ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992

Seria: INFORMATYKA z. 20 Nr kol. 1190

Andrzej GRZYWAK
Katarzyna GRZYWAK-PTASZNIK

JĘZYK OPISU SPRZĘTU CYFROWEGO TYPU VHDL

Streszczenie. Artykuł przedstawia zasadnicze pojęcia i możliwości stosowania języka opisu sprzętu cyfrowego VHDL. Przedstawiono koncepcję języka VHDL zdefiniowaną standardem IEEE-1076. Opis języka zilustrowano przykładem jego zastosowania do zdefiniowania projektu pamięci stałej 64kB*8 bitów.

HARDWARE DESCRIPTION LANGUAGE VHDL

Summary. This paper presents the Hardware Description Language VHDL. The VHDL language is a formal notation intended for use in all phases of the creation of electronic systems. The paper ilustrates the structural description of simple memory board.

DIE HARDWAREBESCHREIBUNGSPRACHE VHDL

Zusammenfassung. Das Artikel darbietet die grundsatzliche Grundbegriffe und Moeglichkeiten der Anwendung der VHDL - Sprache (Hardware Description Language). Die Sprache wurde mit dem Beispiel von ihrer Anwendung fuer das Festspeicherprojekt illustriert.

1. Wstęp

Systemy automatyzacji projektowania urządzeń elektronicznych wymagają w coraz większym stopniu wprowadzania języków opisu sprzętu, stwarzających równocześnie możliwość symulacji projektowanych struktur. Postulat ten wiąże się bezpośrednio z ros-

nącą złożonością układów VLSI i koniecznością wprowadzania ujednoliconych metod projektowania tych układów.

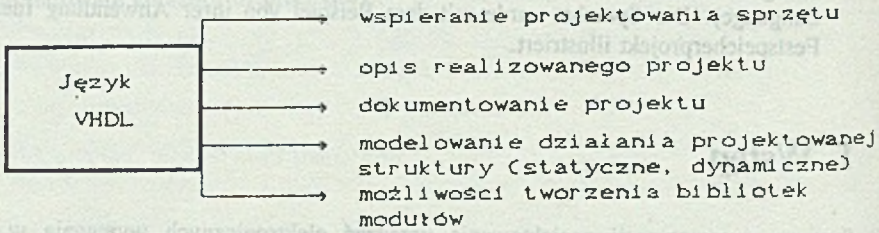
Języki stosowane do opisu sprzętu elektronicznego, a zwłaszcza maszyn cyfrowych, były definiowane od wielu lat. Praktycznie jednak języki te nie były stosowane przez inżynierów projektujących urządzenia elektroniczne. W roku 1983 z inicjatywy Departamentu Obrony Stanów Zjednoczonych powołano grupę roboczą złożoną między innymi z przedstawicieli firm IBM i Texas Instruments, do opracowania języka opisu sprzętu przydatnego do realizacji tzw. programu VHSIC (Very High Speed Integrated Circuits). W roku 1985 powstała pierwsza wersja jeyka VHDL implementowana na MC VAX 11/780 i IBM 370. Artykuł niniejszy dotyczy wersji języka VHDL objętej standardem IEEE 1076-1987 (drugie wydanie) z lat 1987-88.

W zasadniczej części opracowanie bazuje na następujących publikacjach źródłowych:
- VHDL Tutorial for Standard 1076 VHDL, June 1987,
- IEEE Standard VHDL Language Reference.

Opracowanie ma na celu przedstawienie podstawowych danych o języku VHDL i zilustrowanie jego zastosowań do opisu wybranych struktur cyfrowych.

2. Koncepcje języka VHDL i jego zastosowanie

Język VHDL w zakresie projektowania sprzętu spełnia wymagania związane z określeniem projektowanej struktury i jej sprawdzania w całym zakresie projektowania. Język ten umożliwia gromadzenie doświadczeń projektowych. Zadania języka VHDL ilustruje rysunek 1.



