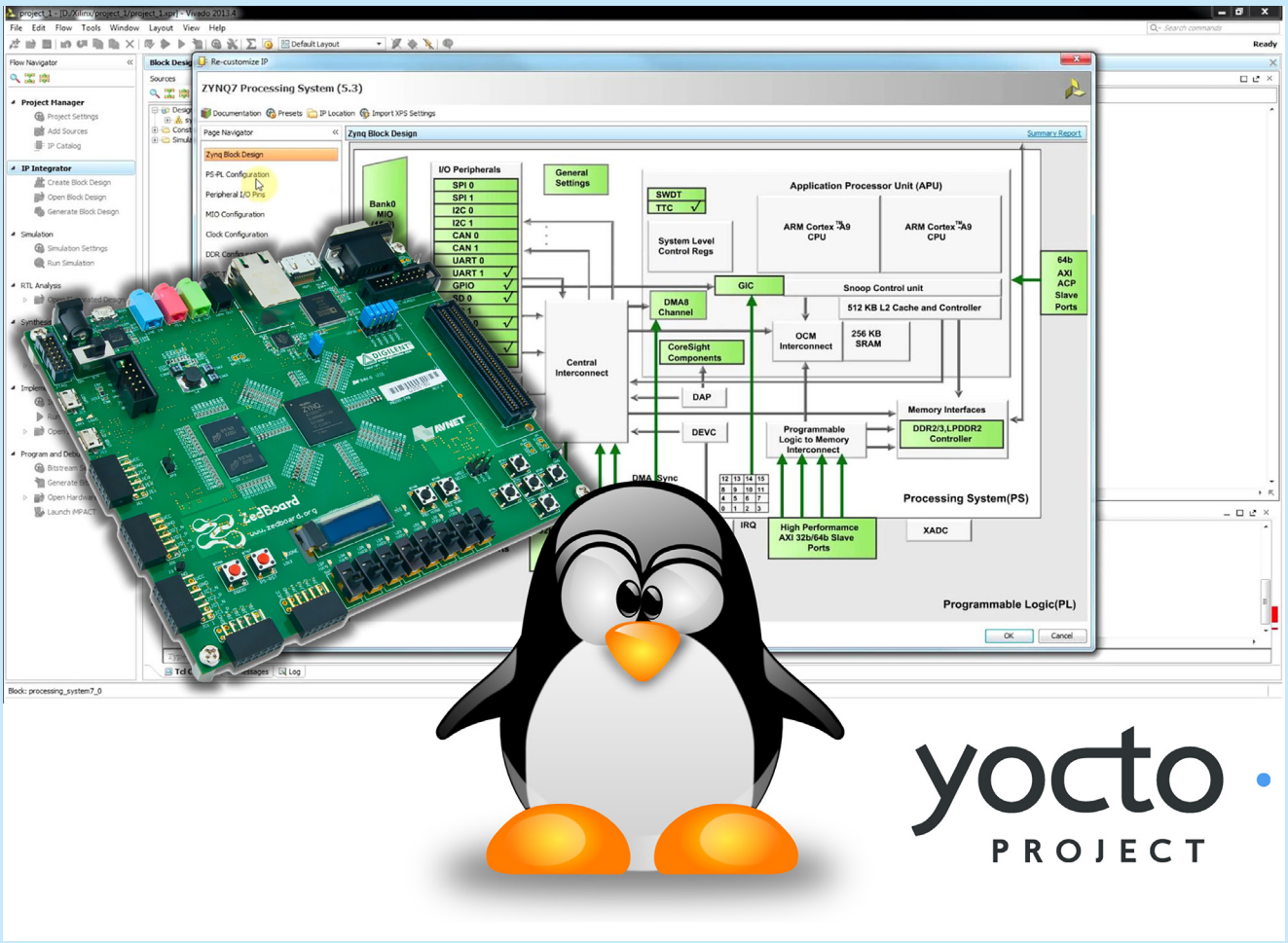


Jak używać układów SoC Xilinx Zynq-7000 z Linuksem – proste przykłady (2)

Choć oczywiście możliwe jest napisanie wielowątkowego, zaawansowanego programu korzystającego ze stosu IP bezpośrednio na procesorze, to jest to dość czasochłonne i najczęściej nieopłacalne. Dużo łatwiej jest skorzystać z dobrodziejstw systemu operacyjnego np. z Linuksa. W tym artykule zostanie opisany proces uruchamiania systemu Linux na płytce ZEDBOARD. Opisany zostanie proces uruchamiania systemu zarówno z karty SD, jak i poprzez sieć.

Płytka Zedboard jest wyposażona w układ Zynq firmy Xilinx. Jedną z jego ogromnych zalet jest wbudowany, dedykowany procesor ARM. Dzięki takiemu rozwiązaniu uruchomienie Linuksa na tym układzie jest stosunkowo proste. Proces przygotowywania potrzebnych plików jest bardzo podobny do typowego procesu uruchamiania Linuksa na procesorze o architekturze ARM.

Linuksa na Zynq można uruchomić z karty SD, z pamięci Flash, bądź przez JTAG. Opisana zostanie pierwsza i trzecia możliwość, gdyż są one najbardziej przydatne na początek. Karta SD daje możliwość łatwego uruchomienia wygenerowanego systemu. JTAG pozwala zaś na zdalne uruchomienie Linuksa z poziomu konsoli komputera, do którego podłączona jest płytka, bez konieczności kopiowania plików na kartę SD, bądź do pamięci Flash. Jeśli dodatkowo skonfiguruje się sieciowy system plików (NFS), edycja systemu plików używanego do uruchomienia Linuksa staje się jeszcze prostsza.

W momencie pisania tego artykułu najnowsza wersja Vivado to 2017.1 i ta wersja wpisana jest we wszystkich poleceniach, gdzie jest to konieczne. Jeśli czytelnik posiada inną wersję, polecenia należy odpowiednio zmodyfikować.

Warto też wspomnieć, że nazwa Linux odnosi się do samego jądra systemu operacyjnego, a na system operacyjny składa się jądro, wraz z systemem plików, gdzie umieszczony jest program Init inne, niezbędne programy i skrypty. W tym i kolejnych artykułach będę jednak stosował skrót myślowy polegający na nazywaniu Linuksem całości systemu.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86735**, pass: **6mqh264k**

Teoria

Do uruchomienia systemu Linux na Zynq z karty SD lub z pamięci Flash potrzebne są cztery pliki:

1. **BOOT.BIN** plik, który składa się z przynajmniej dwóch plików zwanych bootloaderami – FSBL (*first stage bootloader*) i U-Boot – kompleksowe narzędzie do uruchamiania systemu Linux. Opcjonalnie można dołączyć również trzeci plik bitstream z konfiguracją logiki programowalnej FPGA (dalej zwaną PL – *programmable logic*).
2. **uImage** plik z jądrem systemu Linux.
3. **Device Tree Blob** plik binarny z informacjami o systemie komputerowym, na którym ma być uruchomiony Linux.
4. Obraz systemu plików zawierający potrzebne skrypty startowe, programy i inne pliki potrzebne do uruchomienia systemu. Opcjonalnie można utworzyć na karcie pamięci dodatkową partycję i rozpakować tam plik z obrazem systemu plików.

