Spis treści

Zamiast wstępu.

O tym, że rynek kontrolerów do urządzeń wbudowanych zdominowała koncepcja procesora na rdzeniu firmy ARM nie trzeba nikogo przekonywać. Wszyscy liczący się producenci mają w ofercie procesory z tym rdzeniem, z Microchip-em i ATMEL-em włącznie. Trend podchwyciły firmy niezależne, oferując bogatą ofertę modułów mniej lub bardziej wyposażonych w różne peryferia a ostra wojna cenowa sprawia, że oferowane są za naprawdę niewielkie pieniądze.

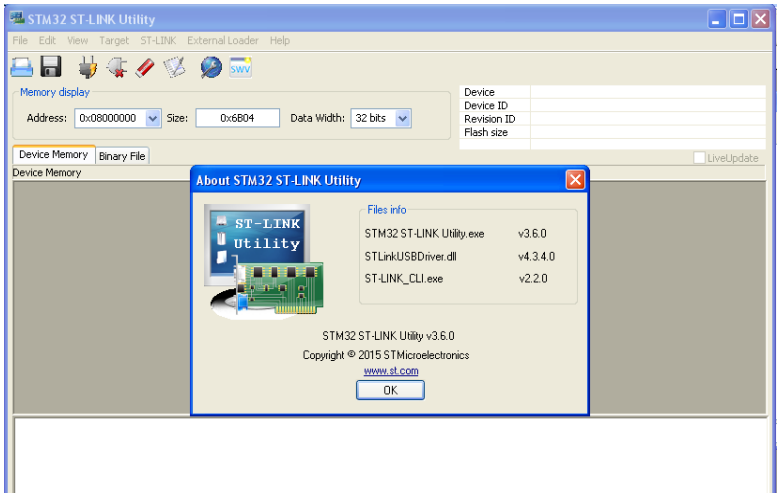
To dobry moment aby wykorzystać je w naszych projektach. Wyrośliśmy już z Bartków i Kacprów a każde radio zawiera minimum 1 procesor. Jeśli uwzględnić trend cyfryzacji wszystkiego i wszędzie, nowe zastosowania będą wymagały nowych umiejętności. Zrozumieliśmy to ponad rok temu i wspólnie z Darkiem SQ5KHA rozpoczęliśmy przygodę z rodziną STM32. To nie przypadek ale świadomy wybór po przejrzeniu Internetu pod kątem projektów wykorzystujących zalety tych procesorów a także dostępność i ceny modułów, i co ważne, środowiska programistycznego.

Dużo czasu zajęło poznanie najbardziej popularnych a przy tym darmowych środowisk, w stopniu wystarczającym na podjęcie ostatecznej decyzji. Wybór padł na Em::Blocks a jako podstawę do ćwiczeń wybraliśmy moduł STM32F4DISCOVERY. Aby ułatwić samodzielny start opiszę pokrótce co trzeba zrobić aby zainstalować środowisko, wymagane programy narzędziowe, wygenerować kod inicjujący procesor z wybranymi peryferiami, skompilować go i uruchomić a na deser linki do przydatnych stron z przykładami do uruchomienia.

STM32 ST-LINK Utility

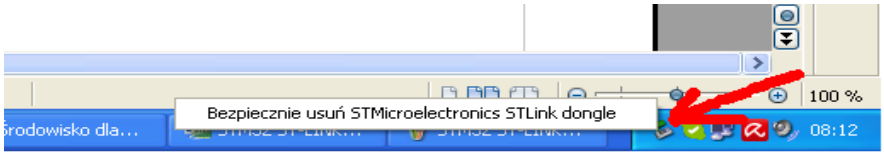
W pierwszej kolejności musimy zainstalować program umożliwiający wymianę informacji i danych między komputerem PC a płytką STM32F4DISCOVERY. Do komunikacji wykorzystywane jest złącze USB umiejscowione na krawędzi płytki od strony programatora (mniejszy mikroprocesor) a więc musimy się zaopatrzyć w odpowiedni kabel łączący.

Pamiętaj! Jak każde urządzenie USB, najpierw instalujemy oprogramowanie a potem podłączamy urządzenie, chyba, że instalator żąda tego wcześniej!



STM32 ST-LINK/V2 pobieramy i rozpakowujemy a następnie instalujemy w domyślnie proponowanym katalogu. Przy zakończeniu instalacji, program instaluje jeszcze sterowniki USB niezbędne do komunikacji z płytką, na co także zgadzamy się domyślnie. Jeśli nie pracujemy na koncie Administratora (co powinno być oczywiste) całą operację trzeba przeprowadzić z jego uprawnieniami. Program nie wymaga żadnych ustawień tak więc dla sprawdzenia poprawności instalacji uruchamiamy go i sprawdzamy wersję z menu Help – About. Następnie podłączamy naszą płytkę i na pasku systemowym powinniśmy zobaczyć ikonkę informującą, że mamy nowe urządzenie, służącą również do jego dezaktywacji. Czasami

przydaje się to w trakcie odpluskwiania programu. Diody LED programatora informują o aktualnym stanie urządzenia, przy braku transmisji świecą światłem ciągłym.



W trakcie pisania i uruchamiania programów sam ST-Link nie będzie nam potrzebny ponieważ IDE mają własne moduły zarządzające programowaniem i debugowaniem ale niezbędne są jego biblioteki dlatego musimy go zainstalować.

EmBlocks

Podstawowym narzędziem służącym do pisania i odpluskwiania programów na STM32 jest program Em::Blocks zwany dalej EM. To kompletne środowisko zawierające prócz edytora kompilator, linker i debuger z pakietu GNU GCC a także zbiór różnego rodzaju dodatków ułatwiających życie programiście. Oczywiście jest też moduł komunikacji z naszą płytką, zatem instalując go na dysku dostajemy komplet możliwości wykreowania, uruchomienia i przetestowania naszego programu.

Po zaakceptowaniu warunków licencji pobieramy, rozpakowujemy i rozpoczynamy instalację programu. Jak zwykle czynimy to z prawami Administratora. Zostawiamy domyślny wybór instalacji, kompilatora i wtyczek a także lokalizacji.

UWAGA! Wczesne wersje pakietów GCC miały problem ze spacjami w nazwach folderów („Program Files"). Nie należy też stosować polskich liter w ścieżkach dostępu do plików środowiska. W razie problemów z kompilacją należy sprawdzić logi kompilacji czy nie wystąpił tego rodzaju problem.