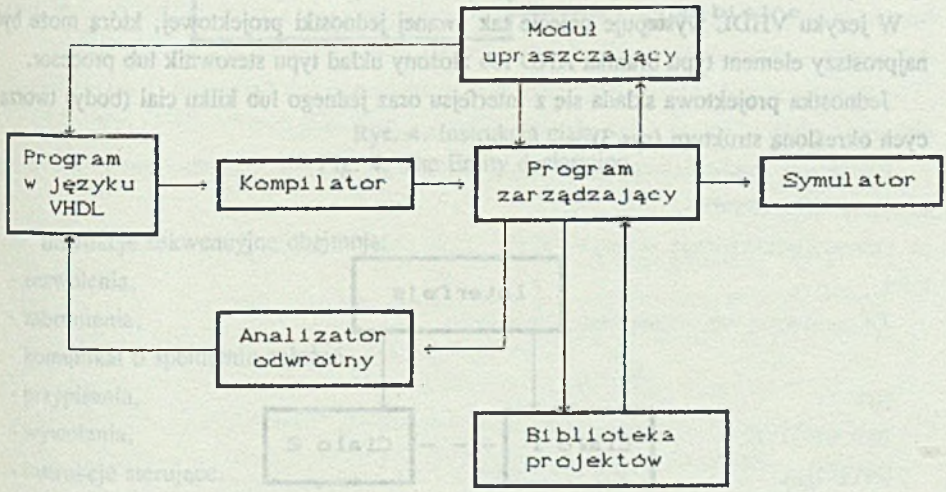
Rys. 1. Zadania języka VHDL
Fig. 1. Functions of VHDL Language

Własności języka VHDL można przedstawić w następujący sposób:
- język umożliwia hierarchiczny opis sprzętu cyfrowego,
- język obejmuje różne poziomy abstrakcji od przesyłów międzyrejestrowych do rozwiązań systemowych,
- możliwe jest modelowanie projektowanych struktur i ich badanie z punktu widzenia statycznego i dynamicznego,
- język jest niezależny od przyjętej metody projektowania sprzętu,
- możliwe jest rozdzielenie opisu części sterującej od części operacyjnej.

Zdefiniowany obecnie język VHDL pozwala opisać topologię układu scalonego.

Środowisko języka tworzą:
- kompilator języka,
- analizator odwrotny,
- programy zarządzające bibliotekami,
- moduł upraszczający,
- symulator,
- biblioteka programów.

Na rysunku 2 przedstawiono środowisko języka VHDL zawierające podane powyżej moduły programowe.



Rys. 2. Środowisko języka VHDL
Fig. 2. Environment of VHDL Language

Kompilator jezyka VHDL tlumaczy postać źródłową programu napisanego w tym języku na format pośredni (format IVAN) weryfikując program statyczny (bez symulacji).

Program zarządzajacy umożliwia przechowywanie projektów w bibliotece i określa sposób korzystania z zasobów biblioteki.

Analizator odwrotny umożliwia dostęp do biblioteki projektów przechowywanych w formacie pośrednim (IVAN) i ich przetwarzanie na tekst zródłowy w języku VHDL.

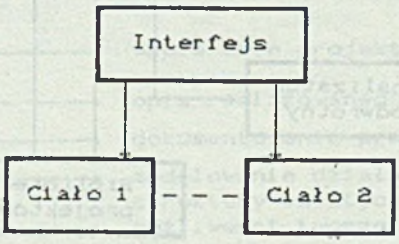
Moduł wprowadzajacy reorganizuje jednostkę projektowaną na postać rownoważną funkcjonalnie jednostce zaprojektowanej. Ten moduł może być pominięty w języku VHDL.

Symulator weryfikuje poprawność funkcjonalną projektu. Symulator dokonuje transformacji opisu na opis w języku ADA modelującym jednostkę projektowaną. Możliwa jest symulacja danego projektu na wielu poziomach abstrakcji.

Biblioteka projektów jest zbiorem rozwiązań projektowych przechowywanym w formacie pośrednim. Projekty są określane w kontekście technologi wykonania lub bloków funkcjonalnych.

