Terrarium dla „gekona”

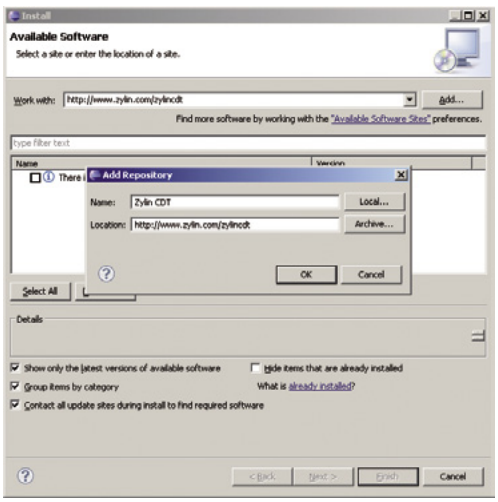
Zintegrowane środowisko programistyczne dla mikrokontrolerów EFM32 (2)

W numerze 9/10 EP została zaprezentowana rodzina mikrokontrolerów EFM32 firmy EnergyMicro (Gecko, TinyGecko). Jeśli ktoś potrzebuje zastosować w swoich aplikacjach układy o niskim poborze prądu z mocą obliczeniową rdzenia ARM Cortex-M3 lub po prostu chce poddać próbie marketingowe hasła producentów tych mikrokontrolerów, to ten artykuł może okazać się bardzo pomocny.

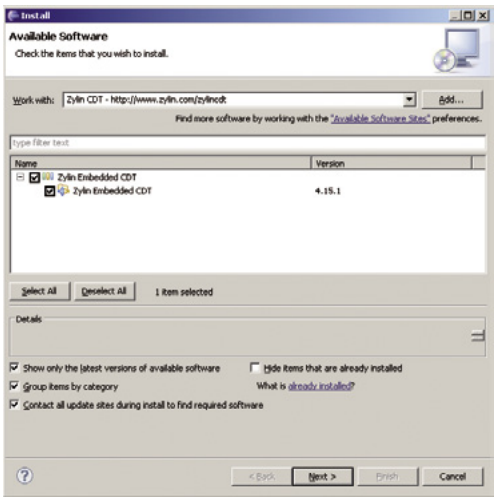
Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 10460, pass: 0646g3n0

Karmienie i doglądanie gekona

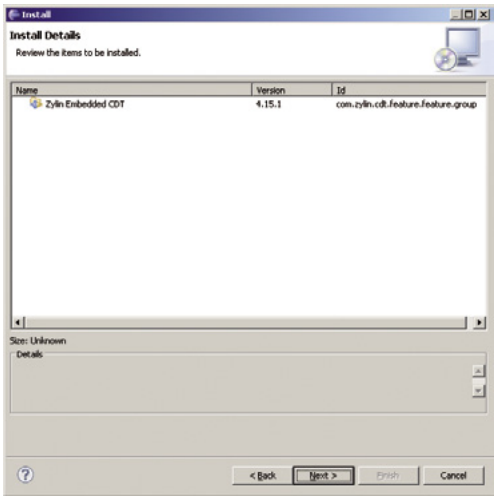
Aby móc programować i debugować nasz mikrokontroler z poziomu *Eclipse*, musimy ściągnąć z Internetu odpowiedni plugg-in i zainstalować sterowniki do znajdującego się na płytce ewaluacyjnej debuggera *SEGGER*. W tym celu klikamy w menu *Help* na *Install New Software*. W nowym oknie klikamy przycisk *Add*, a następnie w polu *Name* wpisujemy „*Zylin CDT*” a w polu *Location* http://www.zylin.com/zylincdt i potwierdzamy *OK* (**rysunek 13**). Zaznaczamy pole *Zylin Embedded CDT*, naciskamy *Next* (**rysunek 14**) i jeszcze raz *Next* (**rysunek 15**). Zaznaczamy akceptację licencji i potwierdzamy przyciskiem *Finish* (**rysunek 16**). W następnej kolejności potwierdzamy autentyczność plug-inu, klikając *OK* (**rysunek 17**). Po zakończonej instalacji należy zrestartować *Eclipse* – przycisk *Restart Now* (**rysunek 18**). Następnie wybieramy w menu *Run -> Debug Configurations*, klikamy dwa razy na *Zylin Embedded debug (Native)* w menu po lewej stronie. W polu *Name* wpisujemy „*EFM32 ROM*”. W zakładce *Debugger* w polu *GDB debugger* wybieramy ścieżkę dostępu do naszego debuggera np. *C:\Program Files\CodeSourcery\Sourcery G++ Lite\bin\arm-none-eabi-gdb.exe*, a pole *GDB command file:* zostawiamy puste



