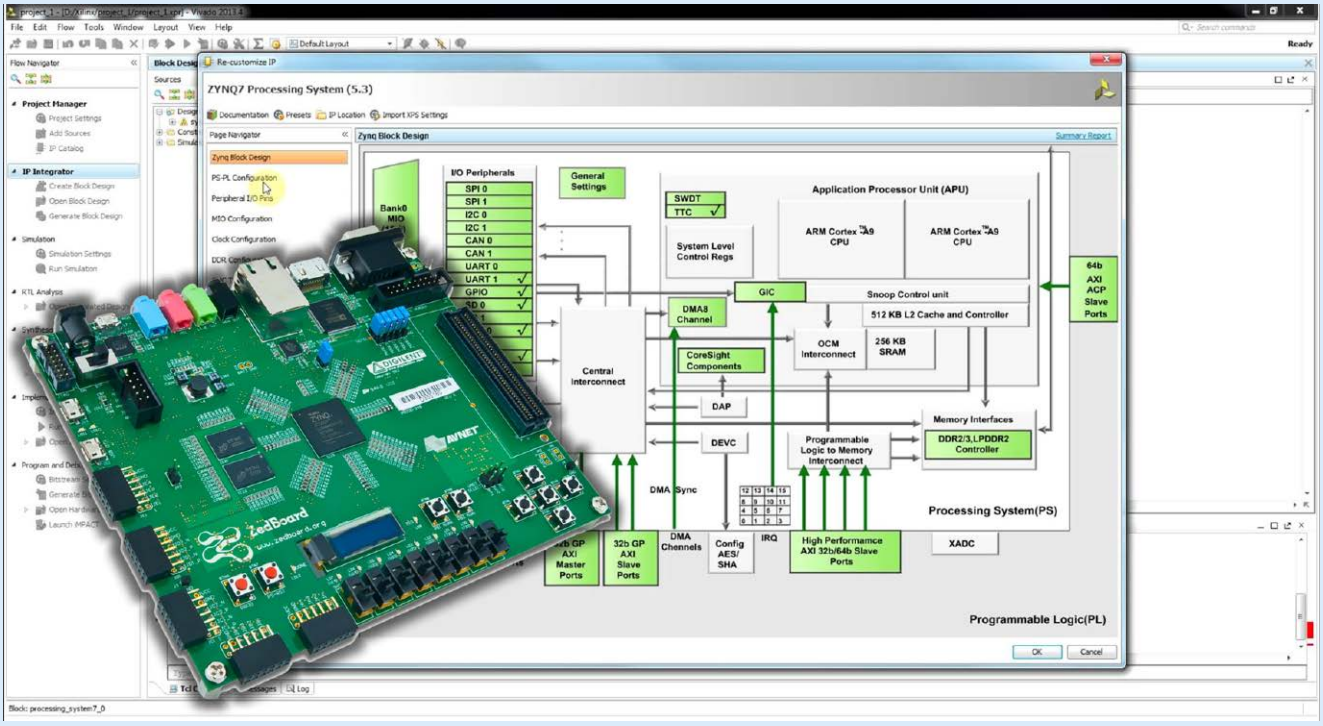


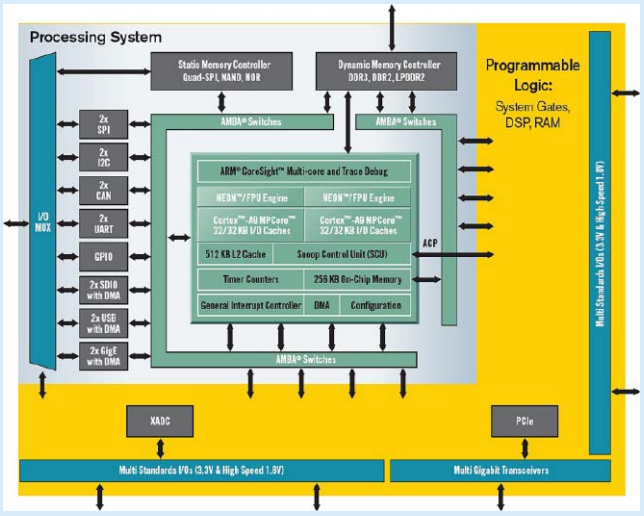
Jak używać układów SoC Xilinx Zynq-7000 z Linuksem – proste przykłady (1)

Układy FPGA pozwalają na tworzenie bardzo specjalistycznych projektów. Ich duża elastyczność ma jednak pewną cenę: podczas projektowania trzeba przestawić się na myślenie współbieżne, co wielu programistom sprawia spore trudności. Szczególnie problematyczne są projekty, w których wymagany jest stos sieciowy z np. serwerem HTTP z interfejsem sterującym. Oczywiście można zaimplementować z nim tzw. soft-procesor (np. NIOS firmy Altera lub Microblaze firmy Xilinx), jednak nawet to rozwiązanie wymaga pewnej wiedzy na temat projektowania systemów komputerowych z użyciem tych komponentów. Rozwiązaniem problemu mogą być układy SoC (System-on-Chip) będące połączeniem FPGA i procesora. W artykule omówiona zostanie płytka Zedboard, której serce stanowi układ SoC z rodziny Zynq firmy Xilinx.