W języku VHDL występuje pojęcie tak zwanej jednostki projektowej, którą może być najprostszy element typu bramka AND lub złożony układ typu sterownik lub procesor.

Jednostka projektowa składa się z interfejsu oraz jednego lub kilku ciał (body) tworzących określoną strukturę (rys.3).



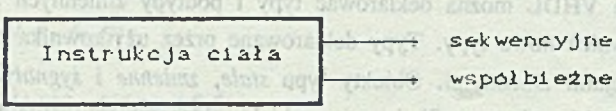
Rys. 3. Jednostka projektowa w języku VHDL
Fig. 3. Design entity

Opis interfejsu definiuje port wejścia i wyjścia oraz parametry czasowe.
W języku VHDL istnieją trzy rodzaje portów:
- port inout - przepływ informacji w obu kierunkach,
- port buffer - (port może mieć tylko jedno zródło informacji),
- port linkage - (do portu można przyłączyć porty o określonych kierunkach przepływu informacji).

Ciała projektu w języku VHDL mogą być opisane na dwa sposoby:
- ciało architektoniczne (architecture body),
- ciało konfiguracyjne (configuration body).

Ciało architektoniczne definiuje strukturę i sposób działania jednostki projektowej. Ciało konfiguracyjne zawiera informacje o hierarchii składników użytych w ciele oraz o przyporządkowaniu aktualnych interfejsów do wyżej wymienionych składników. W skład ciała jednostki projektowej wchodzą deklaracje obiektów (stałe, zmienne, sygnały) oraz pełny zbiór instrukcji związanych z jednostką.

Instrukcje ciała dzielą się na sekwencyjne i współbieżne (rys.4).

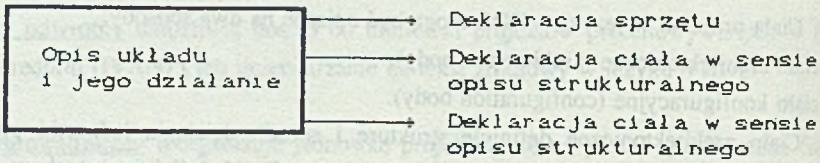


Rys. 4. Instrukcja ciała
Fig. 4. The Entity declaration

Instrukcje sekwencyjne obejmują:
- zezwolenia,
- zabronienia,
- komunikat o spełnieniu założeń,
- przypisania,
- wywołania,
- instrukcje sterujące.

Instrukcje współbieżne obejmują:
- blok (lokalne deklaracje i instrukcje),
- proces (instrukcje sekwencyje obliczające wartości zmiennych i sygnałów),
- powielenie (konkretyzacja regularnej struktury składników jednostki projektowej).

Opis układu cyfrowego i jego działanie w języku VHDL obejmuje:
- deklaracje sprzęgu,
- deklaracje ciała w sensie opisu strukturalnego,
- deklaracje ciała w sensie opisu funkcjonalnego (rys.5).



Rys. 5. Opis układu cyfrowego w języku VHDL
Fig. 5. The design Entity concept

Bardzo istotne w języku VHDL są tak zwane nośniki sygnału, które umożliwiają przekazywanie zmian wartości (historii) sygnałów. Jeżeli konieczne jest zachowanie wartości sygnału, to taki sygnał poprzedza słowo: *memoried*.

W języku VHDL można deklarować typy i podtypy zmiennych i sygnałów. Można tworzyć również nowe typy. Typy deklarowane przez użytkownika mogą być skalarami lub deklaracjami złożonymi. Obiekty typu *stałe, zmienne* i *sygnały* tworzone są przez deklaracje. Deklaracja specyfikuje nazwę obiektu, klasę obiektu i typ danych obiektu.

Podprogramy w języku VHDL mogą być procedurami lub funkcjami. Funkcje służą do obliczeń, a procedury do dekompozycji jednostki projektowej.

