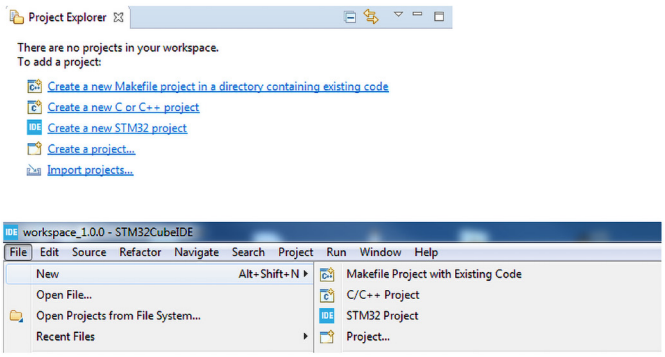
Środowisko projektowe STM32CubeIDE

Pisanie oprogramowania od zawsze wymagało przynajmniej dwu elementów: niezbędnej wiedzy i odpowiednich narzędzi. Narzędzia projektowe tak jak i mikrokontrolery przeszły od początku istnienia olbrzymie przeobrażenia. Dawno temu szczytem marzeń był kompilator asemblera z modułami linkera i bibliotekarza. Wprawny programista używający prostego edytora tekstowego i tych narzędzi potrafił pisać skomplikowane i zaawansowane programy. Jeżeli przy tym napisał sobie odpowiednią „bazę" programów bibliotecznych, to wydajność takiego programowania mogła być jak na tamte standardy wysoka. Wraz z rozwojem elementów, ich powszednieniem i spadkiem ceny okazywało się, że głównym składnikiem kosztów dowolnego urządzenia zawierającego mikroprocesor staje się czas programisty. Skrócenie czasu pracy było możliwe z jednej strony dzięki wprowadzaniu kompilatorów języka wyższego rzędu – najczęściej C, a z drugiej strony dzięki możliwym do kupienia komercyjnym bibliotekom. Na początku i kompilatory, i biblioteki były tak drogie, że mogły sobie na nie pozwolić tylko duże bogate firmy. Ale w pewnym czasie do gry weszły firmy produkujące mikrokontrolery. Okazało się, że opłaca się dostarczać bezpłatnie środowiska i programy narzędziowe po to, by zwiększyć popyt na produkowane mikrokontrolery.

Dzisiaj jest tak, że programiści mają nielimitowany, bezpłatny dostęp do wyrafinowanych narzędzi projektowych: środowisk IDE, konfiguratorów układów peryferyjnych, kompilatorów języka C/C++ oraz coraz lepszych bibliotek. Te ostatnie to nie tylko biblioteki obsługujące układy peryferyjne łącznie z warstwą HAL, ale też middleware takie jak obsługa systemu plików FAT, biblioteki graficzne, stos TCP/IP, Bluetooth, w tym BLE itp.

Większość dużych firm oferuje własne środowiska projektowe z kompilatorami i konfiguratorami, warto tu wymienić Microchipa, czy Renesasa. Do tej pory wyjątkiem był ST oferujący jedne z najbardziej rozpoznawalnych mikrokontrolerów STM32. Programiści programujący STM32 mają do dyspozycji kilka środowisk, od znanego i uznanego µVision Firmy Keil, poprzez dostępne bezpłatnie Atollic True Studio for STM32 czy System Workbench for STM32. Dostępny jest też konfigurator STM32CubeMX. Jednak nie było kompletnego środowiska integrującego wszystko w jednym miejscu jak to jest na przykład w MPLAB X Microchipa czy e2studio Renesasa. Plug-in STM32CubeMX, który teoretycznie można zintegrować na przykład z Atollic True Studio, miał wiele błędów i był praktycznie nie do użycia. Trochę to dziwne, bo STM32CubeMX uruchamiany samodzielnie działał bardzo dobrze. Teraz ma się to zmienić, bo ST zaoferował swoje własne IDE, nazwane STM32CubeIDE.