Zedboard jest płytką przeznaczoną głównie dla studentów i entuzjastów układów FPGA. Mimo relatywnie niskiej ceny w porównaniu do płytek ewaluacyjnych do profesjonalnych zastosowań, ma ona spore możliwości. Jej serce stanowi układ Zynq-7000 AP SoC XC7Z020. Jest ona też wyposażona w zestaw podstawowych peryferii przydatnych do prostych i zaawansowanych projektów i do komunikacji ze światem zewnętrznym:

- Pamięć RAM – 512 MB DDR3.
- Pamięć Flash – 256 Mb Quad-SPI Flash.
- Złącze karty SD.
- 10/100/1000 Ethernet.
- USB OTG 2.0.
- Gniazdo FMC-LPC.
- 5 gniazd Pmod.
- Gniazda HDMI i VGA.
- Wyświetlacz OLED 128×32.
- Interfejs USB-JTAG.
- 9 diod LED.
- 7 przycisków.
- 8 przełączników.

Układ Zynq-7000 składa się z dwóch części – z systemu mikroprocesorowego z procesorem ARM (dwa rdzenie ARM



Rysunek 1. Schemat blokowy układu Zynq7000

Cortex-A9) i logiki programowalnej FPGA (**rysunek 1**). Ta pierwsza często w literaturze nazywana jest skrótowo PS (*Processing System*), zaś druga – PL (*Programmable Logic*). Takie połączenie pozwala na relatywnie proste zaprojektowanie systemu komputerowego, w którym część sterująca stanowi program napisany na system Linux uruchomiony na PS, natomiast zadania wymagające obliczeniowo wykonywane są przez PL. Rozwiązanie to zapewnia dużą oszczędność czasu i zasobów komórek logicznych w stosunku do takiego, w którym system implementuje się wyłącznie w układzie FPGA z soft-procesorem. Ponadto, często jest ono wydajniejsze pod względem szybkości komunikacji w stosunku do systemu, w którym do układu FPGA jest podłączony zewnętrzny, procesor. Co więcej, dzięki takiemu rozwiązaniu dostajemy od razu gotowy kanał komunikacyjny z zaprojektowanymi blokami FPGA i nie trzeba go samodzielnie implementować.

PS może komunikować się z PL za pomocą kilku interfejsów. Podstawowym jest typowa dla procesorów ARM szyna danych AXI, będąca częścią standardu komunikacji AMBA. Tego typu interfejsów jest w sumie dziewięć, a część z nich można dodatkowo konfigurować, wybierając na przykład szerokość szyny danych (32 lub 64 bity). Oprócz szyny AXI PL może się kontaktować z PS poprzez interfejs EMIO (*Extended Multiplexed In/Out Interface*). Może być on skonfigurowany w różny sposób, na przykład jako wyprowadzenie pinów GPIO z PS do PL lub jako port łączący PL z komponentami PS, np. UART lub SPI. Możliwe jest również zgłaszanie przerwań między PL i PS, zarówno od procesora do części FPGA, jak i w stronę przeciwną.

Programmable Logic (PL)

Główną część logiki programowalnej stanowi 13300 tzw. sekcji logicznych (*Logic Slices*). Każda z nich zawiera cztery sześciowejściowe tablice LUT (*Look Up Tables*) i osiem jednobitowych przerzutników (*flip-flops*). Daje to w sumie 53200 LUT i 106400 przerzutników. Ponadto jest do dyspozycji 140 konfigurowalnych bloków pamięci RAM, każdy po 36 kb, co daje w sumie 630 kB pamięci z szybkim dostępem. Jeśli nasz projekt wymaga dużej ilości szybkich obliczeń, można do nich wykorzystać 220 układów dodająco-mnożących (*DSP Slices*), oczywiście również konfigurowalnych. Na styku logiki ze światem zewnętrznym znajdują się bloki wejścia wyjścia (*in/out blocks, IOBs*). Wszystkie wymienione elementy są połączone przez macierz przełączającą (*Switch Matrix*).

Dodatkowo w części PL znajduje się blok konwertera analogowo-cyfrowego (XADC), składający się z dwóch osobnych, 12-bitowych przetworników. Do każdego z nich można podłączyć jeden z 17 zewnętrznych analogowych sygnałów. Dodatkowo, alternatywnie jeden z nich może służyć do pomiarów napięć wewnątrz układu lub jego temperatury.

Do działania każdego synchronicznego układu cyfrowego potrzebny jest zegar. PL może skorzystać z jednego z wyjść zegarowych pochodzących z PS lub z własnych, niezależnych źródeł.

Ostatnim ważnym elementem logiki programowalnej jest port JTAG. Za jego pomocą można zarówno zaprogramować PL, jak i skorzystać z wielu opcji debugowania projektów, np. wysyłanie danych po szynie AXI, bez potrzeby programowania procesora. JTAG ma również możliwość konfiguracji XADC, co znacznie upraszcza np. pomiar temperatury układu.

Processing System (PS)

Główną część systemu mikroprocesorowego stanowi APU (*Application Processing Unit*). Jego najważniejszymi elementami są dwa rdzenie procesora ARM Cortex-A9. Każdemu z nich towarzyszy blok obliczeniowy *NEON Media Processing Engine* (MPE), realizujący sprzętowo obliczenia zmiennoprzecinkowe, oraz rozszerzający listę rozkazów procesora o instrukcje typu SIMD (*Single Instruction Multiple Data*), przydatne przy obliczeniach DSP.

Dostęp do zewnętrznej pamięci RAM typu DDR3 jest zapewniony przez interfejs. Jest do niego podłączone zarówno APU, jak i część PL, poprzez specjalne porty AXI.

