Darmowe narzędzia programowe dla mikrokontrolerów STM32

Mikrokontrolery z rdzeniem ARM7 jeszcze na dobre nie zdążyły się zadomowić na naszym rynku, a już powoli zastępowane są przez mikrokontrolery z rdzeniem CORTEX-M3. Sukces ten zawdzięczamy głównie niskiej cenie, dużej wydajności, oraz optymalizacji pod względem oszczędności energii. Ze względu na bardzo dobry stosunek możliwości do ceny szczególnie dobrze zostały przyjęte mikrokontrolery STM32. Sukces rynkowy danego układu determinowany jest nie tylko przez jego możliwości, ale również przez dostępność odpowiednich narzędzi programowych i sprzętowych.

Dziś w zasadzie koniecznym warunkiem jest dostępność kompilatora C, a jeszcze lepiej C++ , interfejsu JTAG umożliwiającego wygodne programowanie układu w systemie, debugowanie oraz odpowiedniego edytora **IDE** (*Integrated Development Environment*) umożliwiającego wygodne pisanie oprogramowania. Gotowe zintegrowane środowiska programistyczne są produkowane przez wiele firm chociażby *KEIL* , *Rowley*, czy *Raisonance*. Środowiska te są bardzo wygodne w użyciu i umożliwiają utworzenie projektu dla wybranego mikrokontrolera w zasadzie za pomocą kilku ruchów myszką, niemniej jednak pełne wersje komercyjne są poza zasięgiem małych firm, nie wspominając o miłośnikach mikrokontrolerów. Istnieją co prawda wersje demonstracyjne wspomnianych wyżej narzędzi, ale na darmowe wersje są praktycznie prawie zawsze nałożone ograniczenia umożliwiające ich pełne wykorzystanie. Dawniej, wspomniane środowiska IDE najczęściej zawierały komercyjne kompilatory języka C, które niestety często były niezgodne ze standardem ISO-C. Na szczęście współczesne środowiska IDE nie wykorzystują komercyjnych odmian kompilatorów, a jedynie integrują doskonały, darmowy kompilator **GNU -GCC** (*Gnu Compiler Collection*), który umożliwia obsługę wielu języków programowania np. C, C++, Ada, Fortran, Java. W przypadku mikrokontrolerów interesować nas będą głównie języki C i C++. Jedynym znanym autorowi środowiskiem *IDE* dla ARMów, który zawiera własny kompilator, to Keil, ale i on umożliwia używanie GCC.

GCC przewyższył jakościowo wiele komercyjnych kompilatorów. Zapewnia on pełną zgodność ze standardem ISO C/C++ i trudno dziś znaleźć taką architekturę sprzętową ,na którą by nie był dostępny. Współczesne środowiska IDE zawierają w zasadzie odpowiednio opakowany kompilator, GCC z odpowiednimi kreatorami i skryptami.

Istnieje doskonałe darmowe środowisko *IDE*, umożliwiające przygotowanie niewielkim nakładem środków dowolnego środowiska programistycznego. W artykule pokażemy w jaki sposób, bazując na środowisku ***ECLIPSE*** i innych narzędziach *Open Source*, można przygotować kompletne środowisko dla mikrokontrolerów STM32 , w niczym nie ustępujące komercyjnym IDE. Dodatkową jego zaletą jest to, że możemy je uruchomić nie tylko pod Windows, ale i pod Linuksem.

Do projektowania aplikacji dla ARM będziemy stosować zintegrowane środowisko programistyczne (*IDE*) *Eclipse*. Do ich kompilowania posłużymy się doskonałym kompilatorem języka C/C++ **gcc**, natomiast do programowania mikrokontrolerów STM32 oraz debugowania programów posłuży doskonałe narzędzie *openocd*/*gdb*. Kompilator, programator oraz debugger dają się zintegrować ze środowiskiem *Eclipse*, dzięki czemu dostajemy kompletne środowisko do tworzenia aplikacji.

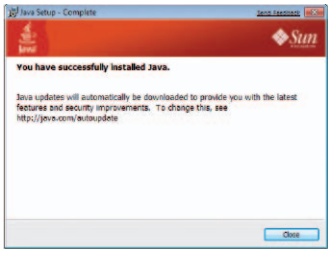
Narzędzia sprzętowe

Mikrokontrolery rodziny **STM32,** jak większość współczesnych mikrokontrolerów, są programowane w systemie **ISP** (*In System Programming*). STM32 zawiera pamięć ROM, w której zawarty jest bootloader umożliwiający programowanie pamięci Flash za pomocą portu szeregowego, bez konieczności stosowania żadnych specjalistycznych narzędzi. Inną możliwością jest wykorzystanie interfejsu JTAG, który dodatkowo umożliwia również debugowanie programu bezpośrednio w systemie. Z uwagi na ten dodatkowy atut, w poniższym artykule skupimy się na wykorzystaniu JTAGa. Do jego obsługi zastosujemy program openocd (*http://openocd.berlios.de/web/*), który umożliwia użycie wielu prostych i tanich interfejsów sprzętowych.