Dodatkowo przydatny może być plik *uEnv.txt*. Zawarte są w nim instrukcje dla programu U-boot, gdzie ma szukać potrzebnych plików i co z nimi zrobić. W przypadku braku tego pliku U-boot wykona domyśle instrukcje, wkompilowane w kod. W wypadku uruchamiania przez JTAG, rolę programu FSBL przejmuje skrypt uruchamiany na komputerze. Jego rolą jest wykonać w konsoli Xilinx System Debugger (XSDB) odpowiednie polecenia:

1. Połączenie z płytką.
2. Reset układu.
3. Załadowanie pliku bitstream do FPGA.
4. Konfiguracja PS.
5. Uruchomienie przekazywania sygnałów pomiędzy PL a PS.
6. Wysłanie do pamięci programu U-boot.
7. Przekazanie kontroli do procesora (uruchomienie programu U-boot).

Do wykonania punktów 3 i 4 jest niezbędny skrypt tcl o nazwie *ps7_init.tcl*. Można go znaleźć w folderze *<ścieżka do projektu Vivado>/adv7511_zed.sdk/system_top_hw_platform_0*, po uruchomieniu SDK, z wyeksportowanym projektem z Vivado.

Proces ładowania Linuksa na Zynq rozpoczyna się od uruchomienia programu zapisanego w pamięci ROM. Program ten jest bardzo mały i robi tylko dwie rzeczy. Na początek sprawdza, jaki jest poziom logiczny specjalnych pinów służących do określenia sposobu ładowania systemu. Jeśli system ma być załadowany z karty SD lub z pamięci Flash, szuka na odpowiednim nośniku pliku BOOT.BIN, ładuje FSBL, będący jego częścią i przekazuje mu kontrolę. Rolą programu FSBL jest odpowiednie skonfigurowanie procesora i załadowanie pliku bitstream do PL. Następnie kontrola jest przekazywana do programu U-Boot, który ładuje do pamięci jądro Linuksa i uruchamia system.

Jeśli piny określające sposób ładowania systemu ustawione są w położeniu, które określa ładowanie systemu poprzez JTAG, to procesor po prostu czeka na odpowiednie instrukcje z zewnątrz. Podobnie zachowa się, jeśli piny wskażą bootowanie z karty SD, a karta nie będzie włożona do czytnika. Proces uruchamiania systemu przez JTAG opiszę w dalszej części artykułu.

Jeśli chcemy w najprostszy sposób uruchomić system z karty SD, przygotowanie jej sprowadza się do skopiowania czterech wspomnianych plików na partycję sformatowaną w systemie plików FAT32. Jak już wspomniałem, opcjonalnie można utworzyć na karcie SD druga partycję, która zostanie zamontowana jako główny system plików i rozpakować tam plik z obrazem systemu plików. Gotowe pliki są dostępne w Internecie, na jednej ze stron Wiki Xilinx (***https://goo.gl/tNjFr8***). W dalszej części artykułu opiszę ten proces bardziej szczegółowo.

Przygotowywanie własnej wersji plików

Cały proces uruchamiania Linuksa, wraz z instalacją potrzebnych narzędzi, opisany jest na stronie wiki Xilinx (***https://goo.gl/JCXJ6K***). Aby jednak móc skorzystać ze wszystkich peryferiów płytki, warto pobrać projekt dla części PL przygotowany przez firmę Analog Devices (ADV), której kilka układów scalonych jest częścią płytki (ten proces opisałem dokładnie w poprzednim artykule – EP9/2017). Wtedy warto też zbudować jądro Linuksa ze źródeł z repozytorium ADV. Składają się na nie komplet źródeł jądra Linuksa skopiowany z repozytorium Xilinx, wraz z dodatkowymi sterownikami i plikami konfiguracyjnymi dedykowanymi również dla płytki Zedboard. W dalszej części artykułu opiszę w jaki sposób pobrać i skompilować te źródła. Proces generowania pliku bitstream ze źródeł od ADV opisałem w poprzednim artykule.

Do wygenerowania systemu plików używam systemu Yocto. Pozwala on na wygenerowanie pełnego obrazu Linuksa, tj. wszystkich wyżej wymienionych plików (poza bitstream). Jest to bardzo zaawansowane narzędzie z niezbyt korzystną krzywą uczenia, więc opiszę dokładnie krok po kroku jak go użyć, bez wgłębiania się w jego działanie (samo Yocto jest tematem na całą serię artykułów). Pewną wadą może być fakt, że przynajmniej według dokumentacji trzeba sobie na niego zarezerwować ok. 50 GB przestrzeni dyskowej. W moim przypadku było to niecałe 40 GB. Lżejszą alternatywą jest system Buildroot, który jest znacznie prostszy i pewnie wystarczający do naszych celów, ale tu opiszę ten proces dla Yocto, którego sam używam. Dodatkowym atutem Yocto jest fakt, że Xilinx opublikował własną tzw. warstwę (layer) o nazwie *meta-xilinx*. Jest to zbiór plików konfiguracyjnych i skryptów znacznie ułatwiających generowanie plików dla Zedboard przez Yocto.

Choć system ten pozwala na wygenerowanie zarówno systemu plików, jak i wszystkich innych potrzebnych plików, pozostałe pliki wygenerujemy ręcznie, aby uprościć proces konfigurowania Yocto, w celach edukacyjnych, aby pokazać jak wygląda ręczna kompilacja jądra Linuksa i programu U-boot i aby widzieć co się dzieje na każdym kroku przy generowaniu każdego z plików.

Na początek upewnimy się, że nasze środowisko jest odpowiednio przygotowane. Użytkownikom systemu Windows radzę zainstalować jakąś dystrybucję Linuksa na maszynie wirtualnej. Tego procesu nie będę opisywał, gdyż wykracza on poza temat artykułu. Sam używam i polecam Ubuntu, ale wiem, że bardzo dobrze nadaje się do tego też np. CentOS. Osobom niezaznajomionym z konsolą Linuksa polecam zapoznać się z poradnikami w tym temacie. W dalszej częśści artykułu zakładam, że czytelnik zna podstawowe komendy konsoli, np. *cd*, *mv*, *rm* itp.