Omawiając część PS, warto też wspomnieć o bloku peryferii wejścia wyjścia (*IO Peripherals*), w którym znajdziemy kilka ważnych interfejsów: USB, gigabitowy Ethernet, SDIO, SPI, I²C, CAN i piny GPIO. Możliwe jest wyprowadzenie każdego z nich na piny wyjściowe układu należące do części PS lub udostępnienie ich części PL. Do konfiguracji tych połączeń służy blok MIO (*Multiplexed In/Out Interface*).

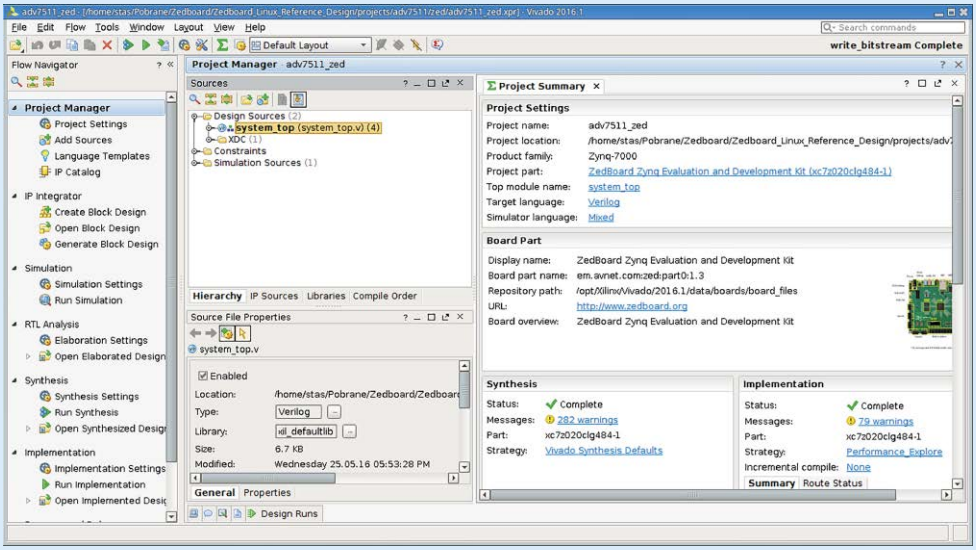
Część peryferii podłączona jest do układu poprzez piny należące do PS, bezpośrednio do sprzętowych interfejsów. Należą do nich: pamięci RAM i Flash, port karty SD, Ethernet, USB OTG, USB UART, przycisk reset, generator zegara 33 MHz, jeden port PMOD, jedna dioda LED i dwa przyciski użytkownika. Pozostałe peryferia podłączone są do części PL i niezbędne jest użycie własnych lub pochodzących z zewnątrz bloków IP-core.

Oprogramowanie i dokumentacja

Podstawowym źródłem informacji o zestawie jest strona ***zedboard.com***. Można tam znaleźć mnóstwo tutoriali i przykładowych projektów. Poniżej zostanie omówionych kilka z nich.

- *ZedBoard Getting Started Guide* – podstawowe informacje o płytce i sposób uruchomienia przykładów.
- *ZedBoard Hardware User's Guide* – informacje o układach Zynq i jego peryferiach.
- *Zynq ZedBoard Vivado Workshop* – tutorial na temat oprogramowania Vivado, służącego do projektowania układów cyfrowych i konfiguracji PS, oraz układu Zynq, na przykładzie płytki Zedboard.
- *Zynq Concepts, Tools, and Techniques on ZedBoard* – obszerny opis podstawowych narzędzi i technik tworzenia projektów dla Zynq. Jest tu również opis uruchamiania gotowego obrazu Linuksa na różne sposoby i zdalnego uruchamiania i debugowania przykładowego programu z poziomu SDK.
- *HDMI Bare Metal Reference Design Using ADV7511 and ADI IP* – ten dokument zawiera opis podstawowego systemu komputerowego na FPGA i podstawowej konfiguracji procesora niezbędnej do uruchomienia Linuksa i niemal wszystkich peryferii dostępnych na płytce. Jest to świetna baza to tworzenia własnych projektów. Warto zwrócić uwagę na to, że stosowane są tu darmowe IP Core firmy Analog Devices, które można pobrać w nowszych wersjach

ze strony AD na Github. Instrukcja budowania tych IP Core'ów znajduje się na stronie ***https://goo.gl/RVuSRy***. Niestety, uruchomienie tych projektów wymaga zainstalowania starszej wersji Vivado (skrypty TCL budujące projekty nie są kompatybilne wstecz), ale warto to zrobić, gdyż jest to znacznie szybsze niż samodzielne budowanie całego projektu od podstaw i jest mniejsze ryzyko błędów. Po zakończonym procesie kompilacji można gotowy projekt zaimportować do najnowszej wersji Vivado.



Rysunek 2. Okno główne środowiska Vivado Integrated Design Environment

Poza wymienionymi materiałami warto zapoznać się z darmową książką *The Zynq Book* dostępną na stronie ***https://goo.gl/45V5F6***. Można tam znaleźć mnóstwo przydatnych informacji zarówno na temat układów z rodziny Zynq, jak i płytki Zedboard. Są tam też opisy metod projektowania systemów wbudowanych, w tym opartych na systemie Linux.

