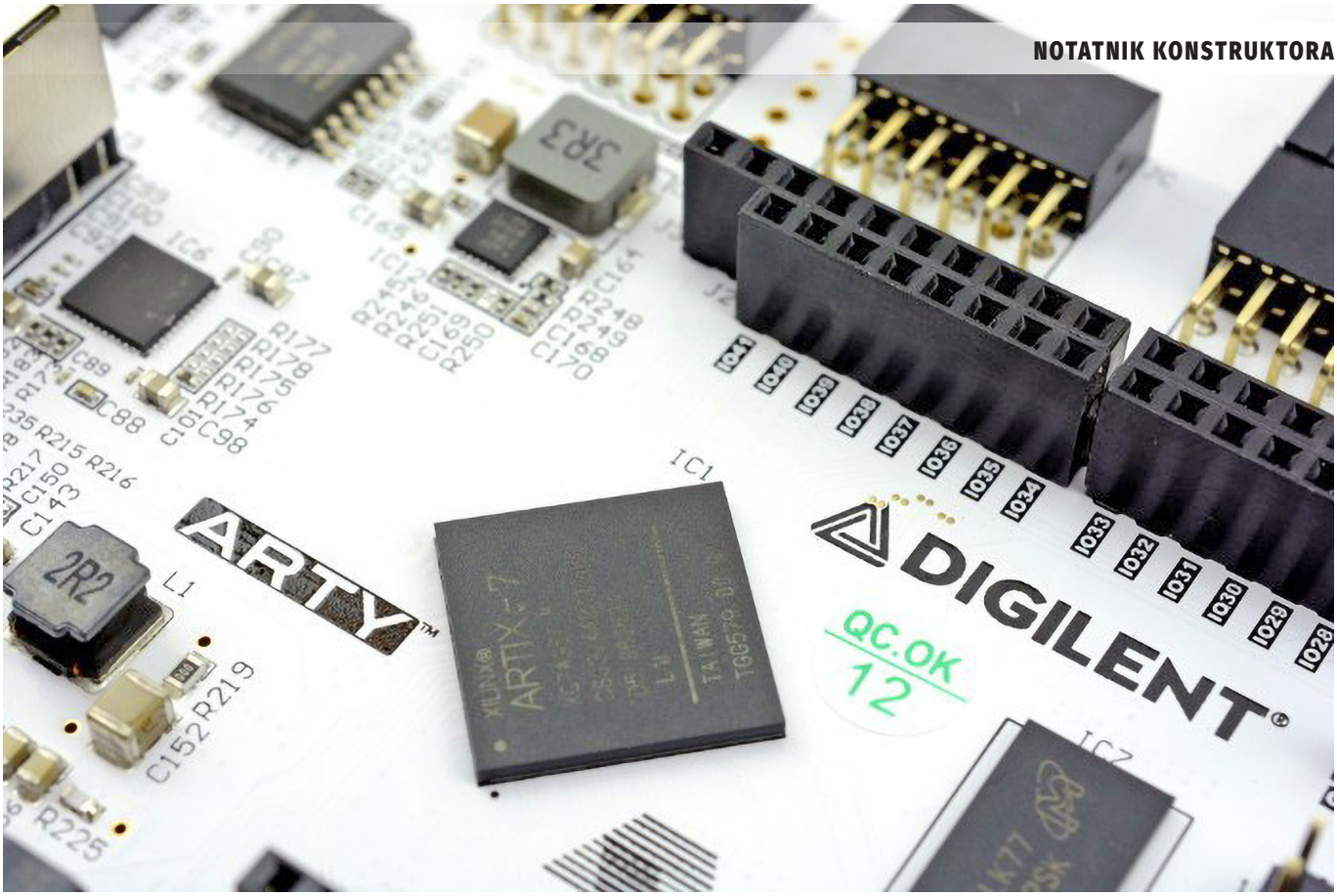


RISC-V

– mikrokontroler open source w FPGA i programowanie w Arduino

W artykule przyjrzymy się, jak łatwo dostępna na rynku płyta Digilent Arty Board, bazująca na układzie FPGA Xilinx Artix-7, może zostać skonfigurowana do pracy jako open-sourceowy mikrokontroler RISC-V, który może być zbudowany ze źródeł RTL i który można następnie zaprogramować za pomocą Arduino IDE lub makefile wbudowanego w zestaw narzędzi GNU toolchain. Do tego celu wymagana jest podstawowa znajomość oprogramowania opartego na Linuksie, systemie git oraz makefile.