Najtańszym rozwiązaniem sprzętowym umożliwiającym obsługę mikrokontrolerów STM32 z wykorzystaniem programu *OpenOCD* jest zastosowanie popularnego interfejsu *Magicor-Wiggler* np. ZL14PRG firmy Kamami. Interfejs ten zawiera bufor dołączany do portu drukarkowego, a jego zaletą jest prostota. Najistotniejszą wadą układu jest niewielka prędkość działania oraz konieczność posiadania portu równoległego LPT, którego większość współczesnych komputerów jest pozbawiona. Zdecydowanie lepszym rozwiązaniem jest wykorzystanie narzędzia *oocdlink* dołączanego do portu USB, którego przykładem jest układ BF-30 (*http://www.boff.pl*). Zapewnia on większą prędkość pracy przy programowaniu oraz debugowaniu mikrokontrolerów z rdzeniem ARM oraz dodatkowo umożliwia programowanie 8-bitowych mikrokontrolerów AVR w trybie kompatybilności z programatorem *USBASP*.

Przygotowanie oprogramowania dla systemu Windows.

Przed rozpoczęciem instalacji programu Eclipse w Windows musimy upewnić się, że na komputerze została zainstalowana maszyna wirtualna Javy (*JRE – Java Runtime Environment*), która jest niezbędna do działania środowiska. Obecność *JRE* możemy sprawdzić poprzez panel sterowania w zakładce *Add/Remove programs*. W przypadku braku maszyny wirtualnej Java musimy pobrać najnowszą wersję ze strony: *http://www.java.com/en/download/*, oraz



Rys. 1.

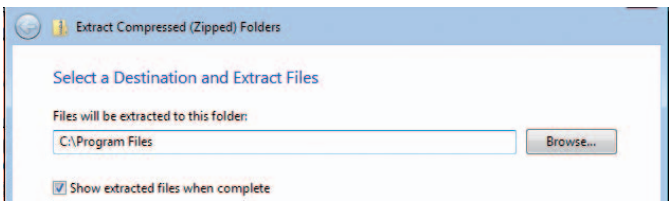
dokonać jej instalacji w sposób typowy dla aplikacji systemu Windows (**rys. 1**).

Po upewnieniu się, że mamy zainstalowaną Javę, należy pobrać najnowszą wersję środowiska IDE eclipse dla programistów C++ (*Eclipse for C/C++ developers*, http://www.eclipse.org/downloads/). Program *Eclipse* jest dostarczany w postaci archiwum ZIP, które należy rozpakować do wybranego folderu np. C:\Program Files (**rys. 2**).

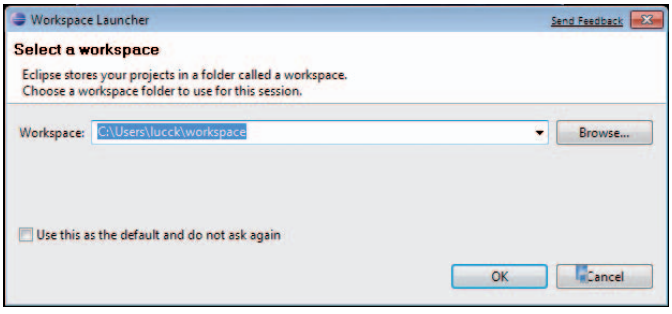
Po rozpakowaniu, w zależności od preferencji, należy utworzyć skrót do programu *eclipse.exe* (znajduje się w podkatalogu eclipse) na pulpicie lub w menu start. Od tego momentu edytor *Eclipse* jest gotowy do pracy, możemy zatem wstępnie przetestować jego działanie klikając w ikonę skrótu. Po uruchomieniu powinno pojawić się okno (**rys. 3**), w którym należy wprowadzić katalog, w którym będą przechowywane domyślnie wszystkie projekty. Jeżeli chcemy aby pytanie nie było ponawiane przy każdym uruchomieniu edytora, to należy zaznaczyć opcję *Use this as the default and do not ask again*, a następnie zatwierdzić wybierając OK. Za chwilę na ekranie powinno pokazać się okno główne programu.

Gdy już upewnimy się, że edytor pracuje poprawnie, przystępujemy do dalszych czynności instalacyjnych. Przystępujemy do instalacji kompilatora GCC dla architektury ARM. Instalator dla systemu Windows możemy pobrać ze strony Codesoucery http://www.codesourcery.com/sgpp/lite/arm/portal/package5356/public/arm-none-eabi/arm-2009q3-68-arm-none-eabi.exe. Po uruchomieniu instalatora, należy odpowiedzieć twierdząco na chęć kontynuacji, oraz zaakceptować warunki umowy licencyjnej. Następnie zostaniemy zapytani o wybór rodzaju instalacji, gdzie należy wybrać opcję *Typical* (**rys. 4**).

W oknie wyboru miejsca instalacji zostawiamy lokalizację domyślną, a w oknie zapytania o dodatnie kompilatora do zmiennej **PATH**, wybieramy opcję *Modify PATH for ALL users*.