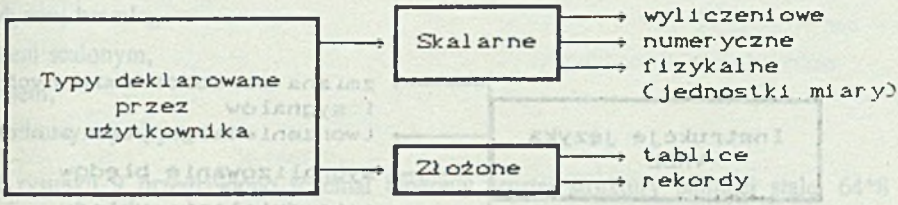
Pakiet jest mechanizmem grupującym deklaracje. Raz zdefiniowany pakiet może być używany wielokrotnie.

Obiekty deklarowane w języku VHDL mają określony typ. Typy stanowią zbiór wartości.

Zdefiniowano pierwotnie następujące typy:

BIT	/0,1/
BIT VECTOR	/tablica/
INTEGER	/liczby całkowite/
REAL	/liczby rzeczywiste/
BOOLEAN	/wartości logiczne/
CHARACTER	/znaki kodu ASCII/
STRING	/tablice kodu ASCII/

SEVERITY LEVEL /poziomy ważności komunikatów/
POSITIVE /liczby całkowite większe od zera/
NATURAL /liczby całkowite nie ujemne/



Rys. 6. Typy deklarowane przez użytkownika
Fig. 6. Entity declaration

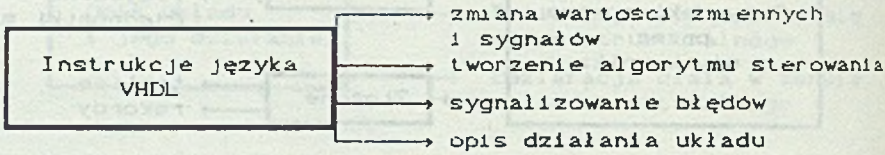
Można definiować nowe typy:
np. zbiór logiki czterowartościowej
type LOG 4 is /0, 1, Z, E/
np. typ numeryczny
type POZIOM NAPIS - 5.5 to +5.5

W języku VHDL istnieją trzy klasy obiektów (rys.7).

Rys. 7. Klasy obiektów
Fig. 7. Data objects

Obiekt deklaruje się przez wyspecyfikowanie nazwy, a także klasy i typu obiektu. Sygnały są obiektami, których wartości mogą być zmienne. Realizacja programu to wykonanie jego instrukcji. Instrukcje mogą:

- zmieniać wartości zmiennych i sygnałów,
- tworzyć algorytm sterowania,
- sygnalizować błędy,
- opisywać działanie jednostki projektowej.



Rys. 8. Wyniki działania instrukcji w języku VHDL
Fig. 8. Results of declaration in VHDL Language

Język VHDL rozróżnia następujące typy instrukcji:

1) sekwencyjne,
2) przypisania,
3) zezwolenia,
4) komunikaty,
5) sterujące,
6) wywołania procedur i funkcji,
7) współbieżne,
8) bloku,
9) procesu (definicja niezależnego procesu),
10) konkretyzacji,
11) powielania.

Ważnym elementem języka VHDL jest modelowanie upływu czasu w makroskali i mikroskali. Mechanizmem grupującym deklaracje typów atrybutów, stałych, podprogramów i składników jest pakiet. Pozwala on wyizolować w opisie sprzętu zależności technologiczne. W języku VHDL można wyrażać wielkości fizyczne, co umożliwia pełne dokumentowanie rozwiązań technicznych. Poniżej przedstawiony zostanie opis pamięci typu ROM zrealizowany w języku VHDL.