Zależności systemu Yocto w niemal pełni pokrywają te dla jądra Linuksa i U-boot'a. Zacznijmy więc od nich. Dla Ubuntu wystarczy wykonać w terminalu komendy *sudo apt install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libsdl1.2-dev xterm*. Komendy instalacji zależności Yocto dla innych dystrybucji można znaleźć na stronie Yocto Project Quick start (***https://goo.gl/CeFiYt***). Dodatkowo, do kompilacji i konfiguracji Linuksa i programu U-Boot potrzebne są trzy pakiety **libncurses5-dev**, **libssl-dev** i **device-tree-compiler**. Przydatny będzie również program Git i narzędzia u-boot, a dokładniej program mkimage. Choć za chwilę będziemy go kompilować razem z programem u-boot, to wygodniej jest mieć go oddzielnie. Polecenie dla Ubuntu wygląda następująco *sudo apt install libncurses5-dev libssl-dev device-tree-compiler git u-boot-tools*. Przed każdym z czekających nas zadań musimy ustawić odpowiednie zmienne środowiskowe, aby program *make* „wiedział", na jaką architekturę kompilujemy dany projekt i jakiego ma użyć kompilatora. W tym celu wpisujemy w konsoli następujące polecenia:

```
source /opt/Xilinx/Vivado/2017.1/settings64.sh
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

Te komendy dotyczą jednej, konkretnej konsoli, w której akurat pracujemy, więc ważne jest, aby po otwarciu nowej konsoli znów je wpisać. Po przygotowaniu środowiska, pobieramy Yocto.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86735**, pass: **6mqh264k**

Polecam utworzyć specjalny katalog i pobierać do niego wszystkie potrzebne pliki. W konsoli z ustawionymi zmiennymi środowiskowymi, z katalogu przeznaczonego dla Yocto, wpisujemy:

git clone -b morty git://git.yoctoproject.org/poky.git
git clone -b morty git://git.yoctoproject.org/meta-xilinx
cd poky
source oe-init-build-env zedboard.

Ostatnia komenda powoduje utworzenie nowego projektu o nazwie zedboard i ustawienie zmiennych środowiskowych dla Yocto.

Po pobraniu plików i skonfigurowaniu środowiska (ostatnia komenda) szukamy w folderze *zedboard/conf/* pliku *bblayers.conf*. Edytujemy zmienną *BBLAYERS*, dodając ścieżkę do folderu zawierającego warstwę *meta-xilinx*. Po edycji zmienna powinna wyglądać mniej więcej tak:

*BBLAYERS ?= „ *
*<ściezka do katalogu zawierającego poky>/poky/meta *
<ściezka do katalogu zawierającego poky>/poky/meta-poky \
*<ściezka do katalogu zawierającego poky>/poky/meta-yocto-bsp *
*<ściezka do katalogu zawierającego meta-xilinx>/meta-xilinx *
„

Zapisujemy plik *bblayers.conf*, zamykamy go i otwieramy właściwy plik konfiguracyjny *local.conf*. Tu znajdą się informacje o tym, jak ma wyglądać proces kompilacji, jaka jest architektura procesora (a dokładniej, dzięki warstwie *meta-xilinx*, na jaką płytkę przygotowujemy obraz), a także, jeśli byśmy chcieli, jakie dodatkowe pakiety mają być zainstalowane w systemie plików.

Otwieramy plik *local.conf* i edytujemy w następujący sposób:

- Na górze, po pierwszym komentarzu, dopisujemy linijki, dzięki którym kompilacja przeprowadzana będzie wielowątkowo

BB_NUMBER_THREADS ?= ${@oe.utils.cpu_count()}
PARALLEL_MAKE ?= „-j ${@oe.utils.cpu_count()}”

- Poniżej liniki: #MACHINE ?= „edgerouter” dodajemy konfigurację architektury

MACHINE ?= „zedboard-zynq7”

- Do zmiennej EXTRA_IMAGE_FEATURES dodajemy eclipse-debug
- EXTRA_IMAGE_FEATURES = „debug-tweaks eclipse-debug”
- Poniżej zmiennej EXTRA_IMAGE_FEATURES dopisujemy zmienną IMAGE_INSTALL_append

IMAGE_INSTALL_append = „libgcc libstdc++ „

Zapisujemy plik i w konsoli przechodzimy do folderu *zedboard*. Z tego folderu wpisujemy w konsoli komendę rozpoczynającą proces budowania obrazu systemu *bitbake core-image-minimal*. Na wykonanie tego polecenia trzeba przeznaczyć dużo czasu. W moim wypadku, na komputerze z dwurdzeniowym procesorem i 4 GB pamięci RAM trwało to około 4 godzin. Trzeba pobrać dużą liczbę plików, więc czas kompilacji będzie również zależał od szybkości połączenia z Internetem.

Po zakończeniu tego procesu, gotowe pliki można znaleźć w folderze *zedboard/tmp/deploy/images/zedboard-zynq7*. Będą nas interesowały trzy pliki:

- *core-image-minimal-zedboard-zynq7-<znacznik czasowy wygenerowania>.rootfs.cpio.gz.u-boot*,
- *core-image-minimal-zedboard-zynq7-<znacznik czasowy wygenerowania>.rootfs.tar.gz*,
- *uEnv.txt* – plik zawierający konfigurację ładowania systemu z karty SD. Więcej o nim napisałem dalej.

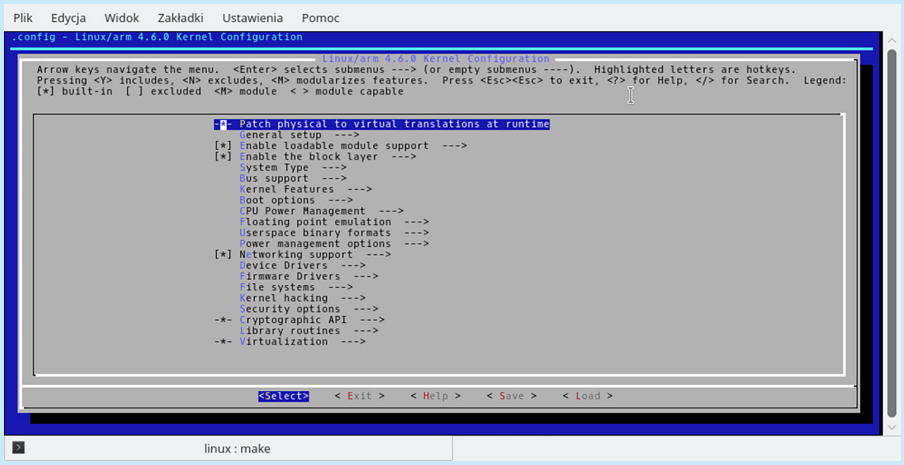
Pierwszy zawiera obraz *initramfs* spakowany jako archiwum cpio i skompresowany przez *gz* z dodanym przez program *mkimage* nagłówkiem dla programu *U-boot*. Jest to obraz gotowy do załadowania przez U-boot. Drugi to archiwum tar zawierające system plików, który będzie można rozpakować na karcie SD.

Więcej informacji na temat Yocto można znaleźć na stronach:

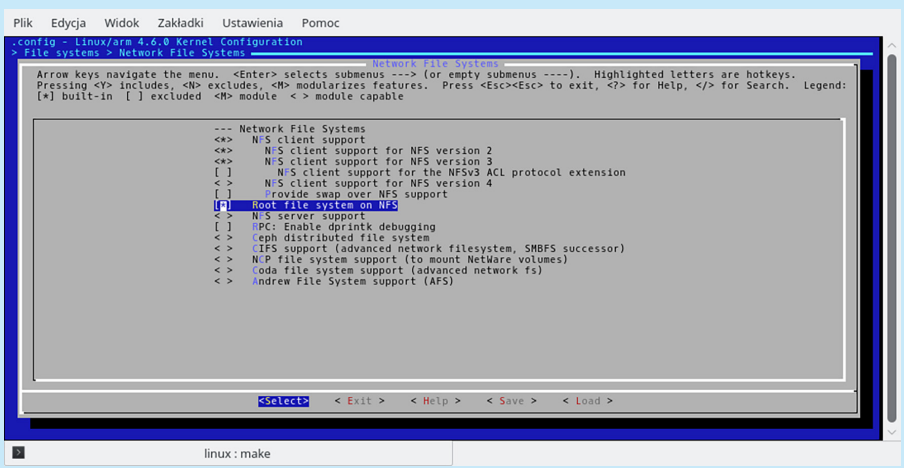
- Yocto Project Quick Start ***https://goo.gl/5wyRjJ*** (instalowanie Yocto).
- Xilinx wiki page Yocto ***https://goo.gl/p2qFfH*** (informacje na temat warstwy *meta-xilinx*).
- Yocto Project Reference Manual ***https://goo.gl/hC7AjE*** (dokumentacja).
- Yocto Project Development Manual ***https://goo.gl/2dNDD4*** (instrukcja zorientowana na konkretne zadania).

U-boot

Kolejny krok to kompilacja programu U-boot. Polecam utworzyć np. w głównym katalogu użytkownika folder o nazwie np. *linux*, do którego będzie można później pobrać źródła Linuksa i U-boot i trzymać tam w dodatkowych podfolderach różne wersje plików *dts* i skompilowanych jąder Linuksa.

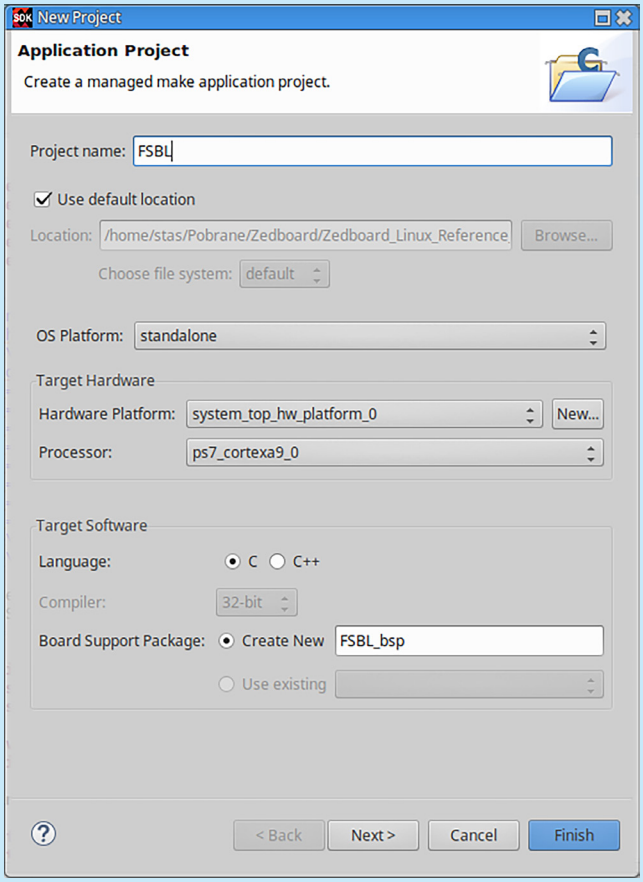


Rysunek 1. Okno konfiguracji kernela systemu Linux

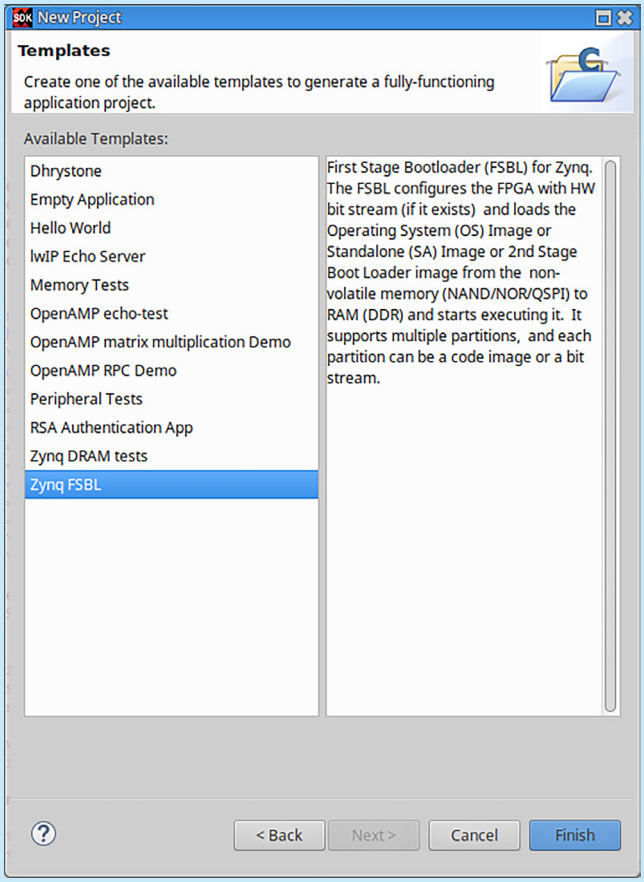


Rysunek 2. Okno konfiguracji kernela – konfigurowanie NFS

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86735**, pass: **6mqh264t**



Rysunek 3. Opcje projektu



Rysunek 4. Ekran wyboru szablonu projektu

W nowej konsoli przechodzimy poleceniem cd do folderu, do którego chcemy pobrać źródła i wykonujemy polecenia:

source /opt/Xilinx/Vivado/2017.1/settings64.sh
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
git clone https://github.com/Xilinx/u-boot-xlnx.git (pobieranie źródeł u-boot)
cd u-boot-xlnx/ (wejście do katalogu u-boot-xilinx)
make zynq_zed_config (konfigurowanie kompilacji)
make (kompilacja)

Wynikowy plik o nazwie u-boot znajduje się w folderze *u-boot-xlnx*. Szczegółowy opis budowania programu U-boot można znaleźć na stronie Wiki Xilinx ***https://goo.gl/kAFA8R***.

Po skompilowaniu programu U-boot możemy przystąpić do kompilacji jądra Linuksa. W standardowej konfiguracji jądra nie jest uwzględnione wsparcie dla NFS, więc trzeba będzie je dodać, poprzez narzędzie *menuconfig*. Uruchamiamy nową konsolę i przechodzimy do wcześniej utworzonego katalogu *linux* lub do innego miejsca, gdzie chcemy trzymać źródła jądra Linuksa i wpisujemy komendy:

```
git clone https://github.com/analogdevicesinc/linux.git
cd linux/
git checkout xcomm_zynq
source /opt/Xilinx/Vivado/2017.1/settings64.sh
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
make zynq_xcomm_adv7511_defconfig
make menuconfig
```

Powinno zostać wyświetlone okno, jak na **rysunku 1**. Używając klawiszy strzałek przechodzimy kursorem do menu *File systems*, wciskamy *enter*, a następnie do *Network File Systems*. Po ustawieniu w tym miejscu kursora wciskamy spację, aby w nawiasach kwadratowych obok napisu *Network File Systems* pojawiła się gwiazdka. Teraz możemy nacisnąć Enter i wejść do tego menu. Tam ustawiamy kursor na opcji *NFS Client Support* i dwukrotnie wciskamy spację. Po jednokrotnym kliknięciu spacji w nawiasie kwadratowym pojawi się literka M, oznaczająca, że dana funkcja zostanie skompilowana jako moduł jądra. My chcemy wkompilować obsługę NFS w samo jądro, więc musimy nacisnąć *enter* jeszcze raz, aby pojawiła się właśnie gwiazdka, oznaczająca wkompilowanie funkcji w jądro. Teraz przechodzimy do opcji *Root file system on NFS* i również klikamy spację dwa razy. Ekran na tym etapie powinien wyglądać jak na **rysunku 2**. Zapisujemy ustawienia, wciskając trzy razy strzałkę w prawo, co ustawi znajdujący się na dole kursor na opcji *Save*. Wciskamy trzykrotnie *enter*, aby zapisać ustawienia i to potwierdzić. Na koniec ustawiamy kursor w polu *Exit* i wciskamy *enter*, aby zamknąć program.

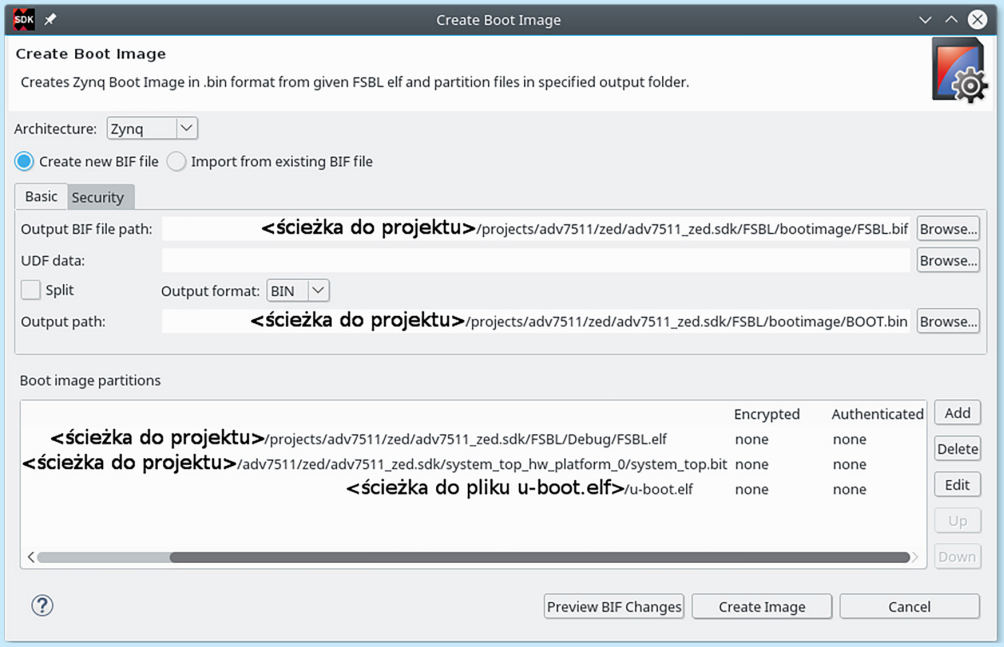
Teraz możemy skompilować jądro Linuksa, wpisując komendę *make uImage LOADADDR=0x00008000*. To powinno potrwać ok. 15…20 minut. Plik ze skompilowanym jądrem Linuksa, uImage, znajduje się w folderze *arch/arm/boot*. Dokładny opis budowania Linuksa od ADI można znaleźć na stronie ***https://goo.gl/dLbLzm***.

Device Tree Blob

Do wygenerowania pliku *Device Tree Blob* (TDB) potrzebne są pliki *dts* (*Device Tree Source*) zawierające informacje o systemie komputerowym. Można je wygenerować za pomocą XSDK, na podstawie pliku *hdf* wygenerowanego przez Vivado. Można też przygotować te pliki samemu lub skorzystać z dostarczonych przez autora projektu Vivado.

Jako, że korzystamy ze źródeł od ADV przystosowanych specjalnie do Zedboard, możemy znaleźć odpowiednie pliki w naszym drzewie źródeł Linuksa. Sama kompilacja plików *dts* na *dtb* wykonywana jest przez program *dtc* (*Device Tree Compiler*), który trzeba skompilować samemu lub pobrać z repozytorium Ubuntu. Ponieważ pobraliśmy już *dtc* (jest na liście zależności dla Linuksa, patrz wyżej), możemy przystąpić do dzieła.

Proponuję utworzyć osobny folder np. o nazwie *dts*, w którym, w osobnych podfolderach, przechowywane będą w przyszłości odpowiednie wersje plików *dts*. W folderze *dts* tworzymy więc folder *clean*, do którego skopiujemy oryginalną wersję plików *dts*. Potrzebne pliki znajdują się w folderze *linux/arch/arm/boot/dts*, jednak nie warto kopiować całego folderu, gdyż potrzebujemy tylko kilku z nich. Wspomniana wcześniej instrukcja przygotowywania Linuksa od ADI mówi, że potrzebny będzie plik *zynq-zed-adv7511.dts*. Aby móc przechowywać osobno inne jego wersje, będą potrzebne również jego zależności. Skopiujmy więc następujące pliki: *skeleton.dtsi*, *zynq-7000.dtsi*, *zynq.dtsi*, *zynq-zed-adv7511.dts*, *zynq-zed-adv7511.dtsi* i *zynq-zed.dtsi*. Tę listę można utworzyć otwierając najpierw plik *zynq-zed-adv7511.dts* i szukając w nim jakie pliki są do niego włączane instrukcją *Include*. To samo robimy z kolejnymi plikami, aż najdziemy wszystkie zależności.



Rysunek 5. Opcje tworzenia obrazu systemu

Gdy dysponujemy już programem *dtc* i gotowymi źródłami *dts*, wystarczy z folderu *clean* wydać komendę *dtc -I dts -O dtb -o <nazwa_pliku_dtb> <nazwa_pliku_dts>*. W naszym wypadku *dtc -I dts -O dtb -o devicetree.dtb zynq-zed-adv7511.dts*. Znaczenie parametrów: **-I** format pliku wejściowego, **-O** format pliku wynikowego, **-o** nazwa pliku wyjściowego. Po wykonaniu tej komendy wynikowy plik *devicetree.dtb* będzie znajdował się w folderze *clean*.

Opis sposobu generowania pliku *dts* na podstawie wygenerowanego przez Vivado pliku *hdf* oraz kompilacji do pliku *dtb* znajduje się na stronie ***https://goo.gl/pqsoiu***.

BOOT.BIN

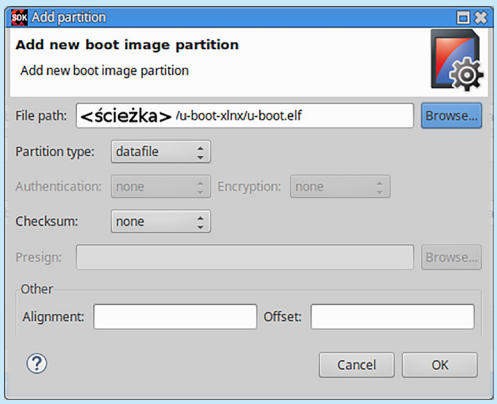
Do wygenerowania pliku BOOT.BIN potrzebne są trzy pliki: *fsbl*, *bitstream* i *u-boot*. Pierwszy z nich można łatwo wygenerować za pomocą XSDK. Po wyeksportowaniu zbudowanego w Vivado projektu (opis w poprzednim artykule) tworzymy w XSDK nowy projekt (menu *File->New->Application Project*). Wybieramy nazwę, np. FSBL i opcje jak na **rysunku 3**. Po kliknięciu *next* powinien się pojawić ekran z wyborem szablonu projektu (**rysunek 4**). Wybieramy szablon *Zynq FSBL*. Po kliknięciu *Finish*, czekamy aż XSDK zbuduje projekt i mamy gotowy FSBL.

