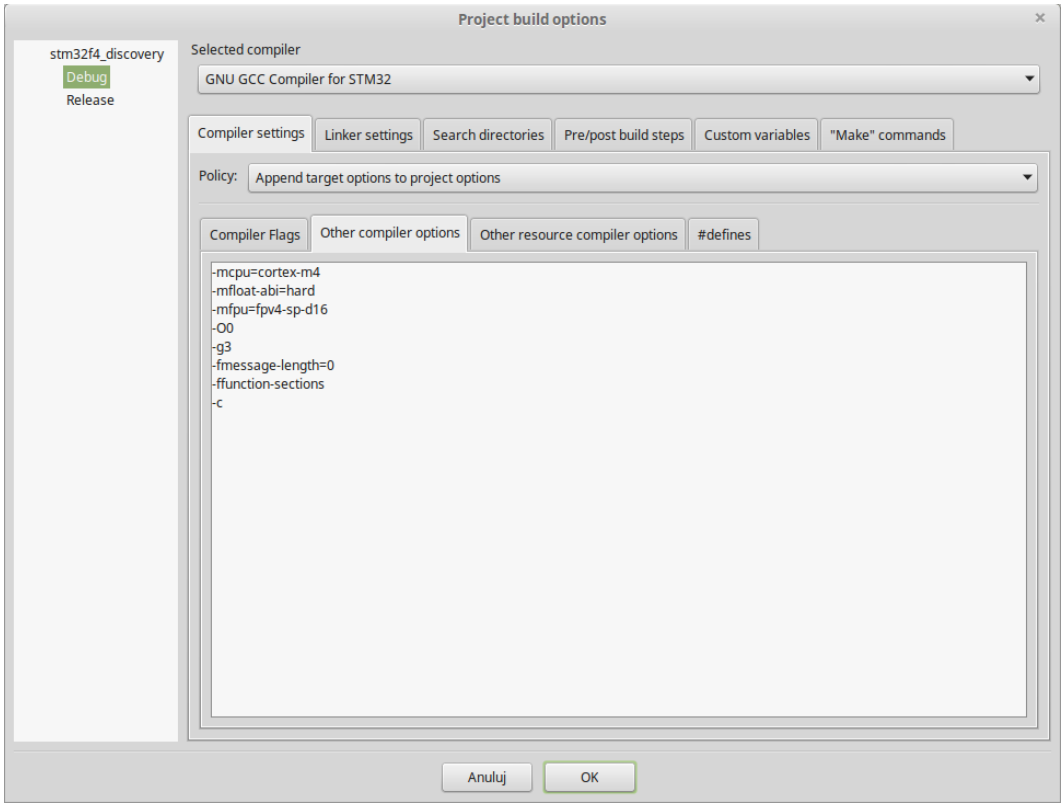
W tym miejscu należy dodać wszystkie flagi dla kompilatora, jakie znajdują się w oknie **[1]** z *rys. 13*, po wybraniu opcji ***MCU GCC Compiler*** **[2].** Są to następujące flagi: `-mcpu=cortex-m4`, `-mthumb`, `-mfloat-abi=hard`, `-mfpu=fpv4-sp-d16`, `-O0`, `-g3`, `-Wall`, `-fmessage-length=0`, `-ffunction-sections`, `-c`

Po zatwierdzeniu konfiguracji kompilatora przyciskiem ***OK*** w oknie `Code::Blocks`, okazało się, że dwie z wymienionych flag są dostępne w oknie ***Compiler settings → Compiler Flags***, co zaskutkowało pojawieniem się komunikatu jak na *rys. 15*.



Rys. 15: Zdublowana konfiguracja

W powyższym oknie należy wybrać przycisk ***Tak*** **[1]**, w wyniku czego zdublowane opcje znikną z zakładki ***Other compiler options***, ale pojawią się zaznaczone w zakładce ***Compiler Flags***. Ostateczny wygląd zakładki ***Other compiler options***, prezentuje się zgodnie z *rys. 16*.