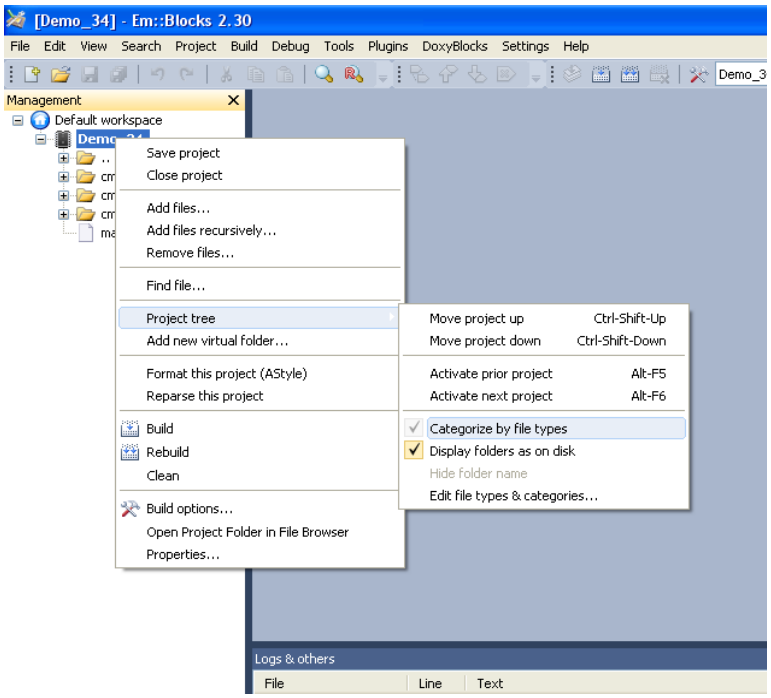
EM może korzystać z różnych kompilatorów, dla każdego tworzonego programu można to określić indywidualnie. Jest to o tyle istotne, że czasami program poprawnie skompilowany konkretną wersją kompilatora działa a inną już nie! (dla przykładu STM32-SDR). Dlatego w skrajnych przypadkach możemy pobrać i zainstalować GCC w innej wersji niż domyślna dla EM i zainstalować go w głównym korzeniu plików (np. C:/GCC/nazwa_wersji)

Jeśli instalacja przebiegła pomyślnie uruchamiamy program i dokonujemy wstępnej konfiguracji. Z menu „Settings" wybieramy opcje „Autosave" i ustawiamy pożądany okres autozapisu. Jeśli nie pasują nam domyślne kolory i inne nastawy, możemy dopasować je do własnych upodobań.

Proponuję też wspólne lokalizacje dla umieszczania plików projektów „**C:\EM_Projects**" (z podkreśleniem między „EM" a „Projects") i archiwów bibliotecznych „**C:\Repository**". Pozwoli to na wymianę projektów skompresowanych w całości i proste ich rozpakowanie bez konieczności modyfikowania ścieżek dostępu.

Teraz możemy już przystąpić do pracy. EM ma wbudowany kreator projektu dla procesorów z rodziny stm32 (a także dla innych) ale również moduł importu projektów z innych środowisk, w tym z popularnego CooCox. Dlatego na początek proponuję import gotowego projektu bowiem będzie możliwość szybkiego przeprowadzenia kompilacji, konfiguracji programatora, załadowania kodu i uruchomienia programu. Za przykład niech posłuży projekt 34-LCD_2x16-Library (STM32F4) z bardzo dobrego portalu poświęconego mikrokontrolerom.

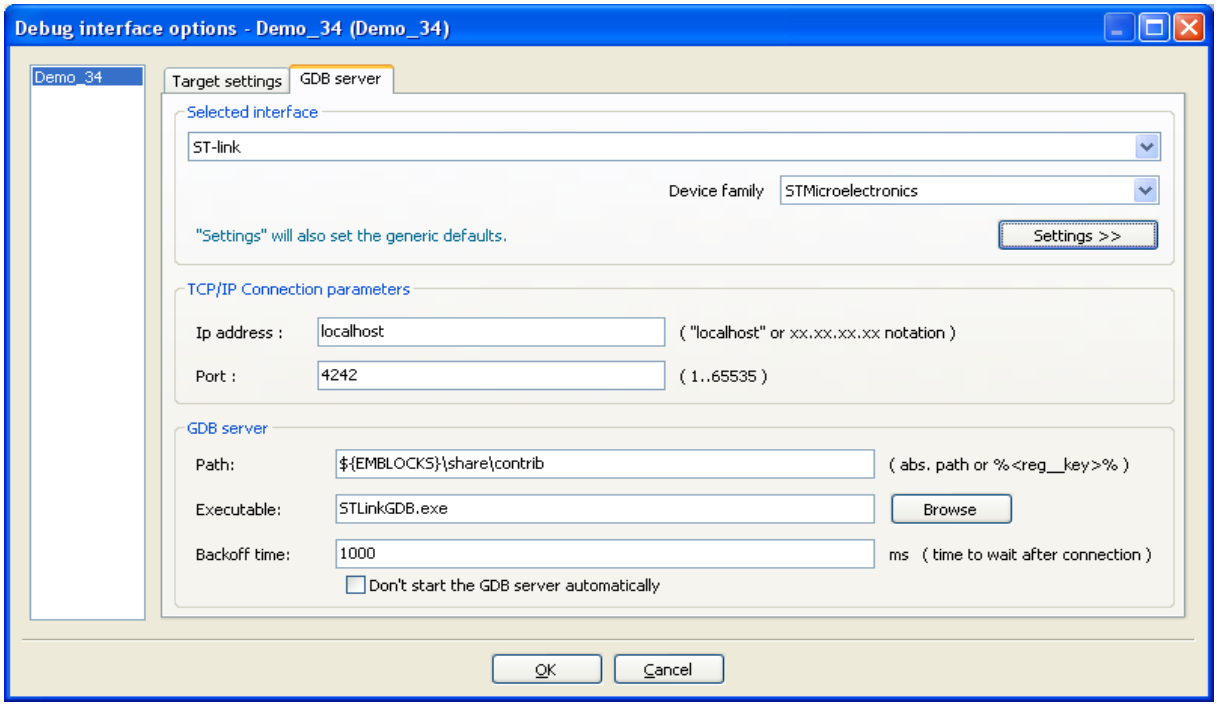
Pobrane archiwum projektu rozpakowujemy w katalogu C:\EM_Projects. Następnie z menu naszego EM wybieramy **File – Import Project** i wskazujemy **CooCox CoIDE Project...** a następnie otwieramy plik **Demo_34.coproj** z folderu **C:\EM_Projects\Demo_34_LCD_2x16.**



W lewym panelu nazywanym **Management** zobaczymy drzewko plików zaimportowanego projektu w domyślnym widoku **Categorize by file types** czyli pogrupowane wg typu (nagłówkowe i źródłowe). Widok ten można zmienić wybierając odpowiednią opcję z menu klikając prawym klawiszem na nazwie projektu. Ja osobiście wybieram widok odzwierciedlający rzeczywiste położenie na dysku.

Ilość narzędzi, ich układ i zawartość określamy z głównego menu **View** co pozwala na elastyczne gospodarowanie powierzchnią ekranu.