3. Opis w języku VHDL modułu pamięci stałej

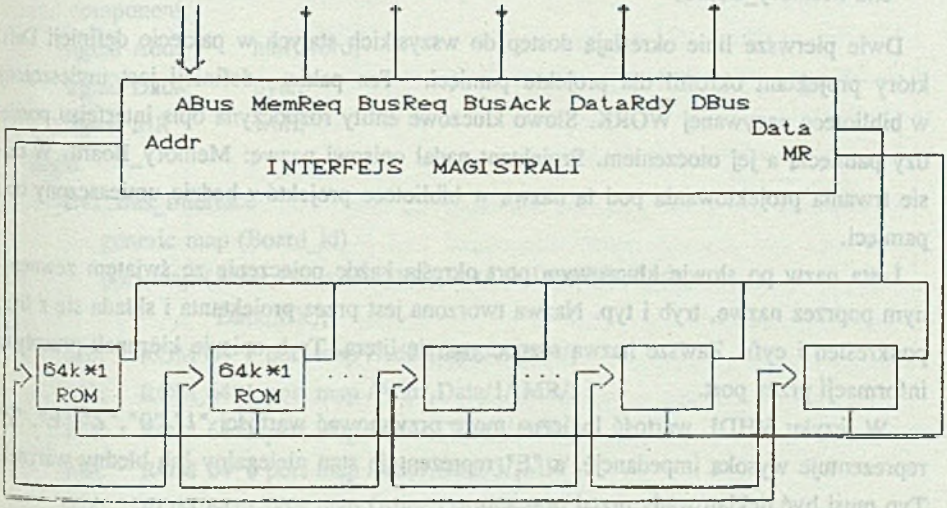
Projektowanie jednostek wyższego poziomu (struktur bardziej skomplikowanych) jest oparte na połączeniach wewnętrznych komponentów jednostki, która może być:

- pojedynczą bramką,
- układem scalonym,
- modułem,
- jednostkowym podsystemem.

Na rysunku 9 przedtawiono schemat blokowy prostej struktury pamięci stałej 64*8 bitów. Na podstawie tego rysunku zostaną pokazane podstawowe zasady strukturalnego opisu.

Interfejs magistrali (jeden z komponetów jednostki) odpowiada za zależności czasowe i sterowanie pamięcią.

Opis pamięci za pomocą języka VHDL składa się z dwóch części:



Rys. 9. Pamięć ROM 64kB*8 bitów
Fig. 9. Memory board ROM 64*8

- definicji interfejsu pomiędzy obiektem projektowanym a światem zewnętrznym,
- projektu samego bloku pamięci.

Poniżej przedstawiono deklarację jednostek dla bloku pamięci:

```
- Interface for the memory board
  Library Work
  use Work Defs. all.
    entity Memory_Board is port
          (
          ABus:     in address;
          DBus:     out byte;
          MemReq:     in wbit;
          BusReq:     out wbit;
          BusAck:     in wbit;
          DataRdy:  out wbit;
          );
    constant   Board_id: wbit_vector:=110;
    end Memory_Board;
```

Dwie pierwsze linie określają dostęp do wszystkich stałych w pakiecie definicji Defs, który projektant określił dla projektu pamięci. Ten pakiet definicji jest umieszczony w bibliotece nazywanej WORK. Słowo kluczowe entity rozpoczyna opis interfejsu pomiędzy pamięcią a jej otoczeniem. Projektant nadał opisowi nazwę: Memory_Board. W czasie trwania projektowania pod tą nazwą w bibliotece projektów będzie umieszczony opis pamięci.

Lista nazw po słowie kluczowym port określa każde połączenie ze światem zewnętrznym poprzez nazwę, tryb i typ. Nazwa tworzona jest przez projektanta i składa się z liter, podkreśleń i cyfr. Zawsze nazwa rozpoczyna się literą. Tryb opisuje kierunek przepływu informacji przez port.

W języku VHDL wartość logiczna moze przyjmować wartości "1","0","Z","E". "Z" reprezentuje wysoką impedancję, a "E" reprezentuje stan nielegalny lub błędny warunek. Typ musi być deklarowany przed użyciem.

Deklaracje typu wbit, address, byte, wbit_vector są umieszczone w pakiecie Defs.

Board_id jest stałą zadeklarowaną, która będzie używana w architekturze: w swej części będzie dotyczyła logicznego dekodowania adresu.

