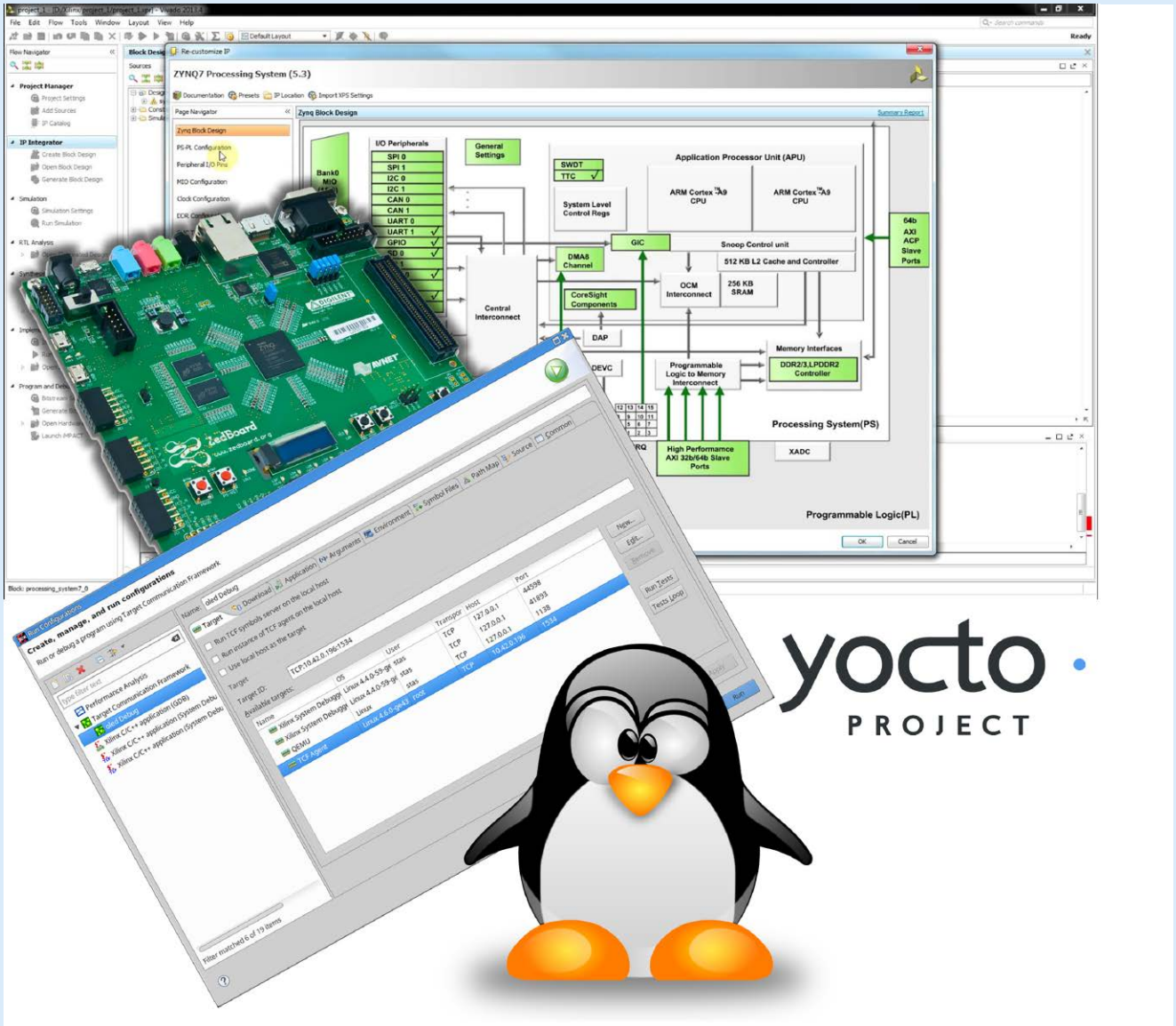


Jak używać układów SoC Xilinx Zynq-7000 z Linuksem – proste przykłady (3)

W poprzednim artykule opisałem różne sposoby uruchomienia Linuksa na płytce zestawu Zedboard. Po udanym uruchomieniu systemu czas na pierwsze programy. Będzie to swoisty „Hello World" zintegrowany z obsługą przycisków oraz przykład obsługi wyświetlacza OLED z interfejsem SPI.