Układy FPGA dają twórcom urządzeń niespotykane dotąd możliwości. W niewielkiej obudowie jest zawarty ogromny potencjał, między innymi możliwość wykonania rdzenia procesora wraz z otoczeniem, niekoniecznie tym najbliższym, ale również z modułami peryferyjnymi.

Mikrokontroler Freedom E310

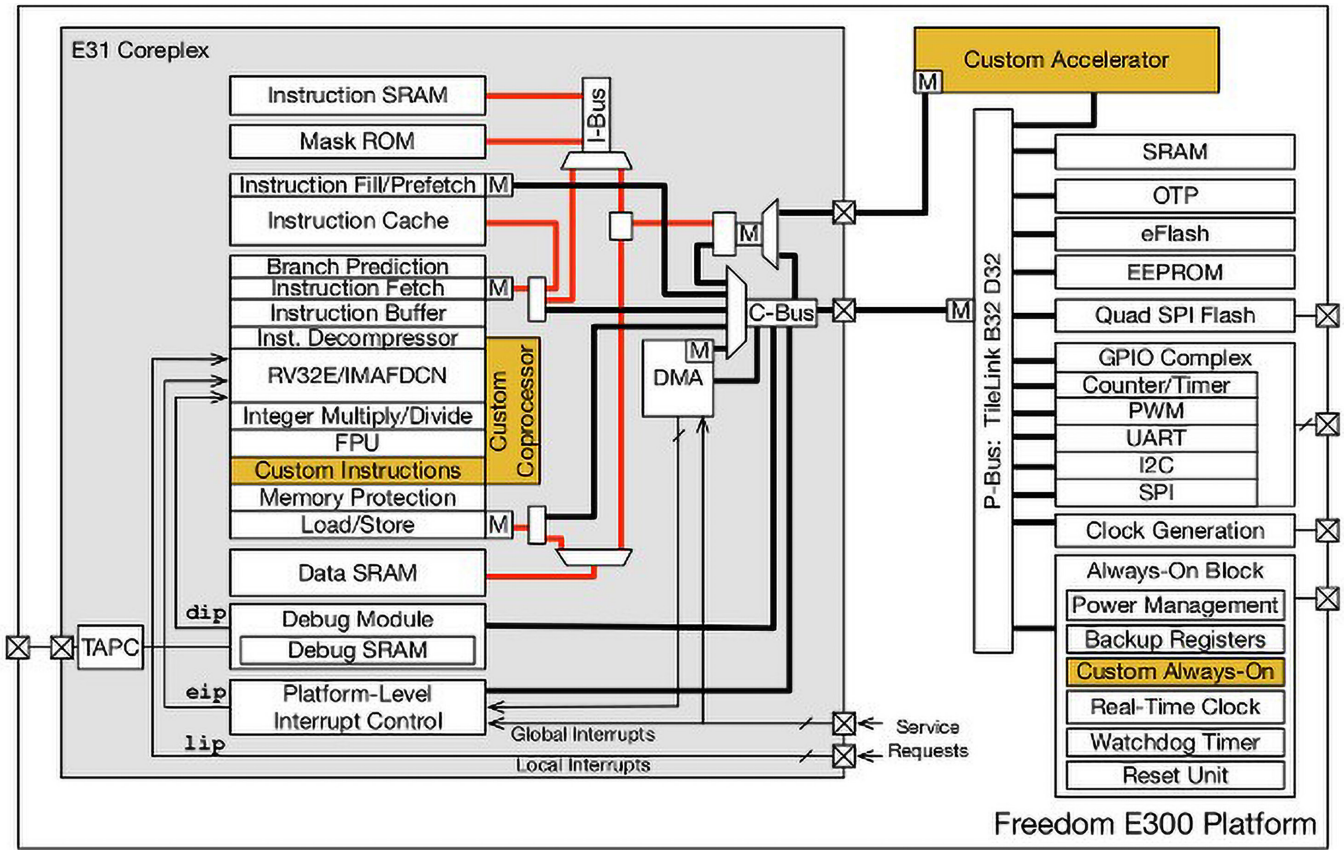
RISC-V to darmowa i otwarta architektura mikrokontrolerowa ISA, opublikowana na postawie otwartej licencji, która zachęca do powszechnego stosowania i umożliwia każdemu implementację architektury we własnym urządzeniu lub symulacji FPGA, lub ASIC. Mogą to być urządzenia klasy ultra-low power IoT, urządzenia mobilne, komputery przenośne czy stacjonarne i serwerowe oraz urządzenia przeznaczone do aplikacji HPC.

