Jak używać układów SoC Xilinx Zynq-7000 z Linuksem – proste przykłady (4)

Sterowanie LED-ami poprzez interfejs Web

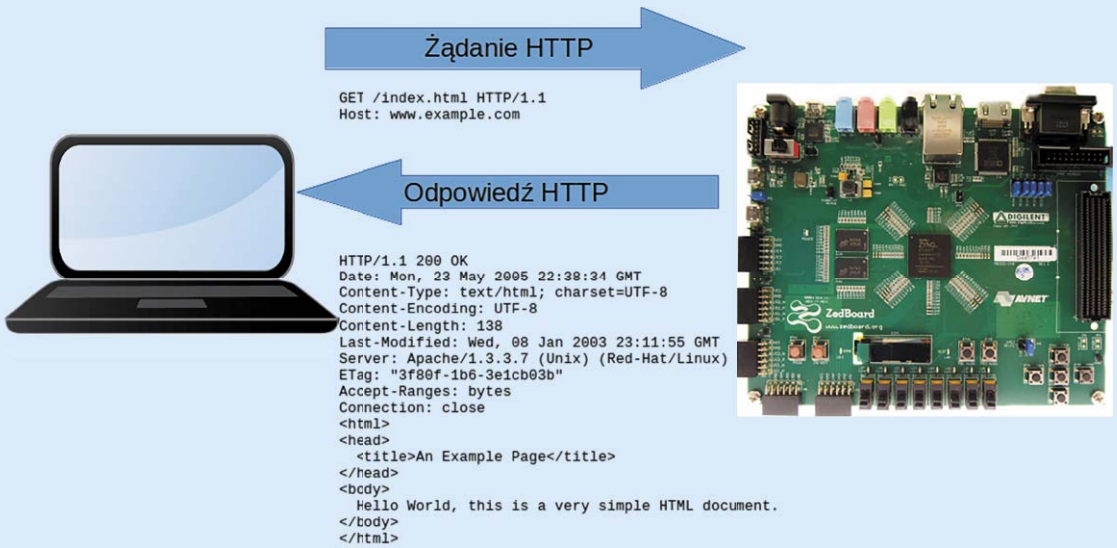
Układy takie jak Zynq bardzo często znajdują się w systemach wbudowanych, do których nie jest przyłączony żaden wyświetlacz czy klawiatura, więc jest potrzebny jakiś inny sposób na sterowanie jego działaniem. Często tę rolę pełni interfejs webowy, dzięki któremu można konfigurować aplikację zdalnie poprzez stronę wyświetloną w przeglądarce. W artykule opisano jeden ze sposobów na wykonane takiego interfejsu na przykładzie sterowania diodami LED.

Aplikację webową można tworzyć za pomocą wielu języków programowania i na wiele sposobów – jako natywną aplikację, poprzez interfejs CGI, itd. Pod spodem zawsze jednak dzieje się to samo – komunikacja poprzez protokół HTTP. Jest to protokół, za którego pomocą każda przeglądarka internetowa może połączyć się z każdym serwerem HTTP i przesłać mu odpowiednie żądanie, np. pobrania kodu HTML strony internetowej lub obrazów, które mają się na niej wyświetlić. Oczywiście w tym kontekście słowo serwer oznacza typ programu, którego rolą jest przyjmować żądania od programu typu klient (w tym przypadku jest to przeglądarka). Ilustruje to **rysunek 1**. Laptop i płytka Zedboard są dołączone do Internetu lub znajdują się w sieci LAN. Uruchomiona na laptopie przeglądarka wysyła żądanie HTTP, a serwer uruchomiony na płytce Zedboard na nie odpowiada.