Rysunek 13. Dodawanie pluginów



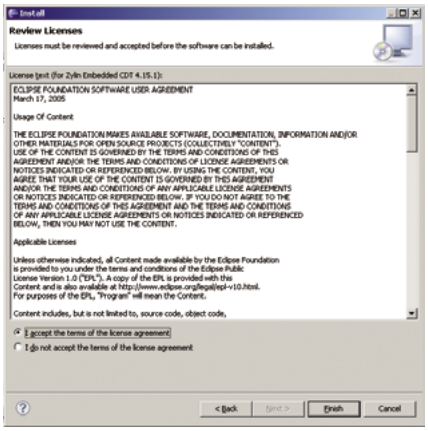
Rysunek 14. Wybór pluginu Zylin Embedded CDT



Rysunek 15. Potwierdzenie

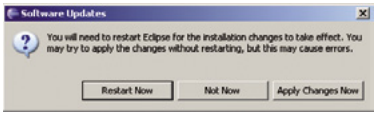
(**rysunek 19**). Zatwierdzamy *Apply*. W zakładce *Commands* w polu *Initialize Commands* wpisujemy kod:

```
target remote localhost:2331
monitor endian little
monitor reset
monitor speed auto
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
monitor flash device = EFM32G890F128
monitor flash download = 1
```

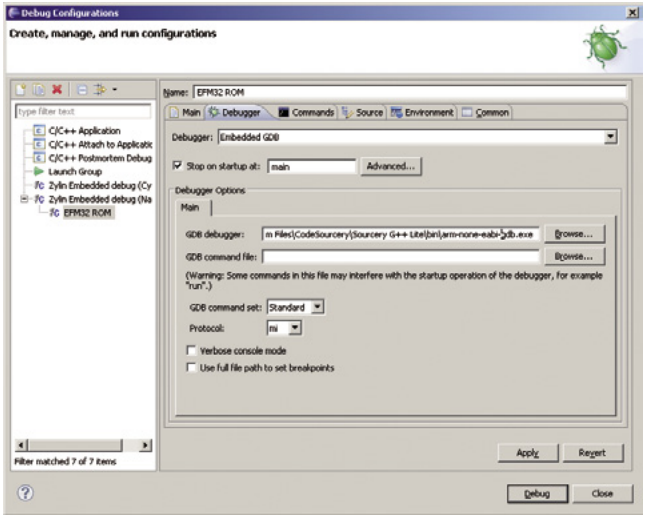


Rysunek 16. Akceptacja licencji

Rysunek 17. Zatwierdzenie nieuwierzytelnionego źródła



Rysunek 18. Restart Eclipse’a

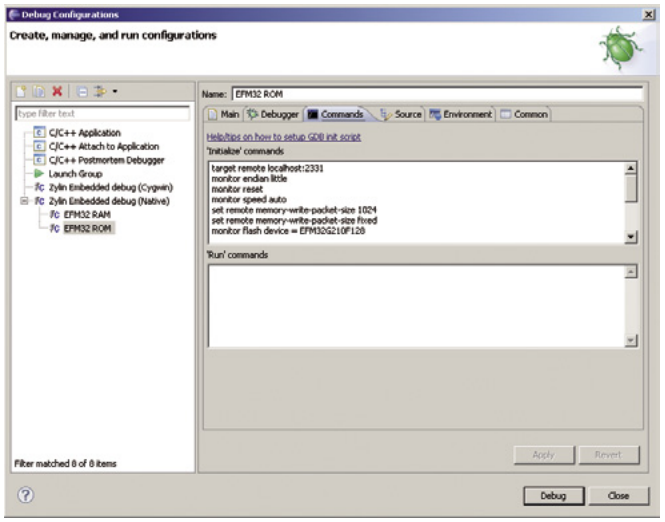


Rysunek 19. Wybór debuggera

```
load
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)
monitor reset
break main
continue
```