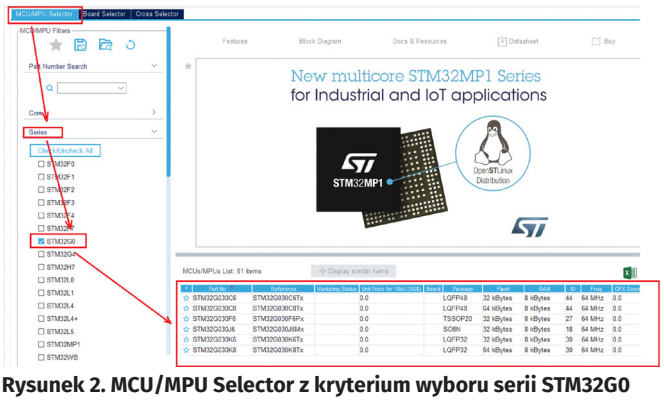
Rysunek 1. Opcje tworzenia nowego projektu

STM32CubeIDE jest zaawansowaną platformą projektową opartą na frameworku Eclipse/CDT przeznaczoną do tworzenia projektów w języku C/C++ i zawiera zestaw narzędzi przeznaczony do edycji kompilowania i debugowania kodu. Kompilator C/C++ wykorzystuje GCC (GNU Compiller Collection), a debugger o GDB (GNU Project Debugger). Co najważniejsze, w końcu STM32Cube IDE jest ściśle zintegrowany z konfiguratorem STM32CubeMX.

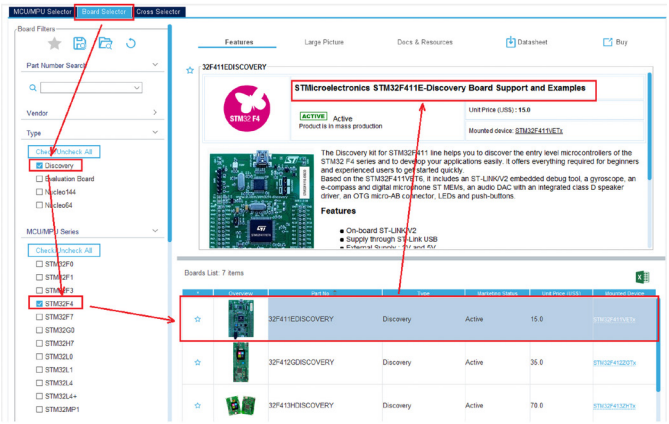
Producent przygotował od razu wersje instalacyjne dla różnych dystrybucji Linuksa, systemu macOS i oczywiście Windowsa. Po ściągnięciu wersji instalacyjnej, zainstalowaniu i uruchomieniu pojawia się okno Project Explorer z kilkoma opcjami rozpoczęcia pracy z nowym projektem. To samo można osiągnąć, używając menu File → New – **rysunek 1**.

Dla nas najbardziej interesującą będzie opcja STM32 Project, bo automatycznie uruchamia kreator projektu znany z STM32CubeMX. Na początku jest uruchamiany selektor mikrokontrolera z zakładkami wyboru:

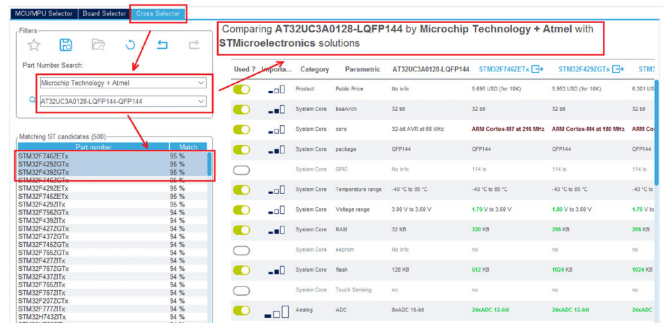
- MCU/MPU Selector przeznaczony do wyboru mikrokontrolera projektu według kryterium właściwości i wyposażenia w układy peryferyjne mikrokontrolerów STM32,
- Board Selector przeznaczony do wyboru mikrokontrolera projektu zamontowanego na wybranej płycie ewaluacyjnej,
- Cross Selector – nowe okno przeznaczone do wyboru mikrokontrolera STM32 o parametrach zbliżonych do mikrokontrolerów innych producentów (Microchip Atmel, Renesas, Infineon, NXP itp.).