Bardzo często aplikacja webowa, niezależnie od tego, w jakim języku programowania jest napisana, nie implementuje protokołu HTTP, tylko podłączona jest do serwera HTTP, który w razie potrzeby ją uruchamia. W tym przykładzie jest jednak inaczej – aplikacja będzie zajmować się również parsowaniem żądań HTTP i formowaniem odpowiedzi. Dzięki temu unikniemy konieczności uruchamiania i konfiguracji serwera HTTP. Uruchomienie takiego serwera nie jest bardzo skomplikowane, ale wymaga wykonania dodatkowych kroków. Implementacją protokołu HTTP zajmie się biblioteka *libmicrohttpd*, której musimy tylko dostarczyć odpowiednie funkcje do obsługi żądań. Dzięki niej, w tym prostym przykładzie cały kod implementujący serwer HTTP mieści się w niecałych 200 linijkach.

Przygotowanie

Aby uruchomić aplikację, potrzebne są wszystkie pliki niezbędne do uruchomienia Linuksa na płytce Zedboard. Sposób ich przygotowania opisałem jednym z poprzednich artykułów kursu. Nie



Rysunek 1. Przykładowa transakcja HTTP pomiędzy komputerem a płytką (https://pl.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

zmienia się sposób bootowania Linuksa, więc nie trzeba zmieniać nic w programie Uboot, ani FSBL. Nie ma też potrzeby modyfikować jądra systemu. Aplikacja będzie sterować diodami poprzez dedykowany sterownik Linuksa, więc potrzebne są odpowiednie wpisy w plikach DST. Tu jednak również nie są potrzebne żadne modyfikacje, ponieważ wpisy dotyczące tych diod są już domyślnie dodane. Potrzebna jest więc tylko jedna modyfikacja: jako że będziemy używać biblioteki *libmicrohttpd*, trzeba ją dodać do systemu plików.

System Yocto w dużym uproszczeniu oparty jest na receptach (*recipes*), które zgromadzone są w warstwach (*Layers*). Po pobraniu repozytorium *poky* dostępne są tylko podstawowe recepty. Aby zbudować podstawowy system dla układu Zynq należy dodatkowo pobrać warstwę *meta-xilinx*. Te warstwy mamy już pobrane. Recepta budowania biblioteki *libmicrohtd* znajduje się w nie pobranej jeszcze warstwie *meta-oe*, w repozytorium *meta-openembedded*, które można znaleźć na Github. Aby je pobrać, należy uruchomić terminal, przejść do folderu Yocto, w którym znajdują się poprzednio pobrane repozytoria *poky* i *meta-xilinx* i wykonać komendę *git clone git@github.com:openembedded/meta-openembedded.git*. To polecenie pobierze całe repozytorium. Trzeba jeszcze dodać ścieżkę potrzebnej warstwy do pliku *bblayers.conf*, do zmiennej *LAYERS*. Po modyfikacji ta zmienna wygląda tak:

```
BBLAYERS ?= „ \
  /home/stas/yocto/poky/meta \
  /home/stas/yocto/poky/meta-poky \
  /home/stas/yocto/poky/meta-yocto-bsp \
  /home/stas/yocto/meta-xilinx \
  /home/stas/yocto/meta-openembedded/meta-oe \
  „
```

Teraz można zmodyfikować główny plik konfiguracyjny – local.conf. Znajdziemy go w katalogu <yocto>/poky/zedboard/conf/ . W pliku *local.conf* szukamy zmiennej *IMAGE_INSTALL_append* i dodajemy do niej bibliotekę *libmicrohttpd* pamiętając o dodaniu na końcu spacji. Po modyfikacji linijka będzie wyglądała następująco – *IMAGE_INSTALL_append = „libgcc libstdc++ libmicrohttpd”*. Następnie ponownie generujemy obraz zgodnie z instrukcją. Nie będzie to trwało tak długo, jak poprzednim razem, gdyż Yocto ma już skompilowane większość potrzebnych pakietów, więc musi tylko skompilować nową bibliotekę i dodać ją do obrazu.