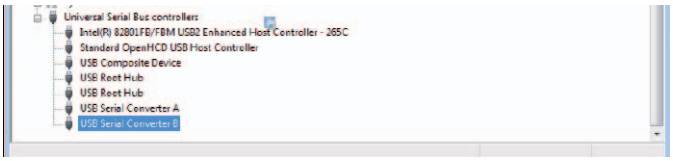
Rys. 2.



Rys. 3.

Dalszy proces instalacji przebiega tak jak w przypadku większości programów dla systemu Windows i nie wymaga komentarza. Prezentowane przykłady do prawidłowej pracy wymagają obecności podstawowych komend Linux, takich jak np. *rm, sed, make*. *Ich* emulację w Windows zapewnia pakietu *MINGW*. Kompletny pakiet zawierający wymagany zestaw poleceń oraz oprogramowanie do obsługi jtaga *BF-30* (*OpenOCD*) w postaci gotowego instalatora, można pobrać ze strony domowej autora http://bryndza.boff.pl/downloads/msysopenocd-installer.exe. Instalacja oprogramowania przebiega typowo i nie wymaga komentarza. Po zakończeniu instalacji **mingw+openocd** mamy w zasadzie kompletne środowisko dla mikrokontrolerów STM32. Pozostało nam jeszcze instalacja sterowników dla JTAG-a, *BF-30*. Proces rozpoczynamy od instalacji sterowników, które można pobrać ze strony: http://www.ftdichip.com/Drivers/CDM/CDM%202.06.00%20WHQL%20Certified.zip. Po ściągnięciu pliku należy go rozpakować do wybranego katalogu. Następnie należy przełączyć **BF-30** w tryb JTAG ustawiając odpowiedni przełącznik na płytce, a potem podłączyć go do gniazda **USB** komputera. System powinien wykryć urządzenie oraz zażądać instalacji driverów. W przypadku *Windows 7* system wyświetla, że nie może znaleźć odpowiedniego drivera i aby zainstalować go poprawnie należy uruchomić *Managera Urządzeń*, kliknąć prawym klawiszem na urządzeniu ***OcdLink*** i wybrać opcję *Install Hardware Driver*, a następnie *Browse my computer for driver software* i jako ścieżkę podać miejsce, gdzie poprzednio rozpakowaliśmy sterowniki. System powinien odnaleźć poprawny sterownik i przystąpić do jego instalacji. Jeżeli sterowniki są prawidłowo zainstalowane, powinniśmy mieć możliwość zobaczenia kompletnych urządzeń *Usb serial Converter A i B* (**rys. 5**).

Rys. 4.

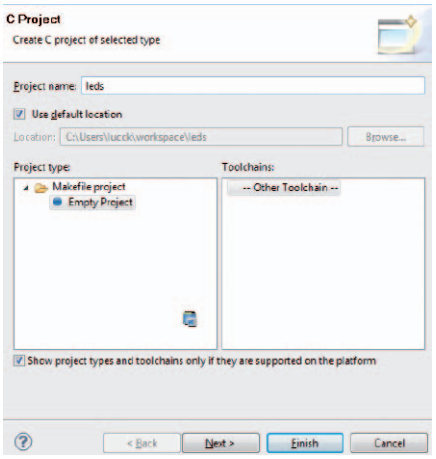


Rys. 5.

Oprogramowanie dla środowiska Linux

Instalacja w środowisku Linux jest zdecydowanie prostsza niż w Windows z uwagi na to, że nie musimy instalować środowiska zapewniającego podstawowe komendy oraz sterowników dla JTAGA BF-30. Instalację opiszemy na przykładzie systemu *Kubuntu Linux 9.10*.

Pierwszą czynnością jest instalacja maszyny wirtualnej Javy z repozytorium. W tym celu należy w managerze pakietów *Kpackage-Kit* zaznaczyć do instalacji pakiet *openjdk-6-jre* lub za pomocą konsoli tekstowej wydać polecenie *sudo apt-get install openjdk-6-jre*. Gdy już mamy zainstalowaną maszynę wirtualną Java, przystępujemy do pobrania edytora *Eclipse for C/C++ Developers* spod adresu http://www.eclipse.org/downloads/. Proces instalacji wygląda w zasadzie identycznie, jak dla środowiska Windows, czyli należy rozpakować *Eclipse* do wybranego folderu np. */usr/local* lub */opt*, a następnie za pomocą edytora menu stworzyć skrót w menu start lub na pulpicie. Kolejną czynnością jest instalacja kompilatora *ARM GCC* dla systemu Linux. Wersję dla architektury 64-bit można pobrać z mojej strony domowej http://bryndza.boff.pl/downloads/cortex-toolchain-20080912_amd64.tar.gz. Dla architektury 32-bitowej można skorzystać z kompilatora dostępnego na stronie *CodeSoucery* http://www.codesourcery.com/sgpp/lite/arm/portal/package5355/public/arm-none-eabi/arm-2009q3-68-arm-none-eabi.bin. W przypadku wersji 32-bitowej sposób instalacji przebiega w sposób analogiczny, jak dla systemu Windows. Instalacja w systemie 64-bitowym sprowadza się do rozpakowania archiwum za pomocą polecenia *sudo tar xvfz cortex-toolchain-20080912_amd64.tar.gz*. Kolejnym krokiem jest sprawdzenie czy mamy zamontowany system plików *USBFS*, za-



Rys. 6.