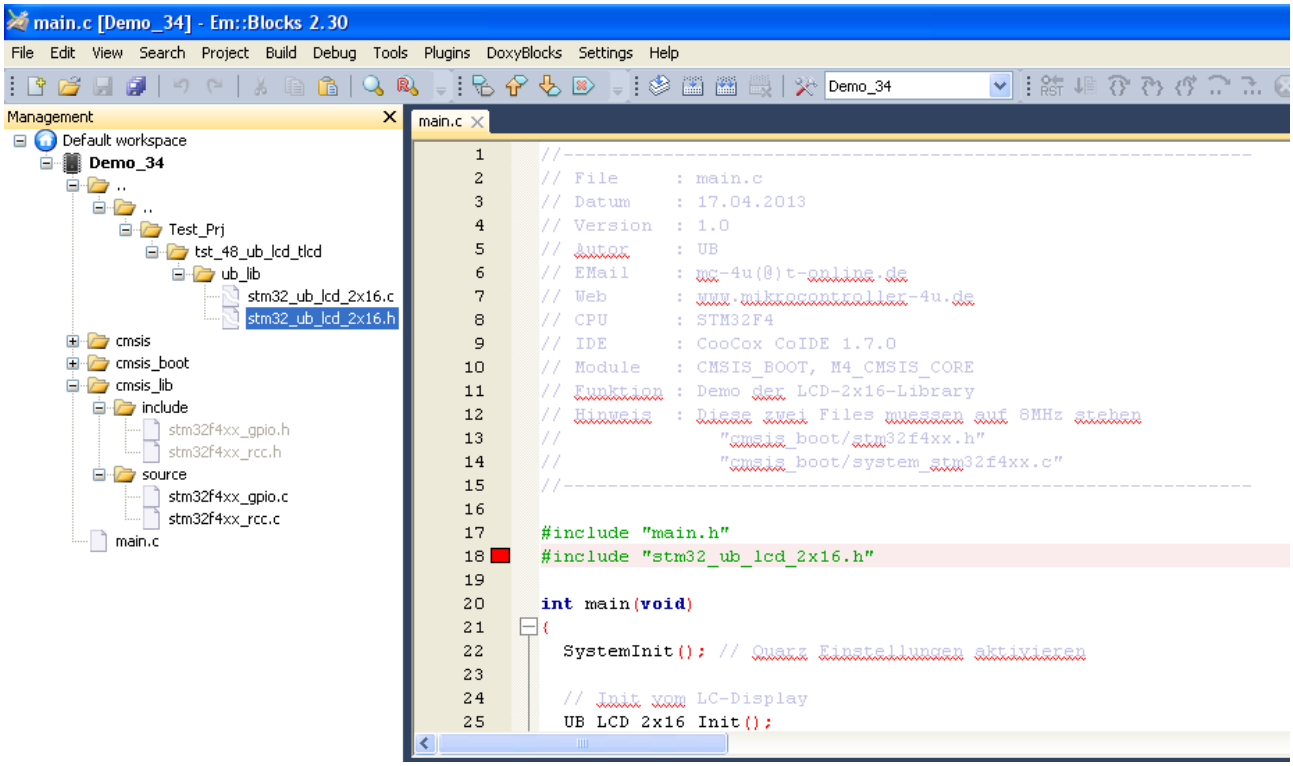
Po załadowaniu projektu aktywuje się opcja **Interfaces** z menu **Debug**. Możemy więc przystąpić do skonfigurowania programatora. Otwieramy formularz **Interfaces** i w zakładce **Target settings** zaznaczamy opcję **Run to Main()**. Pozwoli to ominąć kod inicjacji samego procesora i ustawi pułapkę na głównej funkcji Main(). Przechodzimy do zakładki **GDB server** i z listy **Selected interface** wybieramy opcję **ST-link**. Następnie klikamy na przycisk **Settings>>** i zatwierdzamy nowo otwarte okno przyciskiem **OK**. W rezultacie okno Debug interface options powinno wyglądać następująco:



Potwierdzamy ustawienia klawiszem OK i na tym kończymy konfigurację programatora. Ustawienia te są ważne w ramach projektu, więc kolejny projekt będzie wymagał ponownej konfiguracji.

Teraz możemy już skompilować program. Naciskamy klawisz funkcyjny F7 (lub z menu **Build – Build target**) i po krótkiej chwili ujrzymy wynik działania kompilatora w oknie na dole ekranu nazwanym **Log & others** zawierający następujący komunikat: **fatal error: stm32_ub_lcd_2x16.h: No such file or directory,** co oznacza, że nie znaleziono pliku stm32_ub_lcd_2x16.h W oknie edytora widać czerwony kwadrat w 18-tej linii wskazujący moment wystąpienia błędu.

To częsty komunikat wyniku kompilacji świadczący o niezgodności ścieżek dostępu w projekcie. Wynika to z różnic konwencji stosowanych w poszczególnych środowiskach projektowych, gdzie raz stosuje się zapis relatywnej, kiedy indziej bezwzględniej lokalizacji plików. Rzadko kiedy po zaimportowaniu projektu wszystko jest poprawnie. Dlatego poszukiwania „zagubionego" pliku rozpoczynamy od przejrzenia zawartości drzewka projektu.

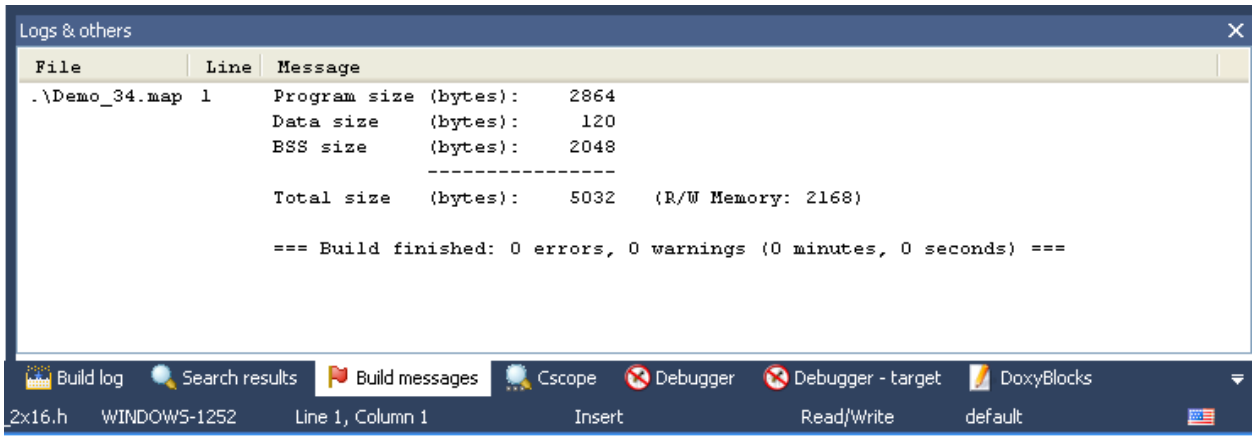


Ja widać powyżej plik istnieje ale kompilator go „nie widzi". Ponadto lokalizacja jest nieco „udziwniona" gdy w rzeczywistości folder ub_lib jest w głównym folderze projektu. Zatem trzeba mu wskazać lokalizację poszukiwania. Najpierw jednak odłączymy tak dziwnie zestawioną ścieżkę wskazując folder bez nazwy u samej góry poprzez kliknięcie prawym klawiszem i wybór opcji **Remove ..*** następnie przyłączając ją ponownie prawym klawiszem na nazwie projektu opcją **Add files recursively...** i wskazując pożądany folder. W ten sposób możemy postąpić, gdy chcemy zaimportować projekt ze środowiska nie obsługiwanego przez moduł importu EM. Nie zawsze jest to takie oczywiste i tu trzeba nabrać pewnej rutyny ćwicząc.

Teraz musimy jeszcze poinformować kompilator, że ma szukać plików w nowym folderze. Zrobimy to przechodząc do opcji projektu. Na nazwie projektu klikamy prawym klawiszem i

wybieramy opcję **Build options...** a następnie zakładkę **Search directories**. W zakładce **Includes** naciskamy przycisk **Add** i w formularzu **Add directory** klawiszem … wskazujemy katalog zawierający oba brakujące pliki czyli **ub_lib**. Odpowiadamy twierdząco na pytanie o to czy chcemy aby ścieżka została zapisana relatywnie i zamykamy okno przyciskiem **OK**.