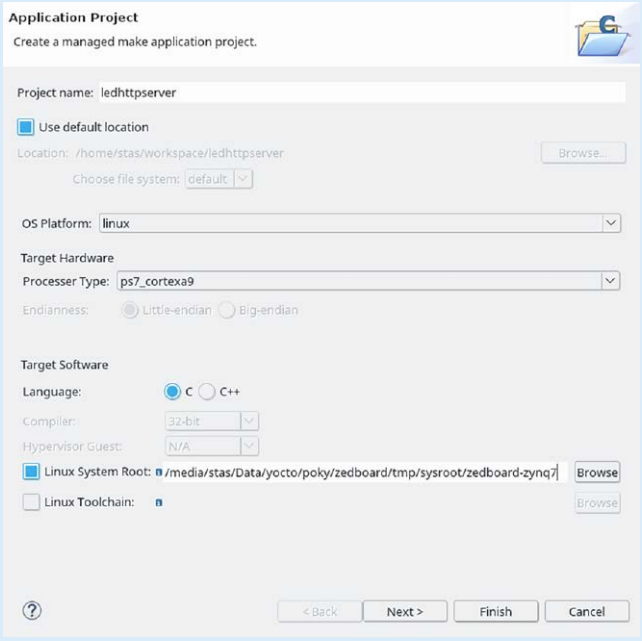
Po wygenerowaniu nowego obrazu, kopiujemy go w odpowiednie miejsce, zgodnie z wybraną metodą uruchamiania Linuksa i uruchamiamy go.

Aplikacja

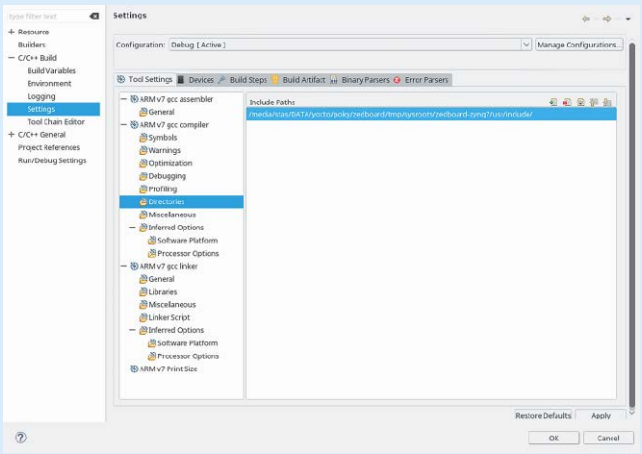
Uruchamiany program potrzebuje dodatkowej biblioteki, więc tym razem potrzebne są dodatkowe kroki, aby poprawnie zbudować aplikację. Niezbędne jest wskazanie dwóch katalogów. W jednym z nich są pliki nagłówkowe, w drugim – biblioteki. Projekt *libmicrohttpd* jest skompilowany jako biblioteka współdzielona, ale i tak podczas budowania jest ona potrzebna.

Zaczynamy od uruchomienia XSDK – z konsoli lub z poziomu Vivado. Przykładowa aplikacja pisana na system Linux, więc nie trzeba koniecznie posługiwać się tą samą, co przy poprzednich aplikacjach przestrzenią roboczą (*workspace*), ale można tak zrobić.

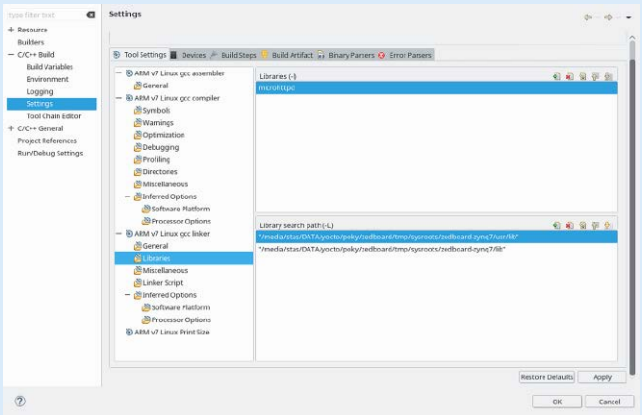
Tak jak w przypadku dwóch poprzednich aplikacji, zaczynamy od utworzenia nowego projektu: *File* → *New* → *Application Project*. Jako nazwę projektu można wpisać *ledhttpserver*. W opcji OS Platform wybieramy Linux, język – C, a na koniec zaznaczamy opcję Linux System Root i wpisujemy tam ścieżkę, w której Yocto trzyma potrzebne do budowania aplikacji zależności. Ta lokalizacja znajduje się w folderze *<yocto>/poky/zedboard/tmp/sysroot/zedboard-zynq7*. Na koniec ekran tworzenia aplikacji powinien wyglądać podobnie jak na **rysunku 2**, na którym ustawiona jest