Rysunek 2. MCU/MPU Selector z kryterium wyboru serii STM32G0



Rysunek 3. Board Selector



Rysunek 4. Zakładka Cross Selector

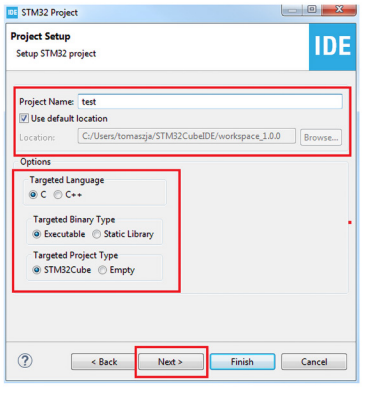
MCU/MPU Selector daje największą swobodę wyboru mikrokontrolera według sporej liczby kryteriów:

- Typu rdzenia: ARM Cortex M0, ARM Cortex M3, ARM Cortex M4 itp.,
- Serii: STM32F0, STM32G0, STM32F4 itp.,
- Obudowy,
- Pozostałych parametrów: ceny, liczby linii I/O, pojemności pamięci, częstotliwości taktowania,
- Wyposażenia w układy peryferyjne.

Używanie tego selektora pokazuje, że lista produkowanych przez ST mikrokontrolerów jest naprawdę imponująca.

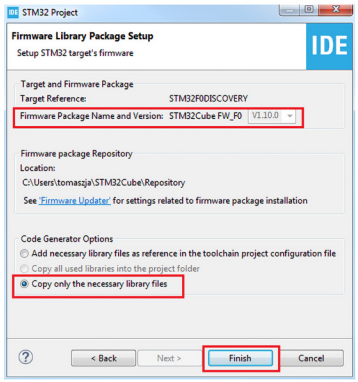
Board Selector wybiera mikrokontroler zabudowany w wybranej płycie ewaluacyjnej. ST oferuje szeroką gamę takich płyt z mikrokontrolerami wszystkich produkowanych serii i z bardzo różnym wyposażeniem, od bardzo prostych, elementów typu dioda i przycisk, po bardzo rozbudowane z dużymi wyświetlaczami LCD/TFT. Board Selector jest świetnym wsparciem dla projektanta wykorzystującego moduły firmowe. Na **rysunku 3** pokazano selektor wyboru modułu z kryteriami: typ modułu Discovery i seria MPU STM32F4.

Zakładka wyboru Cross Selector pomaga projektantowi dobrać mikrokontroler STM32 na podstawie parametrów technicznych wybranych mikrokontrolerów innych producentów. W sytuacji kiedy projektant nie jest przywiązany do konkretnego producenta mikrokontrolera i może wybierać różne opcje, Cross Selector pozwala na szybkie i wygodne porównanie mikrokontrolera STM32 z wyświetleniem procentowej „zgodności" oraz graficznym przedstawieniem takich samych właściwości i różnic.



Rysunek 5. Okno ustawień projektu

Na **rysunku 4** pokazano, jak Cross Selector wybrał zamienniki mikrokontrolera AT32UC3A0128 produkowanego przez Microchipa (Atmel). Pomimo różnic w rdzeniu (32-bit AVR vs Cortex M) uzyskano 95% zgodności funkcjonalnej. Żeby wykonać taką analizę ręcznie trzeba by było poświęcić na nią sporo czasu.