Structure jest nazwą wybraną przez projektanta, dla ustalonej architektury pamięci. Strukturę pamięci przedstawia zapis:

```
architecture Structure of Memory_Board is
component ROM 64*8
   port(
   Addr      :   in half word;
   Data      :   out wbit;
   chip selct:   in wbit);
end component;
component Bus_Interface
   generic (Board_id:   in wbit vector);
   port  (ABus          :in address;
          DBus          :out byte;
          MemReq        :in wbit;
          BusAck        :out wbit;
          DataRdy       :out wbit;
          Addr          :out half_word;
          Data          :in byte;
          MR            :out wbit);
end component;
   signal Addr      half word;
   signal Data      byte;
   signal MR        wbit;
begin
   INT:Bus_Interface
      generic map (Board_id)
      port map (ABus,DBus,MemReq,BusAck,DataRdy,Addr,
                Data,MR);
   M0:   ROM 64*1 port map /Addr,Data/0/,MR/;
   M1:   ROM 64*1 port map /Addr,Data/1/,MR/;
   M2:   ROM 64*1 port map /Addr,Data/2/,MR/;
   M3:   ROM 64*1 port map /Addr,Data/3/,MR/;
   M4:   ROM 64*1 port map /Addr,Data/4/,MR/;
   M5:   ROM 64*1 port map /Addr,Data/5/,MR/;
   M6:   ROM 64*1 port map /Addr,Data/6/,MR/;
   M7:   ROM 64*1 port map /Addr,Data/7/,MR/;
end structure;
```

Taka architektura będzie zapamiętana w tej samej bibliotece projektu co jednostka Memory_Board. Struktura jest podzielona na dwie części przez słowo begin.

Dwie pierwsze deklaracje opisują "komponenty" (składniki), które będą używane. Są tu zadeklarowane dwa typy "komponentów":

- ROM 64*8,
- BUS interface.

Nazwa każdego komponentu następuje po liście portu opisującego połączenia z "komponentem". Każdy port ma swoją nazwę, tryb i typ.

BUS interface posiada także generic list. Użycie tej listy dostarcza całej rodziny "komponentów", które są opisane przez pojedyncze definicje. Wartości muszą być wybrane dla każdego generowanego komponentu wybranego z całej rodziny elementów.

Sygnał (signal) określa drogę sygnału, która może łączyć komponenty przez projektanta. Jeżeli sygnał jest podłączony do portu, typ sygnału i typ portu muszą być takie same.

Stałe przykładowe komponentów zostały wykreowane i połączone w części linii architektury. Każda stała jest kreowana przez linię stałych komponentów. Sala rozpoczyna się zawsze etykietą.

Interfejs magistrali INT ma wygenerowaną mapę, taką jak mapę portów. Stała BOARD id zadeklarowana w deklaracji jednostki jest dostarczana przez wartość parametru generacji. Wartość BOARD id będzie użyta wewnątrz implementacji interfejsu magistrali do definicji maski adresu modułu pamięci.

Strukturalny styl opisu pokazuje dekompozycję urządzenia na składniki (komponenty), które mogą być abstrakcyjnymi jednostkami funkcjonalnymi lub mogą odnosić się do fizycznych komórek, upakowanych układów scalonych lub modułów w dużym systemie. W przeciwieństwie do tego opisu przepływu danych uwypukla się przepływ informacji pomiędzy pamięcią a elementami bramkującymi.

Przepływ danych jest zarządzany i kierowany przez elementy sterujące, które logicznie oddzielają sygnały sterujące od linii danych.