RISC-V nie jest procesorem jako takim – jest to model programowy procesora ISA. Mimo to fundacja RISC-V Foundation zapewnia implementację referencyjnego procesora o nazwie Rocket wraz z narzędziami, które ułatwiają generowanie rdzeni oraz zestaw narzędzi kompilatora GNU.

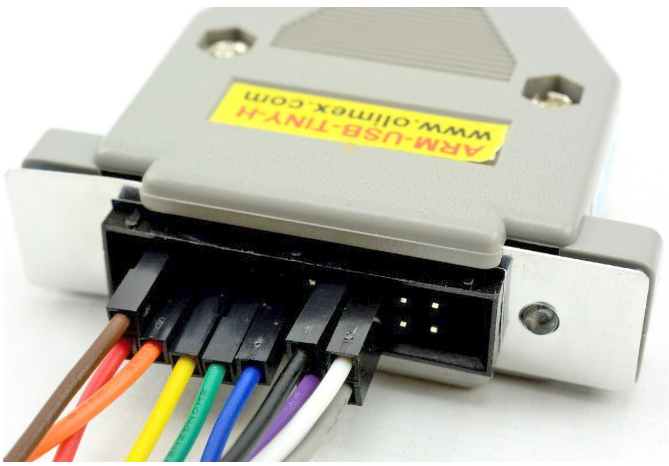
SiFive to organizacja komercyjna, która została założona przez twórców RISC-V, dostarcza produkty, które zawierają rdzenie IP i platformy SoC, opracowane i publikowane jako IP core'y. Ich podstawowa platforma sprzętowa SoC o nazwie Freedom E300 jest oparta o rdzeń E3 Coreplex, dostępny jako IP core do implementacji w układach ASIC, może być także stosowana w układach FPGA. Projekt układu Freedom E310, który umieścimy w pamięci konfiguracyjnej płytki Arty, opiera się na architekturze SoC przedstawionej na schemacie lokowym z **rysunku 1**.

Konfiguracja sprzętowa

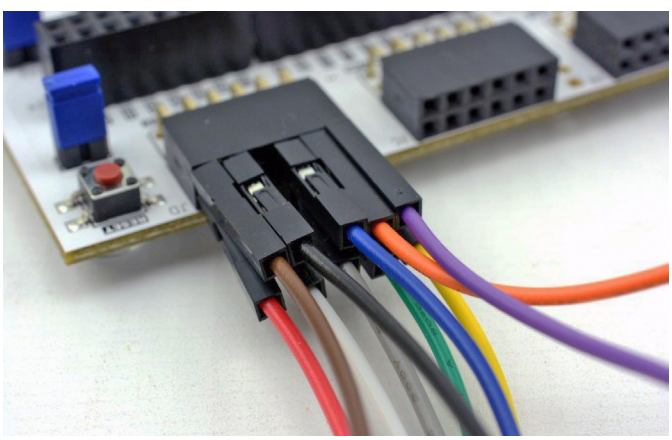
W zestawie Digilent Arty zintegrowano programator USB-JTAG, który może być użyty do programowania bezpośrednio układu FPGA Xilinx Artix-35T oraz pamięci QuadSPI, która zwykle jest używana do konfigurowania FPGA po włączeniu zasilania. Interfejs JTAG ARM-USB-TINY-H USB jest wymagany w celu zapewnienia połączenia z rdzeniem RISC-V do debugowania/programowania tego układu. W ten sposób mamy do dyspozycji w pełni programowalną platformę sprzętową, na której można zarówno dowolnie modyfikować projekt samego mikrokontrolera, jak i załadować własny kod i uruchomić go na platformie.