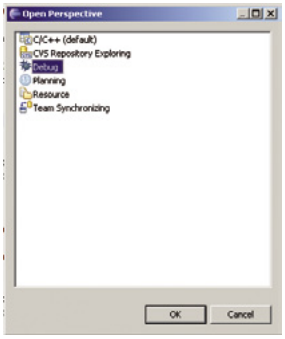
Wpisany kod zatwierdzamy, klikając *Apply* (**rysunek 20**). W menu po lewej stronie ponownie klikamy dwa razy na *Zylin Embedded debug (Native)*. W polu *Name* wpisujemy *EFM32 RAM*. W zakładce *Debugger* w polu *GDB debugger* wybieramy ścieżkę dostępu do debuggera – *C:\Program Files\CodeSourcery\Sourcery G++ Lite\bin\arm-none-eabi-gdb.exe*. Pole *GDB command file* zostawiamy puste. W zakładce *Commands* w polu *Initialize Commands* wpisujemy kod:

```
target remote localhost:2331
monitor endian little
monitor reset
monitor speed auto
load
```



Rysunek 20. Ustawienia debuggera

```
monitor reg r13 = (0x20000000)
monitor reg pc = (0x20000004)
break main
continua
```



Rysunek 21. Otwieranie widoku debuggowania

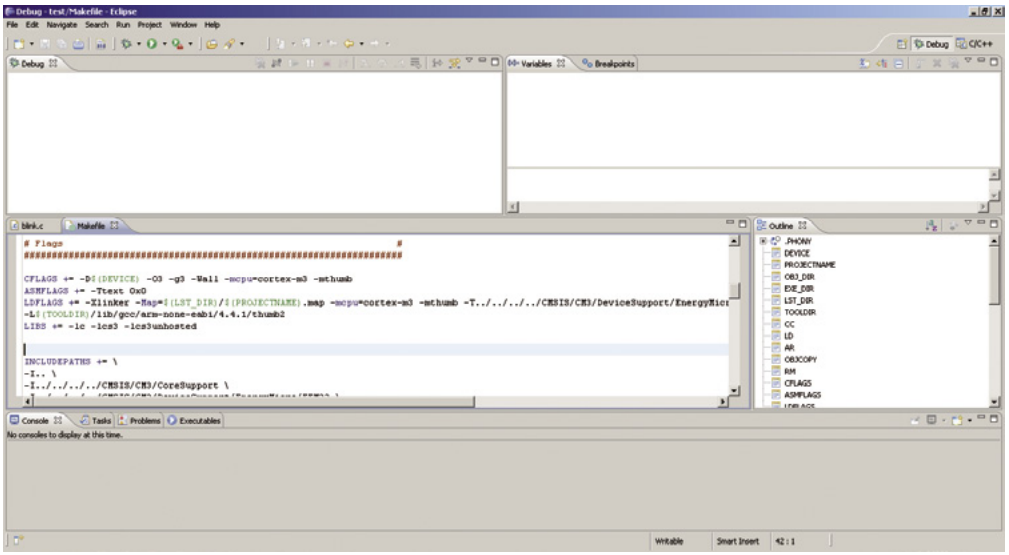
Wpisany kod zatwierdzamy, klikając klawisz *Apply*. Następnie klikamy *Close*. Plug-in jest teraz zainstalowany i skonfigurowany do umieszczania kodu i debugowania zarówno w pamięci *Flash*, jak i *RAM* (w przypadku, gdy chcemy debugować kod umieszczony w pamięci *RAM*, jest niezbędna edycja skryptu linkera w celu umieszczenia całego kodu w pamięci *RAM podczas budowania projektu*).

Ostatnimi plikami, które musimy ściągnąć i zainstalować, są sterowniki do debugera. Najnowsza, stabilna wersja dostępna jest na stronie internetowej *http://www.segger.com/cms/jlink-software.html* (w chwili pisania artykułu jest to wersja 4.20p o wielkości 8862 kB). Instalacji dokonujemy z użyciem parametrów domyślnych.

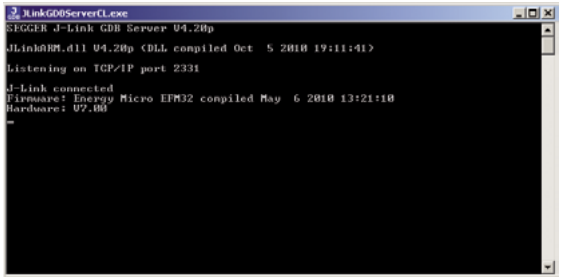
Przed zaprogramowaniem mikrokontrolera otworzymy sobie specjalny widok przeznaczony do debugowania. Wybieramy z menu *Window -> Open Perspective -> Debug* lub, jeżeli nie będziemy widzieć takiej opcji, wybieramy ją z menu *Window -> Open Perspective -> Other* i wybieramy z listy *Debug*. Wybór potwierdzamy klawiszem *OK* (**rysunek 21**). *Eclipse* powinien wyglądać jak na **rysunku 22**.