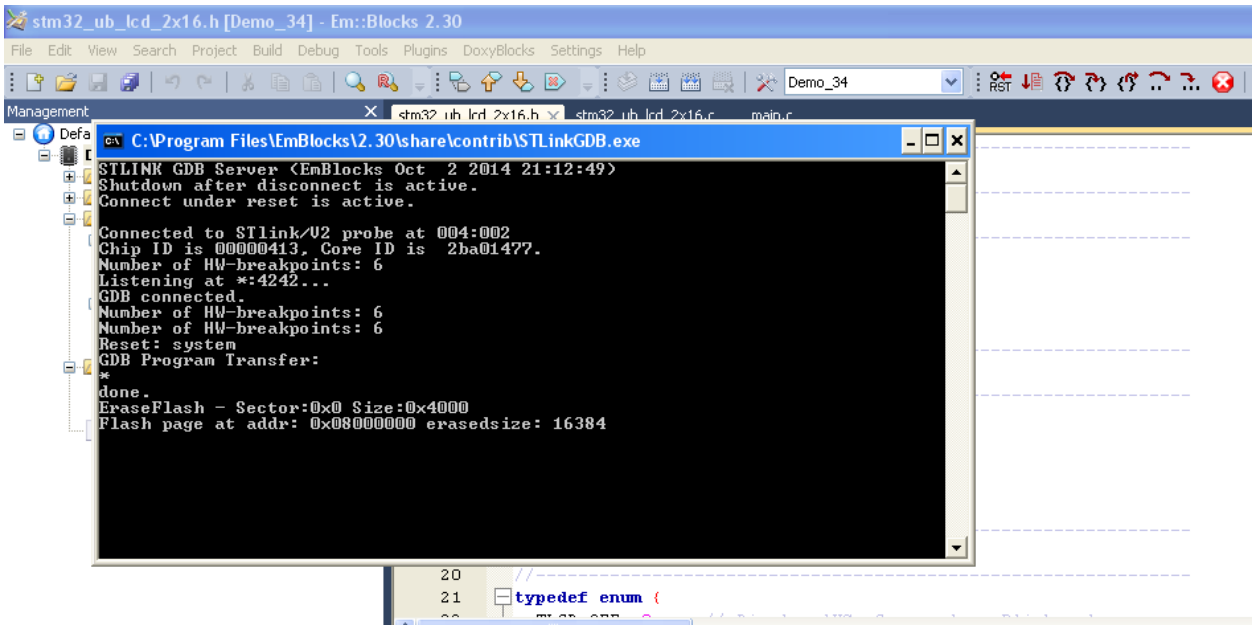
Ponowne wciśniecie klawisza **F7** powinno zakończyć etap budowania programu następującym komunikatem:



W komunikacie tym zawarta jest kwintesencja działania kompilatora i linkera i widać ile pamięci zostanie zużytej na jego alokację. Tym samym możemy załadować nasz program do pamięci procesora STM32F407VG i rozpocząć jego testowanie.

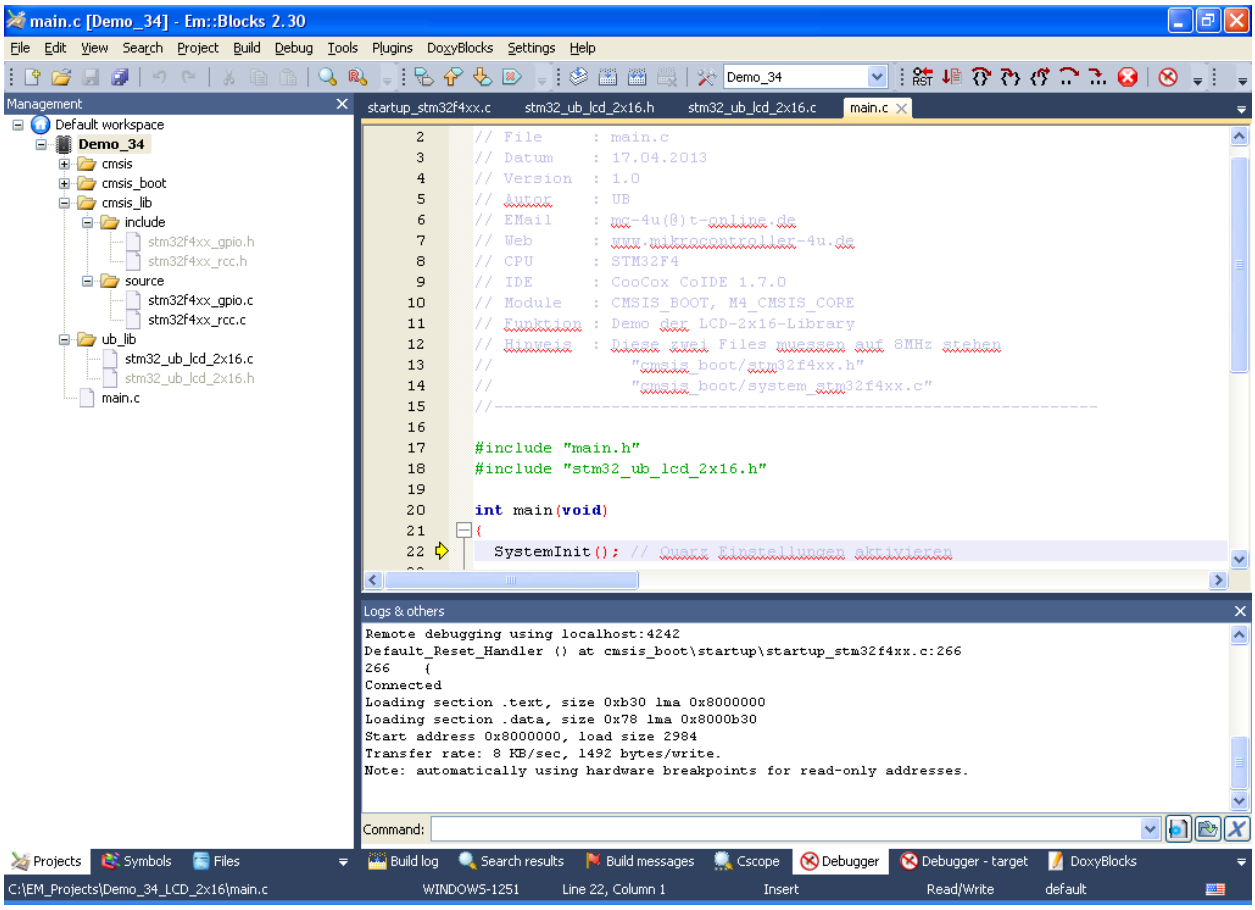
Pamiętaj, że ST-Link nie jest potrzebny do ładowania i testowania programu w EMBlocks. Jeśli jest otwarty, trzeba go zamknąć, w przeciwnym wypadku nie uda się uruchomić serwera GDB!

Ładujemy program klawiszem F8 (lub z menu **Debug** – **Start/Stop Debug Session**) co zostanie potwierdzone wyświetlenie okna DOS z komunikatami programatora w którym widać przebieg operacji programowania.



