Konfigurowanie układów programowalnych Xilinx

przy pomocy mikrokontrolera

Zapewne większość Czytelników zajmujących się tematyką układów programowalnych używa do ich konfiguracji interfejsu JTAG pracującego pod kontrolą portu LPT lub USB komputera PC. Czasami jednak, już po uruchomieniu, umieszczeniu w obudowie czy ewentualnym oddaniu urządzenia klientowi, zachodzi potrzeba wpisania nowej konfiguracji do układu programowalnego.

Wykonanie rekonfiguracji układu programowalnego w warunkach „terenowych", czyli w miejscu, gdzie układ działa, może być utrudnione: urządzenie z układem PLD może być umieszczone w miejscu trudno dostępnym lub otwieranie obudowy urządzenia i podłączanie do niego komputera PC może być niepraktyczne (np. w deszczu lub w warunkach dużego zapylenia). Dużym ułatwieniem może być w tym przypadku zastosowanie w urządzeniu mikrokontrolera mogącego konfigurować ten układ, lub programować jego pamięć konfiguracyjną Flash. Obecnie nawet przeciętny mikrokontroler może obsługiwać rozmaite rodzaje pamięci masowych, na których można dostarczyć nową konfigurację dla układu PLD lub może być wyposażony w rozbudowane interfejsy komunikacyjne, którymi konfigurację tę można przesłać. Zastosowanie mikrokontrolera umożliwia dostarczenie danych do rekonfiguracji na różne sposoby: na karcie pamięci, przy pomocy interfejsów radiowych, lub nawet sieci GSM czy Internetu.

W artykule zaprezentowano możliwość konfiguracji układów programowalnych firmy Xilinx przy pomocy mikrokontrolera. Do artykułu dołączony jest prosty program demonstracyjny dla mikrokontrolera AT91SAM7S64 (lub -S128, -S256) wykonujący konfigurację układu programowalnego. Nową konfigurację przesyła się do mikrokontrolera przy pomocy portu szeregowego i protokołu Xmodem. Projekt ten należy postrzegać raczej jako przykład dydaktyczny niż konkretne rozwiązanie, ponieważ nie zapewnia on uniezależnienia się od zastosowania komputera PC (lecz jest bardzo prosty, co w tym przypadku jest ważniejsze).

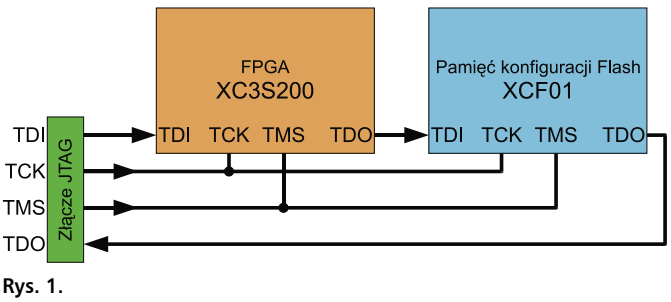
Jak to się dzieje?

Typowym sposobem konfiguracji układów PLD firmy Xilinx jest zastosowanie programu iMPACT obsługującego np. interfejs JTAG podłączany do portu USB lub LPT. Opisana niżej idea konfiguracji logiki programowalnej przy pomocy mikrokontrolera jest natomiast następująca: mikrokontroler generuje sygnały JTAG dla układu programowalnego w taki sposób jak sam program iMPACT. Aby mikrokontroler wiedział, jak ma wysterować JTAG, należy wcześniej „nagrać" sekwencje sygnałów tego portu przy pomocy programu iMPACT. „Nagrywanie" sekwencji będzie polegało na ustawieniu iMPACT-a tak, aby wszystkie czynności, które miałby wykonać na interfejsie JTAG zapisywał do pliku. Dysponując już plikiem z zarejestrowanymi sekwencjami poleceń JTAG dla układu PLD, będzie można ten plik „odtworzyć" przy pomocy mikrokontrolera. Mikrokontroler ma tylko wykonać czynności zadane w zawartości pliku.

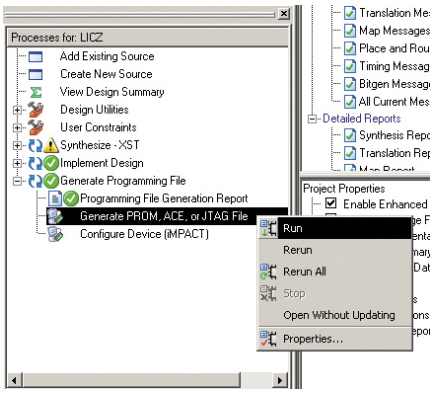
Przykładem plików zawierających sekwencję poleceń JTAG są pliki w formacie XSVF, a fragment programu wykonywany przez mikrokontroler i ustawiający port JTAG na podstawie zawartości XSVF, nazywany jest dosłownie „**odtwarzaczem XSVF**" (*XSVF player*).

Pliki XSVF to binarna odmiana plików SVF (*Serial Vector Format*) zoptymalizowana do przeprowadzania rekonfiguracji układów firmy Xilinx przy pomocy urządzeń *embedded*. Z kolei pliki SVF są plikami zawierającymi polecenia interfejsu JTAG zapisanymi w postaci tekstowej. Jak wiadomo pliki tekstowe są trudniejsze do przetwarzania niż pliki binarne, szczególnie w urządzeniach *embedded*.

W dalszej części artykułu opisano w trzech etapach tworzenie pliku XSVF. Opis dotyczy generowania plików przy pomocy pakietu Xilinx ISE 9.2i na przykładzie płytki ZL10PLD zawierającej układ XC3S200 (Spartan 3) oraz pamięć konfiguracji Flash XCF01S połączonych w łańcuch JTAG (**rys. 1**).



Rys. 1.

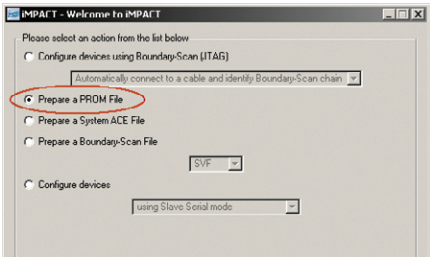


Rys. 2.

Etap I. Generowanie potrzebnych plików wynikowych

Jeśli w projekcie, w łańcuchu JTAG znajduje się tylko układ programowalny, wystarczy wykonać punkt 1 poniższej instrukcji (czyli wygenerować plik *.bit*). Pozostałe punkty dotyczą generowania pliku *.mcs* potrzebnego wtedy, gdy w łańcuchu JTAG znajduje się również pamięć Flash i gdy chcemy mieć możliwość jej zaprogramowania.

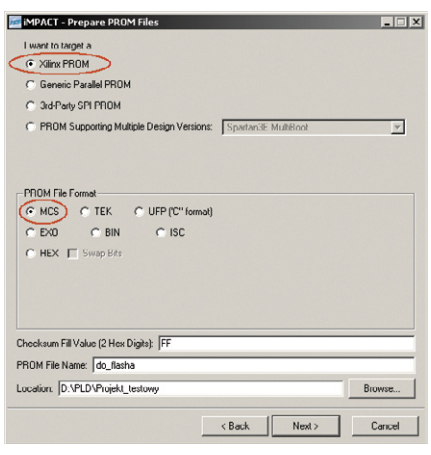
1. W Xilinx Project Manager na zakładce „Processes" klikamy prawym przyciskiem myszy „Generate Programming File" i z menu kontekstowego wybieramy opcję „Run". W tym momencie dysponujemy plikiem *.bit* zawierającym konfigurację FPGA.
2. Klikając na znak „+" obok „Generate Programming File" rozwiniemy dodatkową listę. Pojawia się m.in. element „Generate PROM, ACE, or JTAG File". Jak poprzednio: klikamy na



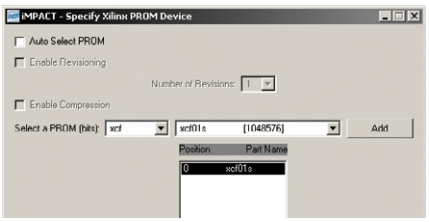
Rys. 3.

nim prawym przyciskiem myszy i wybieramy „Run" (**rys. 2**).

3. W okienku, które się pojawia (**rys. 3**) wybieramy opcję „Prepare a PROM File" i wciskamy „Next".
4. Teraz pojawi się kolejne okno iMPACT-a (**rys. 4**). W obszarze „I want to target a" wybieramy „Xilinx PROM", a w obszarze „PROM File Format" wybieramy „MCS". W polu tekstowym „PROM File Name" wpisujemy dowolną nazwę pliku, który chcemy wygenerować. Pole „Checksum Fill Value" można pozostawić z wartościami domyślnymi. Klikamy „Next".
5. Następne okno (**rys. 5**) pozwala na wybór układu Flash, którym dysponujemy. W przypadku płytki ZL10PLD będzie to XCF01S – nazwę wybieramy obok tekstu „Select a PROM". Wciskamy przycisk „Add" i wybrana kość Flash pojawia się na liście. Zatwierdzamy klikając „Next". W następnym okienku ujrzymy podsumowanie. Jeśli wszystko się zgadza, można wcisnąć przycisk „Finish".
6. W tym momencie powinien pojawić się komunikat informacyjny „Add Device", który potwierdzamy przyciskiem „OK". W okienku wyboru pliku konfiguracyjnego wskazujemy plik, który użylibyśmy do konfiguracji układu FPGA (będzie to najprawdopodobniej plik z rozszerzeniem *.bit*).
7. Gdy pojawi się okno z pytaniem, czy chcemy dołożyć kolejne pliki konfiguracyjne („Would you like to add another device file...") wciskamy „No". Okno iMPACT-a powinno wyglądać podobnie jak na **rys. 6**.
8. W okienku potwierdzającym zakończenie dodawania plików z komunikatem „You have



Rys. 4.



Rys. 5.

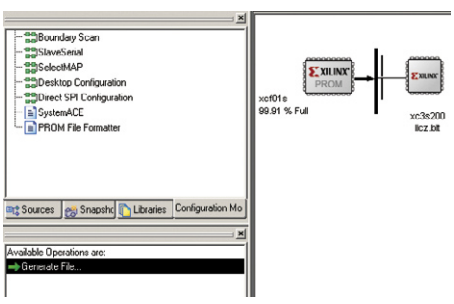
completed the device file entry" wciskamy „OK".

9. W lewej części okna uruchamiamy podwójnym kliknięciem widoczną na rys. 6 opcję „Generate File...". Po chwili powinien pojawić się komunikat „PROM File Generation Succeeded".

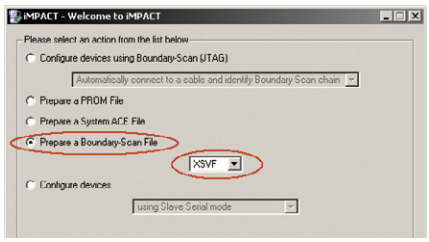
Teraz dysponujemy plikiem *.bit* mogącym skonfigurować układ FPGA oraz plikiem *.mcs*, który może posłużyć do zaprogramowania pamięci Flash na płytce ZL10PLD. Można więc przystąpić do wygenerowania pliku XSVF, czyli docelowego pliku, który będzie przetwarzał mikrokontroler.

Etap II. Przygotowanie łańcucha JTAG i pliku XSVF w programie iMPACT

1. Uruchamiamy program iMPACT. Możemy to uczynić albo z poziomu projektu klikając na „Generate Programming File", albo uruchamiając go jako osobne narzędzie.
2. Aby wygenerować potrzebny plik XSVF w okienku wizarda z opisem „Please select an action from the list below" wybieramy opcję „Prepare a Boundary-Scan File", a następnie, z listy rozwijanej, format pliku „XSVF" (**rys. 7**) i zatwierdzamy wybór przyciskiem „Finish".
3. Pojawi się okno zapisu do pliku XSVF „Create a New XSVF File". Wpisujemy w nim nazwę pliku i wciskamy przycisk „Zapisz".
4. Wciskamy przycisk „OK" w okienku informującym, że wszystkie wykonane operacje będą przekierowywane do pliku XSVF.
5. Teraz powinno być widoczne okno wyboru pliku „Add Device". Z jego pomocą dodamy nowy układ do łańcucha JTAG (iMPACT nie posługuje się teraz żadnym programatorem, więc nie może automatycznie wykryć układów w łańcuchu). Dodamy urządzenia w taki sposób, jak są umieszczone na płytce ZL10PLD, czyli najpierw układ XC3S200, a za



Rys. 6.



Rys. 7.

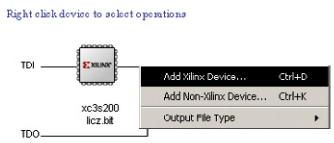
nim konfigurator Flash XCF01A (rys. 1). Zaczynamy od XC3S200, więc z okienka „Add Device" wybierzemy wygenerowany wcześniej plik .bit. Program iMPACT rozpozna, że chodzi nam o dodanie układu XC3S200.

6. Może pojawić się ostrzeżenie dotyczące niepoprawnie wybranego źródła sygnału zegarowego przy starcie FPGA, lecz możemy je zignorować (iMPACT poprawi to ustawienie za nas, więcej szczegółów na ten temat znajdziemy w artykule [4]).
7. Jeśli posługujemy się płytką ZL10PLD, dodajemy drugi element łańcucha JTAG: pamięć Flash XCF01A. Czynimy to klikając prawym przyciskiem myszy za układem XC3S200 (na prawo od niego). Z menu kontekstowego, które się pojawi, wybieramy „Add Xilinx Device..." (**rys. 8**). W oknie wyboru pliku znajdujemy wygenerowany w wcześniej plik *.mcs* z zawartością pamięci Flash.
8. W oknie „Select Device Part Name" (**rys. 9**) wybieramy taki typ pamięci jakim dysponujemy, czyli (w przypadku płytki ZL10PLD) XCF01S, a następnie zatwierdzamy wybór przyciskiem „OK".

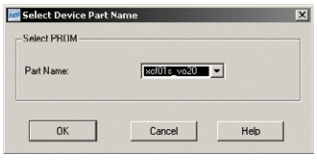
Etap III. „Nagrywanie" operacji JTAG

W tym momencie dysponujemy zdefiniowanym łańcuchem JTAG. Od teraz, każda wykonana operacja (konfiguracja FPGA, kasowanie czy programowanie Flasha) będzie zarejestrowana w pliku XSVF na poziomie poleceń interfejsu JTAG. Innymi słowy, polecenia te będą zapisywane do pliku XSVF, a nie wykonywane przez JTAG. Dotyczy to także poleceń weryfikacji.

Nie musimy wykonywać jednocześnie konfiguracji FPGA i programowania Flasha. Mikrokontroler odtwarzający plik XSVF może zajmować się wyłącznie konfiguracją FPGA lub wyłącznie



Rys. 8.



Rys. 9.

skasować i ponownie zaprogramować nową zawartością pamięć konfiguracyjną Flash.

Dostępne operacje mamy wypisane z lewej strony okna ISE („Available Operations are:"). Teraz programem iMPACT możemy posługiwać się dokładnie tak samo jakbyśmy dysponowali płytką ZL10PLD podłączoną do komputera PC przez JTAG.

Jeśli chcemy zaprogramować układ FPGA na wirtualnej płytce ZL10PLD, kliknięciem uaktywniamy układ XC3S200, a następnie z „Available Operations are:" wybieramy „Program". W oknie „Programming Properties" wybieramy typowe ustawienia i potwierdzamy przyciskiem „OK". Natomiast programując pamięć Flash XCF01 w oknie „Programming Properties" warto pamiętać o zaznaczeniu opcji „Verify" i „Erase Before Programming". Opis, jak wykonywać konfigurację FPGA i programowanie pamięci konfiguracyjnej Flash znajduje się w [4].

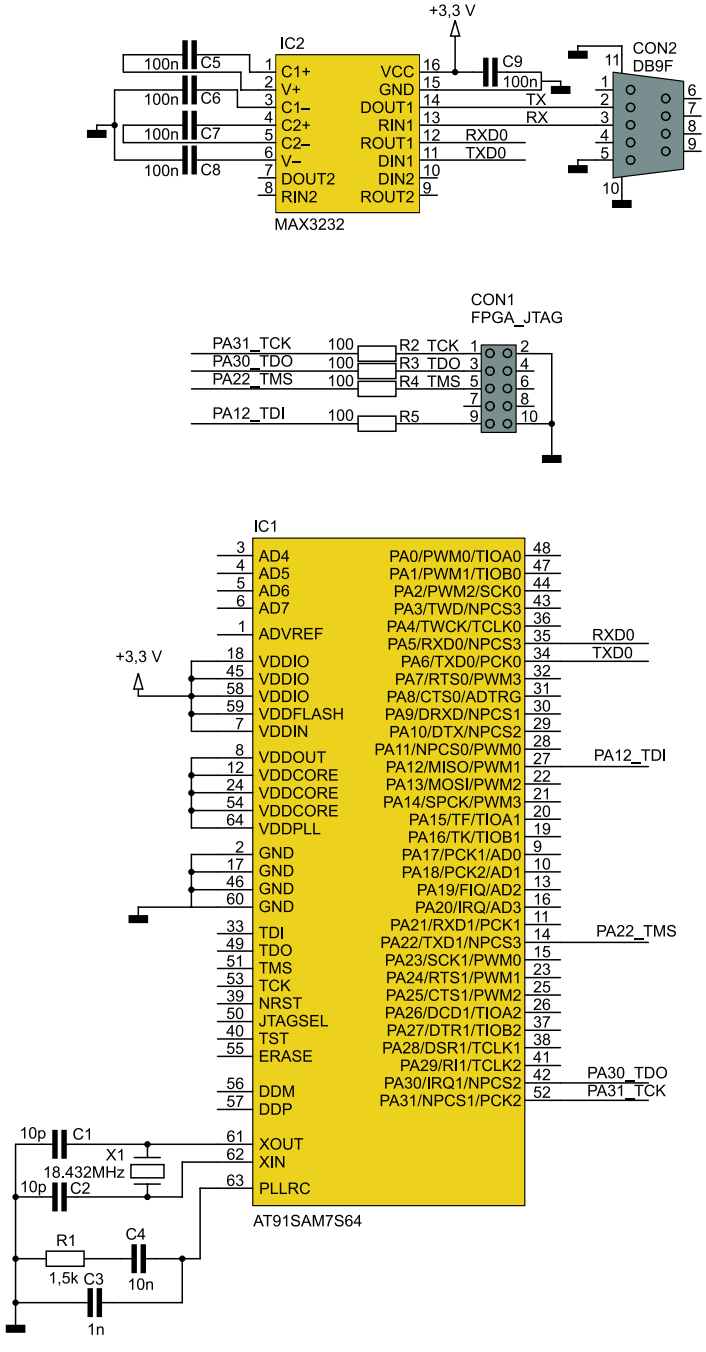
Po zakończeniu wirtualnego programowania (czyli rejestracji akcji do pliku XSVF) klikamy prawym przyciskiem myszy na obszarze okna wyświetlającego zawartość łańcucha JTAG i z menu kontekstowego wybieramy kolejno opcje „Output File Type", „XSVF File" i „Stop Writing to File" dzięki czemu zamykamy plik XSVF.

Dysponujemy już gotowym plikiem XSVF dla mikrokontrolera. Zupełnie innym zagadnieniem jest jak wysłać i zachować ten plik w urządzeniu z mikrokontrolerem.

Przesyłanie i przechowywanie pliku XSVF w urządzeniu

Pliki XSVF mogą mieć dość dużą objętość jak na warunki urządzeń *embedded* – nawet rzędu kilkuset kilobajtów. Przechowywanie pliku XSVF można zrealizować np. przy pomocy pamięci DataFlash (oznaczenia AT45DB...) lub na karcie pamięci (np. SD lub MMC) dołączonej do mikrokontrolera. Dysponując taką pamięcią w urządzeniu, nie musimy stosować pamięci dedykowanych do przechowywania konfiguracji, takich jak np. XCF01. Rozwiązanie to będzie opłacalne ekonomicznie w warunkach, gdy w projektowanym urządzeniu przewidujemy zarówno umieszczenie mikrokontrolera, jak i układu FPGA.

Najprostszym rozwiązaniem na przechowywanie pliku XSVF w pamięci nieulotnej dołączonej do mikrokontrolera jest zapisanie go do tej pamięci „bajt po bajcie", bez zastosowania systemu plików. W takim przypadku przesyłanie pliku można zaimplementować z zastosowaniem łącza szeregowego, a same pliki wysyłać np. programem terminalowym. Tutaj jednak czyha pewna pułapka: programy typu „terminal" mogą źle radzić sobie ze strumieniowym przesyłaniem dużych plików (gdy wysyłane są tylko kolejne bajty pliku nieopatrzone dodatkowymi informacjami, takimi jak sumy kontrolne czy liczniki bajtów). Okazało się to prawdą w przypadku programów HyperTerminal i Bray Terminal. Dodatkowo, w czasie przesyłania sektek kilobajtów mogą wystąpić błędy transmisji. Dlatego w programie przykładowym, do przesyłania pliku XSVF przez łącze szeregowe zaimplementowano protokół Xmodem wysyłający plik pakietami z licznikami i sumami kontrolnymi.



Rys. 10.

Inną metodą przesyłania i przechowywania pliku XSVF jest implementacja urządzenia klasy magazynującej USB. Przykładowy opis takiego urządzenia znajduje się w [6]. Stosując urządzenie magazynujące należy mieć na uwadze, że dane wysłane do karty pamięci przechowującej plik XSVF, będą zapisywane w określonym systemie plików (najczęściej w jednej z odmian systemu FAT). Dlatego stosując to rozwiązanie, należy w urządzeniu dodatkowo zaimplementować obsługę tego systemu plików. Wszystko to może brzmieć dość „groźnie", lecz rozwiązanie z klasą magazynującą i systemem plików może być stosunkowo proste i wydajne, jeśli dysponujemy odpowiednimi bibliotekami: obsługę klasy magazynującej można „zdobyć" z materiałów [6], a biblioteki obsługi FAT, przy pomocy mikrokontrolera, dostępne są powszechnie w Internecie.

Wybór mikrokontrolera

Firma Xilinx dostarcza specjalną bibliotekę do odtwarzania plików XSVF przy pomocy mikrokontrolera. Jest ona dostępna za darmo na stronie [1]. Biblioteka ta składa się z trzech modułów napisanych w języku C, więc skompilowanie jej i uruchomienie na różnych mikrokontrolerach nie powinno sprawić problemu. Problemem natomiast może być zapotrzebowanie odtwarzacza na pamięć RAM. W zależności od tego, ile urządzeń znajduje się w łańcuchu JTAG oraz

jakie są to urządzenia, ilość pamięci RAM wymagana przez bibliotekę może być zmienna.

Pierwotnie rozważano zastosowanie AVR. Niestety jedynym popularnym mikrokontrolerem AVR z wystarczającą ilością pamięci RAM jest ATmega128. W takim wypadku, wbrew pozorom, nieco tańszym i prostszym rozwiązaniem może być zastosowanie niewielkiego mikrokontrolera ARM, np. AT91SAM7S64 (16 KB RAM).

Program przykładowy został opracowany przy użyciu pakietu WinARM, w którego skład wchodzi wersja kompilatora GCC dla mikrokontrolerów z rdzeniem ARM.

Układ testowy

Przykładowy, minimalny schemat elektryczny układu, na którym dostarczony program demonstracyjny zadziała bez modyfikacji przedstawiono na **rys. 10**. Dla przejrzystości nie zaznaczono na nim kondensatorów odsprzęgających linie zasilania mikrokontrolera: 3,3 V i 1,8 V.

Działanie odtwarzacza XSVF można bez większych trudności wypróbować na płytce testowej z dowolnym mikrokontrolerem rodziny SAM7. Jeśli zastosujemy płytkę ewaluacyjną z mikrokontrolerem AT91SAM7S64, -S128 lub -S256 i wyprowadzimy sygnały JTAG z mikrokontrolera tak jak na schemacie z rys. 10, dostarczony program nie będzie wymagał żadnych zmian. W przypadku innych mikrokontrolerów z rodziny SAM7 lub innego podłączenia sygnałów JTAG do mikrokontrolera, niezbędne będą drobne modyfikacje kodu, polegające głównie na zmianie kilku makrodefinicji używanych wyprowadzeń.

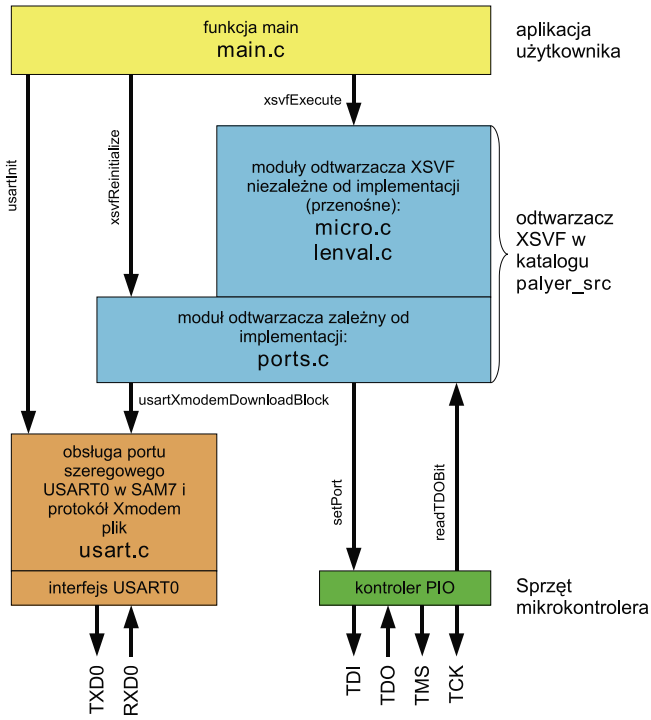
Wszystkie elementy znajdujące się wokół mikrokontrolera są typowe: dotyczy to rezonatora kwarcowego, układu filtru PLL oraz podłączenia zasilania. Elementy o takich wartościach stosowane są na większości (jeśli nie na wszystkich) dostępnych płytkach ewaluacyjnych dla SAM7.

Sygnały JTAG dla układów programowalnych Xilinx generowane są za pomocą kontrolera PIO (*Parallel Input Output*), czyli jako uniwersalne wejścia-wyjścia cyfrowe. Na złączu CON1 sygnały JTAG są ułożone w typowy sposób i pasują np. do bezpośredniego podłączenia modułu ZL10PLD. Rezystory R2...R5 przede wszystkim zabezpieczają przed skutkami różnic w napięciach zasilania mikrokontrolera i układu PLD.

Przesyłanie pliku XSVF do mikrokontrolera, jak zostało już wspomniane, odbywa się przy użyciu asynchronicznego portu szeregowego wbudowanego w mikrokontroler. Do komunikacji wybrno port szeregowy USART0 (*Universal Synchronous-Asynchronous Receiver Transmitter*). Wykorzystywane są tylko dwie linie: RXD0 (odbiór danych) i TXD0 (wysyłanie danych). Na schemacie ideowym z rys. 10 zaznaczono przykładowe podłączenie translatora poziomów logicznych IC2 (MAX3232 lub podobny) i żeńskiego złącza portu szeregowego CON2 (typowe DB-9).

Zastosowanie biblioteki odtwarzacza

Uproszczony schemat przepływu informacji w projekcie demonstracyjnym odtwarzacza został przedstawiony na **rys. 11**. Można przyjąć, że cała jego zawartość „merytoryczna" (realizacja algorytmu) jest zaimplementowana w modułach *micro.c* i *lenval.c*, natomiast moduł *ports.c* służy do dostosowania biblioteki do własnych potrzeb.



Rys. 11.

Od strony programu wykonywanego przez mikrokontroler, konfiguracja układu programowalnego sprowadza się do wywołania funkcji ***xsvfExecute***, której ciało znajduje się w module *micro.c* biblioteki. Funkcja *xsvfExecute* zajmie się odczytem odpowiednich porcji danych z pamięci przechowującej zawartość pliku XSVF oraz wysterowaniem i odczytem wyprowadzeń JTAG. Po zakończonej konfiguracji PLD lub programowaniu pamięci Flash, funkcja zwróci liczbę informującą o przebiegu tego procesu. W pliku *\player_src\micro.h*, projektu przykładowego, znajdują się makrodefinicje odpowiadające wartościom zwracanym przez funkcję *xsvfExecute*. W praktyce najczęściej spotykanymi wartościami są:

- XSVF_ERROR_NONE (0) – gdy wykonywanie pliku XSVF zakończy się pomyślnie,
- XSVF_ERROR_TDOMISMATCH (2) – wystąpi zazwyczaj, gdy nie podłączono interfejsu JTAG do układu programowalnego,
- XSVF_ERROR_DATAOVERFLOW (6) – wystąpi wtedy, gdy nie zezwolono bibliotece odtwarzacza na alokację wystarczającej ilości pamięci operacyjnej.

Odtwarzacz, pomimo iż jego kod jest łatwo przenośny, musi zostać dostosowany do konkretnego mikrokontrolera i kompilatora. Programista powinien zatroszczyć się o napisanie funkcji realizujących opóźnienia, odczyt danych i kontrolę wyprowadzeń portu JTAG. Odtwarzacz w czasie wykonywania pliku XSVF będzie wywoływał te funkcje – dzięki nim może nawiązać komunikację z układem programowalnym i z pamięcią przechowującą zawartość pliku.

Funkcje wywoływane przez odtwarzacz to:
void setPort(short p,short val) W zależności od parametru `p` ustawia linię TCK (gdy `p=0`), TMS (gdy `p=1`) lub TDI (gdy `p=2`) na stan logiczny „1" (gdy `val≠0`) lub na stan logiczny „0" (gdy `val=0`). Zawartość funkcji *setPort* przedstawiono na **list. 1**. Zmienna `once` w tej funk-

List. 1.

```
void  setPort(short  p,short  val)
{
        static  uint8_t  once;
        if  (once  ==  0)
        {
                once  =  1;
                JTAG_PIO->PIO_PER  =  JTAG_OUTPUT_MASK  |  JTAG_INPUT_MASK;
                JTAG_PIO->PIO_OER  =  JTAG_OUTPUT_MASK;
        }
        if  (p==TMS)
        {
                if(val)  TMS_1;  else  TMS_0;
        }
        if  (p==TDI)
        {
                if(val)  TDI_1;  else  TDI_0;
        }
        if  (p==TCK)
        {
                if(val)  TCK_1;  else  TCK_0;
        }
        asm  volatile(„nop"::);
        asm  volatile(„nop"::);
        asm  volatile(„nop"::);
        asm  volatile(„nop"::);
}
```

List. 2.
```
void  readByte(unsigned  char  *data)
{
        *data  =  dataBlock.payload[counter];

        counter++;

        if(counter  >=  USART_XMODEM_PAYLOAD_LENGTH)
        {
                counter  =  0;
                while(usartXmodemDownloadBlock((TUsartXmodemBlock*)&dataBlock)
!=OK);
        }

}
```

List. 3.
```
#define  USART_XMODEM_PAYLOAD_LENGTH       128

typedef  struct  __attribute__((packed))  {
        uint8_t  startByte;
        uint8_t  packetNumber;
        uint8_t  notPacketNumber;
        uint8_t  payload[USART_XMODEM_PAYLOAD_LENGTH];
        uint16_t  checksum;
}TUsartXmodemBlock;
```

List. 4.
```
typedef  struct  var_len_byte
{
        short  len;      /*  number  of  chars  in  this  value  */
        unsigned  char  val[MAX_LEN+1];    /*  bytes  of  data  */
}  lenVal;
```

cji pozwala wykonać inicjalizację wyprowadzeń mikrokontrolera na początku jej pierwszego wywołania. W mikrokontrolerze SAM7 włączany jest wtedy kontroler PIO na wszystkich wyprowadzeniach odpowiadających sygnałom JTAG oraz ustawiane są w tryb wyjściowy wyprowadzenia: TDI, TCK i TMS (TDO jest wejściem więc nie ma potrzeby jego konfiguracji).

unsigned char readTDOBit() Funkcja ma zwracać stan logiczny linii TDO (stan na końcu łańcucha JTAG).

void waitTime(long microsec) Jest to funkcja realizująca opóźnienie. Według dokumentacji firmy Xilinx, powinna ona realizować opóźnienie trwające co najmniej *microsec* mikrosekund. W przypadku niektórych układów programowalnych (np. Virtex II) w czasie trwania opóźnienia stan linii TCK portu JTAG musi się ciągle zmieniać. Funkcjonalność ta została zaimplementowana w projekcie demonstracyjnym odtwarzacza, lecz program nie był testowany z układami Virtex II. Więcej szczegółów na ten temat można znaleźć m.in. w pliku *readme.txt* w archiwum dostępnym na stronie [1].

void readByte(unsigned char *data)

Ta funkcja ma wpisać do miejsca wskazywanego przez *data* kolejny bajt odczytany z pliku XSVF. Zauważmy, że przez takie podejście biblioteka odtwarzacza uzyskuje do zawartości XSVF dostęp sekwencyjny. Z punktu widzenia programisty jest to dość wygodna cecha, ponieważ umożliwia zastosowanie szeregowego przesyłania danych bez potrzeby zapewniania dostępu do dowolnego bajtu z pliku XSVF. Na **list. 2** przedstawiono zawartość funkcji *readByte*. Dane są odczytywane z tablicy *payload* będącej elementem struktury danych *dataBlock*. Gdy licznik odebranych bajtów *counter* przekroczy maksymalny indeks tablicy *payload*, z portu szeregowego pobierany jest następny blok danych przez wywołanie funkcji *usartXmodemDownloadBlock*.

Ponieważ dane są pobierane przy pomocy protokołu Xmodem, komputer PC nie wyśle nowego bloku danych, jeśli nie potwierdzimy odebrania poprzedniego bloku. W omawianej funkcji program oczekuje w pętli *while* na odebranie nowego bloku danych przez interfejs USART.

Struktura *dataBlock* jest typu *TUsartXmodemBlock*, a definicja tego typu (**list. 3**) znajduje się w pliku *usart.h*. Typ *TUsartXmodemBlock* opisuje ciąg danych protokołu Xmodem nadchodzący z programu terminalowego. Warto zwrócić uwagę, że w definicji struktury danych typu *TUsartXmodemBlock* zastosowano atrybut *__attribute__((packed))*, dzięki któremu elementy struktury, niezależnie od ich rozmiaru, ułożone są w pamięci ściśle „jeden po drugim" (co w przypadku 32-bitowych mikrokontrolerów ARM7TDMI wcale nie jest oczywiste).

Implementacja protokołu Xmodem jest specyficzna dla omawianego tutaj projektu przykładowego. W swoim projekcie Czytelnik zapewne będzie musiał wpisać zupełnie inny kod odczytujący kolejny bajt pliku XSVF. Np. może to być odczyt kolejnego bajtu z pliku wykonany przy pomocy biblioteki obsługi systemu plików lub odczytanie kolejnego bajtu z pamięci DataFlash.

Chcąc dostosować odtwarzacz do swojego mikrokontrolera, oprócz napisania powyżej przedstawionych funkcji interfejsowych, należy ustawić długości buforów przechowujących dane dla rejestrów przesuwnych interfejsu JTAG. Aby to uczynić, bynajmniej nie trzeba wnikać w działanie interfejsu JTAG – wystarczy ustawienie jednej makrodefinicji o nazwie **MAX_LEN** w pliku *lenval.h*.

W projekcie testowym makrodefinicja MAX_LEN oznacza wartość 600. Jest to wartość niewiele większa od minimalnej potrzebnej do zaprogramowania pamięci konfiguracyjnej Flash XCF01S na płytce ZL10PLD. Jeśli planujemy programowanie samego układu FPGA (nawet, gdy w łańcuchu JTAG oprócz niego znajduje się pamięć XCF01S), odtwarzacz zadziała z MAX_LEN ustawionym na 128. Na początku pliku *lenval.h* znajduje się opis, jak obliczać wartość MAX_LEN, oraz kilka przykładów ustawień MAX_LEN dla różnych układów firmy Xilinx. Najprostszym sposobem wyznaczania MAX_LEN jest eksperyment – jeśli wartość MAX_LEN będzie za mała, przed zakończeniem „odtwarzania", funkcja *xsvfExecute* zwróci wartość 6, czyli XSVF_ERROR_DATAOVERFLOW.

List. 5.
```
typedef  struct  tagSXsvfInfo
{
        /*  XSVF  status  information  */
        unsigned  char       ucComplete;
        unsigned  char       ucCommand;
        long                          lCommandCount;
        int                            iErrorCode;

        /*  TAP  state/sequencing  information  */
        unsigned  char       ucTapState;
        unsigned  char       ucEndIR;
        unsigned  char       ucEndDR;

        /*  RUNTEST  information  */
        unsigned  char       ucMaxRepeat;
        long                          lRunTestTime;

        /*  Shift  Data  Info  and  Buffers  */
        long                           lShiftLengthBits;
        short                         sShiftLengthBytes;

        lenVal                       lvTdi;
        lenVal                       lvTdoExpected;
        lenVal                       lvTdoCaptured;
        lenVal                       lvTdoMask;

#ifdef    XSVF_SUPPORT_COMPRESSION
        /*  XSDRINC  Data  Buffers  */
        lenVal                       lvAddressMask;
        lenVal                       lvDataMask;
        lenVal                       lvNextData;
#endif    /*  XSVF_SUPPORT_COMPRESSION  */
}  SXsvfInfo;
```

Nie wyjaśnione pozostało jeszcze zagadnienie, jak się ma wartość `MAX_LEN` do zapotrzebowania odtwarzacza na pamięć RAM. Makrodefinicja `MAX_LEN` używana jest w definicji struktury danych `lenVal` (**list. 4**) w pliku *lenval.h*. Widzimy, że każda struktura `lenVal` zajmuje `MAX_LEN`+3 bajty. Z kolei struktury `lenVal` wchodzą w skład struktury danych `SXsvfInfo` (**list. 5**), której definicja znajduje się w pliku *micro.c*. Struktura `SXsvfInfo` służy odtwarzaczowi do przechowywania parametrów, statusu i danych w czasie wykonywania pliku XSVF. Z kolei w skład struktury `SXsvfInfo` wchodzą 4 struktury `lenVal`, czyli one same zajmują 4*(`MAX_LEN`+3) bajtów. Liczymy 4 struktury `lenVal` w `SXsvfInfo` dlatego, że pozostałe 3 z nich nie biorą udziału w kompilacji (zastosowano polecenie kompilacji warunkowej `#ifdef XSVF_SUPPORT_COMPRESSION`). Makrodefinicja w `XSVF_SUPPORT_COMPRESSION` w programie przykładowym została wyliczona, ponieważ nie używamy skompresowanych plików XSVF. Jeśli zatem przy wyłączonej kompresji ustawimy wartość `LEN_VAL` równa 1000, to struktura `SXsvfInfo` (umieszczana na stosie) zajmie ponad 4 KB, a z „odbezpieczoną" kompresją ponad 7 KB pamięci RAM mikrokontrolera.

Działanie przykładowego programu

W funkcji *main* (plik *main.c*) najpierw zostaje wywołana procedura inicjalizacji podstawowych układów peryferyjnych mikrokontrolera, tj. włączenie taktowania kontrolera PIO, uaktywnienie wejścia sygnału *reset* oraz wywołanie funkcji inicjalizującej interfejs USART (*usartInit*).

Po wykonaniu inicjalizacji, mikrokontroler wykonuje nieskończoną pętlę, w której wywołuje na przemian funkcję reinicjalizującą odtwarzacz XSVF (jest to tylko funkcja pomocnicza, nie wchodząca w skład biblioteki odtwarzacza) oraz samą funkcję odtwarzacza: *xsvfExecute*.

Po uruchomieniu odtwarzacza, mikrokontroler wstrzymuje jego działanie w funkcji *xsvfReinitialize* do czasu, gdy nadejdzie pierwszy pakiet danych z portu szeregowego (parametry transmisji to 115200 baud, 8 bitów danych, 1 bit stopu, brak bitu parzystości i brak kontroli przepływu danych). Podczas oczekiwania na plik XSVF, co pół sekundy przez port szeregowy wysyłany jest znak C. Znak ten dla programu terminalowego oznacza, że urządzenie jest gotowe do przyjęcia pliku wysłanego nową (tj. z 16-bitowymi sumami kontrolnymi CRC) odmianą protokołu Xmodem.

Aby wysłać plik XSVF w programie HyperTerminal wystarczy z menu wybrać polecenie „Transfer", a następnie „Wyślij plik". Pojawi się okienko „Wysyłanie pliku". Wybieramy w nim plik, który chcemy wysłać oraz protokół, jakiego zamierzamy używać – dla programu demonstracyjnego wybierzemy „Xmodem". Gdy wszystko ustawimy, wystarczy wcisnąć przycisk „Wyślij". Pojawi się okno wyświetlające informacje o postępie wysyłania pliku i szybkości transmisji.

W czasie wysyłania pliku działa biblioteka odtwarzacza XSVF. Co 128 wywołań funkcji *readByte* (w pliku *ports.c*) wywoływana jest funkcja odczytująca kolejny pakiet przy pomocy protokołu Xmodem (w jednym pakiecie Xmodem mieści się 128 bajtów).

Komputer PC nie wyśle do urządzenia nowego pakietu, jeśli poprzedni pakiet nie zostanie potwierdzony. Może się zdarzyć (np. przy programowaniu pamięci Flash XCF01), że odtwarzacz będzie realizował nawet kilkusekundowe opóźnienie, więc przez ten czas nie będzie pobierał nowych danych. Ważne jest, aby program terminalowy nie zgłosił wtedy np. timeoutu (czyli nie uznał, że urządzenie nie odpowiada). Problem ten przy programowaniu pamięci XCF01 nie występuje dla programu HyperTerminal.

Poprawnie przeprowadzona konfiguracja układu XC3S200 na płytce ZL10PLD jest sygnalizowana włączeniem się diody LED oznaczonej jako D1 (RDY).

Więcej informacji

Przedstawiona implementacja odtwarzacza jest tylko najprostszym przykładem możliwości programowania układów PLD przy pomocy mikrokontrolera.

Oryginalny kod źródłowy biblioteki odtwarzacza plików XSVF można znaleźć na stronie [1]. W archiwum pod tym adresem znajdują się także narzędzia do konwersji plików SVF i XSVF oraz dodatkowe informacje na temat zastosowania odtwarzacza do konfiguracji różnych układów programowalnych. Na stronie [2] znajduje się nota aplikacyjna dotycząca programowania układów PLD firmy Xilinx przy pomocy mikrokontrolera.

Jeśli Czytelnik jest zainteresowany działaniem portu JTAG, polecam rozdział 5 książki [5]. Z kolei Czytelnicy zainteresowani samym formatem XSVF mogą rozszerzyć swą wiedzę zapoznając się z dokumentem [3].

Robert Brzoza-Woch
rabw@poczta.fm

Literatura

[1] Źródła odtwarzacza: ftp.xilinx.com/pub/swhelp/cpld/eisp_pc.zip
[2] XAPP058, adres: www.xilinx.com/support/documentation/application_notes/xapp058.pdf
[3] XAPP503, adres: www.xilinx.com/support/documentation/application_notes/xapp503.pdf
[4] Jacek Majewski, Piotr Zbysiński, Układy FPGA w przykładach, część 7, Elektronika Praktyczna 4/2007
[5] Piotr Zbysiński, Jerzy Pasierbiński, Układy programowalne, pierwsze kroki, wydanie II, BTC, Warszawa 2004
[6] Robert Brzoza-Woch, Czytnik kart SD, Elektronika Praktyczna 5-6/2008