Większość peryferii na płytce Zedboard podłączonych jest do części logiki programowalnej, więc konieczne są specjalne bloki IP core w FPGA, które je obsłużą. Można opracować takie bloki samemu, ale można też skorzystać z propozycji producenta układów scalonych obsługujących np. HDMI, oferującego je bezpłatnie.

Sercem płytki Zedboard jest układ Zynq firmy Xilinx, więc podstawowym narzędziem potrzebnym do tworzenia projektów dla tego urządzenia jest środowisko Vivado. Pozwala ono na zaprojektowanie części logiki programowalnej, skonfigurowanie systemu procesorowego, jak również na symulację pracy całego systemu. Wraz ze środowiskiem Vivado użytkownik dostaje również SDK – środowisko programistyczne oparte na programie Eclipse, służące między innymi do tworzenia aplikacji na PS. Mogą to być zarówno programy typu *bare metal*, czyli uruchamiane bezpośrednio na procesorze, jak i aplikacje na system operacyjny, np. Linux, lub któryś z systemów czasu rzeczywistego. Ponieważ bardzo często w projektach na Zynq stosuje się system Linux, firma Xilinx dostarcza własną dystrybucję tego systemu wraz z narzędziami ułatwiającymi budowanie jądra i innych komponentów potrzebnych do jego uruchomienia.

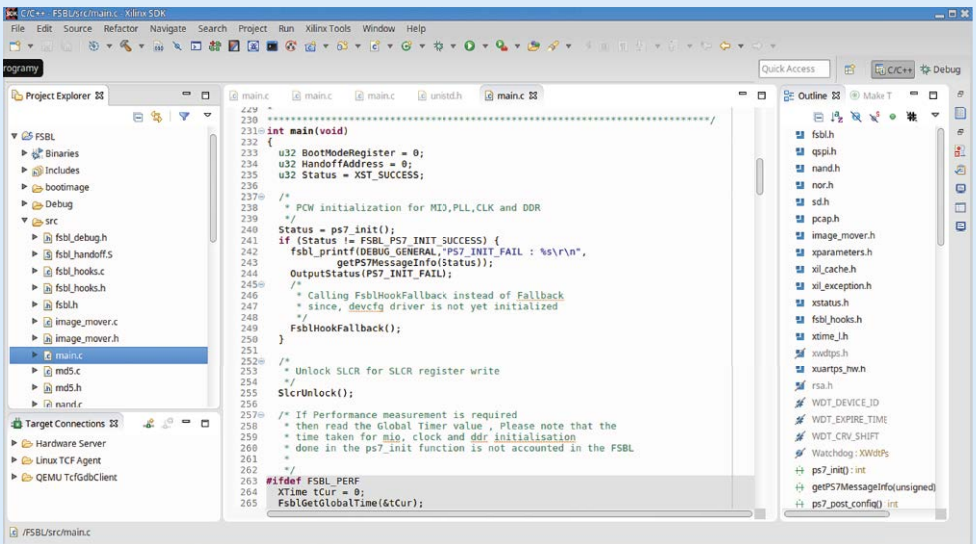
Niektóre z peryferii dostępnych na Zedboard zrealizowane są za pomocą układów scalonych firmy Analog Devices. Firma ta, mając świadomość faktu, że jej układy są stosowane na wielu płytkach z układami FPGA, udostępnia bezpłatnie projekty z gotowymi blokami IP core do obsługi produkowanych przez siebie układów. W dalszej części artykułu zaprezentuję podstawowe narzędzia firmy Xilinx, a następnie pokażę, jak wykorzystać gotowe projekty, aby szybciej przystąpić do realizacji własnych pomysłów.

Vivado Design Suite

Vivado DS jest kompletnym narzędziem do projektowania systemów SoPC. Pozwala ono na tworzenie złożonych projektów, ze szczególnym naciskiem na dekompozycję projektu na mniejsze moduły zwane *Intellectual Property Cores* (IPC). Na każdym etapie projektowania możliwe jest szczegółowe przetestowanie tworzonego projektu lub IPC, które obejmuje symulację, jak również planowanie ścieżek zegarów, zużycie energii, analizę czasową itd.

Podstawowym interfejsem jest GUI zwane Vivado *Integrated Design Environment* (IDE). Na **rysunku 2** pokazano przykładowy widok środowiska, zaraz po otwarciu projektu. Po lewej stronie znajdują się przyciski uruchamiające poszczególne etapy ścieżki projektowej, w środkowej części widać nawigator projektu, zaś po prawej – pole robocze, w którym można otworzyć edytor tekstu, graficzny edytor systemu, symulację itd. Na rysunku widać w polu roboczym podsumowanie projektu.

Oprócz pracy w GUI możliwe jest także pisanie skryptów TCL, które automatycznie mogą stworzyć projekt, odpowiednio połączyć bloki funkcjonalne i wykonać cały proces budowania projektu, zakończony wygenerowaniem pliku Bitstream, który zawiera konfigurację PL.