Po jego zakończeniu wskaźnik krokowej pracy programu (żółta strzałka w linii 22) zostanie

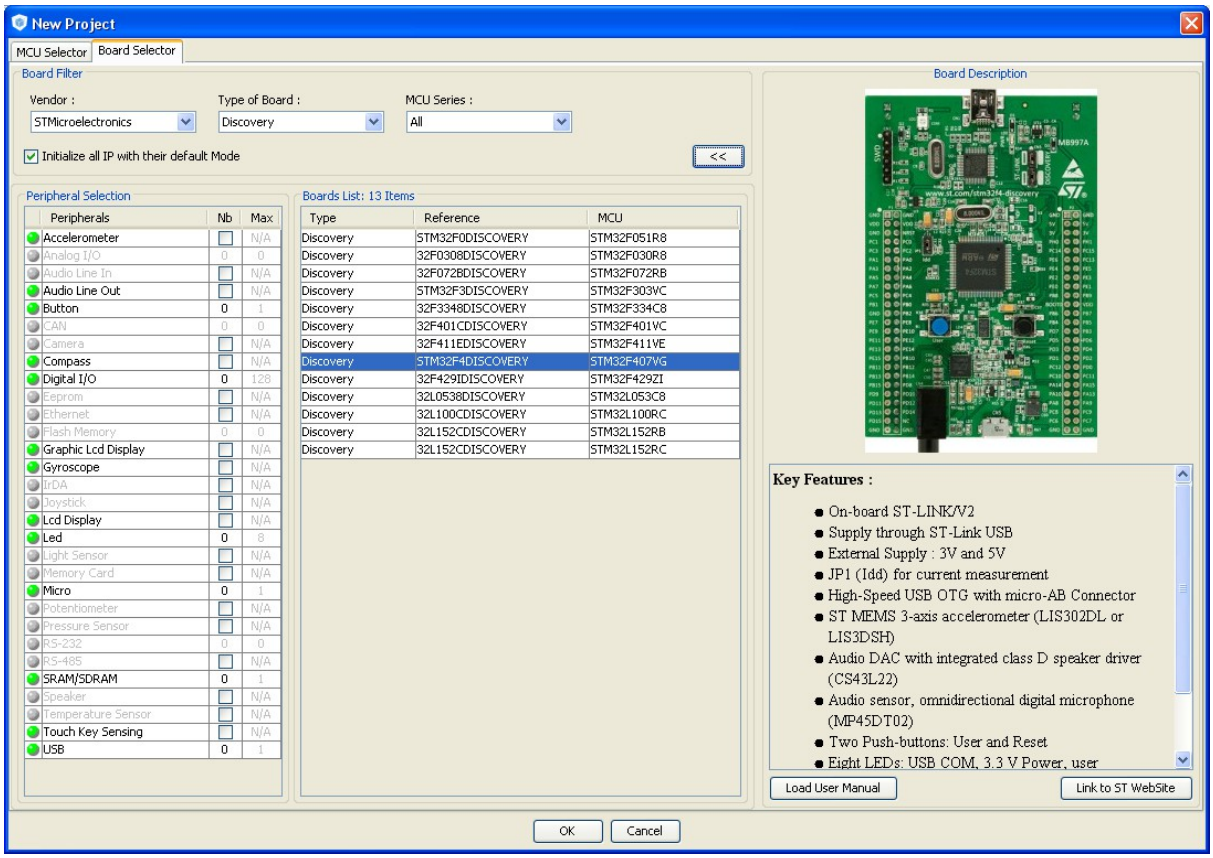
wyświetlony w głównej funkcji Main() i będzie czekał na polecenia debugera. Jednocześnie aktywne staną się klawisze debugera w prawym górnym roku, kolejno: reset procesora, kontynuację programu, następny krok, wejście do podprogramu, wyjście z podprogramu, dwie komendy pracy krokowej asemblera i wreszcie, jako ostatni, zatrzymanie pracy debugera i powrót do edytora.



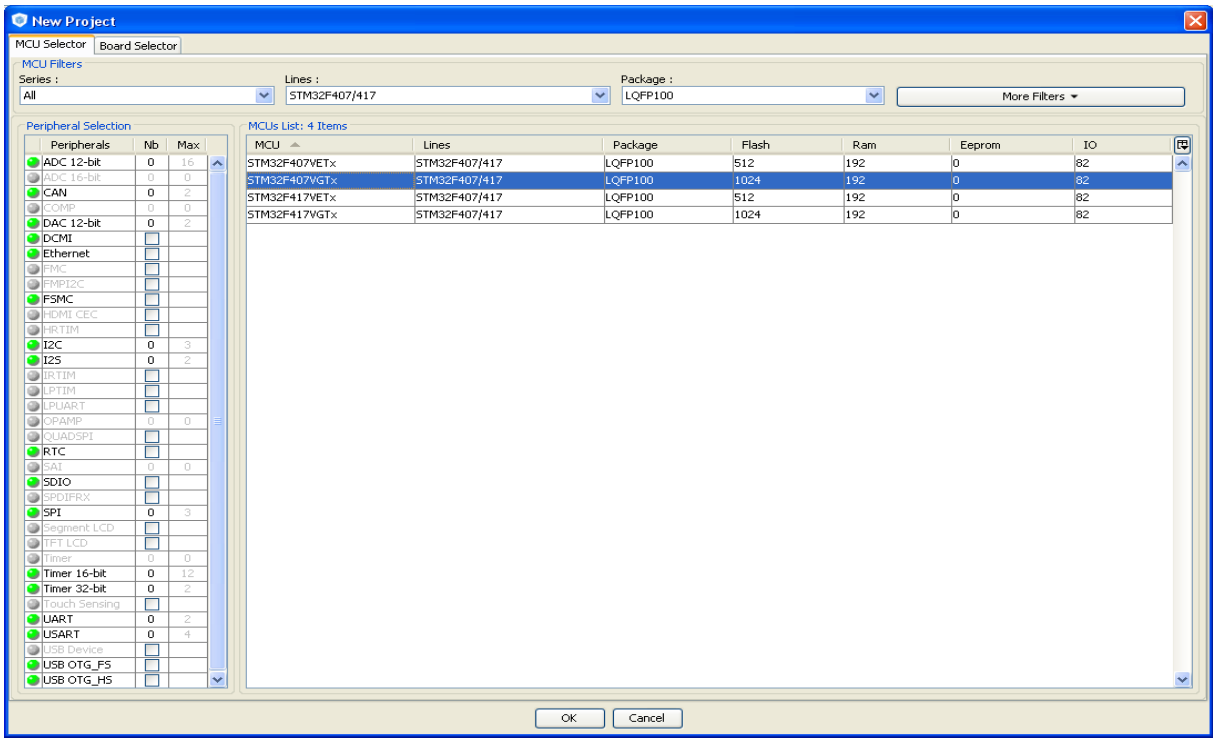
STM32CubeMX

Dla kogoś, kto tak jak ja zaczyna przygodę z procesorami STM32 „od zera” przebrnięcie przez dokumentację liczącą 1700 stron wydaje się zadaniem nie do wykonania a napisanie i uruchomienie najprostszego programu do zapalenia diody LED podłączonej do portu procesora wprawia w frustrację trudną do opanowania. Na szczęście jest na to lekarstwo. STM32CubeMX sprawi, że będziemy mogli ogarnąć zasoby, zdefiniować porty, ustawić ich taktowanie i wygenerować podstawowe kody inicjujące je w procesorze. Pobierze też dla nas standardowe biblioteki oferowane przez ST i zadba o ich aktualność.

Jak zwykle pobieramy program , rozpakowujemy i uruchamiamy instalację. Przy okazji warto pobrać też dokument szczegółowo opisujący założenia i działanie programu. Instalacja nie wymaga żadnych modyfikacji, instalujemy z domyślnie przyjętymi założeniami. Następnie uruchamiamy program i naciskamy **CTRL+N** aby utworzyć nowy projekt. Jeśli mamy zestaw ewaluacyjny wybieramy zakładkę **Board Selector** gdzie możemy wybrać posiadany model i predefinowane nastawy dla zasobów płytki. Po zatwierdzeniu będziemy mogli dokonać edycji projektu.