Rysunek 6. Ustawienia dotyczące bibliotek firmowych

Po wybraniu mikrokontrolera przez selektor przechodzimy do okna ustawień projektu (**rysunek 5**), w którym:

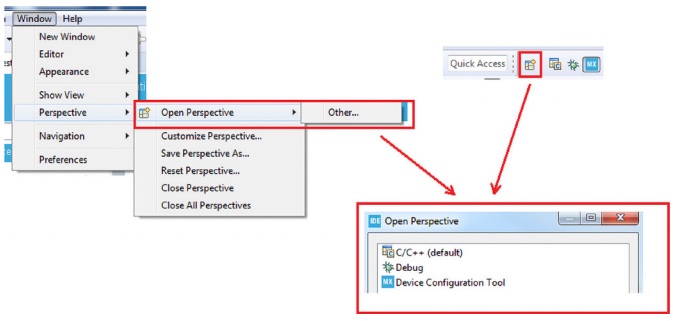
- Nadajemy nazwę projektu i opcjonalnie zmieniamy domyślną ścieżkę dostępu do katalogu projektu,
- Wybieramy język programowania C lub C++,
- Wybieramy rodzaj pliku wynikowego: wykonywalny (hex) lub biblioteczny,
- Generowany szkielet programu: pusty lub na podstawie ustawień konfiguratora STM32CubeMX.

Kliknięcie na przycisk Finish kończy działanie kreatora projektu, a kliknięcie na Next otwiera okno z ustawieniami dotyczącymi firmowych bibliotek dla STM32 (**rysunek 6**). W chwili obecnej można wybrać tylko paczkę z bibliotekami w wersji v1.10.0. Opcje generowania kodu określają sposób dołączania plików bibliotecznych do projektu. W przypadku pracy z STM32CubeMX warto zaznaczyć opcje kopiowania do projektu tylko potrzebnych plików bibliotecznych.

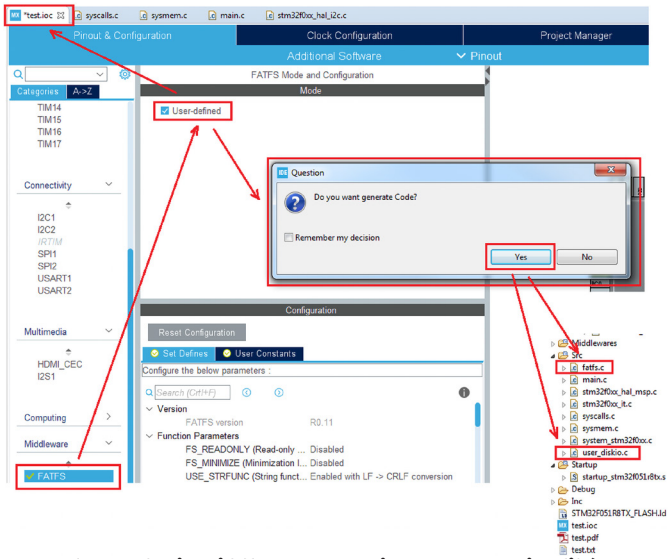
Kliknięcie na przycisk Finish powoduje przejście do kolejnego etapu. Jeżeli uruchamiamy projekt pierwszy raz, to program łączy się z serwerami ST przez Internet i pobiera paczkę z firmware'em dla wybranego mikrokontrolera. Po pobraniu i rozpakowaniu otwiera się okno środowiska STM32CubeIDE.

Środowiska IDE oparte na Eclipse wyglądają podobnie i jeżeli ktoś używał podobnego, to nie będzie miał żadnego problemu z STM32CubeIDE. Ale Eclipse jest dość intuicyjne i nawet dla użytkownikowi mającemu pierwszy z nim kontakt praca nie powinna sprawiać większych trudności.