W projekcie migać będzie zielona dioda podłączona bezpośrednio do procesora. Szybkością migania będzie można sterować za pomocą przycisków BTNL i BTNR. Pierwszy z nich będzie zmniejszał szybkość migania, a drugi zwiększał. Dodatkowo użyjemy przełączników (SW0…SW7) i czerwonych diod (LD0…LD7) i połączymy je w pary. Każdy przełącznik będzie sterował świeceniem jednej diody.

Obsługa GPIO

Wszystkie użyte przez nas przełączniki, przyciski i prawie wszystkie diody są dołączone do PL Zynq, więc jest niezbędna odpowiednia konfiguracji FPGA, umożliwiająca dostęp do nich z poziomu procesora. Najłatwiej jest po prostu przyłączyć je do wewnętrznych pinów GPIO procesora przez EMIO (Extended Multiplexed Input/Output). Na szczęście taka właśnie jest domyślna konfiguracja

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 95777, pass: 53wtjyf6

w projekcie, który mamy już gotowy (opis utworzenia tego projektu można znaleźć w jednym z poprzednich artykułów). Możemy więc przejść do kolejnego kroku – dostosowania Device Tree (DT) poprzez modyfikację plików dts.

Można jednak zadać sobie pytanie, czy rzeczywiście trzeba coś edytować w plikach dts? Jeśli chcielibyśmy tylko w prosty sposób poprzez SysFs manipulować pinami GPIO lub czytać ich wartości, to odpowiedź brzmi – nie, ponieważ ogólny wpis o GPIO już tam jest. Skoro jednak do tych pinów będą dołączone przyciski i diody, to warto by było użyć dedykowanych im sterowników. Dodajmy więc wpisy łączące sterowniki z pinami GPIO, do których podłączone są przyciski i diody. Więcej informacji na temat tych sterowników i niezbędnych wpisów w DT można znaleźć na wiki Xilinx (***https://goo.gl/gFyZWM***).

W poprzednim artykule polecałem, żeby utworzyć katalog dts na różne wersje DT. Powinna być tam „czysta", jeszcze nieedytowana wersja plików dts dostarczona przez Analog Devices (ADV) w folderze *clean*, więc skopiujmy cały ten folder i nadajmy mu nazwę np. *projekt1*.

Przyjrzyjmy się teraz zawartości plików dts pod kątem naszych potrzeb. W pliku *zynq-zed-adv7511.dtsi* znajdziemy na końcu wpis dotyczący czerwonych diod, zatem do kompletu brakuje diody zielonej. Tu warto wspomnieć kilka słów o numeracji GPIO w Zynq. Jest ona dość intuicyjna, jeśli się ją już zna. Pierwsze 54 numery (0…53) to GPIO MIO (*Multiplexed Input/Output*), czyli są to piny doprowadzone bezpośrednio do procesora, natomiast kolejne 64 numery (54…117) należą do EMIO GPIO, wyprowadzonego do części PL i odpowiadają pinom EMIO o numerach 0…63. Sprawdźmy więc, czy diody w DT są odpowiednio ponumerowane, na przykładzie pierwszej diody oznaczonej jako LD0. W pliku *zynq-zed-adv7511.dtsi* widać, że jest przyłączona do pinu GPIO numer 73. Jeśli odejmiemy od tego 54, to uzyskamy liczbę 19. Rzut oka do dokumentacji płytki (np. do pliku *ZedBoard_HW_UG_v2_2.pdf*) powie nam, że ta dioda podłączona jest do pinu T22 Zynq. Plik *zed_system_constr.xdc* (można go znaleźć wśród plików z projektem referencyjnym od Analog Devices, w folderze *hdl-hdl_2016_r2/projects/common/zed*) powie nam zaś, że ten pin podłączony jest do sygnału numer 19 wektora *gpio_bd*, podłączonego dalej do pinów EMIO procesora (dociekliwych odsyłam do plików projektu Vivado). Widać więc, że nie ma pomyłki.

Wiedząc, że zielona dioda podłączona jest do pinu GPIO MIO7 (można to znaleźć w ww. miejscu dokumentacji, lub… po prostu przeczytać z płytki), możemy dodać odpowiedni wpis w pliku *zynq-zed-adv7511.dtsi*, w obszarze o nazwie *leds*:

```
ld9 {
    label = „ld9:green”;
    gpios = <&gpio0 7 0>;
};
```