Rysunek 2. Okno tworzenia nowego projektu



Rysunek 3. Ustawienia kompilatora – ścieżka do plików nagłówkowych



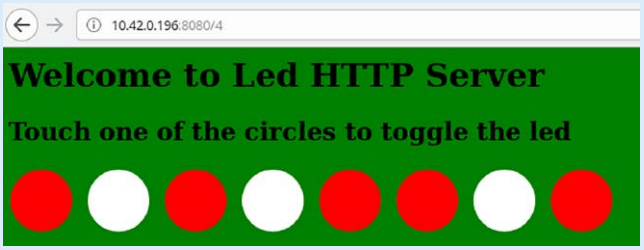
Rysunek 4. Ustawienia linkera – biblioteki zewnętrzne

ścieżka do folderu yocto taka, jak na moim komputerze. Następnie klikamy *Next*, jako szablon aplikacji wybieramy *Empty application* i klikamy *Finish*.

Następnym krokiem jest dodanie biblioteki w opcjach linkera. W tym celu należy zaznaczyć nazwę aplikacji panelu po lewej, rozwinąć menu klikając w nią prawym przyciskiem myszy i wybrać *Preferences*. Tam można się najpierw upewnić czy została dodana

Listing 1. Główna część programu serwera Web

```
#define HEADER „\
<html>\
<head>\
<meta charset=UTF-8>\
</head>\
<body bgcolor=green>\
<h1>Welcome to Led HTTP Server</h1>\
<h2>Touch one of the circles to toggle the led</h2>\
”

#define FOOTER „\
<svg width=’0’\
    height=’0’>\
          <defs>\
                    <g id=’r’>\
                              <circle cx=’32’ cy=’32’ r=’30’ fill=’red’/>\
                    </g>\
          </defs>\
          <defs>\
                    <g id=’w’>\
                              <circle cx=’32’ cy=’32’ r=’30’ fill=’white’/>\
                    </g>\
          </defs>\
</svg>\
</body>\
</html>\
”

int fdleds[8];
int ledState[8];
int a = 0;

static int ahc_ledhttpserver(void * cls, struct MHD_Connection * connection,
    const char * url, const char * method, const char * version,
    const char * upload_data, size_t * upload_data_size, void ** ptr);

static void initLeds();
static int toggleLed(int led);

int main(int argc, char ** argv)
{
    printf(„Starting...\n”);
    struct MHD_Daemon * d;
    if (argc != 2)
    {
        printf(„Usage: %s PORT\n”, argv[0]);
        return 1;
    }
    initLeds();
    d = MHD_start_daemon(MHD_USE_THREAD_PER_CONNECTION, atoi(argv[1]),
    NULL,
    NULL, &ahc_ledhttpserver,
    NULL, MHD_OPTION_END);
    if (d == NULL) return 1;
    (void) getc(stdin);
    MHD_stop_daemon(d);
    return 0;
}

static int ahc_ledhttpserver(void * cls, struct MHD_Connection * connection,
```