Rysunek 3. Środowisko programistyczne XSDK

Xilinx SDK

XSDK jest środowiskiem programistycznym do pisania aplikacji na procesor Zynq i Microblaze. Jest ono oparte na znanym, standardowym środowisku Eclipse (**rysunek 3**). Poza projektowaniem aplikacji, można za jego pomocą między innymi:

- Wygenerować *Board Support Package* (BSP, zestaw funkcji niezbędnych do pisania aplikacji na konkretny system).
- Wygenerować *First Stage Boot Loader* (FSBL – program uruchamiany na początku, po włączeniu procesora, służący do skonfigurowania części PS, jak i załadowania bitstreamu – pliku z konfiguracją logiki programowalnej).
- Wygenerować projekt z platformą sprzętową – projekt zawierający potrzebne dane na temat platformy sprzętowej, na którą pisane są programy.
- Stworzyć obraz uruchomieniowy, zawierający FSBL (*First Stage Boot Loader*), bitstream i SSBL (*Secound Stage Boot Loader*). Funkcję SSBL pełni zwykle program U-Boot służący do uruchomienia Linuksa. Taki obraz jest niezbędny do uruchomienia na Zynq Linuksa z karty SD lub pamięci Flash.
- Zaprogramować część FPGA i pamięć Flash.

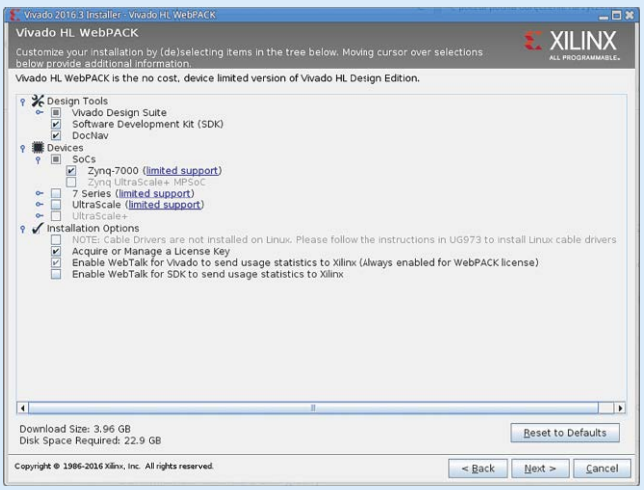
XSDK pozwala na pisanie, uruchamianie i debugowanie projektów, przeznaczonych do uruchomienia zarówno bezpośrednio przez procesor, jak i przez system operacyjny, w tym Linux. Debugowanie linuksowych aplikacji możliwe jest dzięki aplikacji TCF Agent, za pomocą serwera TCF Server uruchomionego na docelowym systemie.

Petalinux

Petalinux to zestaw skryptów ułatwiających konfigurację, budowanie, instalację i uruchomienie systemu Linux. Pozwala na kompilację jądra Linuksa, programu U-Boot, jak również programów dostarczonych przez użytkownika. Z jego pomocą można też wygenerować *rootfs* – podstawowy system plików, zawierający skrypty uruchomieniowe systemu i podstawowe programy. Możliwe jest również wygenerowanie obrazu uruchomieniowego, wyżej opisanego przy środowisku XSDK. Wszystko to można zrobić za pomocą jednej lub zaledwie kilku komend, co znacznie ułatwia proces budowania i uruchamiania systemu.

Instalowanie środowiska

Zanim zostanie utworzony pierwszy projekt, należy przygotować środowisko pracy. W zależności od preferencji można zainstalować Vivado na komputerze z zainstalowanym systemem Windows lub jedną z kilku dystrybucji sytemu Linux. Osobiście zdecydowanie polecam używanie systemu Linux, gdyż ułatwi to w przyszłości budowanie jądra Linuksa, które uruchomimy na Zynq. Kroki opisane w tym tutorialu wykonano na komputerze z zainstalowanym Ubuntu 16.04, ale powinny zadziałać również na innych dystrybucjach. Tam, gdzie będzie to możliwe i konieczne, podam również wskazówki, jak niektóre zadania wykonać na komputerze z systemem Windows.



Rysunek 4. Okno środowiska gotowego do pracy

Instalacja środowiska Vivado jest bardzo prosta. Ze strony ***https://goo.gl/QmK8qn***, po uprzedniej rejestracji, należy pobrać mały (ok. 80 MB) plik instalacyjny odpowiedni dla danego systemu operacyjnego, który po uruchomieniu pobierze i zainstaluje całe środowisko. Pod Windows po pobraniu instalatora po prostu należy go uruchomić, zaś pod Linuksem może być konieczne nadanie mu praw wykonania.

Należy pobrać najnowszą wersję Vivado, konieczną do pracy i jedną ze starszych, która będzie niezbędna do wygenerowania projektu ze skryptów, które zostaną pobrane w następnym kroku. Numer starszej wersji można znaleźć w dokumentacji repozytorium, z którego będziemy pobierali pliki HDL (opis w dalszej części artykułu).

Uruchamiamy konsolę i wpisujemy:

```
cd <ścieżka do folderu, do którego został pobrany plik instalacyjny>
chmod +x <nazwa pliku instalacyjnego>
```

W systemie Linux domyślnym katalogiem, do którego należy instalować programy niebędące w repozytorium, jest katalog */opt*. Należy on jednak do użytkownika *root*, więc żeby było możliwe zainstalowanie tam Vivado, trzeba uruchomić instalator z uprawnieniami *roota*. Nie jest to jednak praktyka rekomendowana przez Xilinksa. Aby uruchomić instalator bez uprawnień administratora, dobrze jest stworzyć w katalogu */opt* podkatalog Xilinx (do którego później zostanie zainstalowane całe środowisko) i nadać mu uprawnienia dostępu dla zwykłego użytkownika. Nie jest to wymagane do poprawnego przebiegu instalacji, ale dużo ułatwia dalszą pracę.