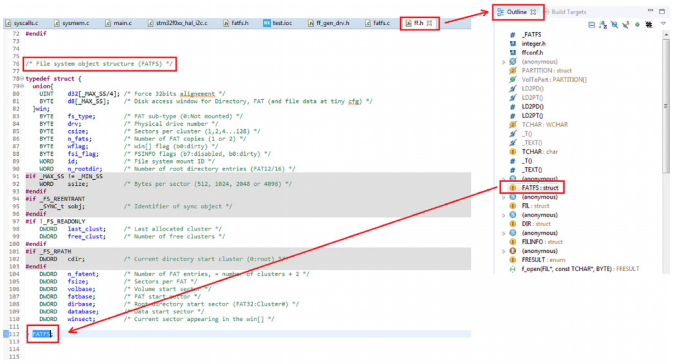
Każde środowisko IDE opiera się na idei projektu skupiającego wszystkie pliki źródłowe, nagłówkowe, typu makefile, ustawienia linkera itp. Praca z projektem wymaga edycji plików źródłowych, kompilacji z obsługą ostrzeżeń i błędów oraz możliwości debugowania i w końcu zaprogramowania pamięci kodem wynikowym. Nowsze pakiety mają również wbudowane konfiguratory peryferii i narzędzia ułatwiające dołączenie i konfigurowanie middleware – na przykład bibliotek graficznych. Żeby ułatwić pracę programistom, Eclipse wprowadza pojęcie pespektywy. Perspektywa



Rysunek 7. Wybór aktywnej pespektywy



Rysunek 8. Dodanie middleware FATFS i wygenerowanie plików źródłowych przez konfigurator



Rysunek 9. Wyszukiwanie definicji struktury za pomocą okna Outline

to zebranie w jednym miejscu wszystkich narzędzi przeznaczonych do pracy nad określonym etapem projektu. Standardowo są dostępne dwie główne perspektywy:

- C/C++ przeznaczona do pracy nad kodem projektu – edycją plików źródłowych i kompilacją,
- Debug – przeznaczona do uruchamiania (debugowania) wygenerowanego kodu.

Te dwie perspektywy są uzupełniane o trzecią przeznaczoną do konfiguracji układów peryferyjnych i dodawania middleware. STAM32CubeIDE oferuje wszystkie trzy perspektywy przełączane z menu Window → Pespective → Open Pespective → Other lub z paska narzędziowego umieszczonego w prawym górnym rogu ekranu – **rysunek 7**.

Wybranie pespektywy Device Configuration Tool otwiera okno z konfiguratorem peryferii STM32CubeMX. Konfigurator działa tak jak niezależnie uruchamiany STM32CubeMX, tylko bez etapu wyboru mikrokontrolera. Jak wiemy, etap wyboru mikrokontrolera działający tak samo jak w niezależnym STM32CubeMX jest uruchamiany w kreatorze projektu na samym początku.

Konfigurator pozwala na skonfigurowanie i dodanie do projektu plików konfiguracyjnych:

- Wyprowadzeń mikrokontrolera, czyli przypisanie im funkcji portu I/O, lub alternatywnych funkcji wyprowadzeń układów peryferyjnych,
- Układu taktowania mikrokontrolera,
- Układów peryferyjnych podzielonych funkcjonalnie na układy analogowe (DAC, DC komparator itp.), układy transmisji szeregowej (SPI, I²C, UART itp.), timery, multimedia (port I²S, port HDMI) itp. zależnie od wyposażenia mikrokontrolera,
- Middleware FAT, RTOS itp.