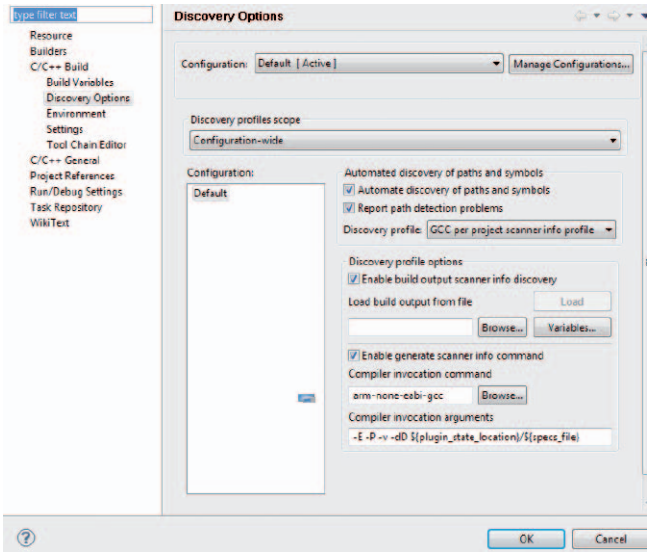
pewniający obsługę urządzeń USB z przestrzeni użytkownika, bez konieczności używania sterowników jądra. W tym celu w terminalu wydajemy polecenie *stat /proc/bus/usb/devices*. Jeżeli otrzymamy komunikat *No such file or directory*, do pliku *rc.local* należy za pomocą edytora dopisać polecenie montujące system *USBFS*: *mount -t usbfs none /proc/bus/usb*. Ostatnią czynnością jest instalacja programu *OpenOCD*, który można pobrać dla obu architektur z mojej strony domowej (i386: *http://bryndza.boff.pl/downloads/openocd-0.3.1_i386.tgz*, AMD64: *http://bryndza.boff.pl/downloads/openocd-0.3.1_amd64.tgz*). Proces instalacji jest trywialny i sprowadza się do rozpakowania archiwum z pomocą polecenia *sudo tar xvfz openocd-0.3.1_architektura -C /*.

Przykładowy projekt – tworzenie, kompilacja, programowanie zestawu

Zintegrowane komercyjne środowiska programistyczne posiadają wbudowane kreatory umożliwiające stworzenie projektu za pomocą kilku kliknięć myszką. Niestety, *Eclipse* nie ma żadnych ułatwień w tym kierunku, ale ten problem można rozwiązać bardzo prosto, wykorzystując pliki z wzorcowego projektu przygotowanego przez autora.

Zaczynając naukę programowania najczęściej pierwszym przykładem prezentowanym w książkach jest aplikacja typu „Hello World" wypisująca tekst na ekranie. Podobnie w przypadku mikrokontrolerów zazwyczaj pierwszym programem jest przykład z mrugającymi diodami LED. Zatem idąc wydeptaną ścieżką, pokażemy krok po kroku jak bazując na wzorcowym projekcie stworzyć własną aplikację mrugającą diodami zestawu *STM32-Butterfly*. Przygotowaną wzorcową aplikację należy pobrać ze strony *http://bryndza.boff.pl/downloads/stm32-examples.tgz*, a następnie uruchomić środowisko *Eclipse*. Z menu aplikacji wybieramy opcje: *File–>New–>C Project*, co spowoduje wyświetlenie okna kreatora aplikacji. W oknie nazwy projektu (*Project Name*) wpisujemy przykładową nazwę projektu, (w tym przypadku *leds*), natomiast pozostałe ustawienia pozostawiamy jak na **rys. 6**.

Po kliknięciu klawisza *FINISH* zostanie utworzony pusty projekt, który następnie należy dostosować do naszego kompilatora. W tym celu w oknie *Project Explorer* klikamy prawym klawiszem myszy na katalogu projektu ***leds***, a następnie z menu kontekstowego wybieramy opcję *Properties*. Pojawi się wówczas okno dialogowe , w którym w zakładce *Discovery Options* należy zaznaczyć opcję *Automate Discovery Path And Symbols*, a w oknie *Compiler invocation command* zmienić nazwę kompilatora *gcc* na *arm-none-eabi-gcc* i potwierdzić wprowadzone ustawienia klawiszem OK. Wprowadzona zmiana umożliwia automatyczne wykrywanie i parsowanie plików nagłówkowych dołączonych do projektu za pomocą dyrektywy *include* (**rys. 7**).



Rys. 7.