Plik *bitstream* był już wcześniej stworzony przez Vivado i powinien być w XSDK w projekcie *system_top_hw_platform_0* (projekt ten jest tworzony po wyeksportowaniu projektu Vivado, opis również w poprzednim artykule). Plik u-boot został zbudowany w jednym z poprzednich kroków. Należy tylko zmienić mu nazwę, dodając rozszerzenie *.elf*, gdyż wymaga tego narzędzie generujące plik *BOOT.bin*. Jesteśmy zatem gotowi do wygenerowania pliku *BOOT.bin*. W tym celu należy zaznaczyć kliknięciem utworzony projekt FSBL, a następnie z menu Xilinx Tools wybrać *Create Boot Image*. W oknie, które zostanie wyświetlone (**rysunek 5**), powinny być już wypełnione niektóre pola. Do listy plików obrazu automatycznie dodaje się zarówno plik FSBL, jak i *bitstream*, ale nie ma w nim plik *u-boot.elf*, gdyż XSDK nie zna jego lokalizacji. Dodajemy go klikając *Add* i wpisując ścieżkę do pliku *u-boot.elf*, jak na **rysunku 6**. Klikamy OK, a następnie *Create Image*. Po kliku chwilach powinniśmy mieć gotowy obraz. Znajdziemy go w folderze *projects/adv7511/zed/adv7511_zed.sdk/FSBL/bootimage/*.

Uruchamianie Linuksa z karty SD

Choć uruchomienie systemu z karty SD jest prostym procesem, można go wykonać na kilka sposobów. Ja przedstawię dwie drogi, różniące się sposobem kopiowania systemu plików na kartę. W pierwszym przypadku *rootfs* będzie skopiowany w postaci pojedynczego pliku *uramdisk.image.gz*, w drugim zaś, na karcie SD zostanie utworzona odrębna partycja, gdzie *rootfs* zostanie rozpakowany. Zaletą pierwszego sposobu jest jego prostota, zaś drugiego – możliwość łatwej edycji plików. W pierwszym ze sposobów system plików ładowany jest do pamięci i po ponownym uruchomieniu systemu wszelkie zmiany nie są w nim zapisywane, w drugim zaś jest odwrotnie.

Najpierw trzeba sformatować kartę. Pierwszy sposób zakłada, że na karcie jest jedna partycja sformatowana jako *fat32*. Można nadać jej etykietę BOOT. Polecam użyć do tego programu *Gparted*. Na **rysunku 7** pokazano zrzut ekranu z oknem opcji formatowania karty. W tym przykładzie karta ma 2 GB pojemności, w wypadku karty o większej pojemności należy zwiększyć rozmiar partycji. Po sformatowaniu karty kopiujemy niezbędne pliki: *BOOT.bin* (z projektu FSBL), *uImage* (jądro Linuksa, znajduje się w folderze *linux/arch/arm/boot*), *devicetree.dtb* (folder *dts/clean*) i plik *core-image-minimal-zedboard-zynq7-<znacznik czasowy wygenerowania>.rootfs.cpio.gz.u-boot* (folder *yocto/poky/zedboard/tmp/deploy/*



Rysunek 6. Dodanie pliku XSDK do obrazu

images/zedboard-zynq7). Nazwę ostatniego z plików należy zmienić na *uramdisk.image.gz*. Po skopiowaniu plików odmontowujemy kartę i podłączamy do płytki. Przed włączeniem płytki należy podłączyć ją do komputera przez USB. Na płytce podłączamy kabel do złącza USB podpisanego jako UART.

Komunikacja z płytką odbywa się przez UART, więc jest potrzebny program do komunikacji za pomocą interfejsu RS232. Sam używam i polecam program *minicom*. Jeśli nie jest zainstalowany, pod Ubuntu instalujemy go za pomocą polecenia *sudo apt install minicom*.

W celu skonfigurowania połączenia najpierw przyłączamy płytkę i włączamy ją bez włożonej karty SD. Po podłączeniu płytki powinna być automatycznie dodana do katalogu */dev jako /dev/ttyACM0*.

Pod Ubuntu może wystąpić problem polegający na tym, że system uznaje płytkę za modem i próbuje się z nią komunikować przez UART, co oczywiście powoduje problemy, gdyż sterownik ten próbuj wysyłać przez UART niepotrzebne dane. Ponadto, domyślnie możemy nie mieć możliwości połączyć się z płytką bez uprawnień administratora. Aby rozwiązać ten problem, trzeba dodać swoje konto użytkownika do grupy *dialout* i dodać odpowiednie reguły do programu *Udev*, określającego między innymi, jakie procesy mają dostęp do jakich urządzeń. W tym celu wykonujemy w konsoli polecenia:

sudo usermod -a -G dialout <nazwa użytkownika> (dodanie użytkownika do grupy dialout)

sudo cp /lib/udev/rules.d/50-udev-default.rules /etc/udev/rules.d/ (kopiowanie pliku 50-udev-default.rules z domyślnej lokalizacji, w celu edycji)

sudo nano /etc/udev/rules.d/50-udev-default.rules (otwieramy plik do edycji)

Pod sekcją z regułami dotyczącymi tty dodajemy linijki:

 # relax the permissions just for ttyUSB0

KERNEL==”ttyUSB0”, MODE=”0666”

 # relax the permissions just for ttyACM0 used on ZedBoard

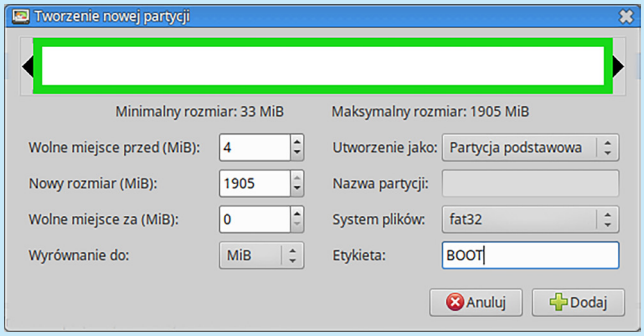
KERNEL==”ttyACM0”, MODE=”0666”

prevent modem-manager from touching USB-UART device that is a target board

SUBSYSTEM==”usb”, ATTR{idVendor}==”04b4”, ATTR{idProduct}==”0008”, MODE=”0666”, GROUP=”dialout”, ENV{ID_MM_DEVICE_IGNORE}=”1”

Wciskamy *ctrl-o* i *enter* aby zapisać i *ctrl-x* zamknąć program. Po przelogowaniu lub restarcie systemu wszystko powinno działać prawidłowo.

Przed uruchomieniem systemu trzeba jeszcze ustawić prawidłowo zworki określające sposób bootowania systemu. Jest to pięć



Rysunek 7. Zrzut ekranu z opcjami formatowania karty SD

zworek znajdujących się w prawym górnym rogu płytki ZEDBOARD. Powinny się one znaleźć w położeniu 01100. Polecam zapoznać się ze znaczeniem ich położenia, które opisane jest w dokumencie *ZedBoard_HW_UG_v2_2.pdf*.

Uruchamiamy program *minicom* poleceniem *minicom -D /dev/ttyACM0*. Zadziała ono przy założeniu, że UART będzie urządzeniem */dev/ttyACM0*. Jeśli nie zadziała, to za pomocą polecenia *dmesg* należy znaleźć odpowiednią nazwę i w razie potrzeby dodać ją do reguł *Udev*.

Teraz przy nadal włączonym programie minicom wyłączamy zasilanie ZEDBOARD, wkładamy kartę SD i włączamy zasilanie zestawu. Po krótkiej chwili powinna się zapalić niebieska dioda sygnalizująca załadowanie pliku bistream do FPGA, a następnie powinniśmy zobaczyć w konsoli proces uruchamiania systemu. Na koniec logujemy się jako *root* (bez hasła).