Opisywany niedawno na łamach „Elektroniki Praktycznej" konfigurator STM32CubeMX w wersji v5, jeżeli nie jest identyczny, to jest bardzo podobny do konfiguratora z perspektywy Device Configurator Tool zaimplementowanego w STM32CubeIDE. Dlatego nie będę powielał szczegółowego opisu działania poszczególnych funkcji. Zmiana konfiguracji wykonywana w perspektywie Device Configuration Tool może być wykonywana w dowolnym momencie pracy nad projektem. Kompletna informacja o konfiguracji jest zapisywana w pliku o rozszerzeniu .ioc umieszczonym w katalogu projektu Inc. Jeżeli zmienimy konfigurację, to ten plik ma status „nie zapisano zmian". Przy każdej próbie zapisania program pyta się o to, czy wygenerować piki zmienionej konfiguracji. Można to zrobić jawnie przez polecenie save lub też przez wydanie polecenia kompilacji (build) projektu. Na **rysunku 8** pokazano dodanie systemu plików FATFS do konfiguracji projektu. Plik test.ioc różni się od ostatnio zapisanego, co oznacza, że w konfiguratorze dokonano zmian. Przy wywołaniu polecenia build IDE otwiera okno z pytaniem, czy wygenerować kod na podstawie bieżącej konfiguracji projektu. Po kliknięciu na Yes konfigurator generuje nowy kod – zostało to również pokazane **rysunku 8**. W tym przypadku dodaje do projektu pliki fatfs.c i user_dikio.c.

Pespektywa C/C++, jak już wspomniałem, jest przeznaczona do edycji i kompilacji kodu. Standardowo z lewej strony okna programu jest umieszczone okno Project Explorer z podkatalogami i plikami projektu przeznaczone do szybkiej nawigacji. Obok tego okna jest umieszczane okno zaawansowanego edytora tekstowego przeznaczonego do edycji kodu C/C++. Edytor oferuje kolorowanie kodu (słowa kluczowe, komentarze, stałe itp.) oraz działający on-line kontroler poprawności syntaktycznej kodu. Po wykonaniu kompilacji edytor wyświetla w liniach kodu informacje o wykrytych błędach i ostrzeżeniach kompilacji. Do sprawnego poruszania się w plikach źródłowych można wykorzystać okno Outline. Jest w nim zawarta lista symboli definicji (ciał) funkcji, stałych, zmiennych, struktur itp. Kliknięcie na element listy przekierowuje na symbol w pliku źródłowym. Jest to bardzo przydatne do poszukiwania fragmentów kodu w dużych plikach źródłowych. Na **rysunku 9** jest pokazane wyszukiwanie struktury FATFS w pliku nagłówkowym ff.h.

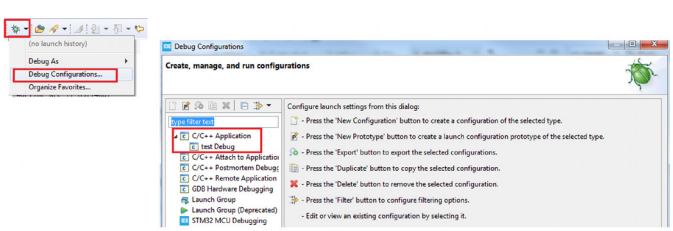
Kompilacja gotowego projektu ma dwie podstawowe konfiguracje:

- Debug – kod wynikowy jest przeznaczony do debugowania,
- Release – kod wynikowy jest kodem przeznaczonym do zaprogramowania ostatecznego pamięci Flash bez możliwości prawidłowego debugowania.

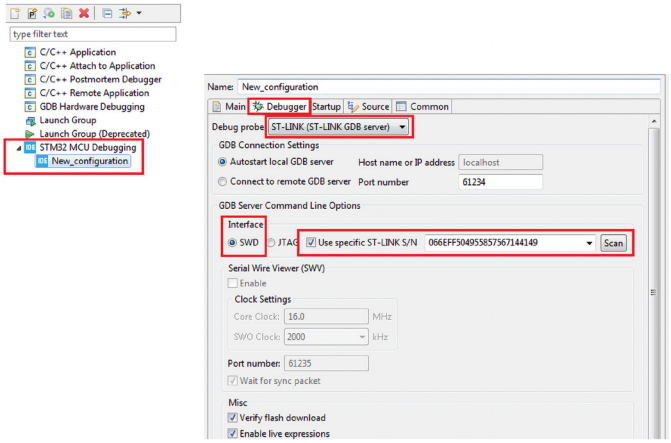
Konfiguracja Debug jest konfigurację podstawową używaną w trakcie pracy nad kodem. Kiedy kod jest gotowy i pozbawiony błędów, trzeba go skompilować jako Release i zaprogramować nim ostatecznie pamięć programu.