Wpis dla przycisków wygląda następująco (dodajemy go powyżej ostatniej, zamykającej klamry w pliku, na tym samym poziomie w drzewie, co wpis dotyczący diod):

```
gpio-keys-polled {
        compatible = „gpio-keys”;
        poll-interval = <30>;
        btnu {
            label = „btnu”;
            gpios = <&gpio0 58 0>;
            linux,code = <103>; /* up */
        };
        btnd {
            label = „btnd”;
            gpios = <&gpio0 55 0>;
            linux,code = <108>; /* down */
        };
        btnl {
            label = „btnl”;
            gpios = <&gpio0 56 0>;
            linux,code = <105>; /* left */
        };
        btnr {
            label = „btnr”;
            gpios = <&gpio0 57 0>;
            linux,code = <106>; /* right */
        };
        btnc {
            label = „btnc”;
            gpios = <&gpio0 54 0>;
            linux,code = <28>; /* enter */
        };
    };
```

Numery pinów GPIO można obliczyć tą samą metodą, co w wypadku diod. Po zakończeniu edycji otwieramy konsolę, przechodzimy do katalogu z nowym zestawem plików dts, kompilujemy pliki dts do pliku dtb poleceniem dtc -I dts -O dtb -o devicetree.dtb zynq-zed-adv7511.dts i kopiujemy w odpowiednie miejsce. Domyśle opcje konfiguracyjne jądra systemu są dla nas satysfakcjonujące, więc możemy pominąć kompilację jądra. Proces kompilacji plików dts i jądra oraz proces uruchamiania systemu opisałem już w artykule poświęconym uruchamianiu Linuksa na Zynq.

Jeśli wszystko zostało zrobione prawidłowo, to po uruchomieniu systemu powinniśmy być w stanie znaleźć pliki reprezentujące dodane do DT przyciski i diody. W pierwszym przypadku powinien być to wspólny dla wszystkich przycisków plik */dev/input/input0*, zaś w drugim – osobny dla każdej diody folder plików *ldx:red* (gdzie *x* to numer diody) i *ld9:green* znajdujący się w katalogu */sys/class/leds*.

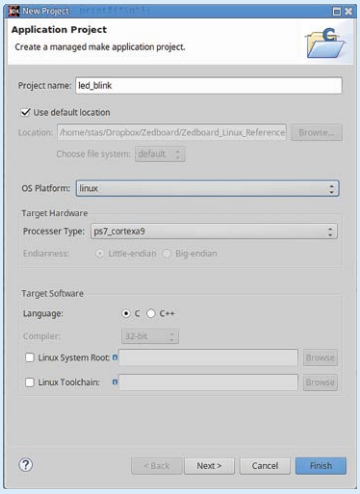
Przed uruchomieniem programu warto go przetestować. W przypadku przycisków można to zrobić np. komendą cat /dev/input/event0 | hexdump, która będzie wypisywać w odpowiednim formacie do konsoli informację o naciśniętym przycisku. Aby uruchomić diodę, należy wpisać liczbę z zakresu od 1 do 255 do pliku *brightness*, znajdującego się w folderze nazwanym tak jak ta dioda. Przykładowo, aby zapalić diodę *ld0*, należy wydać w konsoli komendę echo 1 > /sys/class/leds/ld0:red/brightness. Podobnie można sprawdzić pozostałe diody, zmieniając jednak w ścieżce nazwę diody. Aby ją zgasić, należy do pliku *brightness* wpisać 0. Można się domyślić, że jeśli sprzęt to obsługuje, dioda powinna się świecić z zadaną jasnością, jednak w tym przypadku następuje zwykłe włączanie i wyłączanie diody.

Warto też przy okazji wspomnieć o sposobie obsługi przycisków i diod bezpośrednio przez SysFs. W taki sposób będziemy w programie odczytywać aktualne położenie przełączników. Tym razem zmienia się numeracja pinów – trzeba dodać pewien offset. Jest on zależny od kodu jądra i w momencie pisania tego artykułu wynosi 906. Więcej na temat tego, dlaczego ta liczba jest akurat taka, można znaleźć na wspomnianej już stronie wiki, dla nas jednak istotne jest, jak można łatwo ją znaleźć. Otóż znajduje się ona w nazwie jednego z katalogów, które znajdziemy w katalogu */sys/slass/gpio*. Nazwa tego katalogu zaczyna się od „gpiochip" i kończy właśnie tą liczbą. Jeśli takich katalogów jest w systemie więcej – trzeba eksperymentować. Gdy wiemy już, jaki jest offset, możemy łatwo policzyć numery dla przełączników doprowadzonych do pinów EMIO 11…18, czyli GPIO 65…72 – będzie to zakres 971…978. Pierwszym krokiem w obsłudze przełączników przez SysFs jest wyeksportowanie pinu echo 971 > /sys/class/gpio/export. Powinien pojawić się folder gpio971. Następnie należy skonfigurować pin jako wejście echo in > /sys/class/gpio/gpio971/direction. Na końcu możemy odczytać aktualne położenie z pliku value cat /sys/class/

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **95777**, pass: **53wtjyf6**

gpio/gpio971/value. To wszystko może też oczywiście wykonać program.