Teraz przyszła pora na dołączenie naszej płytki ewaluacyjnej do gniazda *USB*. Sprzęt powinien zostać prawidłowo wykryty i zainstalowany w systemie. W celu komunikowania się oprogramowania ze sprzętem, użyjemy zainstalowanej aplikacji dostarczonej przez *SEGGER*. Tworzymy na pulpicie (lub w innym dogodnym dla nas miejscu) skrót do programu *C:\Program Files\SEGGER\JLinkARM_V420p\JLinkGDBServerCL.exe*. Następnie wchodzimy w jego opcje i w zakładce *Skrót* w polu *Element docelowy* na końcu dopisujemy *-if SWD*.

Spowoduje to, że program będzie uruchamiał się z interfejsem *SWD*. Po uruchomieniu aplikacji na ekranie powinniśmy zobaczyć widok jak na **rysunku 23**. Jeżeli jednak chcemy uruchamiać aplikację *SEGGERa* bezpośrednio z *Eclipse*, to wybieramy z menu *Run -> External Tools -> External Tools Configurations*, a następnie klikamy dwa razy myszką na *Program*. Teraz po prawej stronie w polu *Name* wpisujemy *JLink SWD*, w zakładce *Main* w polu *Location* wskazujemy ścieżkę dostępu do aplikacji – *C:\Program Files\SEGGER\JLinkARM_V420p\JLinkGDBServerCL.exe*. W polu *Arguments* wpisujemy *–if SWD*. Potwierdzamy, klikając na *Apply* i wychodzimy *Close* (**rysunek 24**). Teraz z menu wybieramy *Run -> External Tools -> Organize Favorites*, klikamy na *Add*, zaznaczamy aplikację, potwierdzamy *OK* i jeszcze raz *OK*.

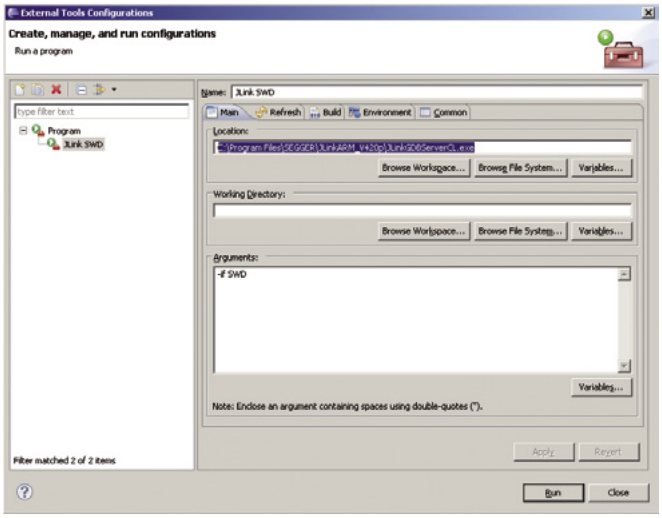


Rysunek 22. Widok Eclipse'a do debuggowania



Rysunek 23. Konsolowy widok programu JLinkGDBServerCL.exe

Aplikację uruchamia się z menu *Run -> External Tools -> JLink SWD*. Niezależnie od tego, który z opisanych sposobów wybierzemy, aplikacja powinna zostać uruchomiona przed przystąpieniem do debugowania. Poprawnie uruchomiona aplikacja w *Eclipse* powinna wyglądać jak na **rysunku 25**. Aby ją zakończyć, zaznaczamy ją, a następnie naciskamy na czerwony przycisk *Stop*. Przy uruchomionej aplikacji wybieramy z menu *Run -> Debug Configurations*, a następnie klikamy na *EFM32 ROM* i potwierdzamy *Debug*. Proces wgrywania programu do pamięci zostaje rozpoczęty. Od tej pory wszystkie komunikaty związane z debugowaniem możemy obserwować w zakładce *Console*. Jeżeli wszystko przebiegnie poprawnie, powinniśmy na swoim ekranie zobaczyć widok jak na **rysunku 26**. Klikamy teraz na przycisk *Resume* lub naciskamy klawisz *F8*. Efektem powinno być miganie diod LED na płytce ewaluacyjnej. Debugowanie zatrzymujemy poprzez naciśnięcie czerwonego przycisku *Stop*.