Jeśli chcemy samodzielnie zdefiniować założenia projektu wybieramy zakładkę **MCU Selector** i rozpoczynamy od wyboru procesora z listy. Możemy posłużyć się filtrami serii, linii, obudowy a także zasobów a program podpowie żądany symbol.

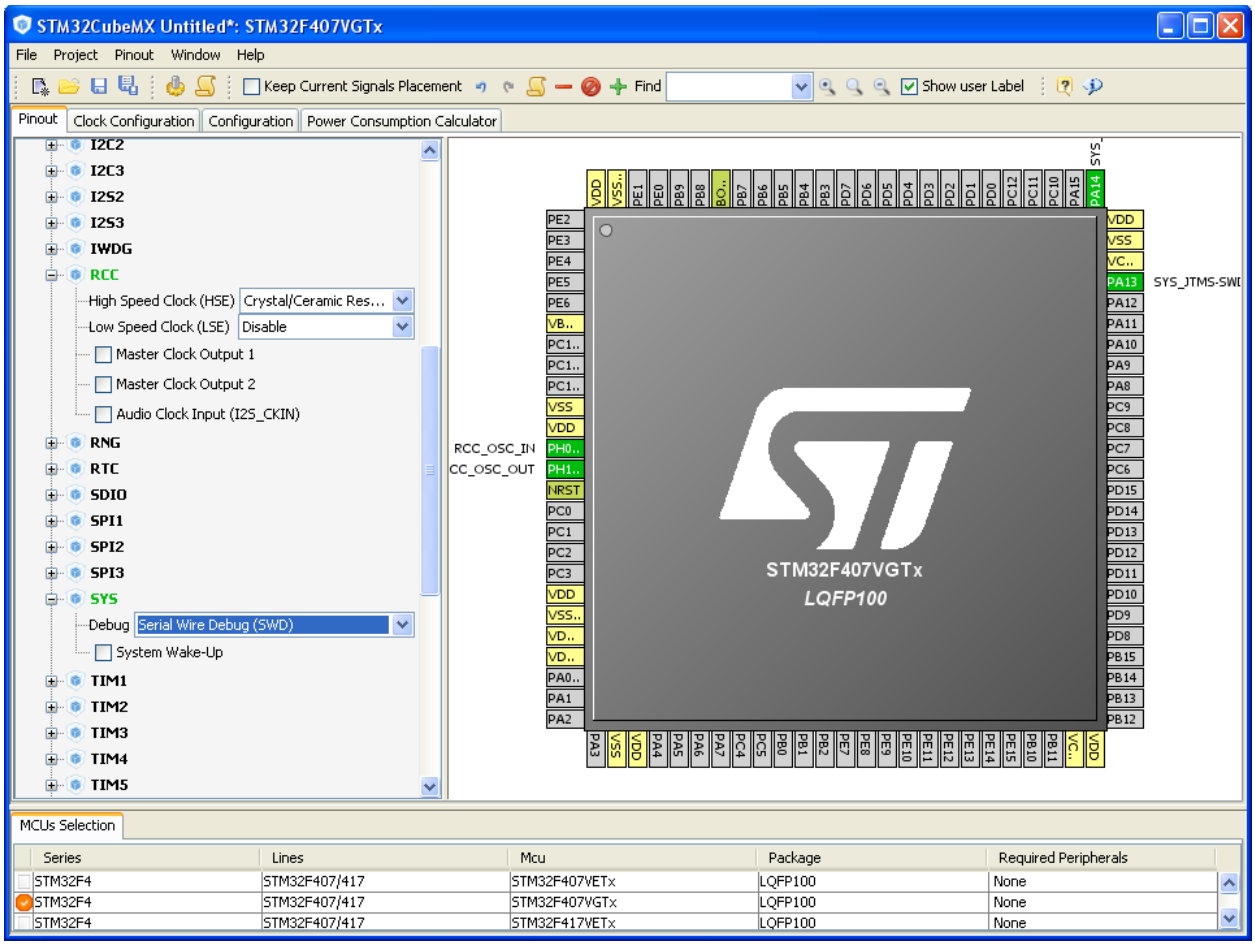


Po zatwierdzeniu wyboru otwiera się okno zawierające podstawowe zakładki projektu:

- **Pinout** pozwalającą podłączyć zasoby do nóżek procesora,
- **Clock Configuration** służącą do konfiguracji taktowania zasobów,
- **Configuration**, która pozwala doprecyzować nastawy zasobów, w tym przerwania i ich priorytety,
- **Power Consumption Calculator** potrafiący oszacować zużycie energii przez procesor na podstawie przyjętych założeń.

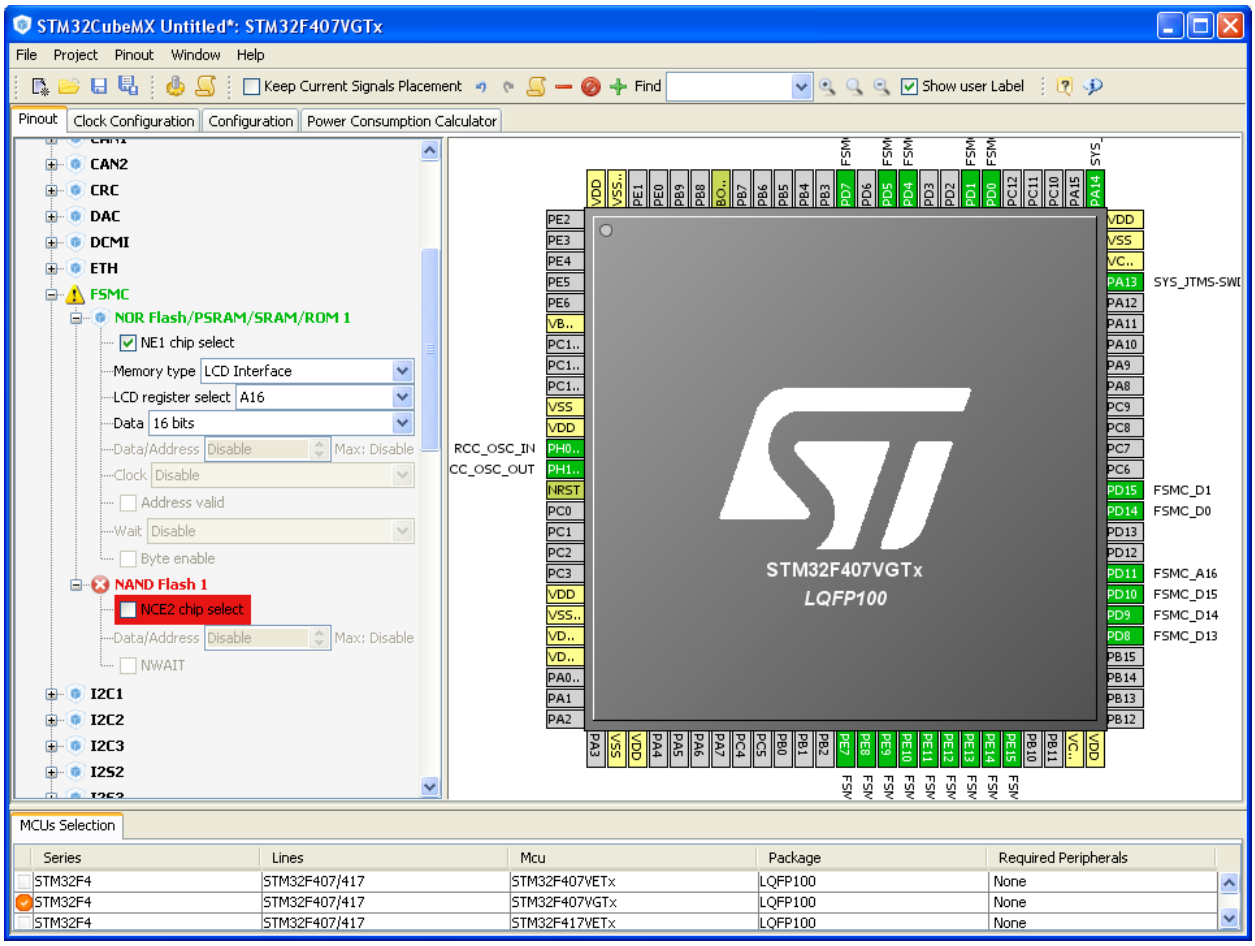
W oknie po prawej stronie mamy rozpisane zasoby procesora, w oknie po lewej rzeczywisty widok obudowy i nóżek procesora. Projektowanie polega na wyborze zasobu i zaznaczeniu odpowiednich opcji a program dokona przypisania ich do nóżek procesora, z uwzględnieniem ewentualnych konfliktów co bardzo ułatwia ten etap projektu.