Kompilacja może obejmować tylko pliki źródłowe, w których od ostatniej kompilacji wykonano zmiany – wykonuje się zazwyczaj szybciej. Możliwa jest tez kompilacja wszystkiego od nowa z uprzednim usunięciem wszystkich plików obiektowych. Przebieg i wynik kompilacji jest wyświetlany w oknie Console.

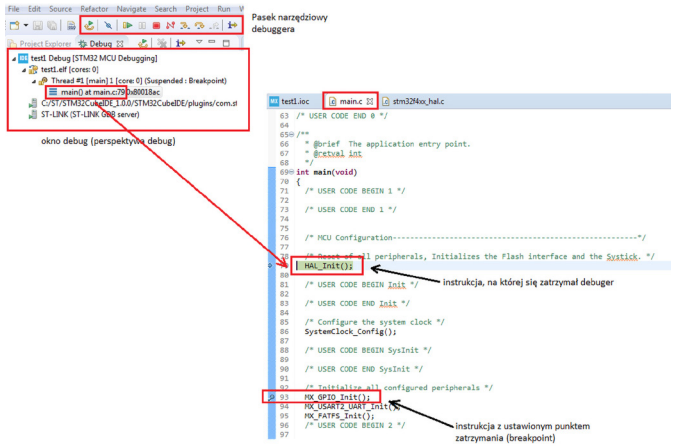
Jedną z podstawowych czynności pracy z kodem jest jego debugowanie, czyli sprawdzanie poprawności działania i usuwanie błędów. Metod debugowania jest wiele. Jedną z prostszych, ale



Rysunek 10. Dodanie projektu do konfiguratora programowego debuggera



Rysunek 11. Konfiguracja sprzętowego debuggera



Rysunek 12. Okno debuggera

zadziwiająco użytecznych jest umieszczanie w kodzie instrukcji typu printf i wyświetlanie za jej pomocą wartości zmiennych czy stanu, w jakim jest program. Ma to też wady, bo zawsze potrzebny jest do tego celu albo port szeregowy (najczęściej UART), albo jakiś inny wyświetlacz. Procedury wyświetlania zaburzają pracę programu, bo potrzebują czasu na wykonanie i w kodzie docelowym ich nie będzie. Z tego powodu lepiej jest użyć standardowych sprzętowych rozwiązań.

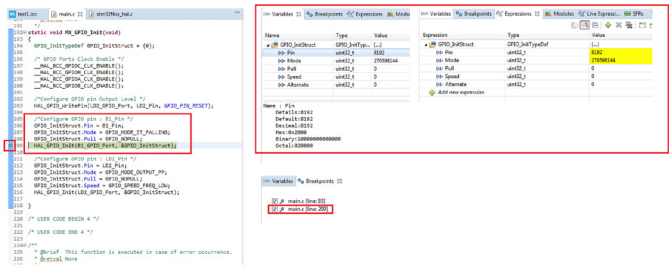
Debugowanie jest na tyle istotne, że mikrokontrolery zostały wyposażone w wewnętrzne układy przeznaczone do tego celu. Może to być standardowy interfejs JTAG lub jakieś własne rozwiązania. Interfejs debugowania współpracuje z układami programatorów/debuggerów i odpowiednio do tego celu generowanym kodem – jak już wspomniałem, w naszym przypadku będzie to kompilowane w konfiguracji Debug. W STM32CubeIDE do debugowania używana jest perspektywa Debug.

Pierwszym krokiem w przypadku każdego nowego projektu musi być skonfigurowanie opcji debuggowania. W tym celu klikamy na ikonkę z zielonym robaczkiem i wybieramy opcję Debug Configurations. Jeżeli mamy zamiar używać programowego symulatora klikamy dwa razy na C/C++ Application. Konfigurator automatycznie doda nasz projekt i będziemy mogli go sobie dalej konfigurować, lub używać do symulacji działania programu (**rysunek 10**).

