Adam ZIĘBIŃSKI, Rafał CUPEK, Artur POREBSKI, Monika NYCZ
POLITECHNIKA ŚLĄSKA W GLIWICACH, WYDZIAŁ AEI, INSTYTUT INFORMATYKI, ul. Akademicka 16, 44-100 Gliwice

Realizacja koprocesora Modbus Slave w układzie FPGA z wykorzystaniem rdzenia procesora Microblaze

Dr inż. Adam ZIĘBIŃSKI

Ukończył studia w 1996 r. w Instytucie Informatyki na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 2002 r. Obecnie pracuje na stanowisku adiunkta w Instytucie Informatyki na wydziale AEI PolŚl. Jego zainteresowania to sprzętowo-programowe projektowanie koprocesorów problemowo zorientowanych z wykorzystaniem układów reprogramowalnych VLSI.

e-mail: adam.ziebinski@polsl.pl

Artur POREBSKI

Student czwartego roku kierunku Informatyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej.

e-mail: artur.porebski@hotmail.com

Dr inż. Rafał CUPEK

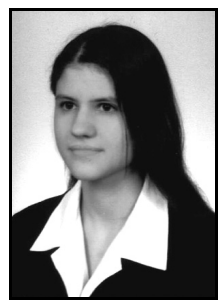
Ukończył studia w 1992 r. w Instytucie Informatyki na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 1999r. Obecnie pracuje na stanowisku prodziekana w Instytucie Informatyki na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej. Jego zainteresowania to komunikacja w przemysłowych systemach komputerowych oraz model komponentowy w systemach informatyki przemysłowej.

e-mail: rafal.cupek@polsl.pl

Monika Maria NYCZ

Studentka czwartego roku kierunku Informatyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej.

e-mail: monika.m.nycz@gmail.com



Streszczenie

W pracy przedstawiono projekt systemu wbudowanego zrealizowanego w układzie FPGA, pełniącego funkcję koprocesora Modbus Slave pracującego w trybie RTU z wykorzystaniem interfejsu RS232. Moduł wykonanego koprocesora składa się z rdzenia procesora Microblaze, modułu UART, timerów i bloków pamięci. Prezentowane rozwiązanie umożliwia zaprojektowanie systemu współpracującego zarówno ze standardowymi szybkościami transmisji danych w sieci Modbus jak i znacznie większymi sięgającymi nawet do 921600 Bd, przy transmisji pomiędzy dwoma systemami z układami FPGA.

Słowa kluczowe: FPGA, Microblaze, Modbus Slave RTU, systemy wbudowane.

Implementation of Modbus Slave coprocessor in FPGA array using soft core processor Microblaze

Abstract

The paper presents design of an embedded system realised on a FPGA array, fulfilling the function of the coprocessor Modbus Slave working in the RTU mode with use of the interface RS232. The realised coprocessor module consists of the soft core processor Microblaze, UART module, set of timers and memory blocks for storing the data. The Modbus Slave algorithm was implemented in C language for the processor Microblaze. The system was implemented on the Xilinx XUPV2P development system with the FPGA XC2VP30. Synthesis and programming were conducted using the Xilinx XPS SDK tools. The maximum frequency of the system operation is above 146 MHz. Correctness of the implemented coprocessor Modbus Slave work under real conditions was tested with use of the program Modbus Poll and Top Server OPC. The presenting solution allows designing the system cooperating with standard speed data transmission in the Modbus networks and considerably greater speeds reaching 128000 Bd (transmission among the personal computer and the FPGA system) or even 921600 Bd (transmission among two FPGA systems).

Keywords: embedded systems, FPGA, Microblaze, Modbus Slave RTU.

1. Wstęp

Protokół komunikacyjny Modbus [3, 4, 15, 16] od wielu lat służy do przesyłania danych w sieciach przemysłowych. Modbus jest też jednym z najczęściej stosowanych standardów komunikacyjnych w większości systemów automatyki przemysłowej. Swoją popularność Modbus zawdzięcza prostocie rozwiązań technologicznych jak i możliwości łatwej rozbudowy sieci o nowe węzły podrzędne (*ang. slaves*).

Najbardziej rozpowszechnionymi wersjami protokołu są: Modbus ASCII i Modbus RTU. Ze względu na większą szybkość przesyłanych ramek przy jednakowych parametrach transmisji częściej, wykorzystywany jest tryb RTU niż ASCII.

Narastające zapotrzebowanie sieci przemysłowych na wyższe szybkości przesyłu i przetwarzania danych skłania ku wykorzystywaniu zaawansowanych technologicznie rozwiązań, specjalizowanych układów VLSI (*Very-large-scale integration*) czy też matryc programowalnych FPGA (*Field Programmable Gate Arrays*). Biorąc pod uwagę, że układy FPGA doskonale nadają się do realizacji systemów prototypowych, rozpoczęte zostały prace nad implementacją protokołu Modbus Slave w układzie FPGA. Implementacja protokołu w FPGA możliwa jest z wykorzystaniem jednego z trzech komponentów:

- własnoręcznie napisanego modułu zawierającego funkcje zapisane w języku opisu sprzętu (HDL - *hardware description language*)
- wbudowanego w układ procesora sprzętowego (np. PowerPC)
- gotowego modułu procesora typu soft core (np. Microblaze)

Najbardziej efektywnym rozwiązaniem jest wykorzystanie gotowego komponentu rdzenia procesora, ze względu na możliwość wykonania szybkiego prototypu systemu. Ponadto rdzeń procesora jest konfigurowalny i programowalny, co zapewnia łatwą integrację z innymi systemami [1]. Mnogość zalet stosowania gotowego komponentu rdzenia procesora skłania do przeprowadzania badań nad możliwościami implementacji koprocesora Modbus Slave pracującego w trybie RTU z wykorzystaniem modułu procesora Microblaze.

2. Opis algorytmu Modbus

Modbus jest protokołem typu Master-Slave posiadającym tryby transmisji RTU (podstawowy) i ASCII (opcjonalny). W trybie transmisji RTU każdy bajt wysyłany jest w kolejności od najmniej do najbardziej znaczącego bitu (ang. LSB to MSB).

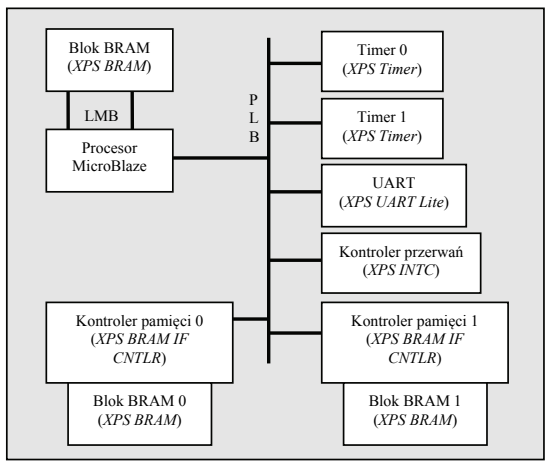
Ramka zapytania (*ang. query*) jak i odpowiedzi (*ang. response*) składa się z takich samych elementów. Mianowicie: bajtu adresu modułu slave'a, bajtu kodu funkcji, N bajtów pola danych oraz 2 bajtów sumy kontrolnej CRC. Każdy bajt w polu danych ramki składa się z dwóch 4-bitowych znaków heksadecymalnych. Maksymalny rozmiar ramki Modbus RTU to 256 bajtów.

Wykrycie rozpoczęcia nowej ramki oraz zakończenia wiadomości odbywa się poprzez pomiar czasu. W trybie transmisji RTU odstępy pomiędzy kolejnymi ramkami wyznacza cisza na łączu trwająca co najmniej 3,5 znaku, a co za tym idzie znacznikiem ukończenia ramki jest czas trwania ciszy trwający również minimum 3,5 znaku.

Całość ramki musi być transmitowana jako ciągły strumień znaków. W przypadku pojawienia się odstępu pomiędzy kolejnymi znakami ramki większego niż 1,5 znaku wiadomość traktowana jest jako niekompletna i powinna zostać odrzucona przez odbiornik [3].

3. Projekt koprocesora

Na potrzeby przeprowadzenia badań zaprojektowano moduł koprocesora Modbus Slave w układzie FPGA Virtex XC2VP30 firmy Xilinx (rys. 1).



Rys. 1. Schemat blokowy koprocesora Modbus Slave w układzie FPGA XC2VP30
Fig. 1. Flowchart of Modbus Slave coprocessor in FPGA XC2VP30

Sercem analizowanego modułu jest procesor Microblaze współpracujący z blokiem pamięci BRAM, zawierającym pamięć RAM jak i przechowującym rozkazy procesora.

Peryferia są podłączone do procesora poprzez magistralę PLB (Processor Local Bus). Spośród nich należy wyróżnić timery (XPS Timer/Counter) oraz kontroler przerwań (XPS INTC). Zastosowanie kontrolera przerwań umożliwiło realizację całości programu realizującego funkcję koprocesora Modbus Slave z wykorzystaniem systemu przerwań [2]. Z kolei wykorzystanie timerów zapewnia spełnienie zależności czasowych transmisji w trybie RTU, odmierzając czas transmisji 1,5 i 3,5 znaku.

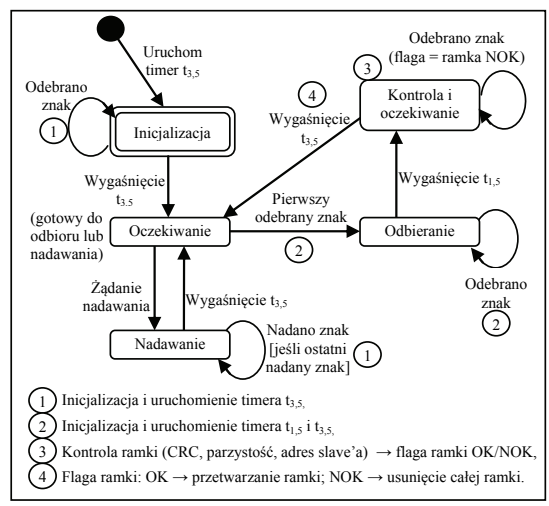
Model danych protokołu MODBUS został odzwierciedlony w dwóch blokach pamięci BRAM (XPS Block RAM) po 64 KB każdy. Dołączone są one do magistrali poprzez kontrolery pamięci (PLB BRAM Controller). Oba bloki pamięci cechuje dwuportowość, pozwalająca na bardzo łatwy dostęp komponentów do zawartych w nich danych. Komponenty mogą zająć się przetwarzaniem danych bez konieczności podłączania się do głównej magistrali PLB (dołączanie zbyt dużej ilości elementów powoduje zmniejszenie maksymalnej częstotliwość pracy całego układu).

Do komunikacji szeregowej użyty został moduł XPS UART Lite, ze względu na fakt, iż posiadał on wszystkie wymagane funkcje (w tym wykorzystywanie przerwań), wymagał zdecydowanie mniej zasobów sprzętowych w układzie FPGA niż bardziej rozbudowany core UART 16550.

4. Opis funkcjonalności programu

Schemat działania aplikacji dla procesora Microblaze, bazuje na opisie diagramu stanów trybu RTU, zawartym w oficjalnej dokumentacji standardu Modbus wydanej przez The Modbus Organization [3]. Dokładny algorytm programu przestawia rysunek 2.

Przy typowych szybkościach RS-232 występujących w sieciach przemysłowych czas generowania nawet najdłuższej ramki Modbus jest nieporównywalnie mniejszy od czasu jej transmisji. Z tego też powodu wystarczające będzie wykorzystanie do budowy systemu rdzenia procesora Microblaze. Program w głównej mierze opiera się na przerwaniach co dodatkowo pozwoli na wykorzystanie czasu procesora także do innych celów niż komunikacja.



Rys. 2. Schemat działania programu
Fig. 2. Program algorithm

W pierwszym kroku następuje inicjalizacja zmiennych i skonfigurowanie układów peryferyjnych. Po czym procesor przechodzi do pętli nieskończonej, w której oczekuje na nadchodzące przerwania. Jednakże pętla może zostać zastąpiona przez realizację innych zadań procesora, na przykład przetwarzanie lub zbieranie danych.Po uruchomieniu programu pomijane są wszystkie odbierane znaki, dopóki nie nastąpi pomiędzy nimi przerwa równa czasowi transmisji trzy i pół znaku (odmierzana przez timer t3.5). Następnie program przechodzi do fazy oczekiwania na obiór lub wysyłanie ramki.

Podczas wysyłania pomijane są wszystkie odbierane znaki. Gdy w czasie oczekiwania zostanie odebrany znak, program przechodzi do fazy odbierania ramki. Po każdym odebranym znaku resetowane są timery t1.5 i t3.5.

W przypadku przepełnienia się timera t1.5 zakładany jest koniec ramki i następuje oczekiwanie na przepełnienie sie timera t3.5. Gdy w międzyczasie zostaną odebrane jakieś znaki, ramka oznaczana jest jako błędna.

Po odczekaniu wymaganego okresu czasu następuje przejście do fazy oczekiwania i jeśli ramka została odebrana prawidłowo, przetworzenia jej. Podczas przetwarzania ramki następuje sprawdzenie CRC, adresu modułu Slave, kodu funkcji oraz poprawności jej argumentów. W zależności od poprawności ramki jest ona wykonywana lub generowany jest wyjątek o odpowiednim

numerze (*ang. exception*). W programie zostały zaimplementowane najważniejsze spośród funkcji protokołu [4]. Wyszczególnienie funkcji zostało zebrane w tabeli 1 wraz z opisem ich działania.

Tab. 1. Zaimplementowane funkcje w programie
Tab. 1. Implemented functions in program

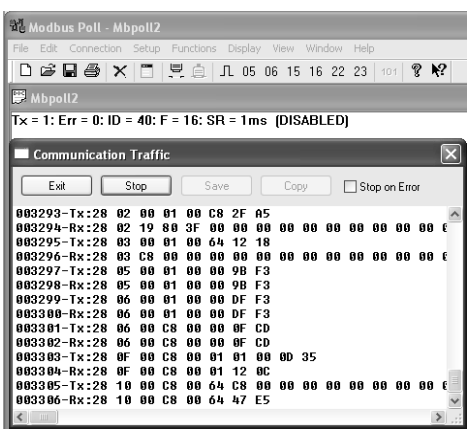
Kod (hx)	Funkcja	Opis
01 (0x01)	Read Coils	*odczyt wyjść bitowych*
02 (0x02)	Read Discrete Inputs	*odczyt wejść bitowych*
03 (0x03)	Read Holding Registers	*odczyt rejestrów*
04 (0x04)	Read Input Registers	*odczyt rejestrów wejściowych*
05 (0x05)	Write Single Coil	*zapis pojedynczego wyjścia bitowego*
06 (0x06)	Write Single Register	*zapis pojedynczego rejestru*
08 (0x08)	Diagnostics (częściowo)	*odczyt liczników diagnostycznych*
15 (0x0F)	Write Multiple Coils	*zapis wielu wyjść bitowych*
16 (0x10)	Write Multiple registers	*zapis wielu rejestrów wyjściowych*

Program jest dodatkowo łatwo konfigurowalny. Pozwala na spójne mapowanie przestrzeni poszczególnych typów danych Modbus w 32-bitową, liniową przestrzeń adresową procesora bez skomplikowanych zmian w kodzie (istnieje możliwość określania adresów początkowych oraz ilości każdego z typów danych).

Zastosowanie w kodzie instrukcji *switch* odpowiedzialnej za rozpoznawanie obsługiwanych funkcji, uczyniło go prostym w modyfikacji. Dopisanie własnej funkcjonalności prócz kodu samej funkcji wymaga zaledwie dopisania pojedynczej linii w programie.

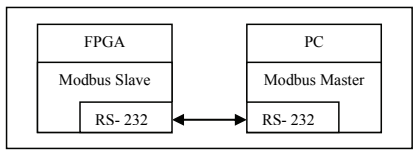
5. Wyniki testów

System zaimplementowano na zestawie uruchomieniowym firmy Xilinx XUPV2P posiadającym matrycę FPGA XC2VP30 [5]. Synteza i programowanie zostały przeprowadzone przy użyciu narzędzi Xilinxa XPS SDK. Raport z syntezy wykazał, że badany projekt zajmuje około 16 % zasobów typu Slices, 9% typu Slice Flip Flops oraz 10% typu 4 input LUTs. Prawie połowę zasobów z każdej grupy zajmuje rdzeń procesora Microblaze. Maksymalna częstotliwość, z jaka może pracować przedstawiony projekt wynosi 146.369 MHz., ale wszystkie testy zostały przeprowadzone przy częstotliwości taktowania 100 MHz.



Rys. 3. Wyniki testów przeprowadzonych w programie Modbus Poll
Fig. 3. Results of tests conducted in the program Modbus Poll

Do sprawdzenia poprawności pracy zaimplementowanego koprocesora Modbus Slave w warunkach rzeczywistych, wykorzystano program Modbus Poll [6] zainstalowany na komputerze PC. Program ten pozwolił na przeprowadzenie testów komunikacji symulując pracę master'a sieci Modbus (rys.3). Przy transmisji pomiędzy FPGA a komputerem PC (rys.4) system pozwolił na poprawną pracę do szybkości 128000 Bd włącznie.



Rys. 4. Schemat testowania systemu FPGA - PC
Fig. 4. Testing FPGA system - PC

Ponadto przeprowadzono testy czasu przetwarzania ramki przy użyciu dodatkowego timera dołączonego do magistrali PLB. Timer ten był włączany, gdy ramka została odebrana, a zatrzymywany gdy program przechodził do fazy oczekiwania. Poszczególne wyniki zapisywane były w blokach pamięci i odczytywane za pomocą odpowiednich funkcji Modbus. Dla każdej z nich przyjmowana była odpowiednia porcja danych zgodnie ze standardem protokołu Modbus [4] (tab. 2).

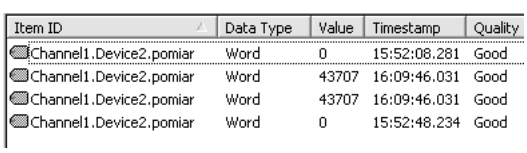
Przeprowadzone testy czasu przetwarzania ramki wykazały że czas wzrasta liniowo w zależności od ilości odczytywanych lub zapisywanych danych.

Tab. 2. Zmierzone czasy generowania i przetwarzania odpowiedzi
Tab. 2. Measured times of frame generation and processing

Kod funkcji	**01**	**02**	**03**	**04**	**05**	**06**	**15**	**16**
Min. liczba danych	5,7 µs		5,9 µs		4,1 µs	3,8 µs	7,3 µs	6,5 µs
Max. liczba danych	104,9 µs		210,8 µs				187,2 µs	90,7 µs

Rozkazy potrzebne do włączania i zatrzymywania timera wprowadziły do pomiarów narzut o czasie 385 ns. (czas wykonywania rozkazów został zmierzony na symulatorze ModelSim 6.5c). Rozwiązanie to, jest więc obarczone stosunkowo niskim błędem pomiaru (1 takt zegara systemu), a jednocześnie jest szybkie w implementacji.

Kolejnym etapem było sprawdzenie pracy systemu w środowisku typowo przemysłowym. Wykonane zostały testy współpracy Top Servera OPC [7] wraz z układem skonstruowanego koprocesora przy standardowej szybkości 9600 Bd. W ramach testu odczytywano wartości z rejestrów koprocesora Modbus za pomocą OPC Quick Client, przy czym maksymalny period odczytów wynosił 10 ms. Rysunek 5 przestawia jeden z wyników odczytywanych wartości, a przeprowadzone testy wykazały kolejny raz poprawność realizowanej komunikacji.

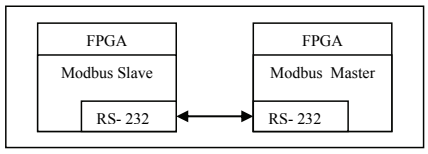


Item ID	Data Type	Value	Timestamp	Quality
Channel1.Device2.pomiar	Word	0	15:52:08.281	Good
Channel1.Device2.pomiar	Word	43707	16:09:46.031	Good
Channel1.Device2.pomiar	Word	43707	16:09:46.031	Good
Channel1.Device2.pomiar	Word	0	15:52:48.234	Good

Rys. 5. Wyniki współpracy systemu w FPGA z Top Server OPC
Fig. 5. Results of system cooperation with Top Server OPC

Następny test został przeprowadzony przy użyciu dwóch systemów uruchomieniowych XUPV2P połączonych kablem typu null-modem (rys. 6). Na jednej z nich zaimplementowany był badany moduł koprocesora, a na drugiej jego zmodyfikowana wersja pełniąca częściowo funkcję Master'a. Miała ona za zadanie

generować ramki Modbus, a następnie czekać na odpowiedz od pierwszej karty oraz sprawdzać poprawność odebranych ramek. Maksymalna szybkość jaką udało się uzyskać to 921600 Bd. Ograniczeniem tutaj okazał się tylko i wyłącznie zastosowany moduł UART, dla którego była to maksymalna szybkość pracy.



Rys. 6. Schemat połączenia układu typu karta - karta
Fig. 6. Connection FPGA - FPGA

Ostatni test dotyczył zajętości procesora Microblaze podczas działania układu. W celu zmierzenia wykorzystania procesora do układu został dodany dodatkowy timer (podobnie jak podczas testu czasu generowania odpowiedzi). W głównej pętli programu umieszczona została ciągła inkrementacja zmiennej. Podczas przepełnienia się timera jej wartość była zapisywana do pamięci, a następnie zerowana. Stosunek wartości zmiennej podczas braku przesyłania ramek oraz podczas ich ciągłej transmisji pozwolił wyznaczyć przybliżony współczynnik wykorzystania procesora. W najgorszym przypadku (funkcja 16) zajętość Microblaze'a wynosiła zaledwie 0,56%, zatem funkcje komunikacyjne mają znikomy wpływ na czas wykonywania głównego kodu procesora, a sam procesor może również zostać wykorzystany do innych celów.

Porównanie naszego rozwiązania przeprowadziliśmy z modułami dedykowanymi dla sterowników PLC (tab.3). Jako że prawie wszystkie pozwalają na dokładną konfiguracje formatu znaku (liczba bitów danych, bitów stopu czy parzystość) różnice występują głównie szybkościach komunikacji oraz posiadanych portach.

Tab. 3. Zestawienie rozwiązań protokołu Modbus
Tab. 3. Modbus protocol solutions

Nazwa modułu	Szybkość min. [Bd]	Szybkość max. [Bd]	Rodzaj portu	Wybór formatu znaku
MVI46-MCM [8]	110	115200	RS232, 485, 422	Tak
IC200CMM020 [9]	1200	19200	RS485	Tak
CM-220-9 [10]	300	38400	RS485	Nie
CM-180-1 [11]	1200	115200	RS232, 485	Nie
EMB-02R [12]	300	230000	RS232, 485	Nie
Modbus FPGA	110	921600	RS232	Tak

6. Podsumowanie

Koprocesor Modbus Slave dostępny jest obecnie w postaci wielu różnego typu rozwiązań, zarówno programowych jak i sprzętowych, Systemy takie są również realizowane w układach FPGA, jednak głównie jako rozwiązania oparte o interfejs Ethernet. Proponowane przez nas rozwiązanie dotyczy implementacji wykorzystującej interfejs RS232. Interfejs ten wraz z interfejsem RS485, jest nadal stosowany w wielu dostępnych urządzeniach automatyki przemysłowej.

Przeprowadzone przez nas testy wykazały, że wykonana implementacja koprocesora Modbus Slave w układzie FPGA z wykorzystaniem rdzenia procesora Microblaze, umożliwia komunikację zarówno z standardowymi rozwiązaniami przemysłowymi wykorzystującymi protokół Modbus, jaki i zwykłym komputerem klasy PC, czy tez innym systemem FPGA. Uzyskane wyniki potwierdzają poprawną pracę systemu zarówno przy standardowych szybkościach dla protokołu Modbus, jak i przy znacznie większych szybkościach przesyłu danych. Uzyskaliśmy szybkość do 128000 Bd przy komunikacji ze zwykłym komputerem klasy PC, 13 razy szybciej niż przy standardowej szybkości. Natomiast komunikacja pomiędzy dwoma systemami FPGA, umożliwia aż 96 razy szybszą transmisję danych, osiągając 921600 Bd. W obu przypadkach główny wpływ na ograniczenie szybkości przesyłanych danych miał zastosowany moduł UART, który w dalszym etapie badań zostanie wymieniony na inny element.

W efekcie wykorzystanie koprocesora Modbus Slave zaimplementowanego w układzie FPGA, może pozwolić na znaczne skrócenie czasu trwania transakcji modbus'owej (cykl od wysłania zapytania do otrzymania odpowiedzi przez układ mastera). Czas ten można obliczyć zgodnie z wzorem 1.

$$czas\_transakcji = \frac{liczba\_bitów \cdot 1000}{szybkosc\_UART} + czas\_przetwarznia \quad [ms] \quad (1)$$

Przy długości ramki zapytania wynoszącej 8 bajtów i długości ramki odpowiedzi wynoszącej 255 bajtów oraz czasowi przetwarzania wynoszącemu 0,21084 ms; liczba bitów wyniesie (8+255)*10=2630, natomiast czas transmisji przy szybkości 9600 Bd zajmie 275 ms, przy szybkości 128000 Bd zajmie 21 ms natomiast przy szybkości 921600 Bd już tylko 3 ms.

Wykorzystanie dodatkowych podsystemów z układami FPGA może pozwolić na wprowadzenie nowej funkcjonalności do systemów automatyki przemysłowej wykorzystujących nadal stosowany protokół Modbus, umożliwiając znaczne zwiększenie szybkości przetwarzania i przesyłania danych.

7. Literatura

[1] http://www.xilinx.com/products/design_resources/proc_central/microblaze_faq.pdf
[2] http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf
[3] http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf
[4] http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf
[5] http://www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf
[6] http://www.modbustools.com/
[7] http://www.toolboxopc.com/
[8] http://www.prosoft-technology.com/content/view/full/135
[9] http://www.ge-ip.com/products/3002
[10] http://www.moeller.pl/artykuly/elektroinstalator_1205_modbus.pdf
[11] http://www.ultima-automatyka.pl/Nowa/Dokumentacja/Moduly/CM-180-1_v01a_ModBus_RTU_slave-ModBus_RTU_slave.pdf
[12] http://www.dcbnet.com/datasheet/emb02ds.html
[13] http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf
[14] http://www.xilinx.com/publications/books/serialio/serialio-book.pdf
[15] Mielczarek W.: Szeregowe interfejsy cyfrowe. Helion, Gliwice, 1993.
[16] Modicon Modbus Protocol. Reference Guide. Modicon, 1991.

otrzymano / received: 07.05.2010
przyjęto do druku / accepted: 04.06.2010 *artykuł recenzowany*