Drugi ze wspomnianych sposobów uruchamiania Linuksa z karty SD również zaczniemy od odpowiedniego sformatowania karty. Tym razem utworzymy dwie partycje. Pierwsza będzie taka sama, tylko mniejsza – musi się na niej zmieścić tylko jądro Linuksa, plik *BOOT.bin*, *devicetree.dtb* i plik *uEnv.txt*, więc powinno wystarczyć ok. 33…40 MB. Resztę karty zajmie druga partycja, sformatowana w systemie plików *ext4*. Można jej nadać etykietę *rootfs*.

Musimy również przekazać jakoś programowi U-boot miejsce, w którym jest system plików. Zrobimy to kopiując na kartę plik *uEnv.txt* wygenerowany przez Yocto, w którym zostaną umieszczone komendy koniecznego uruchomienia systemu i zmodyfikowane zostanie polecenie uruchomienia Linuksa. Uwaga! Sam plik *uEnv.txt* znajdujący się w folderze *yocto/poky/zedboard/tmp/deploy/images/zedboard-zynq7*, to tylko dowiązanie symboliczne. Potrzebujemy pliku docelowego (jego nazwa zaczyna się od *uEnv-zedboard-zynq7* i jest najdłuższa spośród tak się zaczynających). Oczywiście po skopiowaniu pliku na kartę SD zmieniamy jego nazwę na *uEnv.txt*. Teraz trzeba tylko zmienić w nim jedną rzecz

– wpisana jest w nim inna nazwa pliku *.dtb* (*uImage-zynq-zed.dtb*), należy więc zmienić nazwę znajdującego się na karcie SD pliku *devicetree.dtb* na *uImage-zynq-zed.dtb*, lub w pliku *uEnv.txt* zmienić *uImage-zynq-zed.dtb* na *devicetree.dtb*.

Obraz systemu plików rozpakowujemy komendą *sudo tar -C <ścieżka do punktu montowania partycji na karcie> -xzpf <ścieżka do obrazu systemu plików>*. Po skopiowaniu obrazu na pulpit i nadaniu drugiej partycji karty etykietę *rootfs*, u mnie ta komenda wyglądała *sudo tar -C /media/stas/rootfs/ -xzpf ~/Pulpit/core-image-minimal-zedboard-zynq7.tar.gz*. Po skopiowaniu plików na kartę, włożeniu jej do czytnika na płytce i włączeniu płytki, mając włączony terminal UART na komputerze powinniśmy zobaczyć podobne komunikaty uruchomienia systemu, co w poprzednim przypadku.

Uruchamianie systemu przez JTAG i przez sieć

Gdy w trakcie tworzenia projektu często zmieniamy pliki potrzebne do załadowania systemu, wygodniejsze może się okazać ładowanie go poprzez JTAG. Służy do tego drugie gniazdo USB na płytce ZEDBOARD, opisane PROG. W poprzednim artykule pokazywałem, jak w środowisku XSDK, przy użyciu JTAG, poprzez narzędzie System Debugger, można zdalnie uruchomić program na procesorze. Tym razem uruchomimy w podobny sposób system Linux. Tym razem jednak nie będziemy używać XSDK, tylko uruchomimy System Debugger z linii komend i zadamy mu do wykonania skrypt bootujący Linuksa.

Kopiowanie plików przez JTAG ma jedną dużą wadę – niewielką szybkość. Dużo szybciej przesyła się pliki przez sieć. Przez JTAG zrobimy tylko to, co musimy – skonfigurujemy część PS i PL i wyślemy program U-boot. Jądro systemu i *devicetree* zostanie przesłane przez sieć protokołem TFTP, zaś system plików zostanie zamontowany na partycji sieciowej NFS. Dla ułatwienia, skopiujemy na kartę SD plik uEnv.txt, w którym zapiszemy adres IP naszego komputera i procedurę ładowania systemu. Załadowanie systemu całkowicie bez karty SD również jest możliwe, wymaga jednak modyfikacji kodu źródłowego programu U-boot, w celu wpisania tam na stałe adresu IP naszego komputera. Alternatywnie, można w kodzie źródłowym poinstruować program U-boot, aby do załadowania systemu użył protokołu DHCP, lub BOOTP, ale to znowu wymaga odpowiedniego skonfigurowania sieci. Użyjemy więc prostszej metody.

Aby można było pobierać pliki z komputera poprzez protokół TFTP i zamontować partycję NFS, trzeba najpierw zainstalować i skonfigurować serwer TFTP i NFS. Pod Linuksem sprowadza się to do wykonania w konsoli kilku poleceń. Najpierw trzeba zainstalować odpowiednie pakiety poleceniem *apt*, a następnie odpowiednio skonfigurować serwery za pomocą plików konfiguracyjnych. Instalowanie serwerów należy wykonać za pomocą komendy *sudo apt install nfs-kernel-server xinetd tftpd tftp*. Zanim skonfigurujemy serwery, utwórzmy katalogi, które będą udostępnione dla TFTP i NFS. Można je utworzyć gdziekolwiek, proponuję jednak, dla ułatwienia, utworzyć je w katalogu domowym. Proponuję nazwy *tftpboot* i NFS.

Konfigurowanie serwera NFS polega na wpisaniu do pliku */etc/exports* katalogów, które chcemy, aby były dostępne z zewnątrz. Dodatkowo w tej samej linijce wpisuje się konfigurację danego zasobu. W moim wypadku zadziałało dopisanie do pliku */etc/exports* (edytujemy z *sudo*) następującą linijkę */home/stas/NFS *(rw,sync,no_root_squash,no_subtree_check)*. Oczywiście, ścieżkę należy zastąpić własną. Aby serwer zadziałał, trzeba teraz uruchomić go poleceniem *sudo service nfs-kernel-server start*. Do folderu NFS kopiujemy obraz systemu takim samym poleceniem jak przy kopiowaniu go na kartę SD, tylko ze zmieniona ścieżką. (np. u mnie jest to */home/stas/NFS*). Aby mieć pewność, że serwer NFS działa prawidłowo, należy spróbować zamontować go poprzez sieć. Można to zrobić lokalnie, lub, jeśli jest taka możliwość, z innego komputera podłączonego przez sieć. Polecenia testujące NFS:

```
cd
mkdir test
sudo mount 192.168.0.10:/home/stas/NFS test
ls test
```

Jeśli po wykonaniu ostatniego polecenia zobaczymy listę katalogów naszego systemu plików, to znaczy, że serwer działa prawidłowo. Aby skonfigurować serwer TFTP, należy najpierw utworzyć plik */etc/xinetd.d/tftp* za pomocą polecenia *sudo nano /etc/xinetd.d/tftp* i wpisać tam następującą konfigurację:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /home/<user>/tftpboot
disable         = no
}
```

Oczywiście, w parametrze *server_arg* należy wpisać własną ścieżkę do katalogu, w którym będą pliki udostępniane przez TFTP. Po zmianach w konfiguracji należy zrestartować serwer poleceniem *udo /etc/init.d/xinetd restart*. Na koniec musimy zmienić uprawnienia katalogu *tftpboot*, aby był on dostępny dla każdego z zewnątrz. Polecenia do wykonania w konsoli (zakładam, ze folder nazywa się *tftpboot* i znajduje się w katalogu użytkownika):

```
cd
sudo chmod -R 777 tftpboot
sudo chown -R nobody:nogroup tftpboot
```

Serwer FTFP również należy przetestować. Aby to zrobić, wystarczy wykonać w konsoli (najlepiej jakiegoś innego komputera w sieci, ale lokalnie tez można) polecenia:

```
tftp <adres IP komputera z serwerem>
get uImage
```

Po wykonaniu tych poleceń, powinien zostać wyświetlony komunikat zbliżony do następującego *„Received 3603069 bytes in 0.3 seconds”*. Oznacza on, że wszystko jest w porządku.