Nasz przykładowy program to zmodyfikowana wersja programu, który można znaleźć na wspomnianej już stronie wiki Xilinx. Dodałem do niego obsługę przełączników przez SysFs i sterowanie czerwonymi diodami. Program jest napisany w języku C. Jego dokładną analizę zostawiam czytelnikowi.

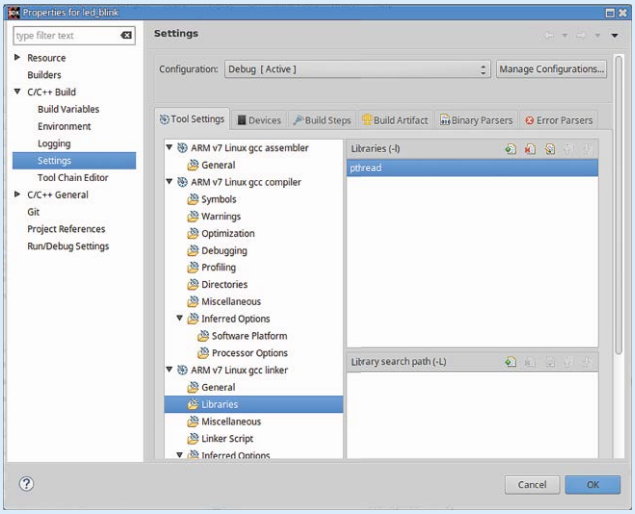


Rysunek 8. Okno właściwości nowego projektu – nadanie nazwy

Uruchamiamy XSDK i tworzymy nowy projekt (*File* → *New* → *Application Project*). Zostanie wyświetlone okno jak na **rysunku 8**. Wpisujemy nazwę projektu np. *led_blink*. Z listy rozwijalnej opcji OS Platform wybieramy *linux*. Resztę opcji pozostawiamy domyślną i klikamy *Next*. W kolejnym oknie wybieramy z listy szablonów po lewej stronie *Linux Empty Application* i klikamy *Finish*. W głównym oknie po lewej stronie powinien się pojawić nasz projekt. Rozwijamy go i klikamy prawym przyciskiem myszy w folder *src*. Z menu wybieramy opcję *Import*. Zostanie wyświetlone okno, za którego pomocą zaimportujemy plik źródłowy programu. Wybieramy opcję *General* → *File system* i klikamy *Next*. W kolejnym oknie na górze wpisujemy lub wyszukujemy ścieżkę, do której został pobrany plik źródłowy, a następnie wybieramy go z listy po prawej (plik *main.c*) i klikamy *Finish*.

Na tym etapie kompilacja jeszcze się nie powiedzie, ponieważ nie dodaliśmy do opcji kompilacji biblioteki *pthread*. Znów klikamy prawym, przyciskiem myszy nasz projekt i tym razem wybieramy *Properties*. Tam szukamy po prawej stronie ustawień kompilacji (*C/C++ Build* → *Settings*). Na środku, w karcie *Tool Settings* rozwijamy *ARM v7 Linux gcc linker* → *Libraries*. Po prawej stronie na górze można tam dodać niezbędną bibliotekę, wciskając ikonkę z zielonym plusem. W oknie, które zostanie wyświetlone, wpisujemy *pthread* i klikamy OK. Po tej operacji okno *Properties* powinno wyglądać jak na **rysunku 9**. Po kliknięciu OK program powinien się już skompilować prawidłowo.