Teraz w zakładce *C/C++ build–>Settings* w oknie *Binary Parsers* należy zaznaczyć opcję *Elf Parser*, co zapewni środowisku możliwość parsowania plików wynikowych aplikacji. Po ustawieniu wymaganych opcji możemy przystąpić do importu projektu *stm32-examples.tgz*. W tym celu w oknie *Project Explorer* klikamy prawym klawiszem na projekcie i z menu kontekstowego wybieramy opcję *Import*. Pojawi się okno źródła importu w którym należy wybrać opcję *Archive File* oraz przejść do kolejnego okna dialogowego (klawisz *Next*). W oknie importu należy wcisnąć przycisk *Browse* i wskazać plik *stm32-examples.tgz* w wyniku czego zostanie wyświetlona lista plików znajdująca się w archiwum. Należy wybrać wszystkie pliki, oraz zakończyć import plików za pomocą klawisza *Finish*. Po zaimportowaniu projektu możemy przystąpić do jego kompilacji (klawisze *Ctrl+B* lub z menu *Project–>Build All*). Jeżeli wszystko przebiegło pomyślnie, na dole w oknie „*Console*" powinniśmy zobaczyć logi z kompilacji natomiast w oknie *Problems* nie powinno być żadnych informacji.

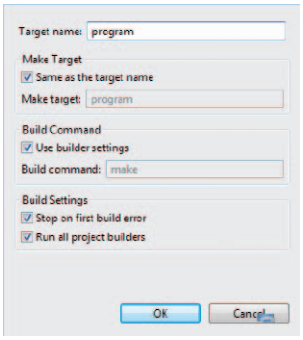
Po skompilowaniu projektu zostało nam zaprogramowanie zestawu *STM32-Butterfly*. Programowanie układu odbywa się poprzez wywołanie polecenia *make install*, które można również wywoływać bezpośrednio z poziomu *Eclipse*. Odbywa się to poprzez stworzenie ikonki dla wybranej reguły make w zakładce *make target*, znajdującej się po prawej stronie okna głównego. Aby to zrobić wystarczy kliknąć ikonkę z symbolem zielonego okręgu znajdującego się na tej zakładce, co spowoduje wyświetlenie okna dialogowego umożliwiającego skonfigurowanie ikony dla wybranej reguły make. W oknie tym należy wpisać regułę **program** a następnie wcisnąć klawisz **OK** (**rys. 8**). Po wykonaniu tej czynności w zakładce *Make Targets* będziemy mogli zobaczyć ikonę z opisem *program* (**rys. 9**).

Aby zaprogramować urządzenie należy do złącza USB podłączyć JTAG BF-30 z przełącznikiem konfiguracyjnym ustawionym w pozycję JTAG, podłączyć zasilanie zestawu *STM32-Butterfly*, a następnie kliknąć we wspomnianą wcześniej ikonę. Po krótkiej chwili układ powinien zostać zaprogramowany, a diody LED *D1*,*D2* powinny migać naprzemiennie.

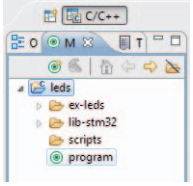
Potrafimy już kompilować programy oraz programować mikrokontroler, do omówienia pozostało nam jeszcze debugowanie aplikacji.

Przykładowy projekt – debugowanie

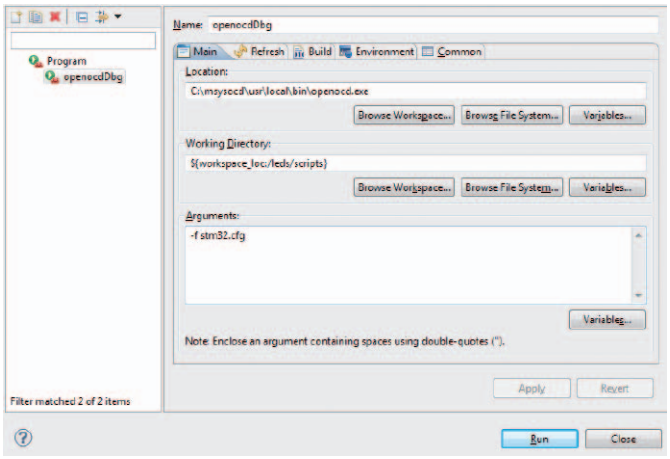
Debugowanie w naszym środowisku odbywa się za pomocą programu debugera *OpenOCD*, którego zadaniem jest bezpośrednia obsługa interfejsu JTAG oraz mikrokontrolera. Program ten nasłuchuje na wybranym porcie (*3333*) na połączenia od programu



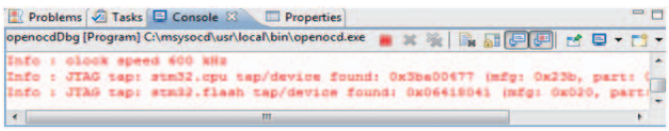
Rys. 8.



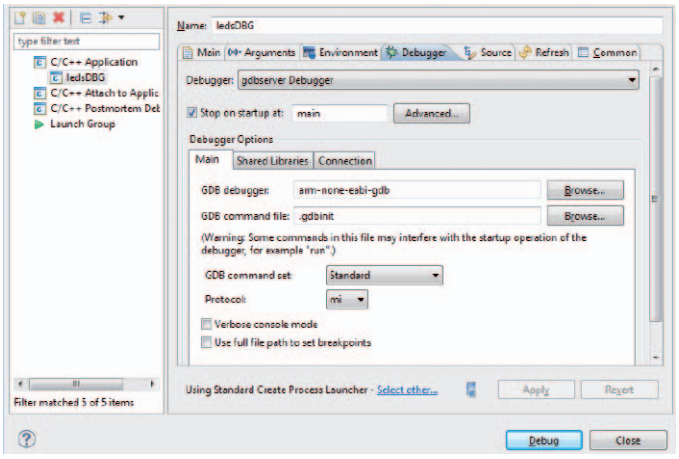
Rys. 9.