Rysunek 1. Schemat blokowy platformy E300



Fotografia 2. Sposób podłączenia przewodów do styków interfejsu ARM-USB-TINY-H



Fotografia 3. Interfejs JTAG ARM-USB-TINY-H USB dołączamy do mikrokontrolera zaimplementowanego w FPGA za pomocą złącza Pmod zestawu Arty

Przewodnik użytkownika zestawu Freedom E300 Arty FPGA (***http://bit.ly/2Ez0g4Z***) zawiera szczegółowe informacje o tym, jak podłączyć adapter, wraz z instrukcjami, jak zbudować narzędzia SoC i GNU toolchain, oraz zaprogramować płytkę itp. Jest to oczywiście podstawowa i oficjalna dokumentacja, do której należy się odnieść jeśli nie jesteś czegoś pewien. Kroki opisane w tym poście oparte są na instrukcjach zawartych w przewodniku.

Interfejs JTAG ARM-USB-TINY-H USB dołączamy do mikrokontrolera zaimplementowanego w FPGA za pomocą złącza Pmod zestawu Arty (**fotografia 3**). Zastosowane zostały kolory przewodów zasugerowane w przewodniku. Nie należy zakładać ani zdejmować żadnych zworek na płytce. Należy natomiast dodać nowe reguły *udav*, aby uzyskać dostęp do ARM-USB-TINY-H:

```
# These are for the Olimex Debugger for use with
E310 Arty Dev Kit
SUBSYSTEM==”usb”, ATTR{idVendor}==”15ba”,
ATTR{idProduct}==”002a”, MODE=”664”, GROUP=”plugdev”
SUBSYSTEM==”tty”, ATTRS{idVendor}==”15ba”,
ATTRS{idProduct}==”002a”, MODE=”664”,
GROUP=”plugdev”
```

Następnie uruchomić:

```
$ sudo udevadm control --reload-rules
```

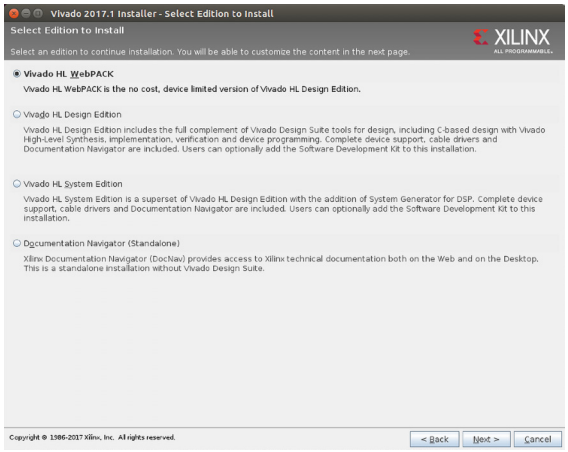
Jeśli twoje konto użytkownika nie znajduje się jeszcze w grupie *plugdev*, należy je do niego dodać, a następnie wylogować się i ponownie zalogować.

Toolchain FPGA

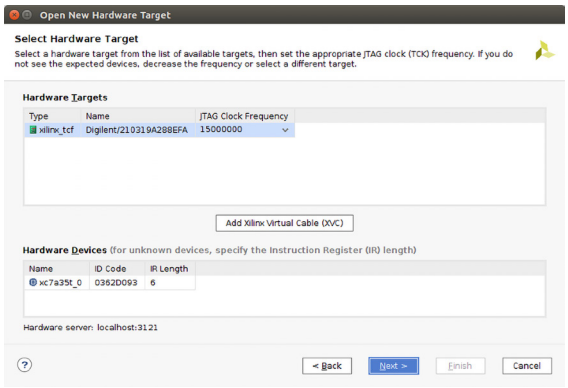
Dostępna jest bezpłatna wersja WebPack narzędzi Vivado HL (**rysunek 4**). Narzędzie do pobrania jest dostępne po zalogowaniu się na stronie Xilinx. Po zakończeniu instalacji należy dodać obsługę płyt Digilent, jest to kwestia uzyskania plików i skopiowania ich w odpowiednie miejsce. Przykładowo, po pobraniu instalatora Vivado 2017.1 procedura wyglądała następująco:

1. Nadaj instalatorowi prawo do wykonywania i uruchom jako root:

```
$ chmod +x Xilinx_Vivado_SDK_2017.1_0415_1_Lin64.bin
```



Rysunek 4. Implementacja projektu wymaga użycia bezpłatnej wersji WebPack system VivadoHLS



Rysunek 5. Po uruchomieniu Vivado otwórz okno *Hardware manager* i połącz się z zestawem Arty

```
$ sudo ./Xilinx_Vivado_SDK_2017.1_0415_1_Lin64.bin
```