Czas teraz na uruchomienie programu. Do uruchomienia aplikacji niezbędne będzie połączenie sieciowe pomiędzy komputerem



Rysunek 9. Okno właściwości nowego projektu – parametry

a płytką. Jeśli jest taka możliwość, polecam ze względu na wygodę użyć do tego switcha lub routera, jednak bezpośrednie połączenie kablem ethernetowym też zadziała. Mając już uruchomiony system na podłączonej do sieci płytce, możemy zaznaczyć nasz projekt i otworzyć okno konfiguracji uruchamiania (prawy przycisk → *Run As* → *Run Configurations*). Po lewej stronie zaznaczamy *Target Communication Framework* i klikamy na górze w ikonkę z żółtym plusem służącą do dodawania nowej konfiguracji. Jeśli przed otwarciem tego okna zaznaczyliśmy nasz projekt, to utworzy się nowa konfiguracja o nazwie *led_blink_Debug*. Teraz w prawej części okna odznaczamy opcję *Use local host as the target*. Nieaktywna dotąd część okna powinna się uaktywnić, a na liście dostępnych targetów powinna pojawić się nasza płytka. Można ją rozpoznać po adresie IP, który powinien się zgadzać z tym z naszej płytki. IP płytki można sprawdzić poleceniem *ifconfig* wykonanym w konsoli systemu uruchomionego na płytce, więc wybieramy odpowiedni target i przechodzimy na kartę *Application*. Tam w polu *Remote File Path* trzeba wpisać ścieżkę dostępu programu na płytce, np. */tmp/led_blink.elf*. Klikamy *Apply*, a następnie *Run*. Jeśli wszystko zrobiliśmy prawidłowo, zielona dioda powinna zacząć migać. Można też zmieniać szybkość migania diody za pomocą przycisków BTNL i BTNR i zapalić czerwone diody, ustawiając przełączniki w położeniu bliższym diodom.

Wyświetlacz OLED

Dostępny na płytce Zedboard, a możliwy do podłączenia do płytki Zybo wyświetlacz OLED wygląda niezwykle efektywnie, a przy tym jest bardzo przydatny, gdy na przykład chce się wyświetlić wartości zmiennych podczas testowania projektów. Jak go jednak uruchomić? Okazuje się, że może być to niezwykle proste.

Gdy po raz pierwszy próbowałem uruchomić wyświetlacz, natrafiłem na duże trudności, gdyż wydawało mi się, że niezbędny będzie moduł jądra Linuksa, czyli po prostu sterownik. Oczywiście dawało to możliwość wyświetlania dowolnego kształtu na wyświetlaczu, ale nadal sam musiałbym generować np. napisy. Gdy już udało mi się uruchomić, na szczęście gotowy, sterownik, okazało się, że jest problem wynikający prawdopodobnie z tego, że jest on niekompatybilny z wersją jądra Linuksa, której używałem. Bez tajemnej, ale przydatnej wiedzy na temat pisania sterowników na Linuksa ani rusz. Tymczasem okazało się, że rozwiązanie jest dużo prostsze – można po prostu sterować pinami GPIO połączonymi z wyświetlaczem, z aplikacji napisanej w przestrzeni użytkownika! Jedyny sterownik, jaki jest potrzebny, to gotowy i działający moduł do obsługi SPI, który na szczęście jest bez problemu dostępny. Co więcej, Digilent niedawno opublikował bibliotekę do obsługi wyświetlacza na płytki PmodOLED dla Zybo, która pozwala na bezproblemowe wyświetlanie tekstu. Jako że Zedboard ma ten sam układ, tylko z nieco większą ilością zasobów, przeportowanie tej biblioteki sprowadza się do zmiany numerów pinów GPIO. Dodatkowo trzeba zrobić odpowiednie modyfikacje w Device Tree, aby zadziałał sterownik SPI. A więc do dzieła!

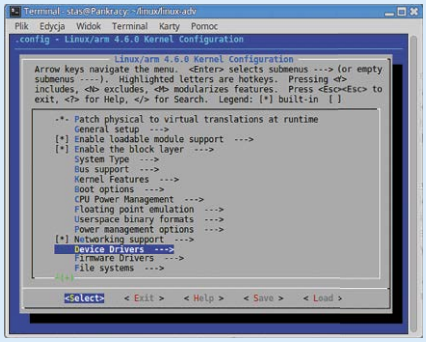
W jednym z poprzednich numerów pokazałem, jak przygotować platformę sprzętową, korzystając ze źródeł od Analog Devices. W tym projekcie wyświetlacz OLED podłączony jest już do pinów GPIO procesora, więc nie trzeba robić żadnych zmian w projekcie. W pliku *zed_system_constr.xdc* (można go znaleźć wśród plików z projektem referencyjnym od Analog Devices, w folderze *hdl-hdl_2016_r2/projects/common/zed*) można znaleźć informację o tym, do których pinów GPIO EMIO (*Extended Multiplexed Input/Output*) jest dołączony wyświetlacz. Mamy więc wszystkie informacje potrzebne do przejścia do kolejnego kroku.

