Ernest JAMRO, Maciej WIELGOSZ, Witold CIOCH, Sławomir BIENIASZ
AKADEMIA GÓRNICZO-HUTNICZA, AKADEMICKIE CENTRUM KOMPUTEROWE CYFRONET,
ul. Nawojki 11, 30-950 Kraków

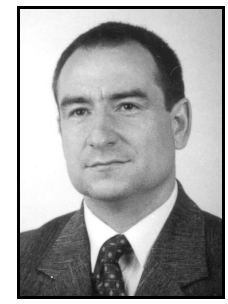
Efektywna komunikacja ARM-FPGA z użyciem interfejsu SPI

Dr inż. Ernest JAMRO

Ukończył studia na AGH na kierunku Elektronika oraz na University of Huddersfield (UK) na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorską w 2001 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Elektroniki na AGH. Jego zainteresowania naukowe to sprzętowa akceleracja obliczeń, niskopoziomowe przetwarzanie obrazów, sieci neuronowe.

e-mail: jamro@agh.edu.pl

Dr inż. Witold CIOCH



Ukończył studia na Wydziale Budowy Maszyn i Lotnictwa PRz, specjalność napędy lotnicze oraz na Wydziale Inżynierii Mechanicznej i Robotyki AGH, specjalność wibroakustyka. Pracę doktorską obronił w 2004. Obecnie pracuje jako adiunkt w Katedrze Mechaniki i Wibroakustyki AGH. Jego zainteresowania naukowe dotyczą diagnostyki techniczne i wibroakustyki. Zajmuje się zagadnieniami cyfrowego przetwarzania sygnałów, sieciami neuronowymi i analizą ryzyka eksploatacyjnego.

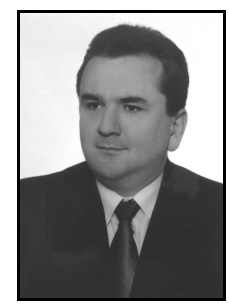
e-mail: cioch@agh.edu.pl

Dr inż. Maciej WIELGOSZ

Ukończył studia na AGH (2005), wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki na kierunku Elektronika i Telekomunikacja. Obecnie jest doktorantem w Katedrze Elektroniki AGH i bierze czynny udział w pracach badawczych realizowanych w zespole rekonfigurowalnych systemów obliczeniowych. Jego zainteresowania naukowe dotyczą sprzętowej akceleracji obliczeń, kompresji obrazu i sieci neuronowych.

e-mail: wielgosz@agh.edu.pl

Dr inż. Sławomir BIENIASZ



Ukończył studia na kierunku Informatyka na AGH. Pracę doktorską obronił w 2006 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Informatyki na AGH. Jego zainteresowania naukowe to agentowe modele symulacji zjawisk fizycznych, programowanie aspektowe, programowanie w środowisku systemu Unix (Linux).

e-mail: bieniasz@agh.edu.pl

Streszczenie

W systemach wbudowanych użycie niezależnego procesora ARM oraz układu FPGA umożliwia uzyskanie dużo większej elastyczności projektowania oraz lepszej wydajności niż w przypadku systemów homogenicznych (opartych na tylko jednej platformie). Wadą takiego rozwiązania jest konieczność zapewnienia wydajnej, szybkiej komunikacji, która w omawianym przypadku została zrealizowana poprzez interfejs SPI. Aby uzyskać większą przepustowość danych zaprojektowano dedykowany moduł sprzętowy wewnątrz układu FPGA obsługujący interfejs SPI, pracujący jako urządzenie typu slave po stronie interfejsu SPI oraz master na magistrali PLB (Processor Local Bus).

Słowa kluczowe: systemy wbudowane, SPI, FPGA, Xilinx EDK.

Efficient ARM-FPGA data transfer employing SPI interface

Abstract

Implementation of fast and reliable data transfer between an FPGA and a processor is a significant challenge for a designer of heterogeneous embedded systems. In the presented system two separate Printed Circuit Boards (PCB) are employed: ARM-based OMAP3530 [4] and FPGA Spartan3 [2]. SPI (Serial Peripheral Interface) [5] is used as a communication interface due to the OMAP3530 limitations in communication interface choice. For the FPGA module, Xilinx Embeded Development Kit (EDK) and soft-processor MicroBlaze are used. The EDK delivers SPI hardware module [9] compatible with the Processor Local Bus (PLB). Nevertheless, this module employs slave interface on the PLB therefore requires the soft-processor MicroBlaze interaction which limits the transfer speed. Consequently, a dedicated hardware module compatible with the PLB and EDK was designed. This module employs master interface on the PLB bus and slave interface on the SPI interface and is further denoted as the xps_spi_master. As a result, the MicroBlaze is not engaged in the data transfer and, therefore, the transfer speed is significantly larger (which resulted in significant increase in the data throughput). FPGA does not generate any wait states and therefore the SPI transfer protocol is simplified. The SPI clock speed is 24 MHz and the measured data transfer is roughly 2 MB/s. Summing up, the designed module xps_spi_master significantly speed-ups data transfer and consumes significantly lower FPGA resources in comparison to the original EDK solution, which employs the MicroBlaze and PLB-slave-based SPI interface.

Keywords: embedded systems, SPI, FPGA, Xilinx EDK.

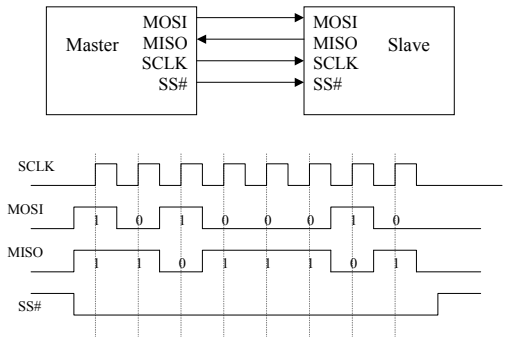
1. Wstęp

W systemach wbudowanych użycie niezależnego procesora ARM oraz układu FPGA umożliwia uzyskanie dużej elastyczności projektowania oraz wydajną pracę urządzenia [1]. Niestety wadą takiego rozwiązania jest konieczność zapewnienia szybkiej komunikacji pomiędzy wspomnianymi systemami. Jest ona kluczowa, dlatego często preferowane jest rozwiązanie homogeniczne, dla którego używa się albo tylko układu FPGA albo tylko procesora ARM. Doświadczenia budowy systemu homogenicznego [2] opartego tylko na układzie FPGA, pakiecie Embedded Development Kit (EDK) oraz soft-procesora MicroBlaze firmy Xilinx pokazało, że rozwiązanie to ma szereg wad [1]. Potwierdzeniem niniejszej tezy jest wprowadzanie przez firmę Xilinx nowej rodziny układów FPGA Zynq-7000 [3], która zawiera procesor ARM Cortex-A9. Dlatego w opisywanym rozwiązaniu kluczowe okazało się zapewnienie wydajnej komunikacji pomiędzy procesorem ARM oraz układem FPGA.

Ponieważ układ FPGA jest układem programowalnym, który nie wymusza standardów komunikacji oraz liczby użytych do tej komunikacji wyprowadzeń, główne ograniczenie wyboru standardu komunikacji leży po stronie procesora ARM. W wybranej płycie ewaluacyjnej [4] z procesorem OMAP3530 opartym głównie o procesor ARM najlepszym dostępnym interfejsem komunikacyjnym jest interfejs SPI (Serial Peripheral Interface) [5]. Oczywiście można rozważyć bardziej zaawansowane interfejsy jak np. Ethernet, ale są one dużo bardziej skomplikowane i jak pokaże wynik niniejszej publikacji programowe obsługiwanie takiego interfejsu wcale nie musi przynieść znaczącej korzyści jeśli chodzi o szybkość komunikacji. Pokazuje to praca [6] dla której uzyskano rzeczywistą transmisję danych przez Ethernet na poziomie 800kB/s. Podobnie jest w przypadku pomiarów dokonanych przez autorów niniejszej pracy, dla których rzeczywista transmisja przez Ethernet pomiędzy omawianym procesorem ARM [4] a komputerem PC była na poziomie 2.6 MB/s. Praca [7] pokazująca obciążenie procesora CPU dla szybkiej komunikacji przez Ethernet podziela niniejsze wyniki.

Warto podkreślić że interfejs SPI jest podstawowym interfejsem służącym do komunikacji pomiędzy procesorem wbudowanym lub DSP oraz urządzeniami peryferyjnymi. Jest to bardzo prosty

interfejs zarówno jeśli chodzi o format przesyłu danych jak i wymagania sprzętowe. Pokazuje to rys. 1.



Rys. 1. Schemat podłączenia urządzenia SPI typu master i slave oraz przykładowy przebieg
Fig. 1. Block diagram of SPI master and slave connection and a typical waveform

2. Model komunikacji

W omawianym zastosowaniu w ramach interfejsu SPI, procesor ARM pełni funkcje urządzenia typu master a układ FPGA pełni funkcję urządzenia typu slave. Po pierwsze wynika to z faktu, że to procesor ARM nadzoruje działanie układu FPGA. Jest to ogólnie przyjęta koncepcja projektowa, że procesor steruje wykonywanie funkcji sprzętowych. Drugim powodem jest to, że system operacyjny Linux (Ubuntu), który został zainstalowany na procesorze ARM, nie wspiera bezpośrednio interfejsu SPI typu slave.

Następnym założeniem w niniejszym projekcie jest przyjęty protokół komunikacji pomiędzy urządzeniem ARM i układem FPGA. Układ FPGA jest programowany za pomocą środowiska Embedded Development Kit (EDK) firmy Xilinx [8]. Umożliwia ono szybkie łączenie różnych gotowych modułów (ang. IP-cores). Dzięki temu możliwe jest między innymi proste wykorzystanie pamięci zewnętrznej SDRAM umieszczonej na płycie z układem FPGA [1] oraz soft-procesora MicroBlaze. Magistralą łączącą poszczególne moduły jest magistrala PLB (Processor Local Bus). Aby umożliwić łatwe kontrolowanie modułów w układzie FPGA poprzez procesor ARM, zastosowano model komunikacyjny poprzez bezpośredni dostęp do pamięci. W konsekwencji procesor ARM może zapisywać i odczytywać pamięć oraz rejestry sterujące w porównywalny sposób jak to wykonuje procesor MicroBlaze. W procesorze ARM napisano następujące funkcje w języku C/C++ służące do transmisji danych:

```
int spi_open(const int clock_speed);
int spi_write(const uchar *buf, uint adr, uint len);
int spi_read(uchar *buf, uint adr, int len);
int spi_close();
```

gdzie: *buf* wskazuje bufor transmitowanych danych znajdujący się na płycie z procesorem ARM; *adr* określa adres na płycie z układem FPGA pod który będą zapisane (funkcja write) lub spod którego będą odczytane (funkcja read) dane; *len* określa wielkość transmitowanych danych w bajtach z ziarnem 4 bajty.

Niskopoziomowy dostęp do interfejsu SPI zawarty wewnątrz wyżej wymienionych funkcji jest ukryty przed użytkownikiem. Warto podkreślić, że interfejs SPI nie określa dokładnego protokołu transmisji, dlatego protokół ten musiał zostać określony na potrzeby niniejszego projektu.

Dokładny algorytm komunikacji, który jest zawarty w funkcjach *spi_wirte* i *spi_read* wymaga oprócz transferu samych danych odpowiedniego przesłania adresu, kierunku i rozmiaru transmisji oraz sprawdzania ewentualnych błędów.

W przypadku dokonywania zapisu danych funkcją *spi_write* wykonywane są następujące czynności:

Listing 1. Algorytm zapisu *spi_write*
Listing 1. Write algorithm, function *spi_write*

```
spi_send(command_wirte, len)
spi_send(adr)
spi_send(data)
error= spi_recieve()
```

Funkcje *spi_send*, *spi_recieve* są to funkcję niskopoziomowe odpowiadające bezpośrednio za wysłanie i obór danych poprzez interfejs SPI. Reasumując wysłanie danych funkcją *spi_write* wymaga odpowiednio wysyłania ustalonej wcześniej komendy zapisu wraz z informacją o rozmiarze danych do zapisu, adresu zapisu oraz wysyłania transmitowanych danych. Po zakończeniu transmisji sprawdzany jest stan modułu odbiorczego po stronie FPGA na wypadek wystąpienia ewentualnych błędów.

Listing 2. Algorytm odczytu spi_read
Listing 2. Read algorithm, function spi_read

```
spi_send(command_read, len)
spi_send(adr)
do
  fifo_state= spi_recieve()
while(fifo_state=empty and not error)
data= spi_recieve()
error= spi_recieve()
```

Algorytm odczytu funkcją *spi_read* w pierwszym przybliżeniu wygląda podobnie jak algorytm zapisu. Wymaga on jednak przed rozpoczęciem odczytu danych sprawdzenia czy bufor FIFO (First-In First-Out) posiada już dane do odczytu. Wspomniany bufor FIFO znajduje się w module obsługującym transmisję SPI po stronie układu FPGA. Warto podkreślić, że zarówno dla odczytu jak i zapisu, podczas samej transmisji danych nie jest sprawdzany stan bufora FIFO, czyli zakłada się, że nie nastąpi jego przepełnienie (zapis) lub opróżnienie (ang. underflow) (odczyt). Stan błędu przepełnienia lub opróżnienia bufora FIFO jest sprawdzany po zakończeniu transmisji. W przypadku wystąpienia błędu konieczne jest powtórzenie transmisji. Warto podkreślić, że główny problem transmisji leży po stronie układu FPGA ponieważ układ ARM pełni funkcję urządzenia typu master, w konsekwencji w przypadku braku gotowości, transmisja jest wstrzymywana – sygnał sclk jest wystawiany przez urządzenie typu master i po przesłaniu pełnego słowa w każdej chwili może być zablokowany. Ponadto procesor ARM posiada wewnętrzny dedykowany moduł sprzętowy wspomagający transmisję danych po SPI wraz z odpowiednim układem DMA (Direct Memory Access).

3. Obsługa interfejsu SPI w układzie FPGA

W układzie FPGA obsługa interfejsu może odbywać się używając albo modułu dostarczonego w ramach pakietu EDK lub też dedykowanego modułu zaprojektowanego na potrzeby niniejszego projektu.

3.1. Moduł Xilinx'a

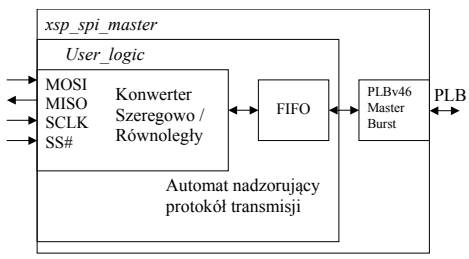
W ramach pakietu Xilinx EDK dostarczony jest moduł o nazwie *xsp_spi* [9], który umożliwia obsługę interfejsu SPI za pośrednictwem magistrali PLB v4.6. Moduł *xsp_spi* na magistrali PLB pełni funkcję urządzenia typu slave. W konsekwencji dla każdej przesłanej przez interfejs SPI danej konieczne jest aby procesor MicroBlaze wykonał odpowiedni odczyt (zapis) modułu *xsp_spi* oraz pamięci. Ponadto, co jest krytyczne czasowo, MicroBlaze powinien również wykonywać inne czynności niezależne od transmisji SPI, co wymusza odpowiednie obsługiwanie przerwania. W konsekwencji zalecany przez Xilinx'a stosunek częstotliwości taktowania magistrali PLB do częstotliwości zegara interfejsu SPI *sclk* wynosi co najmniej 64 [9]. W przeciwnym wypadku procesor MicroBlaze może nie być w stanie odpowiednio zapisać / odczytać buforu FIFO w module *xps_spi* i może nastąpić błąd

transmisji. Dla częstotliwości taktowania magistrali PLB równej 128 MHz, otrzymujemy zalecaną maksymalną częstotliwość interfejsu SPI mniejszą niż 2MHz, co daje teoretyczny maksymalny transfer mniejszy niż **250 kB/s**. Wartość ta jest zdecydowanie poniżej założeń projektowych i jest nie do przyjęcia.

3.2. Dedykowany moduł SPI

Powyższy problem został rozwiązany poprzez zaprojektowanie dedykowanego modułu sprzętowego o nazwie *xsp_spi_master*, który nie wymaga ingerencji procesora podczas transmisji danych. Jest to moduł typu master na magistrali PLB a odpowiednie procedury wykonywane uprzednio w procesorze MicroBlaze zostały przeniesione do automatu stanu znajdującego się w tym module. Moduł ten został napisany w języku VHDL. Schemat blokowy zaprojektowanego modułu został zamieszczony na rys. 2.

Moduł *xsp_spi_master s*kłada się z konwertera szeregowo/równoległego, który konwertuje standard szeregowy SPI na magistralę równoległą i odwrotnie. Dodatkowo niniejszy moduł dokonuje zamiany zegara taktującego SPI *sclk* na zegar systemowy. Moduł *user_logic* składa się głównie z odpowiedniego automatu stanu, który na podstawie aktualnego stanu oraz otrzymanych danych ustala funkcje aktualnie transmitowanych przez SPI danych. Moduł ten jest podłączony do magistrali PLB poprzez odpowiedni mostek PLBv46 Master Burst [10] firmę Xilinx.



Rys. 2. Schemat blokowy wykonanego modułu *xsp_spi_master*
Fig. 2. Block diagram of the designed module *xsp_spi_master*

4. Wyniki doświadczalne

Dedykowany moduł *xsp_spi_master* jest zdecydowanie szybszy od modułu Xilinx'a *xsp_spi*. Posiada on również bufory FIFO 16×32-bit. Dzięki nim moduł zabezpieczony jest przed dłuższym czasem oczekiwania na przyznanie magistrali PLB. Ponadto możliwe jest przeprowadzanie zdecydowanie szybszego transferu blokowego na magistrali PLB o głębokości czterech 32-bitowych słów. W konsekwencji szybkość transmisji danych pomiędzy procesorem ARM oraz układem FPGA jest ograniczona przez szybkość samego interfejsu SPI, który jest standardem szeregowym a więc relatywnie wolnym oraz szybkości obsługi interfejsu SPI po stronie procesora ARM. Doświadczalnie osiągniętą maksymalną częstotliwością transmisji danych jest 24 MHz, przy częstotliwości 48 MHz występują błędy podczas transmisji spowodowane najprawdopodobniej nie najlepszą jakością elektrycznego połączenia płyt ARM i FPGA.

Podsumowując uzyskana maksymalna teoretyczna szybkość transmisji dla częstotliwości 24 MHz wynosi 3 MB/s. Rzeczywista zmierzona szybkość transmisji wynosiła około 2 MB/s. Warto podkreślić, że dla relatywnie dużego bloku transmitowanych danych naddatek przyjętego protokołu transmisji jest znikomy i raczej nie wpływa na powyższą degradacje. Dlatego zmieszenie transmisji należy upatrywać w dodatkowych stanach oczekiwania wprowadzanych w procesorze ARM, układ FPGA zgodnie z przyjętym protokołem nie może wprowadzać dodatkowych opóźnień transmisji.

Tab. 1 pokazuje wyniki implementacji poszczególnych modułów. Moduł *user_logic* stanowi wewnętrzną część modułu *xsp_spi_master* zaprojektowaną przez autorów niniejszego artykułu, nie zawiera on modułu mostka PLBV46 Master Burst [10]. Podobnie moduł *xsp_spi_user* (wchodzący w skład modułu Xilinx'a *xps_spi*) nie uwzględnia dodatkowej logiki potrzebnej do podłączenia magistrali PLB w trybie slave. Porównując zasoby zajmowane przez moduł Xilinx'a *xsp_spi* oraz zaprojektowany moduł *xsp_spi_master* widać że dedykowany moduł zajmuje nieznacznie więcej zasobów układu FPGA. Jednak jak porównamy tylko logikę użytkownika: *user_logic* oraz *xsp_spi_user*, czyli po odjęciu zasobów potrzebnych do pośredniczenia pomiędzy logiką użytkownika a magistralą PLB, zauważymy że zajmowane zasoby są porównywalne. Warto podkreślić, że zasoby zajmowane przez soft-procesor MicroBlaze są dużo większe od opisywanych powyżej modułów.

Tab. 1. Wynik implementacji w układzie Xilinx Spartan 3 (XC3S1500FG456), EDK 12.2.
Tab. 1. Implementation results for Xilinx Spartan 3 (XC3S1500FG456), EDK 12.2.

moduł	FF	LUT	LUTRAM
xsp_spi_master	413	558	64
user_logic	205	354	64
MicroBlaze	626	1242	340
xsp_spi	384	457	64
xsp_spi_user	223	384	64

5. Podsumowanie

Niniejszy artykuł pokazuje jak zaprojektowanie dedykowanego modułu interfejsu SPI poprawia wyniki transmisji danych. Użycie modułu *xsp_spi* dostarczonego wraz z pakietem EDK umożliwia osiągnięcie maksymalnej teoretycznej transmisji rzędu 250 kB/s, w porównaniu z osiągniętym wynikiem 3 MB/s w przypadku użycia dedykowanego modułu *xsp_spi_master*. Zaprojektowany moduł jest kompatybilny ze środowiskiem EDK firmy Xilinx dlatego umożliwia łatwą integrację z pozostałymi elementami systemu wewnątrz układu FPGA. Został on tak zaprojektowany, że nie wymaga żadnej interakcji procesora MicroBlaze znajdującego się wewnątrz układu FPGA. Dzięki temu głównym ograniczeniem szybkości komunikacji ARM-FPGA jest szybkość interfejsu SPI, procesora ARM oraz czasy propagacji wewnątrz układu FPGA. Podsumowując, zaprojektowanie dedykowanego modułu sprzętowego przynosi znaczące oszczędności zarówno z punktu widzenia zajmowanych zasobów sprzętowych jak i zdecydowanej poprawy szybkości transmisji danych.

Praca wykonana w ramach realizacji projektu rozwojowego nr N R03 0061 06.

6. Literatura

[1] Wielgosz M., et. al.: System wbudowany oparty na procesorze ARM oraz układzie FPGA, PAK, przyjęty do publikacji.
[2] Adamczyk J., Krzyworzeka P., Cioch W., Jamro E.: Monitoring of Nonstationary States in Rotating Machitery, WITE Państwowy Instytut Badawczy - Radom, Kraków 2006.
[3] http://www.xilinx.com/products/silicon-devices/epp/zynq-7000/index.htm
[4] Embest Info&Tech Co., DevKit8000 Evaluation Kit, www.embedinfo.com
[5] http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
[6] Sienkiewicz J., Sikoń Sz.: Transmisja danych poprzez sieć Ethernet z wykorzystaniem układu FPGA, Praca dyplomowa inżynierska, AGH, Wydz. EAIiE, Kraków 2007.
[7] Twardy M.: Rekonfigurowany system ochrony transmisji danych typu Firewall dla sieci Ethernet o wielkich przepływnościach implementowany w układach FPGA, Ph.D. Thesis, AGH-UST WEAIiE, Kraków 2011.
[8] Xilinx Inc. EDK Concepts, Tools and Techniques, UG683 EDK 12.2, www.xilinx.com
[9] Xilinx Inc. LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a), DS570, 23-07-2010, www.xilinx.com
[10] Xilinx Inc. PLBV46 Master Burst (v1.01a) DS565 26-05-2010 www.xilinx.com

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*