Rozpoczynamy od wyboru źródła taktowania procesora. Rozwijamy listę zasobu **RCC.** Jeśli chcemy wykorzystać rezonator kwarcowy, wybieramy z listy odpowiednią opcję. Mamy do dyspozycji pozostałe źródła taktowania i wyprowadzania zegarów taktujących na nóżki procesora. Na tym etapie możemy też zdefiniować styk procesora z debugerem. Jeśli jest to płytka STM32F4Discovery wybieramy **SWD** z zasobu **SYS**.



Jak widzimy w prawym oknie program „zapalił” na zielono cztery nóżki procesora opisując je odpowiednio nazwą zasobu. W ten sposób możemy wizualnie ocenić postęp projektu.

Przykładowo uaktywniamy interfejs FMSC dla podłączenia wyświetlacza LCD z magistralą Intel 8080. Rozwijamy listę z okna po lewej dla FSMC i wybieramy stosowne opcje a program zbada czy nie ma konfliktu z innymi zasobami i w razie czego zasygnalizuje to kolorem i pozwoli na wybór innego niż domyślnie przypisania zasobów do nóżek.



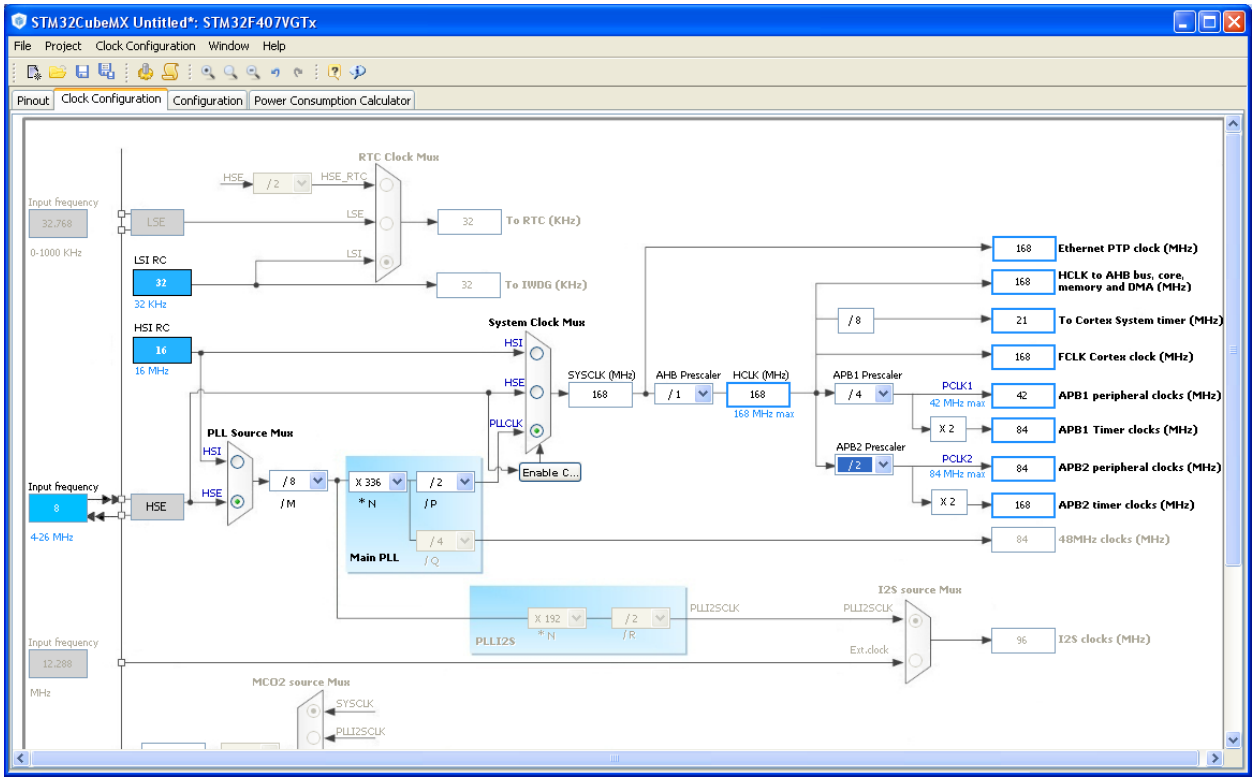
Jak widać program zasygnalizował, że nie ma możliwości podłączenia pamięci typu NAND ze względu na zajęcie **NE2** przez interfejs.

Szczegóły dotyczące zasad deklaracji i przypisywania zasobów znajdziecie w dokumencie pobranym ze wskazanej wcześniej lokalizacji.

Kolejnym krokiem projektu jest przypisanie taktowania do zasobów. STM32 wymaga określenia w jaki sposób zostaną uaktywnione zasoby przez podanie odpowiednich zegarów. W zakładce **Clock Configuration** mamy do dyspozycji różne źródła i przełączniki zegarów. Przykładowa nastawa dla kwarcu 8 MHz i częstotliwości taktowania rdzenia 168 MHz na obrazku poniżej. Szczególną uwagę należy zwrócić na nastawy dla I2S i Ethernetu bowiem istnieje wiele kombinacji z pozoru dających właściwe nastawy jednak wzajemne zależności często się wykluczają. Na forach sporo dyskusji na ten temat. W razie wątpliwości niestety trzeba sięgnąć do manuala dla konkretnego procesora.

W zakładce **Configuration** mamy wypisane zasoby pod klawiszami wyboru gdzie dokonujemy uszczegółowienia nastaw z uwzględnieniem przerwań i ich hierarchii. Znajdą one odzwierciedlenie w kodach generowanych przez program. Możemy więc przyjąć jakieś wstępne założenia a potem je skorygować bowiem kody zawierają sekcje podlegające i nie podlegające odświeżeniu

przy zmianie nastaw (sekcje użytkownika i programu).



Wspomnę tylko, że ostatnia zakładka Power Consumption Calculator pozwala na oszacowanie sumarycznego poboru rdzenia i zasobów dla konkretnych przyjętych nastaw zegarów a tym samym zdecydować czy założenia projektowe zostały spełnione. Nic nie wiem na temat jakości tego szacunku.