Jak już wspomniałem, potrzebny nam będzie sterownik do SPI. Jako że używamy pinów GPIO, niezbędny będzie tzw. sterownik *Bitbang*, który implementuje protokół komunikacyjny na pinach

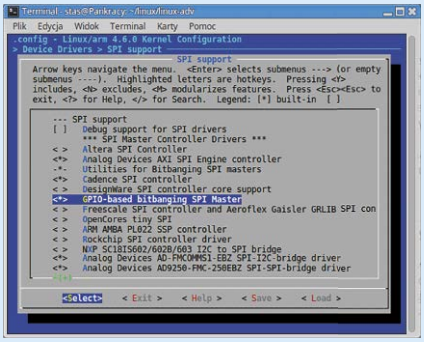
GPIO. Najprostszym dla nas wyjściem będzie wkompilować go w jądro Linuksa.

Jak już napisałem w artykule o budowaniu i uruchamianiu Linuksa na Zedboard, kompilację jądra Linuksa najprościej robi się w linuksowym środowisku. Jeśli ktoś nie używa na co dzień Linuksa, proponuję zainstalować go na maszynie wirtualnej.

Cała procedura jest niemal identyczna jak wcześniej, z jednym tylko wyjątkiem. Już po wstępnym skonfigurowaniu jądra, a jeszcze przed samą kompilacją, uruchamiamy narzędzie do ręcznej konfiguracji jądra komendą *make menuconfig*.



Rysunek 10. Wyświetlenie okna interfejsu w terminalu



Rysunek 11. Zaznaczenie drivera do wkompilowania w jądro Linuksa

W terminalu powinien zostać wyświetlony interfejs, jak na **rysunku 10**. Używając klawiszy kierunkowych na klawiaturze, wybieramy opcję *Device Drivers* i wciskamy *Enter*. W kolejnym ekranie przechodzimy do menu *SPI support*. W następnym ekranie umieszczamy kursor na linijce „GPIO-based bitbanging SPI Master". Aby wkompilować ten driver w jądro, należy dwukrotnie wcisnąć spację. Po pierwszym wciśnięciu przy nazwie sterownika, w nawiasach kwadratowych, pojawi się literka M, oznaczająca, że sterownik zostanie skompilowany jako moduł. Drugie wciśnięcie spowoduje pojawienie się tam gwiazdki (**rysunek 11**) oznaczającej, że driver zostanie zgodnie z naszym zamiarem wkompilowany w jądro. Na koniec przenosimy dolny kursor strzałkami na pole *Save* i wciskamy dwukrotnie *enter*, aby zapisać konfigurację. Program konfiguracyjny zamykamy poprzez kilkakrotne naciśnięcie przycisku *Esc* lub poprzez przejście dolnym kursorem na pole *Exit* i naciśnięcie *Enter*.

Po skonfigurowaniu jądra kompilujemy je, tak jak poprzednio, poleceniem *make uImage* LOADADDR=0x00008000.

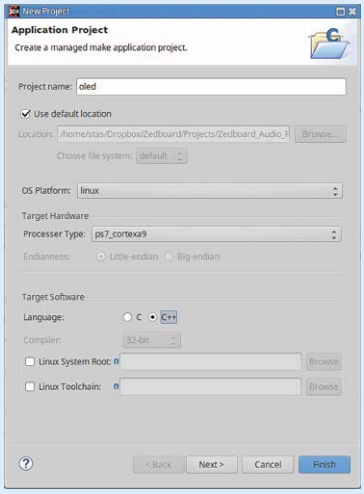
Skopiujmy na początek folder z czystymi plikami dts (jeśli weźmiemy projekt z poprzedniego projektu, to projekt z OLED nie zadziała, gdyż poprzednio używaliśmy sterownika do przycisków, a teraz stan przycisku jest odczytywany przez SysFs) i nazwijmy go np. *projekt2*. Następnie otwórzmy plik *zynq-zed-adv7511.dtsi* i edytujmy go, dodając poniżej wpisu o diodach LED następujące linijki:

```
spioled {
    compatible = „spi-gpio";
    #address-cells = <1>;
    #size-cells = <0>;
    //ranges;
    gpio-sck = <&gpio0 61 0>;
    gpio-mosi = <&gpio0 62 0>;
    num-chipselects = <0>;
    spidev: spidev@0 {
        compatible = „spidev";
        spi-max-frequency = <1000000>;
        reg = <0>;
    };
};
```