2. Zainstaluj sterowniki:

```
$ cd /opt/Xilinx/Vivado/2017.1/data/xicom/
cable_drivers/lin64/install_script/install_drivers
$ sudo ./install_drivers
```

3. Pobierz pliki płytki Digilent i skopiuj je w miejsce docelowe:

```
$ git clone https://github.com/Digilent/vivado-
boards.git
$ sudo cp -r vivado-boards/new/board_files/* /opt/
Xilinx/Vivado/2017.1/data/boards/board_files/
```

4. Wróć do katalogu głównego i ustaw zmienne środowiskowe:

```
$ cd
$ source /opt/Xilinx/Vivado/2017.1/settings64.sh
```

Ten skrypt musi być wykonany za każdym razem, gdy się logujesz lub otwierany jest nowy terminal.

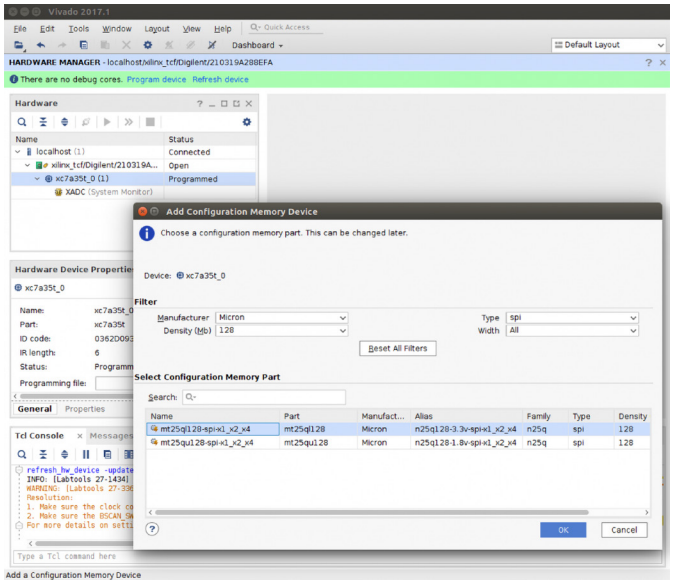
W tym momencie GUI można uruchomić, wpisując:

```
$ vivado
```

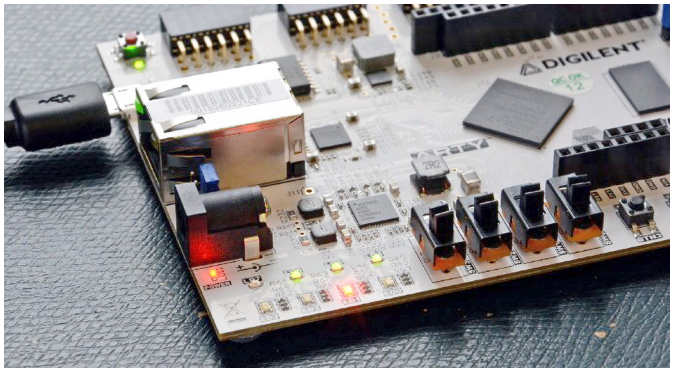
Konfigurowanie sprzętu

Plik do zaprogramowania pamięci Flash, który łączy gotowy strumień FPGA (*pre-built*) i program demonstracyjny, można pobrać ze strony internetowej SiFive (wymagana rejestracja).

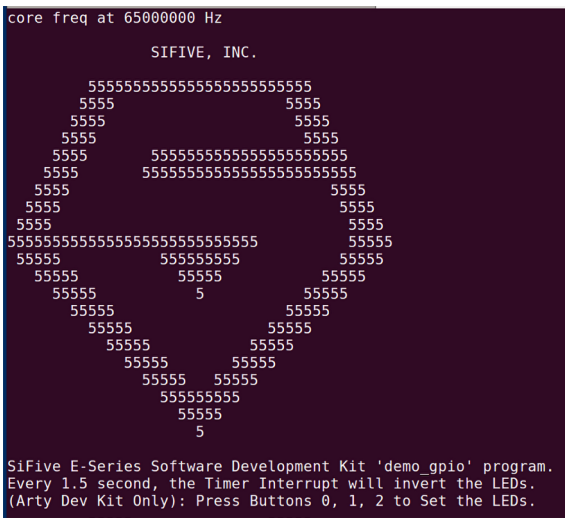
Po uruchomieniu Vivado otwórz okno *Hardware manager* i połącz się z docelową płytką (**rysunek 5**). Następnie kliknij na FPGA prawym przyciskiem myszy, wybierz opcję *Add Configuration Memory Device*, a następnie wybierz *Micron n25q128-3.3v* (**rysunek 6**). Jeśli zdecydujesz się na zaprogramowanie urządzenia, możesz wybrać pobrany plik *.mcs*. Po zakończeniu programowania, jeżeli zostanie naciśnięty czerwony przycisk oznaczony PROG, układ FPGA zostanie skonfigurowany i powinien uruchomić się program demonstracyjny, sygnalizujący swoje działanie za pomocą migania diod LED1 i LED2 (**fotografia 7**).