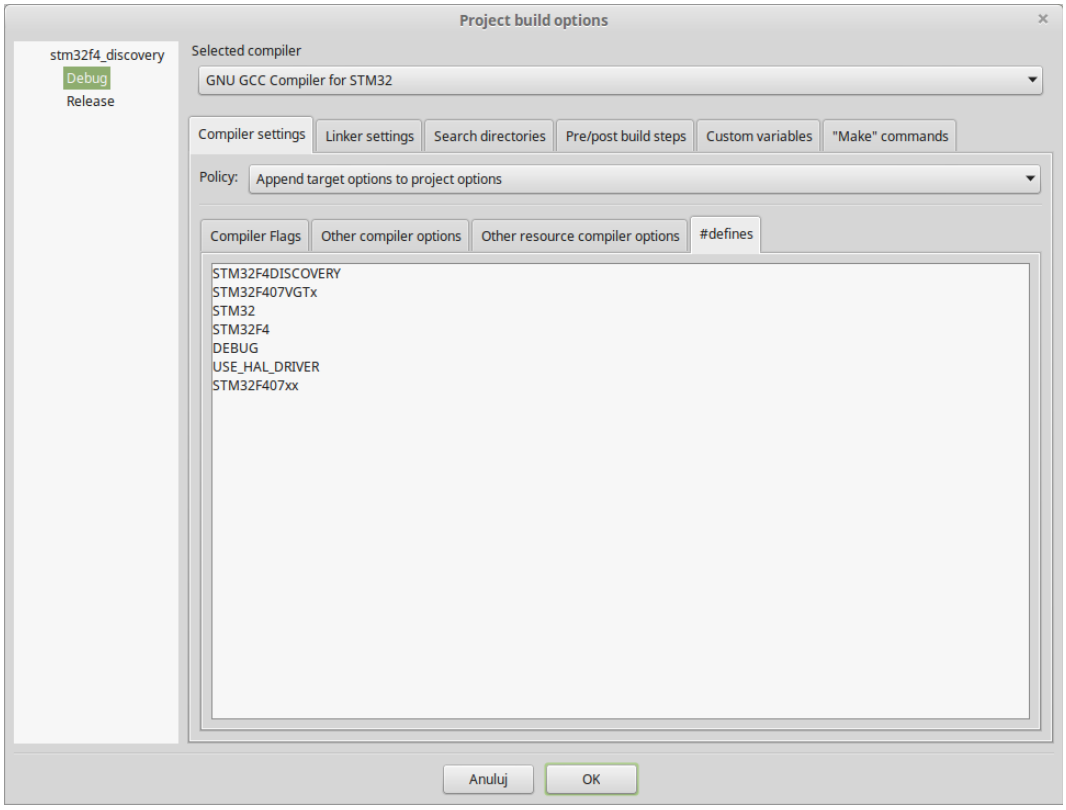


Rys. 16: Konfiguracja zakładki "Other compiler options"

c) Zakładka ***Compiler settings → #defines***

W tym miejscu należy wstawić definicje określające dla jakiego mikrokontrolera/zestawu kompilowany jest program, oraz jakich bibliotek (`SPL`/`HAL`) użyto w kodzie programu. Informacja ta jest zawarta w oknie **[1]** z *rys. 13*, lub po wybraniu podmenu ***Symbols*** we wskazanym oknie. Dla wybranego zestawu discovery oraz bibliotek `HAL`, są to definicje: `STM32F4DISCOVERY, STM32F407VGTx, STM32, STM32F4, DEBUG, USE_HAL_DRIVER, STM32F407xx`

Końcowy wygląd zakładki ***#defines***, przedstawiono na *rys. 17*.



Rys. 17: Konfiguracja zakładki „#defines"

d) Zakładka ***Linker settings***

Po otwarciu tej zakładki należy w pole ***Other linker options:*** wkleić dane pochodzące z ramki **[1]** z *rys. 13*, po uprzednim zaznaczeniu opcji ***MCU GCC Linker*** **[3]**. Jednak w tym przypadku zmieniają się dwie kwestie. Po wklejeniu zawartości, z ramki **[1]**, w oknie konfiguracyjnym `Code::Blocks` należy usunąć wpis dotyczący