Numery GPIO 61 i 62 to numery pinów, do których są podłączone piny SPI i MOSI wyświetlacza. Tę informację, jak już napisałem, można znaleźć w pliku *zed_system_constr.xdc*. Do numeru GPIO, który można znaleźć w pliku, dodajemy 54, gdyż system widzi w tym samym banku wszystkie piny GPIO, również te należące bezpośrednio do procesora, a numeracja pinów EMIO, należących do części PL, zaczyna się właśnie od 54. W moim przypadku fragment pliku *zed_system_constr.xdc* wygląda następująco:

```
set_property -dict {PACKAGE_PIN P16
IOSTANDARD LVCMOS25} [get_ports gpio_bd[0]]
; ## BTNC
set_property -dict {PACKAGE_PIN R16
IOSTANDARD LVCMOS25} [get_ports gpio_bd[1]]
; ## BTND
set_property -dict {PACKAGE_PIN N15
IOSTANDARD LVCMOS25} [get_ports gpio_bd[2]]
; ## BTNL
set_property -dict {PACKAGE_PIN R18
IOSTANDARD LVCMOS25} [get_ports gpio_bd[3]]
; ## BTNR
set_property -dict {PACKAGE_PIN T18
IOSTANDARD LVCMOS25} [get_ports gpio_bd[4]]
; ## BTNU
set_property -dict {PACKAGE_PIN U10
IOSTANDARD LVCMOS33} [get_ports gpio_bd[5]]
; ## OLED-DC
set_property -dict {PACKAGE_PIN U9
IOSTANDARD LVCMOS33} [get_ports gpio_bd[6]]
; ## OLED-RES
set_property -dict {PACKAGE_PIN AB12
IOSTANDARD LVCMOS33} [get_ports gpio_bd[7]]
; ## OLED-SCLK
set_property -dict {PACKAGE_PIN AA12
IOSTANDARD LVCMOS33} [get_ports gpio_bd[8]]
; ## OLED-SDIN
set_property -dict {PACKAGE_PIN U11
IOSTANDARD LVCMOS33} [get_ports gpio_bd[9]]
; ## OLED-VBAT
set_property -dict {PACKAGE_PIN U12
IOSTANDARD LVCMOS33} [get_ports gpio_bd[10]]
; ## OLED-VDD
```

Widać, że pin SCK (SCLK) jest doprowadzony do GPIO numer 54+7=61, a pin MOSI (SDIN): 54+8=62. Po zapisaniu pliku kompilujemy pliki DTS do DTB, tak jak poprzednio dtc -I dts -O dtb -o devicetree.dtb zynq-zed-adv7511.dts. Biblioteka jest dostępna na serwerze ftp. Rozpakowujemy ją, aby móc później zaimportować te źródła do projektu w XSDK. Dodatkowo załączone są tam klasy do obsługi GPIO.



Rysunek 12. Tworzenie nowego projektu

Otwórzmy więc XSDK i stwórzmy nowy projekt (menu *File* → *New* → *Application Project*). W oknie,

Listing 1. Przykładowy program

```
extern „C” {
#include „OLED.h”
#include <unistd.h>
#include <stdio.h>
#include <string.h>
}
#include „GPIOClass.h”
#include <iostream>

using namespace std;

void delay(int mSec);

string val;
OledClass OLED;
string OffButton = „960”;  // 906+54; Center button, to terminate
program
GPIOClass tButton = GPIOClass(OffButton);
int xpos = 0;
int ypos = 0;
int xdir = 1;
int ydir = 1;

void setup()
{
    OLED.begin();
    //Choosing Fill pattern 0
    OLED.setFillPattern(OLED.getStdPattern(0));
    //Turn automatic updating off
    OLED.setCharUpdate(0);
    tButton.export_gpio();
    tButton.setdir_gpio(„in”);
}

int main()
{
    setup();
    while (1)
    {
        OLED.clearBuffer();
        OLED.moveTo(xpos, ypos);
        OLED.drawString(„ZedBoard”);
        OLED.updateDisplay();
        xpos += xdir;
        ypos += ydir;
        if (xpos >= 64 || xpos <= 0) xdir *= -1;
        if (ypos >= 24 || ypos <= 0) ydir *= -1;
        delay(20);
        tButton.getval_gpio(val);
        if (val == „1”) break;
    }
    OLED.end();
    return 0;
}

void delay(int mSec) {
    usleep(mSec * 1000);
}
```

