SNOVANJE DIGITALNIH VEZIJ Z OPISNIM JEZIKOM VHDL

Marjan Štrakl, Zmago Brezočnik, Bogomir Horvat, Tatjana Kapus
Laboratorij za digitalne in informacijske sisteme
Tehniška fakulteta Maribor, Elektrotehnika, računalništvo in informatika

Ključne besede: jeziki računalniški, vezja digitalna, VHDL jezik, opis jezika, razvoj jezikov, omejitve pri snovanju, jeziki opisni, snovanje vezij digitalnih, strukture vezij, arhitektura vezij, obnašanje vezij, načrtovanje vezij

Povzetek: Članek predstavi standardizirani jezik VHDL za opis digitalnih vezij. V VHDL-u lahko opišemo obnašanje in/ali strukturo vezja. Tako obnašanje kot strukturo lahko predstavimo na različnih abstraktnih nivojih, od arhitekturnega nivoja do nivoja logike. Prvi koraki razvoja VHDL-a so bili narejeni v zgodnjih osemdesetih letih. Končna verzija predlaganega jezika, poznana kot VHDL Verzija 7.2, je bila pripravljena leta 1985 in leta 1987 sprejeta kot standard IEEE-1076. Glavne prednosti VHDL-a so: javna dostopnost, podpora različnih metodologij načrtovanja in načinov implementacije vezja, neodvisnost od tehnološkega procesa, možnost opisa na različnih abstraktnih nivojih, prenosljivost opisov, modularnost, ponovna uporabljivost in podpora vlade ZDA. Ker je VHDL standardiziran, je na voljo vedno več načrtovalskih orodij, ki uporabljajo za vhod opise vezij v VHDL-u. Zadnja standardizirana verzija jezika je IEEE-1076-1993. Trenutno se razvija razširitev VHDL-a za opis analognih vezij.

Digital Circuit Design with VHDL Description Language

Keywords: computer languages, digital circuits, VHDL language, language description, language development, design limitations, digital circuit design, circuit structure, circuit architecture, circuit behaviour, circuit planning

Abstract: This paper presents the standardised VHDL digital hardware description language. VHDL is a language for describing digital electronic systems. It arose out of the United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1990. There was a need for standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC Hardware Description Language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE).

VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design, that is how it is decomposed into sub-design, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language constructs. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

A history of VHDL can be separated into three phases: the definition phase, the development phase and the deployment phase. In the definition phase the basic concepts and elements that form the character of the hardware description language were defined. At the same time the academic and industrial environments necessary for the eventual acceptance of VHDL as an industry standard were cultivated. The principal activity of the development phase has been the development of software to support the use of VHDL for electronic product design and documentation. After successful development the deployment phase began. In this phase VHDL can be widely used in industry, university education and research. The final version of the language, known as VHDL Version 7.2, was made available and standardised by IEEE as IEEE Standard 1076 in 1985 and 1987, respectively. In 1992 IEEE revised the standard. Some new features in signal assignment, new predefined operators, functions and attributes, and detailed specified timing model needed for the simulation cycle were added. These revisions were accepted as IEEE-1076-1993 standard.

Circuit description in VHDL can be behavioural and/or structural. Each kind of description covers different abstraction levels, from the architecture level down to the logic level. The full set of design representations that a VHDL can specify is described in the followig matrix.

Level	Behavioural domain	Structural domain
Architectural	Perforn1ance specifications	Logical connections of processors, memories, controllers, buses
Algorithmic	Procedural behaviour, manipulation of data structures	Data structures, procedural partitions
Functional	Concurrent operations, register transfer, state sequencing	Physical connections of functions, including arithmetic and logic units, multiplexers, and registers
Logic	Boolean equations	Physical connection of gates,latches

VHDL represents the full behavioural and structural domains. The behavioural domain describes what a circuit or device must do. The structural domain describes a component's logical layout: partitioning, decomposition (hierarchy), and interconectivity.

The main advantages of VHDL are: public availability, different design methodology and technology support, technology and process independence, wide range of descriptive capability, design exchange, modularity, design reuse and USA government support. A few years ago VHDL was used only in academic societies, but now powerful tools for making integrated circuits that use VHDL entry description are available. The advantage of VHDL in comparison to other hardware description languages is that VHDL is standardised by IEEE and it is in public availability.
Currently, VHDL analog extensions are under development.