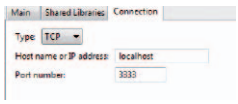
Rys. 10.



Rys. 11.



Rys. 12.



Rys. 13.

debugera *gdb*, który umożliwia bezpośrednią interakcję z użytkownikiem. *GDB* komunikuje się z użytkownikiem za pomocą konsoli, więc użytkownicy systemu Windows mogą z jego obsługą mieć problemy. W przypadku środowiska *Eclipse* odpowiednie polecenia dla debugera wysyła samo środowisko, umożliwiając wygodną pracę bez konieczności wpisywania dodatkowych poleceń.

Przygotowanie projektu do debugowania, rozpoczniemy od dodania programu *OpenOCD*, tak aby uruchamiał się po wciśnięciu pojedynczego przycisku. W tym celu z menu start wybieramy opcję *Run –>External Tools–>External Tools Configuration*, co spowoduje wyświetlenie okna dialogowego umożliwiającego konfigurację uruchomienia zewnętrznych programów. W tym oknie należy wybrać ikonę *New Launch Configuration*, która utworzy nową konfigurację uruchomieniową. W oknie konfiguracyjnym (**rys. 10**), należy wprowadzić ścieżkę do programu **OpenOcd** ,wskazać katalog roboczy, oraz argumenty dla polecenia.

Po wprowadzeniu konfiguracji możemy spróbować uruchomić aplikację wybierając przycisk *Run*. Aplikacja zostanie uruchomiona w wewnętrznej konsoli, gdzie wypisze komunikat o znalezieniu mikrokontrolera oraz pozostanie aktywna w tle. Zatrzymanie aplikacji możliwe jest po wciśnięciu ikony z symbolem *stop* (**rys. 11**).

Potrafimy już uruchomić **OpenOCD** pozostało nam jeszcze skonfigurowanie *Eclipse*, tak aby potrafił połączyć się serwerem **OpenOCD** i przejść do trybu debugowania. Aby to zrobić należy z menu wybrać opcję: *Run–>Debug Configurations*, co spowoduje wyświetlenie okna dialogowego z ustawieniami konfiguracyjnymi. W oknie tym, po lewej stronie, należy kliknąć na ikonę z opisem *C/C++ Application*, która wygeneruje nową konfigurację. W oknie konfiguracji należy wybrać projekt, który chcemy debugować (w tym przypadku *leds*), co powinno zaowocować automatycznym przypisaniem pliku binarnego aplikacji w polu *C/C++ Application*. Teraz należy przejść do zakładki *Debuger*, gdzie w polu debugger należy z listy wybrać opcję *gdbserver debuger*, oraz ustawić nazwę pliku programu debugera *GDB* (**rys. 12**).

Następnie w pod-zakładce *Connnection* należy wybrać typ połączenia (*Connection type*) **TCP**, nazwę hosta (*hostname*) należy ustawić na *localhost*, natomiast w pole port należy wpisać wartość *3333* (**rys. 13**).

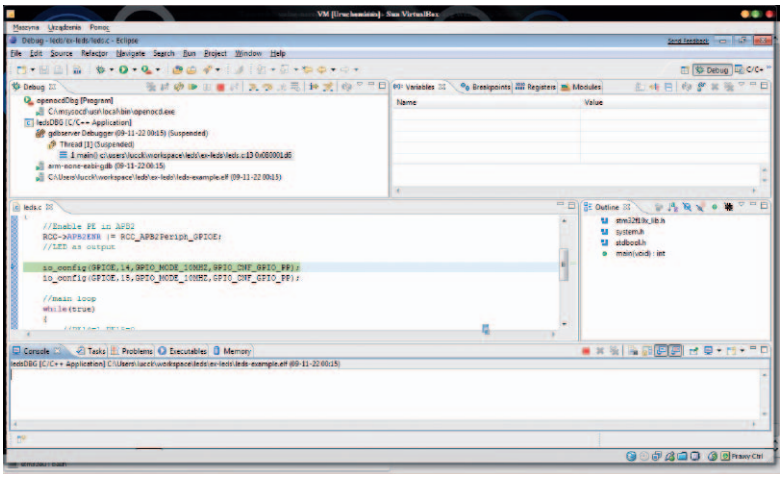
Konfiguracja w zasadzie dobiegła końca należy jeszcze wcisnąć klawisz *Apply*, a następnie zamknąć okno przyciskiem *Close*. Przed sprawdzeniem konfiguracji w praktyce jeszcze jedna drobna uwaga odnośnie optymalizacji kompilacji programu. W pliku projektu *ex-led/Makefile* na początku pliku znajduje się zmienna OPT=2, która umożliwia sterowanie procesem optymalizacji kodu. Opcja 2 jest polecana w przypadku, generowania docelowych programów, i zdaniem autora zapewnia dobry wybór pomiędzy wielkością aplikacji a szybkością jej działania. Niestety podczas debugowania wybranie tej opcji może powodować, iż debugger będzie pomijał niektóre linie podczas pracy krokowej, lub nie będzie dostępny podgląd niektórych zmiennych. Dzieje się tak ponieważ kompilator, często optymalizuje dany fragment kodu tak, że przestaje on bezpośrednio mieć odzwierciedlenie w postaci linii kodu źródłowego. Właśnie ze wspomnianego wyżej powodu zalecane jest aby przed debugowaniem skompilować całą aplikację z opcją 0, zaprogramować mikrokontroler, a następnie po tej czynności – rozpocząć sesję debuggera.