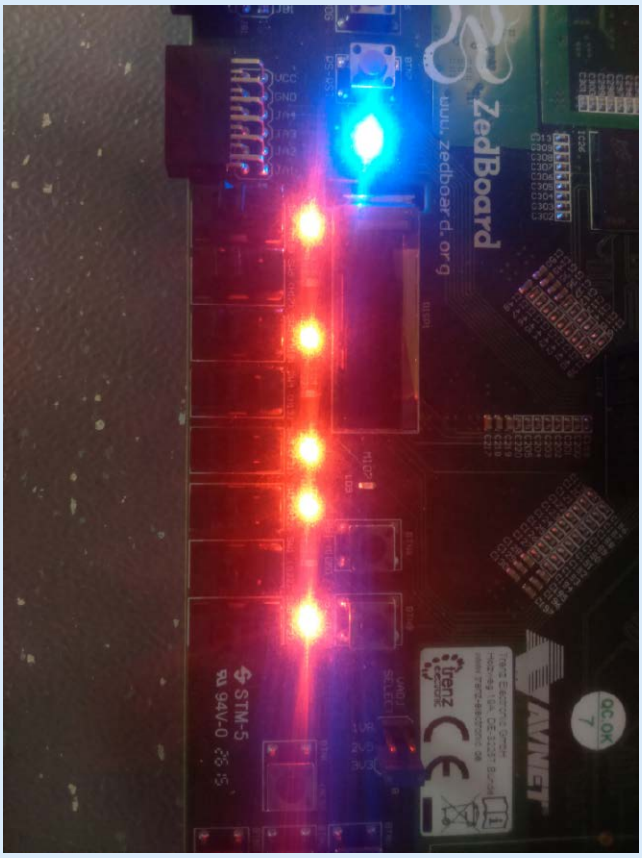


Rysunek 5. Widok strony wyświetlonej w przeglądarce

lokalizacja plików nagłówkowych (wybierając opcje kompilacji jak na **rysunku 3**), a następnie przejść do opcji linkowania bibliotek (jak na **rysunku 4**). Tam należy dodać bibliotekę przyciskiem z zielonym plusem, znajdującym się po prawej stronie na górze. W oknie, które się pojawi wpisujemy *microhttpd* (celowo pomijamy przedrostek *lib!*). Z pewnych względów trzeba jeszcze przejść do opcji *Miscellaneous* i dodać flagę linkowania o nazwie *sysroot*, której wartość to znów ścieżka do lokalizacji folderów z plikami potrzebnymi do budowania aplikacji (**rysunek 5**). Klikamy OK.

Aplikacja

Kod aplikacji znajduje się w pliku *main.c*, który jest dostępny na serwerze ftp zawierającym materiały dodatkowe do artykułów. Po pobraniu dodajemy go do projektu. Jest to przerobiona wersja przykładowego programu, który można znaleźć w dokumentacji biblioteki *libmicrohttpd* (***https://goo.gl/iSS2as***). Program należy uruchomić z parametrem PORT oznaczającym numer portu, na którym aplikacja będzie nasłuchiwać żądań. Po wpisaniu w przeglądarce adresu IP płytki i portu, powinien się pojawić ekran z szeregiem białych kółek, po jednym na każdą diodę. Jeśli



Rysunek 6. Zaświecone diody odpowiadają temu, co jest widoczne w przeglądarce

Listing 1. cd.

```
    const char * url, const char * method, const char * version,
    const char * upload_data, size_t * upload_data_size, void ** ptr)
{
    char *page;
    char header[] = HEADER;
    char footer[] = FOOTER;
    char *lines[8];
    int size = 0;
    static int dummy;
    struct MHD_Response * response;
    int ret, i;
    //Checking for various things
    if (0 != strcmp(method, „GET”)) return MHD_NO; /* unexpected method */
    if (&dummy != *ptr)
    {
        /* The first time only the headers are valid,
        do not respond in the first round... */
        *ptr = &dummy;
        return MHD_YES;
    }
    if (0 != *upload_data_size) return MHD_NO; /* upload data in a GET!? */
    *ptr = NULL; /* clear context pointer */
    //End of checking
    int toggledLed = -1;
    if(strlen(url) > 1)
    {
        if((int)url[1] >= ‚0’ && (int)url[1] <= ‚7’);
        toggledLed = url[1] - ‚0’;
        int res;
        res = toggleLed(toggledLed);
        ledState[toggledLed] = res;
    }
    char *imgStr = „<a href=/%i><svg width=\”70\” height=\”70\”><use xlink:href=\”#%s\” /></svg></a>\n”;
    for (i = 0; i < 8; i++)
    {
        lines[i] = (char*) malloc(strlen(imgStr) + 50);
        sprintf(lines[i], imgStr, i, ledState[i] == 1 ? „r” : „w”);
        size += strlen(lines[i]);
    }
    size += strlen(header);
    size += strlen(footer);
    page = (char*) malloc(size + 1);
    strcpy(page, header);
    for(i = 7 ; i >= 0 ; i--)//Inverting, because 7th led is on the left
    {
        strcat(page, lines[i]);
        free(lines[i]);
    }
    strcat(page, footer);
    response = MHD_create_response_from_buffer(strlen(page), (void*) page,
        MHD_RESPMEM_PERSISTENT);
    ret = MHD_queue_response(connection,
    MHD_HTTP_OK, response);
    MHD_destroy_response(response);
    return ret;
}
```