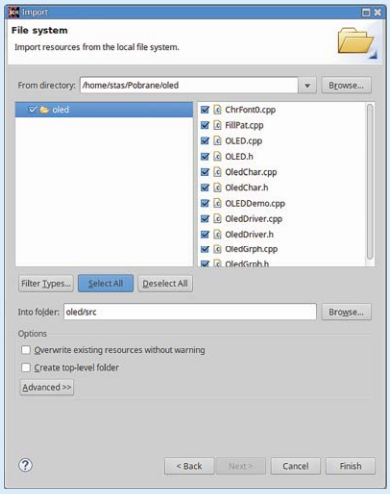
które zostanie wyświetlone (**rysunek 12**), w polu *Project name* wpisujemy nazwę projektu, np. *oled*, zmieniamy platformę systemową (OS platform) na Linux i wybieramy C++ jako nasz język programowania. Resztę opcji pozostawiamy domyślnie. Klikamy *Next* i w następnym ekranie wybieramy *Empty application* i klikamy *Finish*.

Po utworzeniu projektu rozwijamy go w sekcji z lewej strony, przechodzimy do katalogu *src* i usuwamy całą jego zawartość. Następnie klikamy projekt prawym przyciskiem myszy i wybieramy opcję *Import*. W oknie, które się pojawi, wybieramy *General → File System* i klikamy *Next*. W następnym oknie (**rysunek 13**), w polu From Directory, wpisujemy ścieżkę, do której wypakowaliśmy wcześniej pliki z biblioteką OLED. Zaznaczamy wszystkie pliki i klikamy *Finish*.

Aby biblioteka OLED skonfigurowana dla Zybo zadziałała dla Zedboard, trzeba ją odrobinę zmodyfikować. Dalej pokazane są linijki po edycji (plik *OledDriver.cpp*):

```
const int DataCmd   = 59;  //Pin EMIO GPIO 5
const int Reset     = 60;  //Pin EMIO GPIO 6
const int VbatCtrl  = 63;  //Pin EMIO GPIO 9
const int VddCtrl   = 64;  //Pin EMIO GPIO 10
const int GpioBase  = 906;
const char* SpiName = „/dev/spidev32766.0”;  //
filename of spidev node
```

Pierwsze cztery zmienne definiują numery pinów, do których są podłączone wyprowadzenia wyświetlacza. Wcześniej opisałem już, skąd się wzięły te liczby. Kolejna zmienna to *GpioBase*. Definiuje ona przesunięcie, jakie jest w numeracji pinów w przestrzeni użytkownika, czyli z perspektywy zwykłego programisty, który nie modyfikuje kodu źródłowego jądra Linuksa. Ta liczba jest zależna od wersji Linuksa. W chwili, gdy piszę ten artykuł to przesunięcie jest równe 906, ale w przyszłości może się zmienić. Szerzej opisałem zagadnienie obsługi GPIO w poprzednim artykule.



Rysunek 13. Wpisanie ścieżki dostępu do biblioteki

Ostatnia zmienna to nazwa pliku reprezentującego urządzenie SPI. Powinna zaczynać się od *spidev* i kończyć jakimś numerem. W moim przypadku ten numer jest taki jak powyżej, ale może być też inny. Trzeba to sprawdzić po uruchomieniu systemu.

Drugi plik, który trzeba zmodyfikować, to plik *OLEDDemo.cpp*. W związku z tym, że te pliki pochodzą z przykładu dla zestawu płytek PmodPack dla Zybo i jest to demo obejmujące wszystkie te płytki, większość kodu z tego pliku nie będzie nam potrzebna. Można więc usunąć go w całości i wpisać tam kod z **listingu 1**.

Jeśli wszystko zostało wykonane poprawnie, można teraz podłączyć płytkę i uruchomić Linuksa. Szczegółowo opisałem, jak to zrobić, w poprzednich artykułach. Mając uruchomioną płytkę z działającym Linuksem, możemy uruchomić program. W tym celu klikamy prawym przyciskiem myszy nasz projekt w ekranie po lewej i z menu wybieramy *Run as à Run Configurations…* Konfiguracja jest niemal identyczna z tą z poprzedniego artykułu, więc pominę jej opis. Po kliknięciu przycisku Run powinniśmy zobaczyć na wyświetlaczu latający napis *Zedboard*.

Dokładną analizę kodu źródłowego programu zamieszczonego na listingu 1 zostawiam czytelnikowi. Jako zadanie dodatkowe polecam modyfikację projektu tak, aby do odczytywania stanu przycisku nie używać SysFs, tylko dedykowanego sterownika, tak jak w poprzednim projekcie.

Stanisław Aleksiński