Pora wreszcie na ostateczny moment działania programu czyli wygenerowania kodów w postaci plików źródłowych i nagłówkowych projektu. Najpierw jednak trzeba dokonać wyboru w jakiej konwencji zostaną one wygenerowane i gdzie umieszczone a także gdzie są umieszczone standardowe biblioteki dostarczane przez firmę ST.

Kombinacją **ALT+P** otwieramy okno Project Settings konfiguracji programu i w zakładce Project nadajemy nazwę projektu i wskazujemy gdzie zostaną umieszczone pliki wygenerowane przez program np. w **C:/EM_Projects**. Z listy **ToolChain/IDE** wybieramy **TrueSTUDIO**.

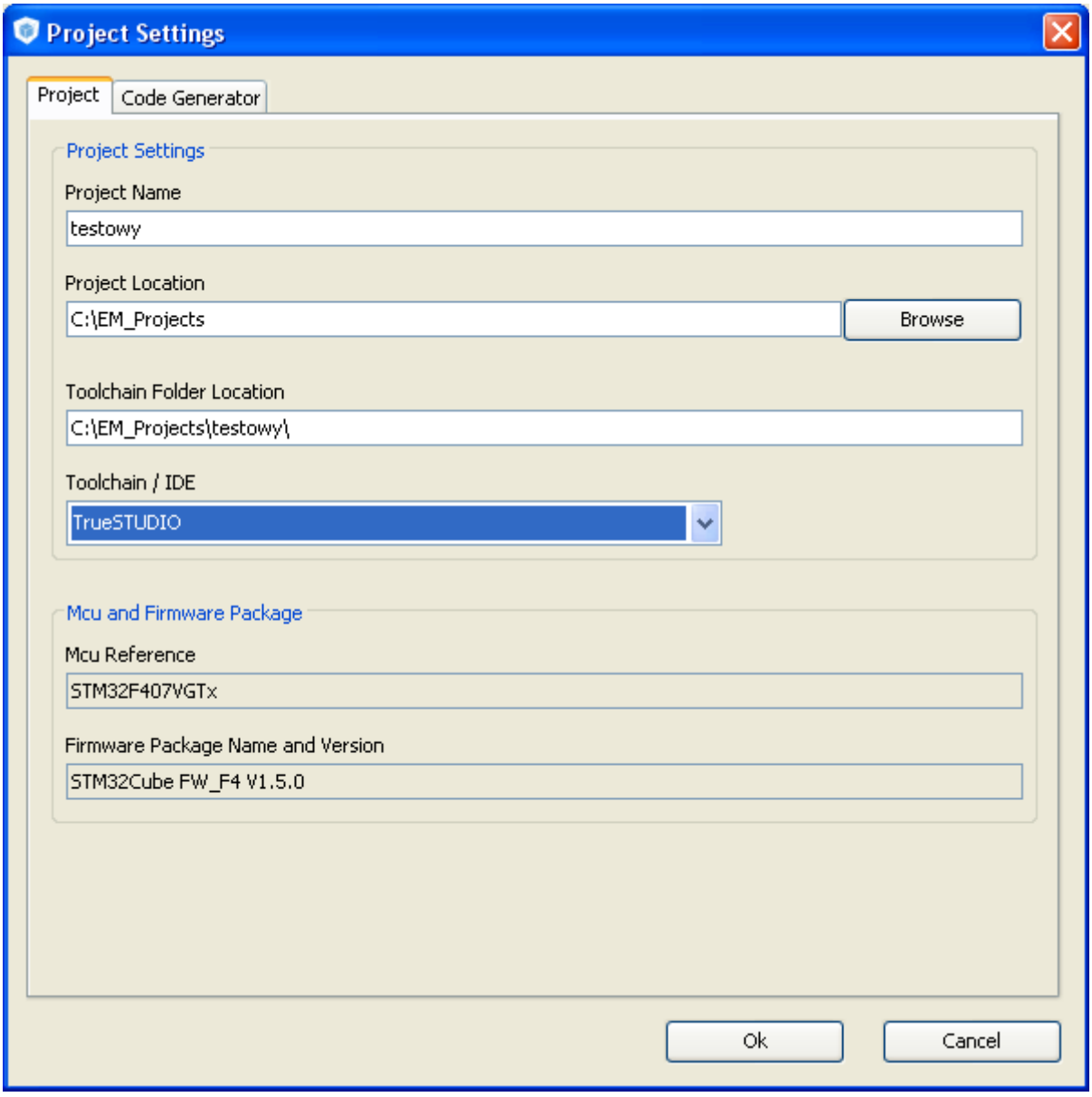
To ważne bo tylko ta konwencja pozwoli w następnym kroku na konwersję do formatu EmBlocks!

W zakładce **Code Generator** zaznaczamy opcję **Copy only the necessary library files** oraz **Generate peripheral initialisation as pair of '.c/.h' per IP**. Pierwsza z nich pozwala znacząco zmniejszyć czas i objętość wygenerowanego projektu poprzez pominięcie zbędnych bibliotek, druga ma znaczenie czysto umowne dla języka C.

Jeśli chcemy włączyć opcję pełnego śledzenia wykonania programu zaznaczamy dodatkową opcję **Enable Full Assert.** Możemy też włączyć kod ustawiający wszystkie nieużywane nóżki procesora jako wejścia dla obniżenia poboru prądu.

Ostatnią czynnością jaką mamy do wykonania przed wygenerowaniem kodu jest ustawienie miejsca gdzie ulokowane są standardowe biblioteki producenta. Jest to o tyle ważne, że kompilator będzie je musiał za każdym razem odszukać i pobrać, jeśli ścieżka dostępu jest długa i

zagmatwana, w listingach będzie nieczytelna a i może się zdarzyć, że będzie zawierać niedozwolone znaki lub spacje przyprawiające o ból głowy. Dlatego proponuję zmienić ją na C:/Repository i tam przenieść biblioteki z lokalizacji domyślnej. Zrobimy to po komendzie **Alt+S** wybierając utworzony wcześniej folder. Możemy też zmienić nastawy dotyczące aktualizacji i połączenia do Internetu



Jeśli wszystko mamy już ustawione wybieramy **Ctrl+Shift+G** uruchamiając generator kodów a po zakończeniu możemy otworzyć folder zawierający pliki źródłowe i nagłówkowe, niezbędne biblioteki i pliki projektu STM32CubeMX. **Ctrl+R** pozwala wygenerować szczegółową dokumentację projektu. Pozostaje nam tylko dokonać konwersji formatu projektu z konwencji TrueSTUDIO na format EmBlocks. Dokonamy tego skryptem CubeMC2EmBlocks z Forum EmBlocks. W ten sposób możliwe będzie włączenie kodów inicjacji procesora do całego powstającego projektu i jego uruchomienie w środowisku EmBlocks.

Skrypt CubeMX2EmBlocks

Pobieramy skrypt, umieszczamy go np. w C:/EM_Projects i uruchamiamy. Wystarczy teraz wskazać folder zawierający nasz projekt a skrypt dokona jego konwersji i zasygnalizuje poprawny koniec stosownym komunikatem. **TO WSZYSTKO!**

Otwieramy plik projektu z rozszerzeniem **.ebp** i możemy go skompilować by sprawdzić czy wszystko „OK" po czym przystąpić do tworzenia naszego programu.

POWODZENIA !

73 – Piotr

SP9FKP

Przydatne odnośniki