Uruchamiamy konsolę i wpisujemy:

```
cd /opt
sudo mkdir Xilinx
sudo chown $USER:$USER Xilinx
```

Następnie przechodzimy do katalogu, w którym znajduje się plik instalacyjny i uruchamiamy go. Pierwszy ekran umożliwia zmianę konfiguracji połączenia sieciowego i liczbę wątków używanych w instalacji. W drugim kroku należy się zalogować, używając danych, które podało się przy rejestracji na stronie ***www.xilinx.com***. W kolejnym kroku należy w trzech miejscach zaznaczyć akceptację licencji. Następny ekran umożliwia wybór edycji Vivado. Jeśli ma się wykupioną licencję na płatną edycję, należy ją wybrać, w przeciwnym wypadku należy zaznaczyć darmową wersję WebPACK. W kolejnym kroku wybieramy komponenty do instalacji. W tym przypadku niezbędne będzie SDK, które zaznaczamy, ale nie będzie potrzebna obsługa układów innych niż zynq-7000, więc pozostałe można odznaczyć (będzie możliwość doinstalowania ich, jeśli w przyszłości będą potrzebne). Po wybraniu odpowiednich opcji ekran powinien wyglądać jak na **rysunku 4**.

Powyższe ustawienia dotyczą najnowszej wersji Vivado. Dla wersji potrzebnej do wykonania skryptów budujących projekt od ADV nie jest potrzebne SDK, więc przy instalacji tej wersji można je odznaczyć.

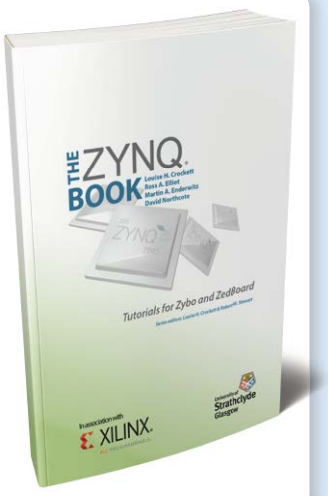
W kolejnym kroku wybieramy ścieżkę instalacji. Jeśli zostały wykonane kroki dotyczące utworzenia folderu Xilinx w folderze */opt* i zmiany właściciela, to można zostawić opcje domyślne. Jeśli nie – należy wybrać odpowiedni folder lub wyłączyć instalator i wykonać wspomniane wyżej czynności. Ostatni ekran zawiera podsumowanie i jeśli wszystko jest w porządku, kliknięcie przycisku *install* spowoduje pobranie plików i zainstalowanie środowiska.

Dokładną instrukcję instalacji środowiska Vivado można znaleźć w instrukcji UG973 na stronie firmy Xilinx.

Instalowanie sterowników

W systemie Windows sterowniki instalują się automatycznie podczas instalacji środowiska Vivado, jednak pod Linuksem należy to zrobić ręcznie. W tym celu otwieramy konsolę i wpisujemy:

```
cd /opt/Xilinx/
Vivado/<wersja_
vivado>/data/xicom/
cable_drivers/lin64/
install_script/
install_drivers
sudo ./
install_drivers
```



Fanom układów SoC polecamy dostępną bezpłatnie książkę poświęconą praktycznym aspektom stosowania tych układów w praktyce. Szczegółowe informacje są dostępne pod adresem: ***www.zynqbook.com***

Budowanie projektu

Jak już wspomniałem, firma Analog Devices udostępnia projekty z blokami IP core do obsługi produkowanych przez nią układów. Projekty te dostarczane są w postaci skryptów, które tworzą projekt i konfigurują go, wybierają odpowiednie bloki IP core oraz łączą je i budują projekt. Za pomocą jednej komendy można wygenerować docelowy plik *bitstream*, który potem ładuje się do FPGA.

Skrypty pobieramy ze strony ***https://goo.gl/hKYxWT***. W momencie pisania tego artykułu należało wybrać wersję pre-release (*branch hdl_2016_r2*), ponieważ pojawia się problem z aktualizacją Vivado w wersji 2015.4 do wersji 2015.4.2, potrzebnej do wygenerowania projektu z wersji *stable*. W przyszłości ta wersja prawdopodobnie stanie się wersją stabilną, więc trzeba zwrócić na to uwagę. Ta wersja potrzebuje Vivado w wersji 2016.2.

Po pobraniu archiwum należy rozpakować je w miejscu, gdzie chce się przechowywać pliki z projektami. Interesujący nas projekt nazywa się *adv7511* w wersji na Zedboard. Znajduje się on w katalogu *projects/avd7511/zed*. Aby ten projekt zadziałał, trzeba również zbudować projekty zależne, definiujące bloki IP core.

Sposób uruchomienia skryptów zależny jest od systemu operacyjnego. W systemie Linux wszystko sprowadza się do wykonania w konsoli następujących komend:

```
source /
opt/Xilinx/
Vivado/2016.2/
settings64.sh
cd <folder,
w którym zostało
rozpakowane
archiwum>/
projects/adv7511/
zed
make
```

Budowanie projektu powinno zająć około 15 minut, w zależności od zasobów komputera.