Nie mogę polecić żadnego serwera NFS i TFTP pod Windowsa. W razie problemów, można spróbować uruchomić serwery na maszynie wirtualnej z Linuksem, mając skonfigurowaną wirtualną kartę sieciową w trybie *bridged*, dzięki czemu maszyna wirtualna będzie widoczna w sieci jako dodatkowy host.

Foldery, w którym trzymamy pliki dla NFS i TFTP mogą być fizycznie w systemie plików Windowsa i udostępnione Linuksowi na maszynie wirtualnej. Mając skonfigurowane serwery TFTP i NFS możemy zająć się plikiem *uEnv.txt*. Utwórzmy go i wpiszmy tam następującą treść:

```
kernel_image=uImage
devicetree_image=devicetree.dtb
autoload=no
serverip=<adres IP naszego komputera>
bootargs=console=ttyPS0,115200 root=/dev/nfs nfsroot=192.168.0.10:/home/stas/NFS rw ip=dhcp
uenvcmd=echo Booting via TFTP and network... && dhcp && setenv serverip <adres IP naszego komputera> && tftpboot 0x2080000 $serverip:$kernel_image && tftpboot 0x2000000 $serverip:$devicetree_image && bootm 0x2080000 - 0x2000000
```

Należy pamiętać o wpisanie w dwóch miejscach adresu IP komputera, na którym jest serwer NFS i *tftp*.

Analizując jego zawartość zauważymy, że zgodnie z poleceniami tam zawartymi, program U-boot najpierw spróbuje ustalić własny adres IP za pomocą protokołu DHCP. Jeśli płytka podłączona jest do sieci, w której nie działa DHCP, musimy edytować plik, dodając zmienną *ipaddr* i przypisując jej statyczny adres IP płytce. W takim przypadku trzeba oczywiście usunąć ze zmiennej *uenvcmd* polecenie *dhcp* i wpisać w *bootargs* statyczny adres IP. Po zmianach plik będzie wyglądał następująco:

kernel_image=uImage
devicetree_image=devicetree.dtb
*serverip=<**adres IP naszego komputera**>*
*ipaddr=<**statyczny adres IP plytki**>*
*bootargs=console=ttyPS0,115200 root=/dev/nfs nfsroot=<**adres IP naszego komputera**>:/home/stas/NFS rw ip=<**statyczny adres IP plytki**>*
uenvcmd=echo Booting via TFTP and network... && tftpboot 0x2080000 $serverip:$kernel_image && tftpboot 0x2000000 $serverip:$devicetree_image && bootm 0x2080000 - 0x2000000

Po zapisaniu, kopiujemy plik *uEnv.txt* na kartę SD, na pierwszą partycję, na której zwykle znajdowały się pliki niezbędne do uruchomienia systemu.

Cała procedura uruchamiania systemu zaczyna się od uruchomienia konsoli debugowania XSDB. Wykonuje ona skrypt tcl, który konfiguruje części PS i PL Zynq, uruchamia połączenie między nimi, ładuje do pamięci program U-boot i uruchamia go. Kod tego skryptu wygląda następująco:

```
connect
targets 2
rst
fpga -f <ściezka do pliku bitstream>/system_top.bit
source <ściezka do pliku ps7_init.tcl>/ps7_init.tcl
ps7_init
ps7_post_config
dow <ścieżka do pliku u-boot.elf>/u-boot.elf
con
```

Oczywiście, należy dostosować ścieżki do własnej konfiguracji. Pliki *ps7_init.tcl* i *system_top.bit* znajdują się w jednym z projektów *system_top_hw_platform_x* znajdujących się w katalogu roboczym XSDK. Jak je wygenerować, opisałem w poprzednim artykule. Powyższy skrypt można zapisać na przykład w katalogu Linux pod nazwą *network-boot.tcl*.

Przed uruchomieniem systemu musimy skopiować wszystkie potrzebne pliki w odpowiednie miejsca. Tak więc, najpierw skopiujmy pliki *uImage* i *devicetree.dtb* do katalogu *tftpboot*. Można też skopiować tam dla porządku pliki *ps7_init.tcl*, *system_top.bit* i *u-boot.elf*. Po skopiowaniu plików do katalogu *tftpboot*, trzeba pamiętać, aby każdorazowo wykonać, wspomniane już przy konfiguracji *tftp*, polecenia ustawiające uprawnienia dostępu dla tych plików na 777 i ustawiające ich właściciela i grupę odpowiednio na *nobody* i *nogroup*.

Do katalogu NFS musimy wypakować obraz systemu plików. Robi się to dokładnie tak samo, jak w przypadku karty SD, tylko trzeba odpowiednio zmienić katalog, do którego wypakowujemy pliki. Polecenie *sudo tar -C ~/NFS -xzpf <ścieżka do obrazu systemu plików>*. Jeśli wszystko wykonaliśmy poprawnie, czyli: wszystkie pliki są na miejscu (w katalogach *tftpboot* i NFS) i z odpowiednimi uprawnieniami, karta SD z plikiem *uEnv.txt* (najlepiej tylko z tym plikiem) jest w czytniku na płytce i mamy gotowy skrypt *tcl*, możemy włączyć zasilanie zestawu ZEDBOARD. Otwieramy dwie konsole: jedną do komunikacji z płytką po UART, w której uruchamiamy program minicom i drugą, w której wykonamy w celu uruchomienia systemu następujące polecenia:

```
source /opt/Xilinx/Vivado/2017.1/settings64.sh
xsdb <ścieżka do skryptu network-boot.tcl>/network-boot.tcl
```

Trzeba pamiętać, ze zworki określające sposób ładowania systemu muszą być w położeniu bootowania **z karty SD**. Tylko wtedy odczytany zostanie plik uEnv.txt. Jeśli chcielibyśmy uruchomić system z zamontowanym systemem plików na NFS z ustawieniem zworek dla JTAG, konieczna byłaby modyfikacja kodu źródłowego programu U-boot.

Po wykonaniu powyższych komend, powinniśmy zobaczyć w terminalu UART log uruchamiania systemu, podobny do poprzednich.

Podsumowanie

Przedstawiony został proces tworzenia własnej dystrybucji systemu Linux i sposób jej uruchomienia z karty SD. W tym momencie można już pisać aplikacje na płytkę i uruchamiać je zdalnie, za pomocą komputera podłączonego przez USB do JTAG. W kolejnym artykule pokażę jak uruchomić przykładowe projekty na płytce.

Stanisław Aleksiński