Powracając do głównego wątku, spróbujmy uruchomić debugger z naszą aplikacją. W tym celu najpierw uruchamiamy program *OpenOCD* (klikając na ikonę *Launch External Tool*), a następnie ikonę z symbolem „robaka", co powinno zaowocować uruchomieniem programu w trybie *Debug* oraz zatrzymaniem się aplikacji w funkcji *main()* (**rys. 14**). Przy pierwszym uruchomieniu sesji należy wybrać opcję *Menu–>Debug Configuration* i wybrać program *ledsDBG*, a następnie wcisnąć klawisz *RUN*. Teraz wciskając klawisz F6 (*Step Over*) możemy śledzić wykonanie poszczególnych linii programu. *Eclipse* umożliwia również ustawianie pułapek w określonych miejscach programu poprzez kliknięcie myszką na niebieskim obszarze przed wybraną linią. Umożliwia też śledzenie zmiennych. Inny użyteczny skrótów klawiszowy to F5 (*Step Into*), opcja która umożliwia zagłębienie się do wnętrza funkcji.

Tryb debugowania *Eclipse* zapewnia możliwości porównywalne z innymi środowiskami komercyjnymi. Opis wszystkich opcji przekracza łamy niniejszego artykułu, niemniej jednak środowisko to jest na tyle intuicyjne, że nie powinno być większych problemów ze znalezieniem poszukiwanej funkcji.

Budowa projektu

Dla użytkownika używającego zintegrowanego środowiska programistycznego jedynym zmartwieniem jest utworzenie plików źródłowych, ponieważ wszystkie skrypty potrzebne do pracy generowane są automatycznie. Podejście takie zazwyczaj jest zadowalające, jeżeli wszystko działa poprawnie. W przypadku, w którym pojawią się jakieś problemy, użytkownik tak naprawdę nie wie w jaki sposób one działają. Nie ma też wpływu na działanie kreatorów, co może być przyczyną wielu problemów. Środowisko *Eclipse* nie zapewnia żadnego specjalistycznego wsparcia w przypadku projektów dla mikrokontrolerów, w związku z tym wszystkie pliki należy stworzyć samodzielnie. Może to być zadaniem stosunkowo skomplikowanym, dlatego najwygodniej będzie skorzystać z gotowego projektu bazowego dla danej rodziny mikrokontrolerów, zmieniając tylko potrzebne opcje. Zaprezentowany projekt



Rys. 14.

bazowy dla mikrokontrolerów STM32 ma strukturę hierarchiczną (**rys. 15**). W katalogu *scripts* zawarte są skrypty odpowiedzialne za przebieg kompilacji oraz programowanie urządzenia docelowego. Plik *stm32.mk* jest plikiem dla narzędzia *make*, zapewniającego automatyzację procesu kompilacji. Stanowi on bazę, która jest dołączana do głównego pliku *Makefile*. Został napisany w taki sposób, aby wszystkie pliki źródłowe C, C++, oraz S (assembler) utworzone w pliku projektu były kompilowane automatycznie.

Plik *stm32.cfg* jest plikiem konfiguracyjnym dla programu debugera OpenOCD i zawiera opis konfiguracji JTAG-a BF-30 oraz mikrokontrolerów rodziny STM32.

Plik *stm32.ld* jest skryptem linkera, w którym mamy możliwość zdefiniowania wielkości pamięci, która może być różna zależnie od wybranego układu z danej rodziny. W linii:

```
RAM (xrw) : ORIGIN = 0x20000000,
LENGTH = 48K
```

możemy definiować adres bazowy oraz rozmiar wielkości pamięci RAM, natomiast w linii:

```
FLASH (rx) : ORIGIN = 0x8000000,
LENGTH = 128K
```