Interfejs magistrali

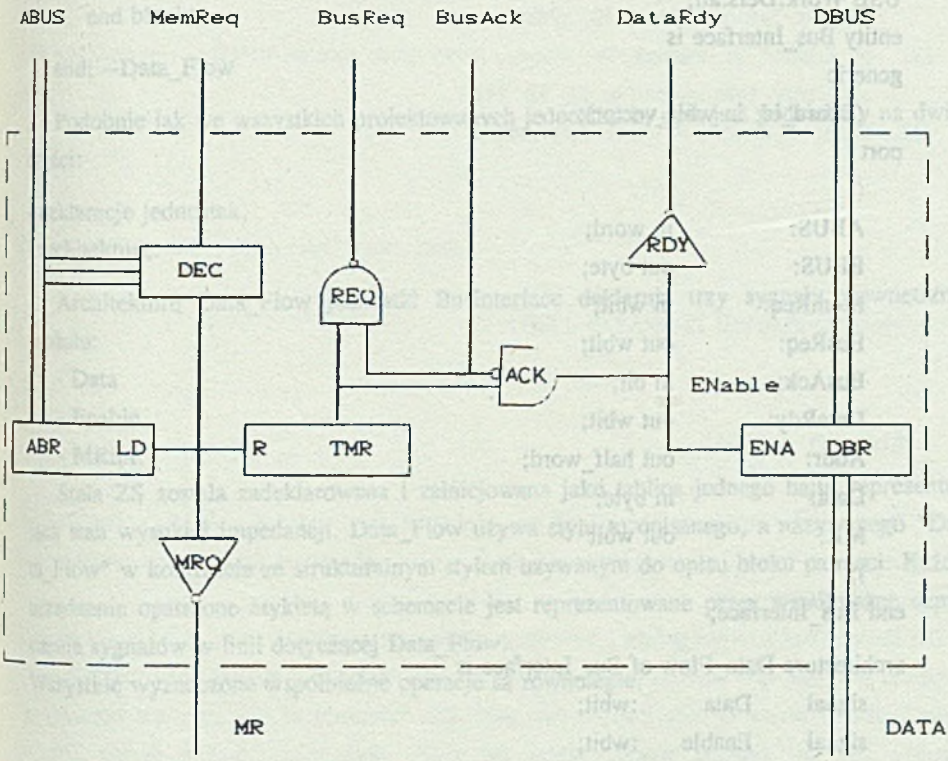
Schemat interfejsu magistrali został przedstawiony na rys.10.

Interfejs posiada trzy porty podłączone do pamięci ROM:

- Addr - który steruje liniami adresowymi pamięci,
- MR - sygnał połączony z linią "select",
- Data - linie danych wyjściowych z pamięci.

Reszta portów BUS interface podłączona jest do zewnętrznej magistrali. Wszystkie sygnały realizujące "handshaking" mają niski poziom aktywny.

Cykl dostępu do pamięci rozpoczyna się, gdy MemReq przyjmuje stan niski i na bitach 16 do 18 (adresu) na magistrali ABUS ustali się adres bloku. Dekoder DEC po zdekodowaniu numeru bloku i uzyskania sygnału MRint generuje sygnał stanu wysokiego.



Rys. 10. Dane i sygnały sterujące w pamięci ROM
Fig. 10. Schematic diagram of the bus interface

MRint steruje załadowaniem adresu wejściowego do rejestru "latch" ABR oraz wyzwala element opóźniający TMR i bufor MRQ. MRQ steruje sygnałem "chip select" wszystkich pamięci ROM.

Wyjścia DANE elementu TMR wystawiają sygnał wysoki na 175 ns po zboczu MRint. Sygnał DANE utrzymuje stan niski sygnału BUSReq aż sygnał BUSAck nie zostanie odebrany z magistrali głównej. Sygnał magistrali BUSAck jest negowany i łącznie z sygnałem DANE tworzy iloczyn logiczny, który generuje sygnał Enable. Sygnał Enable jest negowany i stanowi sygnał wyjściowy DataRdy. Sygnał Enable ponadto wyzwala trójstanowe wyjście DBR. Element DBR steruje przepływem danych z pamięci.