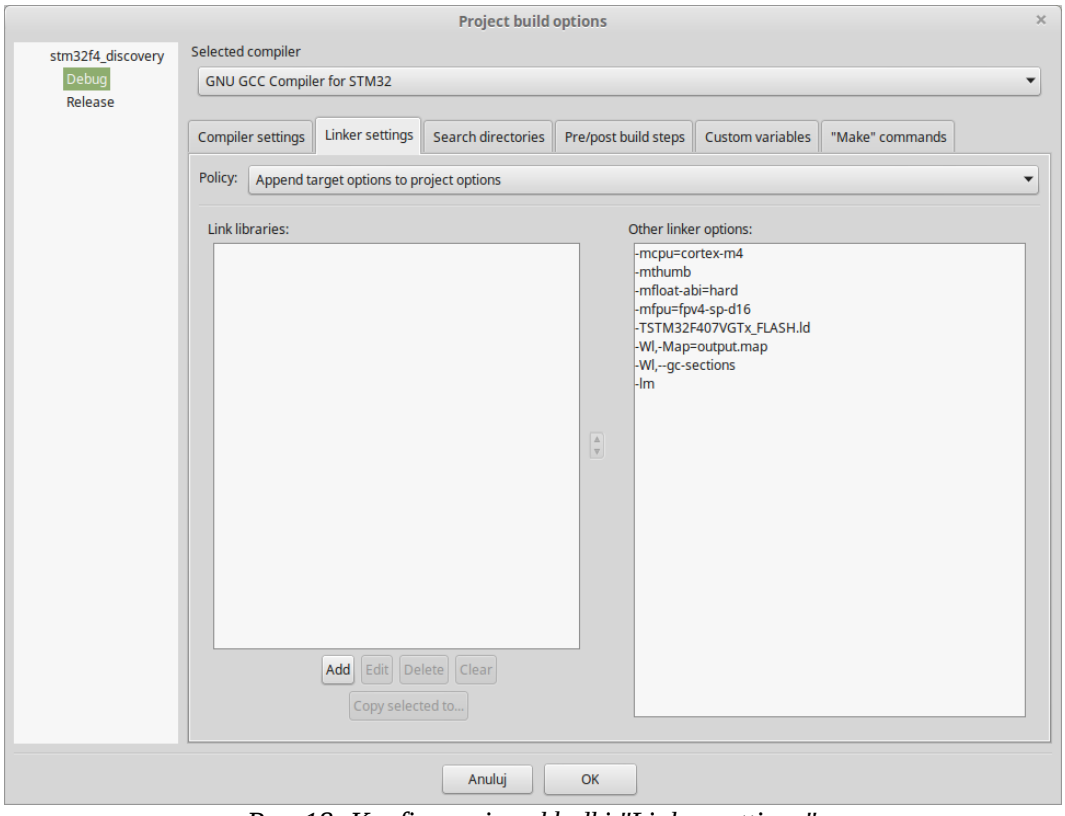
dołączania skompilowanej biblioteki `HAL`, ponieważ `Code::Blocks` takiej biblioteki nie tworzy. Przykładowy wpis ma postać:
`-L"/mnt/Elektronika/workspace/stm32f4discovery_hal_lib/Debug"`

Druga zmiana dotyczy wpisu pliku odpowiedzialnego za linkowanie kodu źródłowego. W ustawieniach `SW4STM32` wpis ten ma postać:
`-T"/mnt/Elektronika/workspace/stm32f4_discovery/LinkerScript.ld"`

Jednak `STM32CubeMX`, generuje ten plik pod nazwą: `STM32F407VGTx_FLASH.ld`, w związku z czym prawidłowy wpis powinien mieć postać:
`-TSTM32F407VGTx_FLASH.ld`

W końcowym efekcie w polu ***`Other linker options:`*** powinny się znaleźć następujące opcje: `-mcpu=cortex-m4`, `-mthumb`, `-mfloat-abi=hard`, `-mfpu=fpv4-sp-d16`, `-TSTM32F407VGTx_FLASH.ld`, `-Wl,-Map=output.map`, `-Wl,--gc-sections`, `-lm`

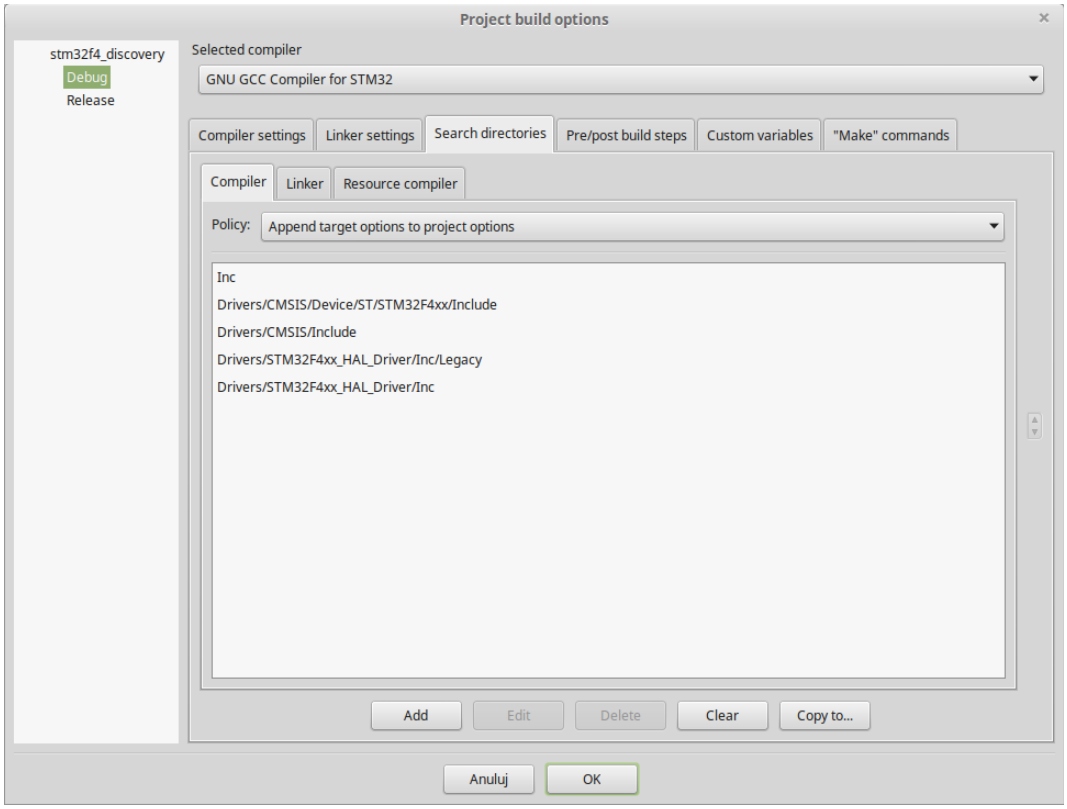
Ostateczny wygląd zakładki ***`Linker settings`***, przedstawiono na *rys. 18.*