adres bazowy oraz wielkość pamięci FLASH w zależności od używanego układu

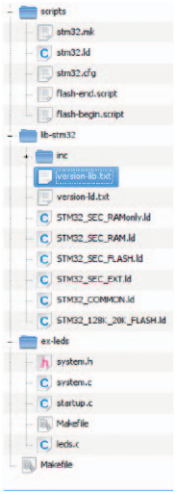
W katalogu *lib-stm32*, znajduje się część biblioteki *STM32 Perhipheral library*, z której wyciągnięto pliki nagłówkowe(katalog *inc*), oraz skrypty dla linkera (**.ld*). Pliki nagłówkowe zostały zmodyfikowane aby zawierały tylko struktury opisujące układy peryferyjne. Zdaniem autora biblioteka *STM32 Perhipheral Library* nie stanowi przejawu kunsztu programistycznego i niepotrzebnie zajmuje cenne miejsce w pamięci mikrokontrolera. Właściwy przykład błyskający diodami LED zawarty jest w katalogu *ex-leds*. Głównym plikiem sterującym przebiegiem kompilacji projektu jest plik *Makefile*. Plik ten zawiera kilkę użytecznych opcji, które mogą być zmieniane przez użytkownika w zależności wymagań. Najistotniejszy fragment pliku z punktu widzenia użytkownika przedstawiono niżej:

```
# Automatic makefile for GNUARM (C/C++)
# Copyright (C) Lucjan Bryndza
<lucjan.bryndza@ep.com.pl>
# http://www.boff.pl
#tutaj wpisz nazwe pliku hex
TARGET      = leds-example
#Format wyjsciowy (moze byc
elf,hex,bin)
FORMAT  = hex
#Optymalizacja [0,1,2,3,s]
# 0 - brak optymalizacji, s
-optymalizacja rozmiaru
OPT     = 2
#Common flags
COMMON_FLAGS = -Wall -pedantic
-DSTM32F10X_CL
```

W zmiennej TARGET mamy możliwość wpisania nazwy pliku wynikowego, który powstanie w wyniku kompilacji projektu. Pole format umożliwia określenie rodzaju pliku wynikowego wygenerowanego w wyniku kompilacji. Skrypt przewiduje możliwość generowania pliku w formatach *ELF*, *Intel Hex* lub bezpośrednio w formacie binarnym. Kolejną istotną opcją jest tryb optymalizacji kodu wynikowego, który został opisany we wcześniejszej części tekstu. Mamy tutaj możliwość sterowania optymalizacją od opcji **0** (brak optymalizacji) poprzez **3** (maksymalna optymalizacja na szybkość programu), do **s** (optymalizującą program pod kątem jak najmniejszego miejsca zajmowanego w pamięci).

Ostatnią istotną zmienną jest *COMMON_FLAGS*, która umożliwia podanie dodatkowych wspólnych opcji dla kompilatora C oraz C++. Istotna tutaj jest definicja *-DSTM32F10X_CL*, powodująca zdefiniowanie typu mikrokontrolera dla biblioteki *STM32 Standard Perhipheral Library*. Mamy tutaj możliwość zdefiniowania następujących symboli: *STM32F10X_LD* (mikrokontrolery STM32 *Low Density Line*), *STM32F10X_MD* (STM32 *Medium Density Line*), *STM32F10X_HD* (STM32 *Hi Density Line*), STM32F10X_CL (STM32 *Conectivity Line* – jak w zestawie *STM32 Butterfly*). Pozostałe pliki są plikami źródłowymi w języku C, i odpowiadają bezpośrednio za działanie aplikacji. Warto tutaj wspomnieć o pliku *startup.c*, w którym została zdefiniowana globalna tablica przypisująca funkcję do wektorów przerwań.

```
/*-------------------------------
----------------------*/
//Interrupt vector table
__attribute__ ((section(".isr_
vector")))
void (* const exceptions_vectors[])
(void) =
{
  &_estack,          // The initial
stack pointer
  reset_handler,     // The reset
handler
  unused_vector, //NMIException
  unused_vector, //HardFaultException
  unused_vector, //
MemManageException
  unused_vector, //
BusFaultException
  unused_vector, //
UsageFaultException
```



Rys. 15.

W pliku należy przypisać funkcję, która będzie odpowiedzialna za realizację obsługi danego przerwania. Domyślnie wszystkie wektory mają zdefiniowaną funkcję *unused_vetor()* która zawiera nieskończoną pętlę *while(1)*. W naszym prostym przykładzie nie wykorzystujemy żadnych przerwań więc nie ma potrzeby dopisywania tutaj żadnych funkcji.

Zakończenie

Zaprezentowane w tym artykule zintegrowane środowisko programistyczne złożone z komponentów Open Source umożliwia zupełnie wygodne i niczym nieograniczone pisanie oraz uruchamianie aplikacji dla mikrokontrolerów STM32. Po poświęceniu odrobiny czasu na przygotowanie, dostajemy pełną wersję wygodnego środowiska, która w przeciwieństwie do innych komercyjnych IDE, może pracować poza systemem Windows, na przykład w Linuksie. Przy tworzeniu nowego projektu musimy pamiętać, aby przed rozpoczęciem pracy ustawić kilka opcji konfiguracyjnych we właściwościach projektu. W *Eclipse* nie mamy wbudowanych kreatorów, umożliwiających stworzenie projektu za pomocą kilku kliknięć myszką, ale bazując na przykładowej aplikacji po nabyciu odrobiny wprawy, mamy możliwość rozpoczęcia projektu równie szybko i wygodnie. Co najważniejsze, mamy pełne panowanie nad procesem kompilacji oraz skryptami konfiguracyjnymi. Równie dobrze kompilacje możemy przeprowadzić poza edytorem, np. w systemie automatycznego relasowania oprogramowania, co jest w zasadzie niewykonalne w przypadku komercyjnych narzędzi.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl