Inteligentny interfejs VXI dla elektroniki sterownika wnęk rezonansowych SIMCON 3.1

Waldemar Koprek
Konrad Hejn

Przedmiotem pracy jest inteligentny interfejs VXI umożliwiający programowanie z poziomu zbioru komend SCPI (*Standard Commands for Programmable Instruments*) elektroniki sterowników wnęk rezonansowych typu SIMCON 3.1, stosowanych w akceleratorze VUV-FEL (DESY Hamburg). Projekt interfejsu zapisano w kodzie VHDL, a następnie zaimplementowano w układach FPGA. Elektronika sterowników SIMCON 3.1 jest widziana w środowisku sprzętowym VXI i programowym zbioru komend SCPI jako urządzenie komunikatowe podwładne (*message-based servant*). Oznacza to, że interfejs zawiera rejestry komunikacyjne niezbędne do działania protokołu WSP (Word Serial Protocol).

A smart VXI Interface for Electronics of Resonance Cavities Controller SIMCON 3.1

This work is about a VXI interface with local intelligence that understands SCPI (Standard Commands for Programmable Instruments). The circuitry of an interface logic is described in VHDL and implemented in FPGA that the electronics of SIMCON 3.1 possesses. SIMCON 3.1 is a resonance cavity controller used in linear accelerator VUV-FEL (DESY Hamburg). Thanks to VXI interface in question, the SIMCON 3.1 is seen as a message-based servant. It means that VXI interface contains all communication registers that make a Word Serial Protocol (WSP) working.

1. Wstęp

Technologia VXI/SCPI [1] została po raz pierwszy zastosowana do konstrukcji inteligentnych urządzeń pomiarowych około 15 lat temu. Jej zaczynem były zamówienia armii USA skierowane do najpoważniejszych producentów elektronicznego sprzętu pomiarowego (HP, Tektronix, Racal-Data, NI, Wavetek, Corolado Data System, Bruel&Kjaer). Ze względu na wysoką cenę potrzebnych systemów pomiarowo-sterujących (SPOM), konieczne było podporządkowanie ich projektowania ogólnie akceptowalnym standardom. Tylko wtedy integracja sprzętu i oprogramowania okazała się procesem o akceptowalnym horyzoncie czasowym, a perspektywa użyteczności systemu mogła być liczona przynajmniej w dziesiątkach lat. Obecnie problem polega na tym, że dostępne normy aparaturowe są w znakomitej większości trudne do zrozumienia i implementacji. Dlatego mniejsi producenci (nie mówiąc o laboratoriach badawczych) zostali całkowicie wyeliminowani z atrakcyjnego rynku SPOM. Dopiero pojawienie się efektywnych układów FPGA z wbudowanym procesorem stworzyło potencjalną szansę uwolnienia projektanta modułów SPOM od wielu, wprawdzie dostępnych szczegółów, ale jednocześnie głęboko ukrytych w plikach obowiązujących go norm. Nawet doświadczony konstruktor staje się bezradnym, bo zakres potrzebnej mu wiedzy, jak i jej szczegółowość są porażające. Dlatego projektant otrzymując do dyspozycji wspomagające narzędzie zawierające interfejs VXI z dekoderem (sprzętowym lub programowym) komend wspólnych oraz z uniwersalnym szkieletem parsera komend SCPI [2], będzie mógł swój specyficzny projekt zamknąć w rozsądnych granicach czasowych i finansowych, a także będzie mógł interakcyjnie testować funkcje projektowanej elektroniki i obsługujące ją drzewa komend SCPI.

Waldemar Koprek – e-mail: wkoprek@elka.pw.edu.pl
Konrad Hejn – e-mail: K.Hejn@ise.pw.edu.pl
Politechnika Warszawska, Instytut Systemów Elektronicznych

2. Koncepcja

Punktem wyjściowym tej pracy był interfejs VXI dla urządzeń rejestrowych [3], wykorzystywany w eksperymencie VUV-FEL (DESY Hamburg) [4] do komunikacji na poziomie binariów ze sterownikiem wnęk rezonansowych SIMCON 2.1. Obecna wersja sterow-

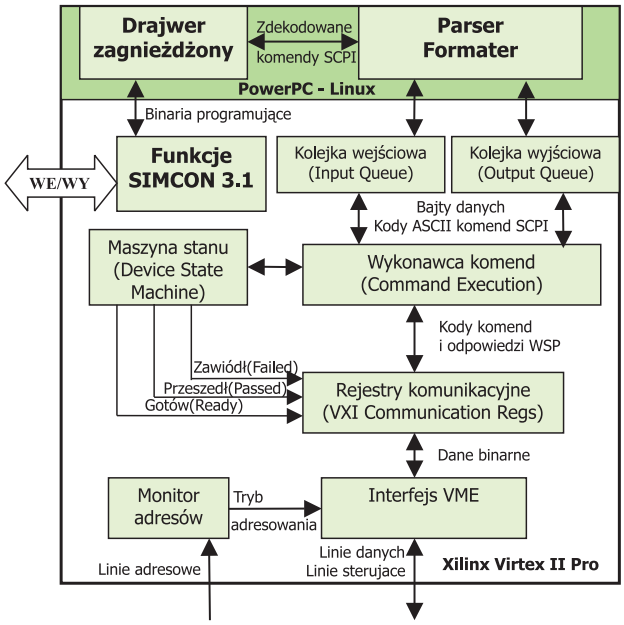
nika wnęk rezonansowych SIMCON 3.1 ma znacznie więcej zaimplementowanych funkcji, stąd liczba jego roboczych rejestrów (sterujących i stanu) przekracza 100. W nowej konstrukcji zastosowano nowoczesny układ FPGA firmy Xilinx (Virtex II Pro), który ma wbudowany procesor PowerPC. W związku z tym pojawiła się możliwość zaprojektowania interfejsu VXI z lokalną inteligencją pozwalającą komunikować się na poziomie znaków ASCII (komend wspólnych standardu IEEE 488.2 [5] oraz zbioru komend SCPI [2]). Istotna część programowego drajwera mogła być przeniesiona z komputera osobistego do modułu SIMCON 3.1 i zagnieżdżona w układzie FPGA. Procesor PowerPC odbiera z interfejsu kasety VXI (E1406) komunikaty SCPI, interpretuje je, a wynik ładuje do rejestrów roboczych modułu SIMCON 3.1. Zaleta komunikacji na poziomie znaków ASCII polega na ukryciu znaczenia zawartości rejestrów roboczych (także skomplikowanych zależności pomiędzy nimi) w procesorze PowerPC. Przy czym funkcje modułu SIMCON 3.1 mogą zostać opisane przez zrozumiałe i czytelne dla użytkownika ciągi znaków ASCII (np.: przez komunikat programujący: SOURCE:ADC1:OFFSET 10).

3. Realizacja

Projekt interfejsu VXI dla podwładnego urządzenia komunikatowego został zapisany w kodzie VHDL i zaimplementowany w układzie FPGA znajdującym się na płycie sterownika wnęk rezonansowych SIMCON 3.1 (rys. 1).

W projekcie wykorzystano wykonany wcześniej interfejs VXI dla sterownika komór rezonansowych SIMCON 2.1, który będąc typowym urządzeniem rejestrowym (zgodnie z normą VME [6]), zawierał już następujące komponenty: *Interfejs VME* i *Monitor adresów*, a także obligatoryjne *Rejestry konfiguracyjne* VXI. W projekcie interfejsu dla elektroniki sterownika SIMCON 3.1 komponent rejestrów VXI został zmodyfikowany przez dodanie *Rejestrów komunikacyjnych* (VXI C*ommunication Registers*). Całkowicie nowym komponentem jest Wykonawca komend (*Command Execution*), którego zadaniem jest interpretacja i wykonanie komend protokołu WSP (*World Serial Protocol*). Ponadto powstały też niezależne komponenty automatu sterującego w postaci *Maszyny stanu* (*Device State Machine*) oraz *Kolejki wejściowej* (*Input Queue*) i *Kolejki wyjściowej* (*Output Queue*). Te dwa ostatnie komponenty służą zarówno do buforowania poleceń odbieranych z magistrali VXI, jak i odpowiedzi wystawianych na magistralę VXI.

Rejestry komunikacyjne. Oficjalna specyfikacja definiuje 16-bitowe rejestry, które musi mieć każde urządzenie komunikatowe wykonane zgodnie ze standardem VXI [1]. Rejestry te znajdują się w 64-bajtowej przestrzeni adresowej dostępnej dla sterownika systemowego w trybie A16/D16. *Rejestry konfiguracyjne* zajmują adresy od 0_{16} do 6_{16}, natomiast *Rejestry komunikacyjne* zajmują adresy od 8_{16} do 14_{16} (rzeczywiste adresy są obliczane względem adresu urządzenia VXI). Przedstawiono to na rys. 2. Dla podwładnego urządzenia komunikatowego tylko trzy spośród ośmiu Rejestrów komunikacyjnych są obligatoryjne: *Protocol*, *Response* i *DataLow* (pogrubiona czcionka na rys. 2). Znajdują się one w komponencie VHDL o wspomnianej już wcześniej nazwie *Rejestry Komunikacyjne*.



Rys. 1. Schemat interfejsu VXI dla podwładnego urządzenia komunikatowego SIMCON 3.1

Fig. 1. Block diagram of VXI interface for message-based servant SIMCON 3.1

0x3F	
	DEVICE DEPENDENT REGISTERS
0x1F	
	VXIbus RESERVED REGISTERS
0x18	
	A32 Pointer
0x14	
	A24 Pointer
0x10	
0x0E	**Data Low**
0x0C	Data High
0x0A	**Response** / Data Extended
0x08	**Protocol** / Signal Register
	CONFIGURATION REGISTERS
0x00	

Rys. 2. Rejestry obligatoryjne podwładnego urządzenia komunikatowego VXI (A16/D16)

Fig. 2. Obligatory registers of VXI message-based servant (A16/D16)

Word Serial Protocol (WSP). Standard VXI korzysta ze ściśle zdefiniowanych protokołów do przesyłania komunikatów (poleceń, zapytań, odpowiedzi) pomiędzy urządzeniem skonfigurowanym jako przełożony (commander) a urządzeniem skonfigurowanym jako podwładny (servant). Obligatoryjny przy tym jest protokół WSP (asynchroniczny z dwuprzewodowym zaklepaniem), który jest ustawiany domyślnie jako aktywny po włączeniu zasilania. Specyfikacja VXI definiuje kilkadziesiąt komend obsługujących WSP, ale tylko 6 z nich: *Begin Normal Operation, Abort Normal Operation, End Normal Operation, Clear, Read Protocol, Read Protocol Error* jest wymaganych w każdym urządzeniu komunikatowym. Zostały one zaimplementowane w komponencie Wykonawca Komend (*Command Execution)*. Komendy te, jak i stowarzyszone z niektórymi z nich odpowiedzi, są słowami 16-bitowymi przesyłanymi zawsze za pośrednictwem rejestru *DataLow*.

Word Serial Data Transfer Protocol

Byte Available (komenda)

1	0	1	1	1	1	1	END	BAJT DANYCH

Byte Request (komenda i odpowiedź)

1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1

1	0	1	1	1	1	1	END	BAJT DANYCH

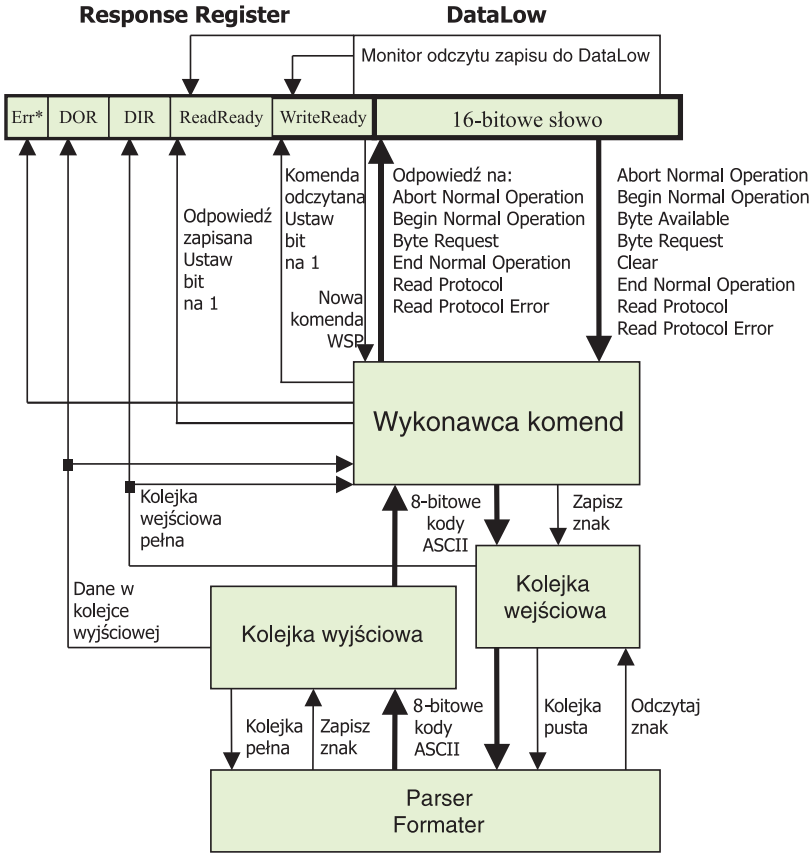
Rys. 3. Format komend i odpowiedzi w Word Serial Data Transfer Protocol

Fig. 3. Commands and responses in Word Serial Data Transfer Protocol

Jest to protokół opcjonalny. Jeżeli jednak urządzenie komunikatowe ma być zgodne ze standardem IEEE 488.2, to protokół staje się obligatoryjnym i jest wykorzystywany do przesyłania komend oraz odpowiedzi. Do realizacji tego protokołu zostały dodane do WSP dwie komendy o nazwach *Byte Available* i *Byte Request*. Służą one odpowiednio do wysłania i odczytania bajtu danych przez urządzenie przełożone. Komendy te tworzą mechanizm zaklepania asynchronicznego przesłania, zwany *Byte Transfer Protocol*. Bajt danych jest zagnieżdżonym parametrem komendy lub odpowiedzi. W przypadku przesyłania komend jeden bajt oznacza kod ASCII jednego znaku komendy. Rys. 3 przedstawia sposób umieszczania bajtów danych w słowie komendy Byte Available i w słowie odpowiedzi na komendę *Byte Request*. Bit END – jeśli jest ustawiony na 1 – wskazuje, że jest to ostatni element przesyłanego ciągu bajtów.

Kolejki danych. W celu usprawnienia transmisji dużej liczby znaków ASCII stosuje się kolejki FIFO. Kolejka wejściowa służy do przechowywania znaków wysłanych za pomocą komendy *Byte Available*. Zaś kolejka wyjściowa przechowuje znaki odpowiedzi, które są pobierane przez komendę *Byte Request*. Przepływ znaków pomiędzy urządzeniami jest regulowany bitami *DIR* i *DOR*, które znajdują się w rejestrze komunikacyjnym *Response Register*. Wartość 0 bitu *DIR* oznacza, że urządzenie nie jest gotowe na przyjęcie kolejnych bajtów i jest to na ogół związane z przepełnieniem kolejki wejściowej. Wartość 1 bitu *DOR* oznacza, że dane są gotowe do odczytania, a w kolejce wyjściowej znajduje się co najmniej jeden znak odpowiedzi.

Należy tutaj rozróżnić zastosowanie bitów *Read Ready, Write Ready* oraz bitów *DIR* i *DOR*. *Bity Read Ready* i *Write Ready* służą do synchronizacji przesyłu komend WSP (również *Byte Available* i *Byte Request*) pomiędzy *Rejestrami komunikacyjnymi* a *Wykonawcą komend* (dwuprzewodowy handshake). Bity *DOR* i *DIR* służą do synchronizacji *Byte Transfer Protocol*. Urządzenie przełożone odczytując rejestr *Response*, testuje bity DIR, DOR i w zależności od ich stanu wysyła odpowiednie komendy *Byte Available* i *Byte Request*. Jeżeli urządzenie przełożone wyśle komendę *Byte Available*, a kolejka wejściowa będzie pełna lub wyśle komendę *Byte Request*, a kolejka wyjściowa będzie pusta, to *Wykonawca komend* musi zgłosić błąd (odpowiednio *DIR Violation* lub *DOR Violation*).



Rys. 4. Zasada działania bloku Wykonawca komend

Fig. 4. Principles of Command Execution block

Wykonawca komend (*Command Execution*). Komendy protokołu WSP są interpretowane i wykonywane w komponencie VHDL o nazwie *Wykonawca komend* (rys. 4). Operacja przesłania i wykonania komend dzieli się na dwa etapy. Pierwszy etap polega na zapisaniu słowa komendy do rejestru *Data Low* w komponencie *Rejestry komunikacyjne*, który jest widziany przez komunikatowe urządzenie przełożone w przestrzeni adresowej szyny VXI.

Przesłanie komendy do rejestru komunikacyjnego *Data Low* podlega standardowemu cyklowi zapisu na szynie VME i jest wykonywane w tle omawianego protokołu WSP. Zgodnie ze specyfikacją na rejestrze *Data Low* jest umieszczony monitor, który informuje blok *Wykonawca komend* o zatrzaśnięciu nowego słowa – komendy WSP. Monitor ustawia bit *Write Ready* na 0. Drugi etap to wykonanie komendy. Blok *Wykonawca* komend śledzi na bieżąco stan bitu *Write Ready*. Jeżeli jest on ustawiony na zero, to znaczy, że w rejestrze *Data Low* pojawiła się nowa komenda WSP. *Wykonawca komend* pobiera komendę, dekoduje ją i wykonuje. Jednocześnie *Wykonawca komend* natychmiast po pobraniu komendy ustawia bit *Write Ready* na wartość 1. Urządzenie przełożone odczytując na szynie VXI zawartość rejestru komunikacyjnego Response urządzenia podwładnego, sprawdza bit Write Ready. Jeżeli jest on ustawiony na 1, to znaczy, że urządzenie podwładne jest gotowe do przyjęcia następnej komendy. W przypadku komendy WSP, która jest zapytaniem, *Wykonawca komend* zapisuje wynik wykonania do rejestru *Data Low* i ustawia bit *Read Ready* na 1. Urządzenie przełożone oczekując na wynik zapytania, odczytuje na szynie VXI rejestr *Response* urządzenia podwładnego i sprawdza bit *Read Ready*. Jeżeli bit jest ustawiony na 1, to znaczy, że odpowiedź jest gotowa. W następnym cyklu odczytu na szynie VXI urządzenie przełożone odczytuje rejestr *Data Low* urządzenia podwładnego, w którym jest odpowiedź, a monitor odczytu rejestru kasuje zawartość bitu *Read Ready*.

Poniżej jest przedstawiony przykładowy przebieg wykonania komendy *Byte Request*:

- zapis do rejestru *Data Low* komendy *Byte Request* przez urządzenie przełożone (standardowa operacja zapisu na VME, która kończy się załadowaniem 16-bitowego słowa do *Data Low*)
- monitor zapisu do rejestru *Data Low* ustawia bit *Write Ready* na 0 (bit *Ready Ready* jest cały czas ustawiony na zero). *Wykonawca komend* sprawdza bit *Write Ready* (0 oznacza, że w rejestrze *Data Low* jest komenda WSP)
- *Wykonawca komend* odczytuje rejestr *Data Low* (16 bitów)
- *Wykonawca komend* ustawia bit *Write Ready* na 1 (jeżeli zostałaby zapisana tylko komenda WSP zamiast zapytania, to urządzenie przełożone mogłoby już wysłać kolejną komendę WSP, np. ciąg komend *Byte Available*)
- *Wykonawca komend* sprawdza 7 najstarszych bitów komendy (dekodowanie komendy – tutaj *Byte Request*)
- *Wykonawca komend* przechodzi do przygotowania odpowiedzi (w tym przypadku odczytuje z kolejki wyjściowej jeden znak – jeżeli kolejka wyjściowa była pusta, to ustawia bit *Err** w rejestrze komunikacyjnym *Response.* Po odczytaniu znaku z kolejki wyjściowej *Wykonawca komend* tworzy słowo 16-bitowe zgodnie z rys. 3)
- *Wykonawca komend* zapisuje słowo do rejestru *Data Low*, jednocześnie z zapisem słowa ustawia bit *Read Ready* na 1
- urządzenie przełożone odczytując rejestr *Response*, testuje bit *Read Ready*
- jeżeli urządzenie przełożone stwierdzi, że *Read Ready* jest ustawiony na 1, to odczytuje zawartość rejestru komunikacyjnego *Data Low* (odpowiedź)
- monitor odczytu rejestru *Data Low* ustawia bit *Read Ready* na 0.

4. Podsumowanie

Inteligentny interfejs dla podwładnego urządzenia komunikatowego został przetestowany w kasecie przystosowanej do modułów VXI o rozmiarze C. Funkcję sterownika osadzonego podsystemu VXI pełnił moduł kieszeni zerowej HP E1406, który ze zrozumiałych względów nie ma w swoich zasobach drajwera dla SIMCOM 3.1. Komunikuje się on z komputerem personalnym (sterownikiem swobodnym) za pomocą magistrali GPIB i biblioteki I/O firmy Agilent. Znajdujący się w nim dotychczasowy drajwer programowy został zmodyfikowany i zagnieżdżony w inteligentnym interfejsie sterownika komór rezonansowych SIMCON 3.1. W ramach przykładowego eksperymentu zostało opracowane drzewo komend SCPI, które pozwala na podjęcie przez sterownik SIMCON 3.1 wszystkich jego podstawowych funkcji. Przeprowadzone badania potwierdziły poprawność współpracy komputera osobistego ze sterownikiem wnęk rezonansowych SIMCON 3.1 zainstalowanym w kasecie VXI jako podwładne urządzenie komunikatowe (*message-based servant*).

5. Literatura

1. *VXI-1 System Specification*, VXIbus Consortium, Inc., Rev. 2.0 ed., August 24, 1998.
2. SCPI Consortium, *Standard Commands for Programmable Instruments*, Version 1999.0, May 1999.
3. W. Koprek, K. Hejn, *A flexible electronic tool for VXI register-based device development*, Proc. of SPIE, Bellingham, WA, USA, TOM 5948, 2005, pp. 81–90.
4. tesla.desy.de
5. IEEE Std 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*, December 1992.
6. ANSI/VITA 1-1994, *American National Standard for VME64*, American National Standards Institute, Inc., April 10, 1995.