Rysunek 6. Wybór pamięci konfigurującej Flash w systemie Arty



Fotografia 7. Program demonstracyjny sygnalizuje swoje działanie za pomocą migania diod LED1 i LED2



Rysunek 8. Komunikat startowy konsoli

Jeśli używasz emulatora terminala do połączenia z /dev/ttyUSB1, powinieneś zobaczyć komunikat wyjściowy (**rysunek 8**). W tym momencie można pominąć następną sekcję i przejść bezpośrednio do programowania SoC za pośrednictwem Arduino IDE/GNU toolchain, a w późniejszym etapie przystąpić do samodzielnego budowania SoC.

Budowanie układu

Najpierw musimy rekursywnie sklonować repozytorium GitHub, które zawiera źródła **Chisel RTL** (***http://bit.ly/2QKa4QM***) dla platform SiFive E300 i U500 (**rysunek 9**):

```
$ git clone --recursive https://github.com/sifive/
freedom.git
```

Może to zająć trochę czasu, ponieważ klonowane są wszystkie pod moduły git, takie jak Rocket Chip Generator and Chisel.

Gdy już mamy wszystko gotowe, źródła Chisel mogą być skompilowane do Verilog przez wykonanie polecenia:

```
$ make -f Makefile.e300artydevkit verilog
```

Aby utworzyć plik konfiguracji pamięci, należy najpierw upewnić się, że zmienne środowiskowe Vivado zostały ustawione przez pozyskanie odpowiedniego pliku powłoki, jak wspomniano wcześniej, a następnie wprowadzić:

```
$ make -f Makefile.e300artydevkit mcs
```

To doprowadzi do syntezy układu FPGA Xilinx wraz z place-and-route itp., i po chwili powinieneś mieć nowy plik .mcs, który może być użyty do zaprogramowania konfiguracji Flash.

Zauważ, że domyślnie wygenerowany obraz Flash nie zawiera programu demonstracyjnego, tak jak w wersji *pre-built*. Jednak program można wypalić w obrazie, edytując odpowiedni plik makefile.

Kompatybilność z Arduino

Zdecydowanie najprostszym sposobem na rozpoczęcie tworzenia oprogramowania dla Freedom E310 jest skorzystanie z faktu, że istnieje wsparcie dla Arduino IDE z przeznaczeniem dla płyty Arty (**rysunek 10**). Wszystko, co musimy zrobić, to najpierw przejść *File* → *Preferences* → *Settings* i dodać następujący *URL* do *Board Managera*: ***http://bit.ly/2Gv6ghB***.

Następnie możemy przejść do *Tools* → *Board* → *Board Manager*, wyszukać SiFive Freedom Boards i zainstalować wsparcie dla nich. Teraz możemy wybrać płytkę Freedom E300 Arty DevKit, przed otwarciem przykładu migania, a następnie skompilować i wgrać go na zestaw. Jeśli w tym momencie programowanie się nie powiedzie, możliwe, że USB-JTAG nie został poprawnie podłączony do złącza Pmod, *udev* nie jest skonfigurowany albo nie jesteś w grupie *plugdev*.

freedom-e-sdk

Jeśli używasz własnego edytora tekstowego i użycie makefile jest trudne, nie obawiaj się, ponieważ równie łatwo można zbudować samodzielny GNU toolchain, **freedom-e-sdk**.