natomiast w ścieżce adresu wpisana zostanie liczba od 0 do 7, jedna z diod zapali się lub zgaśnie.

Główna część programu widoczna jest w **listingu 1**. Otwierają go dwie stałe preprocesora: HEADER i FOOTER. Są to odpowiednio początek i koniec kodu HTML, który zostanie wysłany w odpowiedzi. Funkcja *main* zawiera kod parsujący parametry wywołania programu, inicjalizację sterownika diod LED i funkcję *MHD_start_daemon*, która uruchamia nasłuchiwanie serwera. Zawiera ona wiele argumentów, których dokładny opis znajduje się w dokumentacji, a najważniejszy z nich to funkcja *ahc_ledhttpserver*, która będzie wywoływana w odpowiedzi na żądanie HTTP. W przypadku skomplikowanych i długich żądań potrzebne są jeszcze dodatkowe funkcje, ale w tym przypadku nie ma takiej konieczności.

Funkcja *ahc_ledhttpserver*, po wykonaniu koniecznych testów, sprawdza czy w ścieżce adresu jest liczba od 0 do 7 i jeśli tak, to gasi lub zapala odpowiednią diodę. Na koniec generowany jest kod HTML, który później wyświetli białe i czerwone kółka. Całość kodu HTML kopiowana jest do jednego stringu, na którego początek wskazuje wskaźnik *page* i odpowiedź podawana jest jako argument funkcji *MHD_create_response_from_buffer*, a następnie do funkcji *MHD_queue_response*.

Aby uruchomić aplikację, należy zaznaczyć projekt w panelu po lewej stronie, rozwinąć menu klikając go prawym przyciskiem myszy i z menu wybrać *Run as* → *Run Configurations*. Konfiguracja jest taka sama jak przy poprzednich aplikacjach, z jednym dodatkiem – trzeba dodać argument wywołania. W tym celu wybieramy kartę *Arguments* i w polu *Program Arguments* wpisujemy jakiś numer poru, np. 8080. Mając podłączoną i włączoną płytkę i uruchomiony system Linux klikamy *Run*.

Aby przetestować aplikację, otwieramy przeglądarkę i wpisujemy adres IP płytki, a po dwukropku wpisujemy numer portu. Jeśli wszystko zostało wykonane poprawnie, to przeglądarka powinna pokazać komunikat powitalny i szereg białych kółek. Po kliknięciu w kółko, odpowiadająca mu dioda powinna się zapalić. Po włączeniu kilku diod, ekran przeglądarki wygląda jak na **rysunku 5**, a odpowiadająca mu konfiguracja zaświeconych diod, jak pokazano na **rysunku 6**.

Podsumowanie

W artykule opisano sposób uruchomienia kolejnej aplikacji na platformie Zedboard. Jako że korzysta ona z zewnętrznej biblioteki współdzielonej, trzeba było dodać ją do systemu plików. Aby ją prawidłowo skompilować, trzeba było również ustawić odpowiednie opcje kompilacji i linkowania. To daje dobrą bazę do kolejnych, bardziej rozbudowanych projektów.

Stanisław Aleksiński