Program *Make* z pomocą odpowiednich plików konfiguracyjnych jest w stanie zrobić wszystko co niezbędne, aby zbudować projekt, łącznie z utworzeniem projektów zależnych. W systemie Windows sprawa nie jest już taka prosta, gdyż nie ma możliwości domyślnie skorzystać z dobrodziejstw programu *Make*. Są w związku z tym dwie możliwości. Pierwsza z nich to zainstalowanie programu *Cygwin*, który umożliwia uruchomienie programu *Make* w systemie Windows. Druga to ręczne wykonanie skryptów *tcl* budujących najpierw projekty będące zależnościami projektu *adv7511*, a następnie tą samą metodą zbudowanie głównego projektu. W tym celu trzeba wykonać komendę:

```
vivado -mode batch -source <nazwa_projektu>_ip.tcl
```

z folderu, w którym znajduje się każdy z projektów zależnych, umieszczonych w folderze *libraries*. W naszym przypadku są to projekty: *axi_clkgen*, *axi_hdmi_tx*, *axi_i2s_adi*, *axi_spdif_tx*, *util_i2c_mixer*. Na koniec przechodzimy do folderu, w którym przechowywany jest główny projekt (*projects/adv7511/zed*) i budujemy go za pomocą komendy:

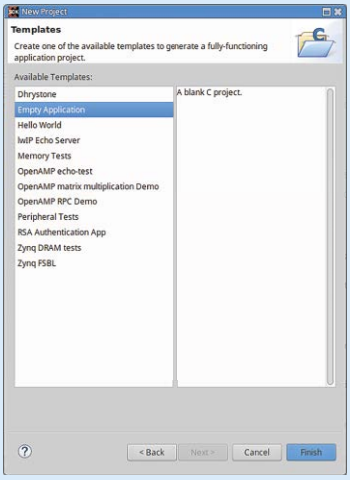
```
vivado -mode batch -source ./system_project.tcl
```

Szczegółowy opis budowania projektów znajduje się na stronie: ***https://goo.gl/7J1M8s***.

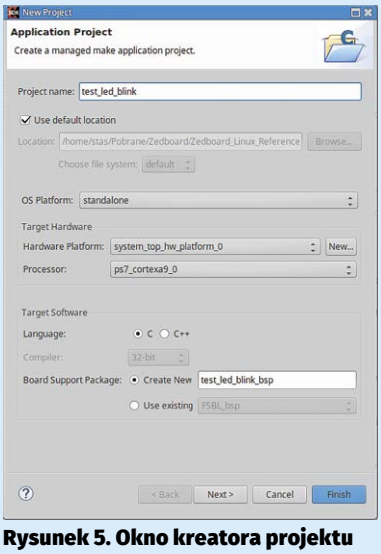
Po zbudowaniu projektu, rola Vivado w starszej wersji kończy się i możemy otworzyć Vivado w wersji najnowszej. W tym celu korzystamy z utworzonego skrótu albo (pod Linuksem) wykonujemy w konsoli komendę (dla wersji 2017.1, najnowszej w chwili pisania tego artykułu)

```
/opt/Xilinx/
Vivado/2017.1/bin/
vivado
```

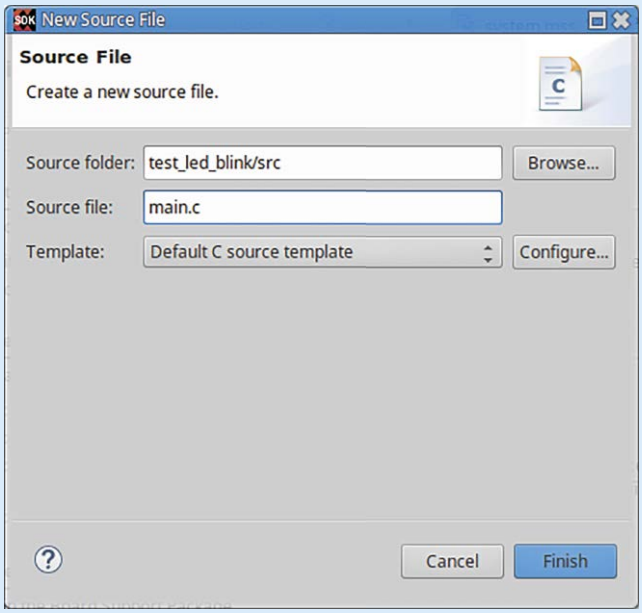
W oknie, które się pojawi klikamy, *Open Project* i wskazujemy gdzie znajduje się zbudowany przed chwilą w starszej wersji Vivado projekt (ścieżka jak wyżej, plik projektu nazywa się *adv7511_zed.xpr*). Po chwili pojawi się okno ostrzegające o otwarciu projektu ze starszej wersji Vivado. Wybieramy opcję *Automatically upgrade to current version*. Po chwili otworzy się projekt i pojawi się okno



Rysunek 6. Okno wyboru szablonu lub projektu



Rysunek 5. Okno kreatora projektu



Rysunek 7. Okno dialogowe

informujące o tym, że należy sprawdzić status bloków IP, gdyż są nowe wersje. Wybieramy opcję *Report IP status*. Na dole pojawi się lista bloków, które należy zaktualizować. Robimy to, klikając *Upgrade Selected* i potwierdzając to w oknie dialogowym przyciskiem OK. Po chwili może się pojawić okno informujące o krytycznych ostrzeżeniach. Spokojnie można je zignorować, klikając OK. Jednocześnie powinno się pojawić okno *Generate output products*, gdzie klikamy *Generate* i chwilę czekamy. Po chwili pojawi się okno potwierdzające wygenerowanie *Output products* (potwierdzamy przyciskiem *OK*), a następnie w głównym oknie zobaczymy *Block Design* naszego projektu referencyjnego. Można mu się przyjrzeć lub od razu ponownie zbudować projekt, wybierając w panelu z lewej strony przycisk *Generate Bitstream* i w oknie dialogowym potwierdzając to najpierw przyciskiem *yes* (jeśli pojawi się takie okno), a potem przyciskiem *OK*. Ten proces znów potrwa ok. 15 minut.