Opis omawianego interfejsu magistrali zapisany w języku VHDL przedstawia się następująco:

```
USE Work.Defs.all;
entity Bus_Interface is
generic
    (Board_id: in wbit_vector);
port
    (
    ABUS:       in word;
    BBUS:       out byte;
    MemReq:     in wbit;
    BusReq:     out wbit;
    BusAck:     in bit;
    DataRdy:    out wbit;
    Addr:       out half_word;
    Data:       in byte;
    MR:         out wbit
    );
end Bus_Interface;

architecture Data_Flow of Bus_Interface is
    signal    Data      :wbit;
    signal    Enable    :wbit;
    signal    MeInt     :wbit;
    constant  ZS        :byte:="ZZZZZZZ";
begin
    RDY:    DataRdy<=not Enable  after 5ns;
    MRQ:    MR     <=not MRint after 5ns;
    ACK:    Enable <=Data and not BusAck after 5ns;
```

```
REQ:    BusReq <=not(Data and BusAck) after 5 ns;
TMR:    Data   <=MRint and MRint' deleyed(175 ns);
DBR:    with Enable select
        DBus <= Data after 9 ns when 1
          ZS after 7 ns when 0
          ZS  when others;
ABR:    block/(MRint='0'/
begin
    Addr= guarded ABus(0 to 15) after 5ns
end block;
end; --Data_Flow
```

Podobnie jak we wszystkich projektowanych jednostkach, opis jest podzielony na dwie części:

- deklaracje jednostek,
- architekturę.

Architekturę Data_Flow jednostki BusInterface deklarują trzy sygnały wewnętrzne modułu:

- Data
- Enable,
- MRint.

Stała ZS została zadeklarowana i zainicjowana jako tablica jednego bajtu reprezentująca stan wysokiej impedancji. Data_Flow używa stylu tu opisanego, a nazywanego "Data_Flow" w kontraście ze strukturalnym stylem uzywanym do opisu bloku pamięci. Każde urządzenie opatrzone etykietą w schemacie jest reprezentowane przez współbieżne oznaczenie sygnałów w linii dotyczącej Data_Flow.
Wszystkie wyznaczone współbieżne operacje są równolegle.

4. Wnioski końcowe

Przedstawione powyżej dane o języku opisu sprzetu typu VHDL wskazują na możliwości jego stosowania w projektowaniu ukladów złożonych systemów cyfrowych. Opracowanie dla tego języka standardu (standard IEEE) stwarza możliwości stosowania go

w projektowaniu i symulacji przez organizacje przemysłowe w Stanach Zjednoczonych i w innych rozwiniętych krajach przemysłowych.

LITERATURA

[1] VHDL Tutorial for IEEE Standard 1076 VHDL second draft, June 1987.
[2] IEEE Standard VHDL Language Reference Manual, 1989.
[3] H. Gizdoń, A. Pawlak, W. Wrona: VHDL ADA języków opisu i projektowania sprzętu, Informatyka nr 10, 1988.
[4] H. Gizdoń, A. Pawlak, W. Wrona: Język opisu sprzętu VHDL - Podstawowe Mechanizmy, Informatyka nr 11, 12, 1988.
[5] Część 2, artykuł poz. [4], Informatyka nr 1, 1989.

Recenzent: Doc. dr hab. inż. Adam Mrózek

Wpłynęło do Redakcji 11 września 1990 r.

Abstract

This article presents the Hardware Description Language VHDL. The VHDL language is a formal notation intended for use in all phases of the creation of electronic system (figure 1).
The design entity is the primary hardware abstraction in VHDL cell, chip, board or subsystem that has well defined inputs and outputs and performs a well defined function.
Each design entity is described in two parts the definition of the interface between the entity and the outside world, called the entity declaration, and a design for the input/output transformation itself, called the architecture body (figure 3).
VHDL supports three distinct styles for the description of hardware architectures. The first of these in structural description, in which the architecture is expressed as a hierarchical arrangement of interconnected components. The second is data-flow description in which the architecture is broken down into a set of concurrent register assignments, each of which may be under the control of gating signals.

Data flow description presents the style of description embodied in register transfer languages. The last is behavioral description in which the transformation is described in sequential program statments that looks like any modern, high level computer programing language. All three styles may be intermixed in a signal architecture (figure 5).
The paper ilustrates structural description of simple memory board (figure 9).