1. Najpierw musimy uzyskać zależności budowania:

```
$ sudo apt-get install autoconf automake libmpc-
dev libmpfr-dev libgmp-dev gawk bison flex texinfo
libtool libusb-1.0-0-dev make g++ pkg-config
libexpat1-dev zlib1g-dev
```

2. Następnie sklonuj źródła i zbuduj:

```
$ git clone --recursive https://github.com/sifive/
freedom-e-sdk.git
$ cd freedom-e-sdk
$ make tools
```

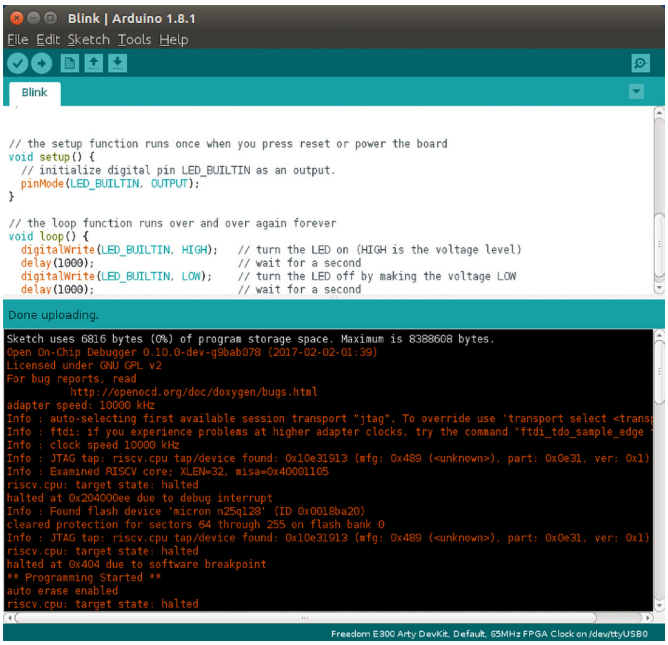
3. Aby zbudować, a następnie załadować program demonstracyjny, wpisz:

```
$ make software PROGRAM=demo_gpio
BOARD=freedom-e300-arty
$ make upload PROGRAM=demo_gpio
BOARD=freedom-e300-arty
```

```
source script/cfgmem.tcl
# lassign $argv mcsfile bitfile datafile
# set iface spix4
# set size 16
# set bitaddr 0x0
# write_cfgmem -format mcs -interface $iface -size $size \
#   -loadbit "up ${bitaddr} ${bitfile}" \
#   -loaddata [expr {$datafile ne "" ? "up 0x400000 ${datafile}" : ""}] \
#   -file $mcsfile -force
Command: write_cfgmem -format mcs -interface spix4 -size 16 -loadbit {up 0x0 obj/syste
m.bit} -loaddata {} -file obj/system.mcs -force
Creating config memory files...
Creating bitstream load up from address 0x00000000
Loading bitfile obj/system.bit
Writing file obj/system.mcs
Writing log file obj/system.prm
==================================
Configuration Memory information
==================================
File Format        MCS
Interface          SPIX4
Size               16M
Start Address      0x00000000
End Address        0x00FFFFFF

Addr1         Addr2         Date                    File(s)
0x00000000    0x0021728B    Jun  5 15:54:36 2017    obj/system.bit
0 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
write_cfgmem completed successfully
INFO: [Common 17-206] Exiting Vivado at Mon Jun  5 15:55:22 2017...
make[1]: Leaving directory '/var/silicon/freedom/fpga/e300artydevkit'
cp /var/silicon/freedom/fpga/e300artydevkit/obj/system.mcs /var/silicon/freedom/builds
/e300artydevkit/sifive.freedom.everywhere.e300artydevkit.E300ArtyDevKitConfig.mcs
andrew@snow:/var/silicon/freedom$
```

Rysunek 9. Wygląd okna terminala po zaprogramowaniu pamięci konfigurującej Flash



Rysunek 10. Do programowania prezentowanej platformy można używać środowiska Arduino

Podsumowanie

Jak widać, przy niewielkich nakładach i w krótkim czasie możesz być gotowy do pracy dzięki konfigurowalnej platformie mikrokontrolerów, w której możesz swobodnie uruchomić projekt SoC lub jeżeli jest to na razie zbyt skomplikowane, używać platformy z Arduino.

Andrew Back

Miłośnik Open Source (hardware i software!), skarbnik i dyrektor Free Silicon Foundation, organizator festiwalu technologicznego Wuthering Bytes i założyciel Open Source Hardware User Group.