Rysunek 24. Uruchamianie programu JLinkGDBServerCL.exe bezpośrednio z Eclipse'a

Czasami okazuje się, że aplikacja debugująca *GDB* nie jest kompatybilna z debuggerem *J-Link* znajdującym się na płytce. Zdarza się to przy nowych wersjach *GDB*. Zatem w przypadku, gdy podczas uruchamiania konfiguracji *EFM32 ROM* w tym tutorialu pojawi się komunikat o błędzie „*Remote ‚g' packet reply is too long…*", wtedy jednym ze sposobów pozbycia się go jest odinstalowanie *Sourcery G++ Lite Edition* i zainstalowanie innej wersji tego pakietu (np. starszej). Aby ściągnąć i zainstalować inną niż najnowsza wersję pakietu, należy wejść na stronę *www.codesourcery.com*, a następnie wybrać z menu *Products -> Sourcery G++ -> Editions -> Lite*. Na nowej stronie wybieramy interesującą nas wersję, czyli *ARM*, a następnie *Download the current release*, wersję *EABI*, ale link *All versions*. Wybieramy odpowiadającą nam wersję, a następnie klikamy na link do pliku instalacyjnego przeznaczonego dla systemu *Windows*, czyli *IA32 Windows Installer*.

Podsumowanie

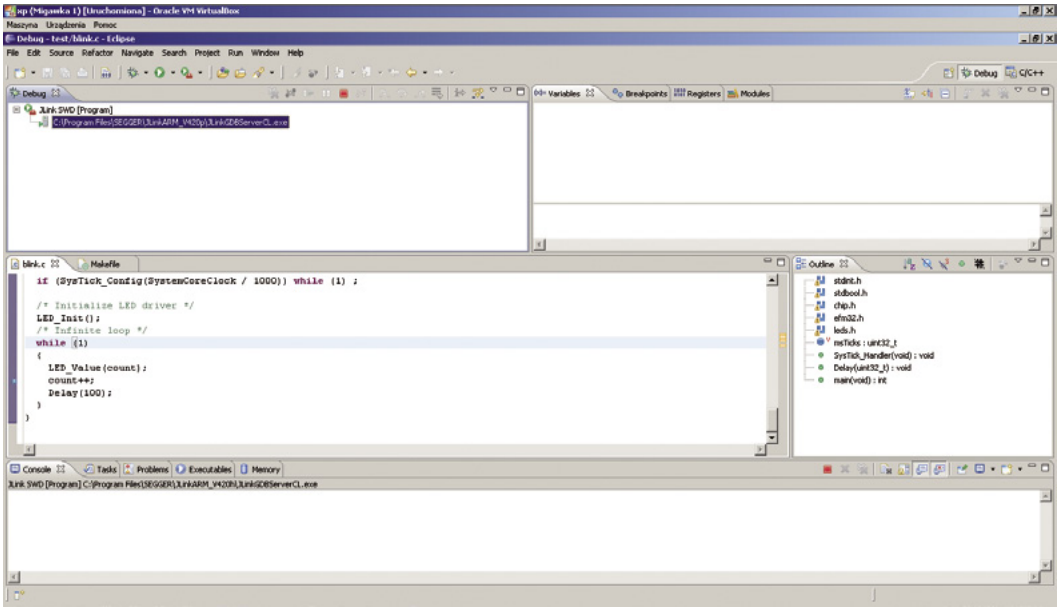
Jak można zauważyć, prawidłowe skonfigurowanie kompletnego środowiska do programowania i debugowania mikrokontrolerów firmy *EnergyMicro* nie jest łatwe i zajmuje trochę czasu. Jednak raz skonfigurowane będzie nam dobrze służyło i co najważniejsze – ilość kodu ograniczać będzie tylko pamięć Flash mikrokontrolera. Zaprezentowane środowisko jest również alternatywą w przypadku, gdy tworzymy projekt za pomocą dołączonego do płytki ewaluacyjnej *IAR Embedded Workbench* i w pewnym momencie zacznie nam doskwierać ograniczenie wielkości kodu wynikowego do 32 kB. Wtedy szybka migracja do przedstawionego w tym poradniku środowiska z pewnością pomoże nam na bezproblemowe dokończenie projektu.