Dużo lepszym pomysłem jest zastosowanie sprzętowego symulatora ST-Link z interfejsem SWD, bo taki jest wbudowany w praktycznie każdy firmowy moduł ewaluacyjny. Dostępne są również niezależne programatory innych producentów. Żeby skonfigurować sprzętową symulację działania programu, trzeba w oknie Debug Configuration kliknąć 2× na STM32 MCU Debugging. W oknie konfiguracji przechodzimy do zakładki Debugger. W oknie Debug Probe wybieramy ST-Link, w oknie Interface SWD. Jeżeli do komputera jest podłączony moduł z programatorem/debuggerem, to po zaznaczeniu USE specific ST-LINK S/N i kliknięciu na scan powinien się pojawić numer ID tego programatora. W trakcie testów do portu USB podłączyłem wybrany w selektorze projektu moduł ewaluacyjny. Zostało to pokazane na **rysunku 11**. Przed uruchomieniem debuggera IDE wykrył, że ST-Link zainstalowany w module ma wgrana starą wersję firmware'u i przy okazji została ona zaktualizowana.

Po skonfigurowaniu debugger jest gotowy do pracy. Otwieramy perspektywę Debug i program automatycznie się zatrzymuje na pierwszej instrukcji pliki main.c. Podstawowe czynności wykonywane w trakcie uruchamiania kodu to:

- Krokowe wykonywanie instrukcji języka C/C++ – Step Into (klawisz F5),
- Krokowe wykonywanie programu bez „wchodzenia" w wywoływane funkcje – Step Over (klawisz F6),



Rysunek 13. Wyświetlenie wartości zmiennej (struktury) zdefiniowanej w programie po zatrzymaniu na breakpont

- Uruchomienie ciągłego wykonywania programu Resume (klawisz F8),
- Zatrzymanie ciągłego wykonywania programu Terminate (klawisze Ctrl+F2),
- Wykonanie zerowania mikrokontrolera.

Istnieje możliwość krokowego wykonywania instrukcji asemblera. W krytycznych fragmentach kodu programista może sobie podejrzeć, jak kompilator implementuje instrukcje języka.

Debugger oferuje możliwość ustawiania i usuwania punktów zatrzymań (breakpoints) w liniach instrukcji języka. Ciągle wykonywany program po dotarciu do breakpoint zostaje zatrzymany.

W momencie, kiedy program jest zatrzymany, możemy podejrzeć wartości zmiennych zdeklarowanych w programie. Do tego celu używane jest okno watch – **rysunek 13**. Poza tym można sobie wyświetlić stan wszystkich bitów rejestrów SFR.

Podsumowanie

IDE firmowane przez ST to bardzo dobra wiadomość dla użytkowników mikrokontrolerów STM32. W końcu jest do dyspozycji kompletne środowisko integrujące selektor mikrokontrolerów STM32, znany już konfigurator układów peryferyjnych STM32CubeMX, edytor kodu źródłowego i debugger GDB.

IDE jest oparte na frameworku Eclipse/CDT i ma kompilator GCC bez ograniczeń wielkości kodu. Pierwsze wrażenie z użytkowania tego pakietu jest bardzo pozytywne. Wszystko działało poprawnie, co na przykład w przypadku debuggera nie musi być normą. Kreator projektu automatycznie pobierał paczkę z firmware'em i nie musiałem się martwić o to, by ją ręcznie pobrać i zainstalować (rozpakować). Gwarantuje to, że w momencie tworzenia projektu mamy do dyspozycji najbardziej aktualną wersję. ST zadbał też o możliwość importu projektów tworzonych dla innych środowisk IDE.

Oczywiście moje wstępne testy i przedstawiony tu opis nie wyczerpuje wszystkich możliwości pakietu, bo te są bardzo duże. Pewnie będą wypuszczane kolejne wersje z ulepszeniami i usuniętymi błędami.

Tomasz Jabłoński