Po zbudowaniu projektu trzeba go wyeksportować do SDK. W tym celu należy otworzyć projekt w Vivado i z menu wybrać *File* → *Export* → *Export Hardware...* W okienku, które się pokaże, zaznaczamy się opcję *include bitstream* i klikamy OK.

Listing 1. Źródło przykładowego programu

```
#include <stdio.h>
#include „sleep.h”
#include „xgpiops.h”

int main()
{
    XGpioPs psGpioInstancePtr;
    XGpioPs_Config *GpioConfigPtr;
    int xStatus;
    int emioLed = 73; // LD0 is connected to EMIO pin 19 (GPIO 73 (54 + 19))
    int mioLed = 7; // Led LD9 is connected to MIO pin 7
    int emioButton = 54; // Button center connected to pin 0 EMIO (GPIO 54)

    u32 outputPinDirection = 0x1;
    u32 inputPinDirection = 0x0;
    //GPIO Intialization
    GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
    if (GpioConfigPtr == NULL)
        return XST_FAILURE;
    xStatus = XGpioPs_CfgInitialize(&psGpioInstancePtr, GpioConfigPtr,
        GpioConfigPtr->BaseAddr);
    if (xStatus != XST_SUCCESS)
        print(„ GPIO INIT FAILED \n\r”);
    //PS GPIO LED pin setting to Output
    XGpioPs_SetDirectionPin(&psGpioInstancePtr, mioLed, outputPinDirection);
    XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, mioLed, 1);
    //EMIO LED Setting to Output
    XGpioPs_SetDirectionPin(&psGpioInstancePtr, emioLed, outputPinDirection);
    XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, emioLed, 1);
    //EMIO Button set to input
    XGpioPs_SetDirectionPin(&psGpioInstancePtr, emioButton, inputPinDirection);
    //Main Loop
    while (1) {
        XGpioPs_WritePin(&psGpioInstancePtr, mioLed, 1);
        XGpioPs_WritePin(&psGpioInstancePtr, emioLed, 1);
        usleep(300000);
        XGpioPs_WritePin(&psGpioInstancePtr, mioLed, 0);
        XGpioPs_WritePin(&psGpioInstancePtr, emioLed, 0);
        usleep(300000);
        if (XGpioPs_ReadPin(&psGpioInstancePtr, emioButton) == 1)
        {
            break;
        }
    }
    return 0;
}
```

Uruchomienie przykładowego projektu

W celu pokazania, że nasz świeżo zbudowany projekt definiujący PL i konfigurujący PS działa, można uruchomić przykładowy, prosty program np. migające diody. Jedna z diod podłączona jest do części PS, a druga – do FPGA w części PL. W projekcie wykorzystany jest także przycisk służący do zakończenia programu, podłączony do PL.

Aby uruchomić SDK, należy otworzyć Vivado i z menu File wybrać opcję *Launch SDK*. Otworzy się okno dialogowe, w którym zostawiamy domyślne opcje i klikamy *OK*. Po chwili powinien się uruchomić program Eclipse, który, razem z zainstalowanymi wtyczkami od firmy Xilinx, stanowi XSDK.

Po uruchomieniu XSDK, tworzymy nowy projekt, klikając menu *File* → *New* → *Applicaton Project*. W oknie tworzenia projektu (**rysunek 5**) wpisujmy nazwę projektu (np. *test_led_blink*), nie zmieniając pozostałych domyślnych opcji i klikamy *Next*. W kolejnym oknie (**rysunek 6**) z listy szablonów projektów wybieramy *Empty application* i klikamy *Finish*. W panelu z lewej strony, na liście projektów szukamy nowo utworzonego projektu i rozwijamy go. W folderze *src* tworzymy nowy plik źródłowy (klikamy lewym przyciskiem myszy i z menu *New* wybieramy *Source file*). W oknie dialogowym (**rysunek 7**) wpisujemy nazwę pliku, np. *main.c* i klikamy *Finish*. Usuwamy całą treść pliku i wpisujemy tam kod programu taki jak na **listingu 1**.

Przed uruchomieniem programu trzeba zaprogramować FPGA. W tym celu podłączamy płytkę kablem USB do komputera (wybieramy gniazdo PROG na płytce), włączamy ją (przełącznikiem znajdującym się w lewym górnym rogu płytki) i wybieramy menu *Xilinx Tools* → *Program FPGA*. W oknie dialogowym zostawiamy opcje domyślne i klikamy przycisk *Program*. Po kilku sekundach na płytce powinna się zapalić niebieska dioda sygnalizująca zakończenie programowania.

Aby uruchomić program, na liście projektów trzeba zaznaczyć nasz projekt, kliknąć jego nazwę prawym przyciskiem myszy i z menu wybrać *Run as* → *Lanuch on hardware (System Debugger)*. Jeśli wszystko zostało wykonane prawidłowo, diody powinny zacząć migać. Naciśnięcie środkowego przycisku (*BTNC*) na ok. 0,5 s (przycisk musi być wciśnięty w momencie, gdy diody zgasną) powinno zakończyć program.

Stanisław Aleksiński

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 47858, pass: 9seghusa