Duża ilość dokumentacji dobrze opracowanej przez producenta bardzo pomaga w stawianiu pierwszych kroków w programowaniu EFM32. Zachęcają do tego ceny płytek ewaluacyjnych oraz samych mikrokontrolerów. Jeśli do tego dodamy wbudowany debugger oraz darmowe środowisko programistyczne bez ograniczenia wielkości kodu, to z pewnością produkt ten jest wart naszej uwagi.

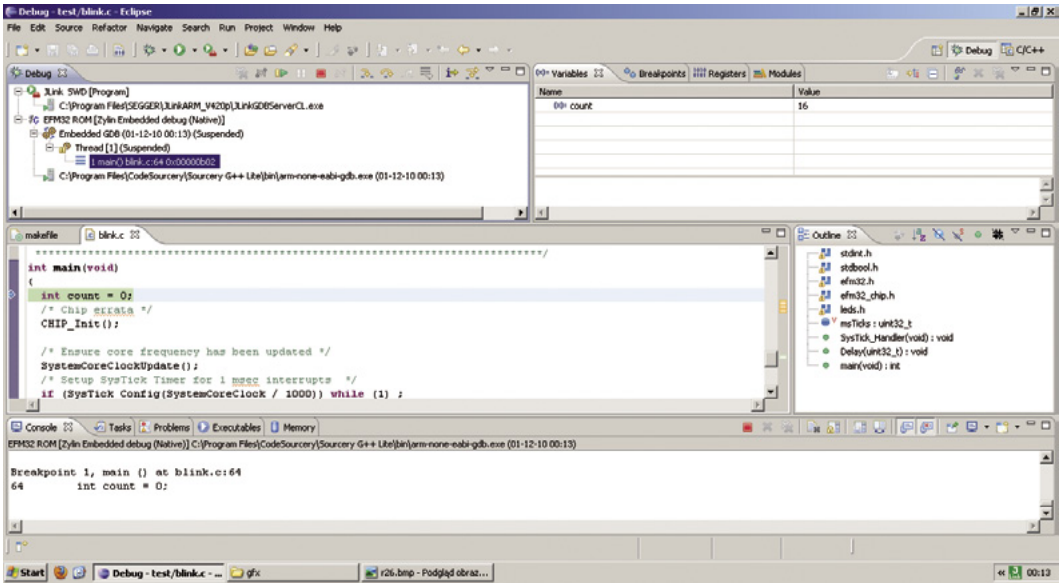
Inżynierowie z *EnergyMicro* nie próżnują. Śledząc internetową stronę firmową, przez cały czas jesteśmy informowani o nowych produktach i koncepcjach. Z ważniejszych wiadomości należy przytoczyć chęć wprowadzenia do sprzedaży w niedalekiej przyszłości dużych gekonów (*Giant Gecko*) z obsługą interfejsu USB oraz powiększoną do 1024 kB pamięcią Flash, a także mikrokontrolerów ze zintegrowanymi modułami radiowymi. Niedawno na stronie pojawił się też film przedstawiający małą płytkę rozszerzającą użyteczność płytki testowej (*STK*) i pokazujący możliwości gekonów w zakresie zasilania z baterii oraz innych źródeł „zielonej energii".

Jednak firma nie poprzestaje tylko na rozwoju sprzętu. W planach jest wprowadzenie na rynek w pierwszym kwartale 2011 roku darmowej, wielomodułowej aplikacji do wspomagania projektowania systemów opartych na rodzinie *EFM32* pod wdzięczną nazwą *Simplicity Studio*. Kluczowymi modułami mają być już dostępne programy *energyAware Profiler* i *energyAware Designer*. Oprócz tego firma obiecuje nowe, ciekawe narzędzia wchodzące w skład aplikacji, zaprojektowane zarówno przez nią samą, jak i przez oddzielne firmy.

Wojciech Gelmuda
Katedra Elektroniki
Akademia Górniczo-Hutnicza
gelmi@student.agh.edu.pl



Rysunek 25. Widok poprawnie uruchomionego programu JLinkGDBServerCL.exe z poziomu Eclipse’a



Rysunek 26. Wygląd Eclipse’a podczas poprawnie uruchomionego debuggowania