Rys. 18: Konfiguracja zakładki "Linker settings"

e) Zakładka ***Search directories → Compiler***

W tej zakładce należy dodać wszystkie katalogi w projekcie zawierające pliki nagłówkowe `*.h`, aby kompilator wiedział gdzie szukać odniesień do plików. Należy zwrócić uwagę, że każdy folder zawierający pliki `*.h` należy dodać **oddzielnie**. Po wciśnięciu przycisku ***Add*** oraz wybraniu folderu z plikami `*.h` należy wcisnąć ***OK***. Pojawi się okienko z zapytaniem: ***Keep this as a relative path?*** Jeżeli ścieżka ma być dodana jako względna, należy wcisnąć ***Tak***. Katalogi dodane do przykładowego projektu przedstawiono na *rys. 19*.



Rys. 19: Konfiguracja zakładki "Search directories"

f) Zakładka **`Pre/post build steps`**

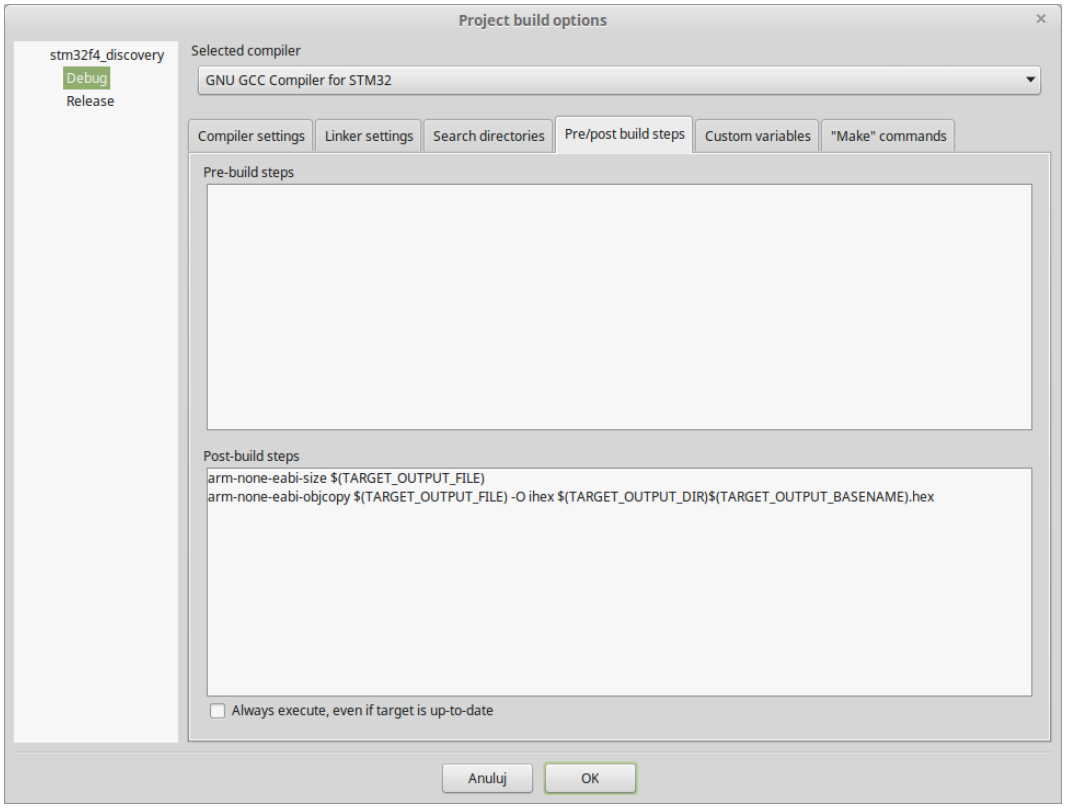
W tej zakładce w polu ***Post-build steps*** należy dodać informacje, jakie dodatkowe pliki mają zostać wygenerowane po kompilacji projektu. Zawarte zostały tutaj informacje o dwóch plikach. Pierwszy z nich to plik wynikowy projektu (często zapisywany z rozszerzeniem `*.elf`), zawierający dodatkowe informacje, dotyczące np. debugowania kodu. Drugi to plik `*.hex`, będący wsadem dla mikrokontrolera. Aby generować dodatkowe pliki, należy dodać wpis:

```
arm-none-eabi-size $(TARGET_OUTPUT_FILE)
```


oraz:

arm-none-eabi-objcopy $(TARGET_OUTPUT_FILE) -O ihex $(TARGET_OUTPUT_DIR)$(TARGET_OUTPUT_BASENAME).hex

Widok skonfigurowanego okna przedstawiono na *rys. 20*. Generowane pliki zapisywane są w folderze z projektem: ***./stm32f4_discovery/bin/Debug/***. Na tym etapie można zakończyć konfigurację kompilatora. Możliwe jest już generowanie plików, które posłużą do zaprogramowania mikrokontrolera i uruchomienia procesu debugowania.



Rys. 20: Konfiguracja zakładki "Pre/post build steps"

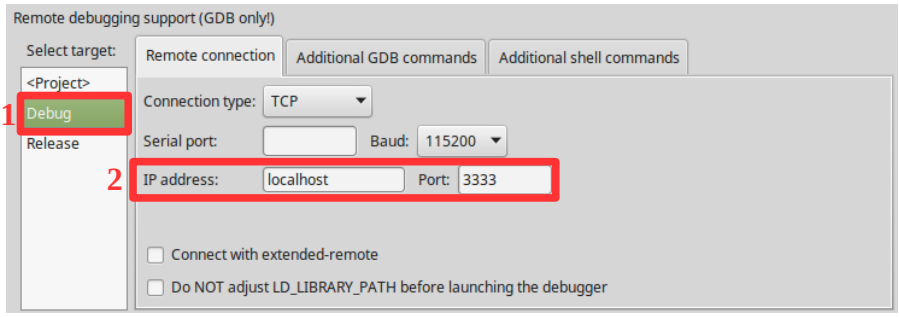
10.3. Konfiguracja debugowania kodu w środowisku `Code::Blocks`

Aby umożliwić debugowanie kodu, konieczna jest konfiguracja debuggera sprzętowego (`OpenOCD`). Aby tego dokonać należy w menu głównym wybrać: ***Project → Properties…*** a następnie przejść do zakładki ***Debugger***.

a) Zakładka ***Debugger → Remote Connection***

Po przejściu do wskazanej zakładki należy wybrać tryb ***Debug*** [1], a następnie wskazać sposób komunikacji z debuggerem sprzętowym. Ponieważ komunikacja będzie następować wewnątrz jednego systemu operacyjnego, zrealizowana zostanie jako wewnętrzna usługa sieciowa wykorzystująca wirtualne urządzenie sieciowe `loopback`.

Konfiguracja polega na wypełnieniu pól: [2] ***IP address:*** wpisem ***localhost*** oraz ***Port:*** wpisem ***3333***. Konfiguracja taka oznacza, że `OpenOCD` oczekuje na połączenie pod adresem IP `127.0.0.1` (wymiana danych wewnątrz jednego hosta) na porcie `3333`. Fragment zakładki ***Debugger*** przedstawiający powyższą konfigurację, przedstawiono na *rys. 21*.



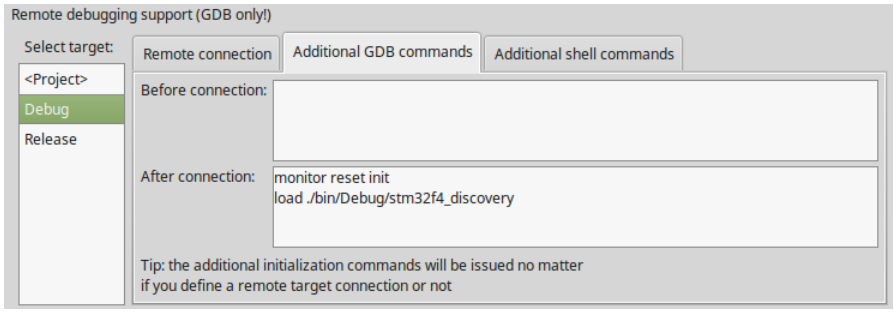
Rys. 21: Konfiguracja zakładki "Debugger"

b) Zakładka ***Debugger → Additional GDB commands***

Ostatnim krokiem konfigurującym połączenie z debuggerem sprzętowym `OpenOCD`, jest wskazanie jakie komendy ma wykonać debugger programowy `GDB`, aby poprawnie zainicjować połączenie z mikrokontrolerem. Po przejściu do wskazanej zakładki należy w polu ***After connection:*** dopisać polecenia:

```
monitor reset init
load ./bin/Debug/stm32f4_discovery
```

Pierwszy wpis odpowiada za przygotowanie mikrokontrolera do działania, drugi natomiast wskazuje jaki plik należy wykorzystać do zaprogramowania oraz debugowania mikrokontrolera. Jest to oczywiście plik wygenerowany na etapie kompilacji. Na *rys. 22*, przedstawiono opisaną powyżej konfigurację, po wykonaniu której można rozpocząć debugowanie kodu.



Rys. 22: Konfiguracja zakładki "Debugger" c.d.

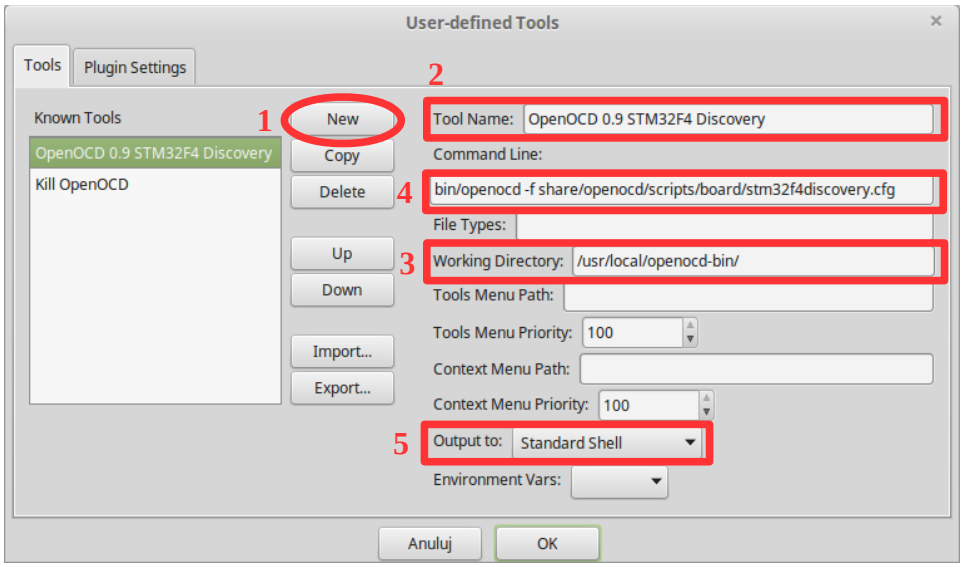
11. Uruchomienie debugowania

Po zakończeniu konfiguracji, nadszedł czas na uruchomienie debugowania kodu. W tym celu przed uruchomieniem trybu debugowania w `Code::Blocks`, należy podłączyć zestaw `STM32F4DISCOVERY` (lub inny programator/debugger), a następnie uruchomić debugger `OpenOCD`. Aby `OpenOCD` działał prawidłowo należy wskazać mu przez jaki interfejs oraz z jakim mikrokontrolerem będzie następowało połączenie. Do tego celu są wykorzystywane skrypty, znajdujące się w folderze z zainstalowanym pakietem `OpenOCD`. W tym przypadku jest to lokalizacja ***/usr/local/openocd-bin/share/openocd/scripts/*** Znajdują się tam katalogi ***interface*** - przechowujące skrypty programatorów/debuggerów, ***target*** - przechowujące skrypty mikrokontrolerów oraz ***board***, o czym za chwilę. Ogólna postać uruchomienia OpenOCD ma postać: `openocd -f interface/conf_interface.cfg -f target/conf_target.cfg` Gdzie `conf_interface.cfg` oznacza wybrany do komunikacji programator/debugger, natomiast `conf_target.cfg` oznacza rodzinę mikrokontrolera, z jakim chce się nawiązać połączenie.

Ponieważ używana wersja OpenOCD nie jest dodana do ścieżki systemowej, aby z niej skorzystać, należy podczas wywoływania programu w terminalu podać pełną ścieżkę dostępu, w tym przypadku jest to: ***/usr/local/openocd-bin/bin/openocd -f interface/stlink-v2.cfg -f target/stm32f4x.cfg*** Powyższe polecenie jest prawidłowym wywołaniem umożliwiającym nawiązanie połączenia z zestawem `STM32F4DISCOVERY`. Można też nieco skrócić powyższą komendę. W katalogu ***board*** znajdują się pliki konfiguracyjne przygotowane dla wybranych zestawów ewaluacyjnych. Wywołując polecenie ***/usr/local/openocd-bin/bin/openocd -f board/stm32f4discovery.cfg*** uzyska się ten sam efekt co w poprzednim poleceniu. Pliki w katalogu ***board*** zawierają po prostu odniesienia do odpowiednich plików w katalogach ***interface*** oraz ***target***. Aby rozpocząć debugowanie kodu w `Code::Blocks`, należy mieć podłączony zestaw debuggera z mikrokontrolerem oraz aktywne połączenie `OpenOCD` z wybranym zestawem.

Ważne!!! Debugger `OpenOCD` działa niezależnie od `Code::Blocks` i musi być uruchomiony przed rozpoczęciem debugowania, aby odwołujący się do niego `Code::Blocks` miał się do czego odwołać. `OpenOCD` musi pozostać uruchomiony przez cały czas trwania debugowania, a jego zamknięcie jest również niezależne od wyłączenia `Code::Blocks`.

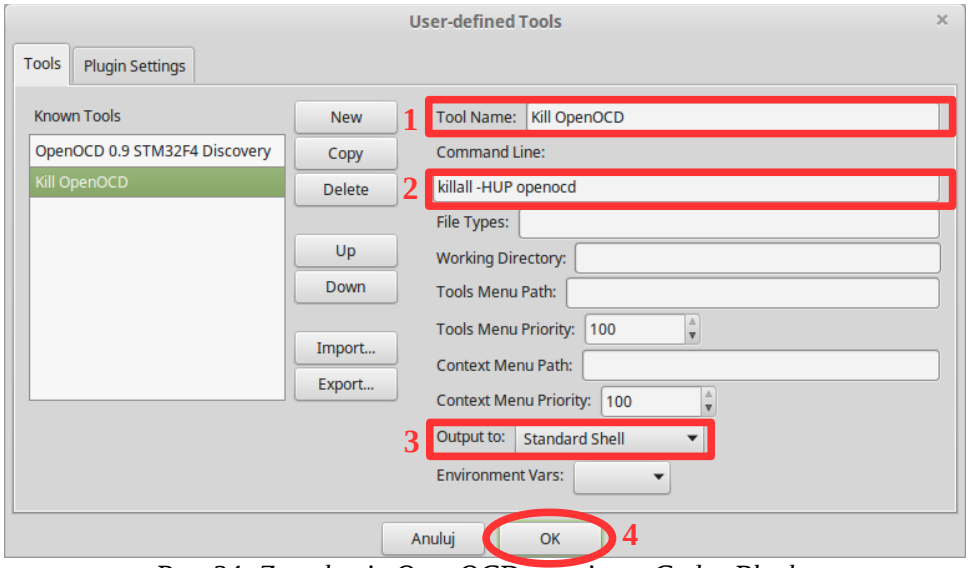
Aby uniknąć konieczności ciągłego, ręcznego wpisywania poleceń w linii komend, można w `Code::Blocks` skonfigurować dodatkowe narzędzie, które będzie uruchamiane poprzez wybór z menu głównego programu. W tym celu należy z menu wybrać ***Tools+ → Configure Tools…*** Pojawi się okno jak na *rys. 23*. Należy wybrać przycisk tworzenia nowego narzędzia ***New*** [1]. W polu ***Tool Name:*** wpisać dla niego nazwę [2]. W polu ***Working Directory:*** [3] wpisać początek lokalizacji, w której należy szukać pliku wykonywalnego realizującego wykonane polecenie, oraz uzupełnić pole ***Command Line:*** przechowujące składnię polecenia [4]. Należy tutaj podać bezwzględną lokalizację dla pliku binarnego `OpenOCD` oraz pliku z konfiguracją `*.cfg` (początek tej ścieżki zaczyna się w polu [3]). W polu ***Output to:*** [5] należy wybrać, czy narzędzie ma się uruchamiać w tle, w niezależnej konsoli (niewidoczne dla użytkownika – ***Standard Shell***), czy też w oknie `Code::Blocks` (widoczne dla użytkownika – ***Tools Output Window***). Podczas testów trzecia opcja ***Code::Blocks Console***, nie działała poprawnie, uniemożliwiając debugowanie.



Rys. 23: Uruchamianie OpenOCD z poziomu Code::Blocks

Można również zautomatyzować proces wyłączania OpenOCD. W tym celu w powyższym oknie należy ponownie wcisnąć przycisk ***New***, a następie uzupełnić pola zgodnie z *rys. 24*. Tym razem wystarczy uzupełnić pole z nazwą narzędzia [1], pole z wykonywaną komendą [2] oraz ustawić wyjście na ***Standard Shell*** [3]. Po czym wcisnąć w oknie konfiguratora ***OK*** [4]. Wpis ***killall -HUP openocd*** oznacza, że zostaną zamknięte wszystkie aktualnie otwarte procesy o podanej nazwie.

Od tej chwili włączanie OpenOCD jest dostępne z menu ***Tools+ → OpenOCD 0.9 STM32F4 Discovery***, a jego wyłączanie w menu ***Tools+ → Kill OpenOCD***.



Rys. 24: Zamykanie OpenOCD z poziomu Code::Blocks

Była to ostatnia czynność jaką należało wykonać, by mieć w pełni sprawne środowisko deweloperskie oparte na Code::Blocks. Po skompilowaniu kodu - menu ***Build → Build*** można uruchomić program poleceniem ***Build → Run***. Po uruchomieniu OpenOCD, debugowanie działa w standardowy sposób i można je wywołać za pomocą menu ***Debug*** lub bezpośrednio w podręcznym menu w oknie głównym Code::Blocks. Jeżeli przykładowy kod z STM32CubeMX został skonfigurowany i uzupełniony według wcześniej wskazanego kursu i dioda naprzemiennie gasi się i zaświeca, można w łatwy sposób zweryfikować działanie opcji debugowania kodu, wznawiając i zatrzymując wykonywanie programu.

12. Zakończenie

Powyższy opis zrealizowany został na podstawie płytki STM32F4DISCOVERY. Jednak nic nie stoi na przeszkodzie, aby na jego bazie skonfigurować środowisko do współpracy z innymi mikrokontrolerami o ile znane są użytkownikowi parametry, jakich wymaga dany kompilator. W przypadku rodziny STM32 poznanie niezbędnych parametrów kompilacji ułatwia wykorzystanie programu SW4STM32.

Przedstawiony sposób nie wyczerpuje wszystkich zagadnień konfiguracji środowiska dla rodziny STM32, wręcz przeciwnie. Jest zaledwie wierzchołkiem góry lodowej, jednak pozwala poznać podstawy konfiguracji środowiska Code::Blocks, na bazie którego można rozwijać swoją wiedzę na temat tego narzędzia. Nie jest to również jedyna „słuszna” metoda konfiguracji. Wskazywane opcje można dostosowywać na wiele różnych sposobów, a do wielu opisywanych opcji można się dostać innymi metodami, niż wskazane w opisie.

Konfiguracja środowiska zrealizowana została w systemie operacyjnym Linux, jednak wszystkie wykorzystane w opisie narzędzia, są dostępne również dla systemu Windows (a część z nich, być może i wszystkie również dla systemu Mac OS X). Możliwe jest zatem przeprowadzenie analogicznej konfiguracji dla tego systemu, zwracając uwagę na różnice nazewnictwa katalogów systemowych, „literowy” dostęp do dysków i rozszerzenia „*.exe” blików binarnych. Jedynie ostatni wpis szczególnie polecenie ***killall –HUP openocd*** ze względu na to, że nie występuje w systemie Windows, wymagać będzie zastąpienia inną metodą, odpowiednią dla tego systemu.

Bibliografia

- Instalacja kompilatora GCC ARM Embedded Toolchain
  - https://launchpad.net/gcc-arm-embedded/+download – pliki instalacyjne
  - http://gnuarmeclipse.github.io/toolchain/install/ - opis instalacji
- Instalacja debuggera OpenOCD
  - https://sourceforge.net/projects/openocd/files/openocd/0.9.0/ - kody źródłowe
  - http://hertaville.com/stm32f0discovery-part-1-linux.html – opis instalacji
- Instalacja STM32CubeMX
  - http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242 – dokumentacja STM32CubeMX
  - http://www.stm32.eu/stm32cube-1 - krótki kurs obsługi, cz. 1
  - http://www.stm32.eu/stm32cube-2 - krótki kurs obsługi, cz. 2
  - http://www.stm32.eu/stm32cube-3 - krótki kurs obsługi, cz. 3
- STM32CubeF4
  - http://www2.st.com/content/st_com/en/products/embedded-software/mcus-embedded-software/stm32-embedded-software/stm32cube-embedded-software/stm32cubef4.html – dokumentacja biblioteki HAL dla rodziny STM32F4
- Instalacja System Workbench for STM32
  - http://www.openstm32.org/Downloading+the+System+Workbench+for+STM32+installer – pliki instalacyjne
- Konfiguracja Code::Blocks dla STM32F0 (StdLib)
  - http://www.hackvandedam.nl/blog/?p=707 – konfiguracja Code::Blocks
- Dodatkowe informacje
  - http://www.freddiechopin.info/pl/artykuly/35-arm/59-arm-toolchain-tutorial – instalacja kompilatora GCC oraz debuggera OpenOCD w środowisku Windows
  - http://www.freddiechopin.info/pl/download/category/4-openocd – skompilowany OpenOCD dla systemu Windows