1. Uvod

VHDL (VHSIC Hardware Description Language) je standardizirani jezik za opis digitalnih vezij /1,9/. Je rezultat programa VHSIC (Very High Speed Integrated Circuits), začetega leta 1980, katerega cilj je bil razvoj zelo hitrih integriranih vezij. V okviru tega programa se je pojavila potreba po standardnem jeziku za opis tako strukture kot tudi obnašanja integriranih vezij.

VHDL zapolnjuje številne potrebe v procesu načrtovanja vezij. Omogoča opis strukture vezja, v katerem se poda, iz katerih modulov je vezje zgrajeno in kako so ti moduli med seboj povezani. Omogoča tudi specifikacijo funkcije vezja na način, ki je podoben standardnim programskim jezikom. Opis vezja lahko simuliramo ter tako hitro primerjamo alternativne rešitve in testiramo pravilnost zasnove še pred izdelavo prototipa. Tako obnašanje kot strukturo lahko predstavimo na različnih abstraktnih nivojih, od arhitekturnega nivoja do nivoja logike.

Postavlja se vprašanje, ali je za predstavitev vezja primerneje uporabiti opisni jezik ali shematski opis. Nekateri načrtovalci menijo, da je shematski opis najprimernejši, vendar postanejo sheme nepregledne, ko vezje preseže nekaj 10000 logičnih vrat. Zato se vedno več načrtovalcev odloča za uporabo jezikov za opis vezij.

Članek najprej predstavi nastanek in razvoj jezika VHDL, njegove značilnosti in prednosti. V nadaljevanju so opisani glavni konstrukti jezika VHDL in primeri opisov obnašanja in strukture vezja. Sledi razprava o primernosti izbire jezika VHDL za načrtovanje digitalnih vezij. Na koncu so opisane še omejitve pri načrtovanju z jezikom VHDL ter njegov nadaljnji razvoj.

2. Razvoj jezika VHDL

Razvoj jezikov za opis vezij se je pričel v šestdesetih letih, ko so nastali prvi višji programski jeziki. Prvi jezik za opis vezij je bil CDL (Computer Design Language) /5/. CDL ima nekatere konstrukte programskih jezikov in objekte, ki so gradniki digitalnih vezij (registri, dekodirniki, stikala, urni signali, ...). Nekoliko kasneje je nastal jezik DDL (Digital System Design Language), ki že ima možnost opisa vezja na arhitekturnem nivoju in na nivoju Boolovih funkcij /5/. V sedemdesetih in osemdesetih letih je prišlo do poplave najrazličnejših jezikov za opis vezij. S tem je nastajala zmeda, saj jeziki med seboj niso bili združljivi.

Zgodovino jezika VHDL lahko razdelimo na tri faze: fazo definicije jezika, fazo razvoja in fazo uporabe /6/. Faza definicije jezika je potekala približno od leta 1980 do 1986. V njej so bili definirani osnovni koncepti in značilnosti jezika. Za začetke definiranja VHDL-a označujemo znanstveno delavnico *The Hardware Description Language Summer Study*, ki je bila junija 1981 v mestu Woods Hole v Massachusettsu /10/. Tam so izoblikovali priporočila za jezik, ki bi bil primeren za opis digitalnih sistemov. Na osnovi teh priporočil je prevzelo iniciativo obrambno ministrstvo ZDA in določilo program razvoja VHDL-a. Formalno se je delo na definiciji jezika pričelo poleti 1983, ko je obrambno ministrstvo ZDA vzpostavilo stik s podjetji Intermetrics, Texas Instruments in IBM. Ta podjetja so definirala osnovne konstrukte jezika. Njihovo delo je kulminiralo poleti leta 1985, ko je bila predstavljena definicija jezika VHDL Verzija 7.2.

Druga faza, faza razvoja jezika, sega približno od leta 1986 do 1993. Glavni dejavnosti v tej fazi sta standardizacija jezika in razvoj programskih orodij, ki uporabljajo za načrtovanje in dokumentiranje digitalnih vezij opis v VHDL-u. Po definiciji VHDL-a je njegovo standardizacijo pričel IEEE. V letu 1986 je IEEE sponzoriral več srečanj, kjer bi se definiral jezik za opis vezij. Osnova je bila definicija VHDL Verzija 7.2. Decembra 1987 je bil sprejet standard IEEE-1076, ki opisuje definicijo jezika VHDL.

Po definiciji in razvoju VHDL-a se prične faza njegove uporabe. Za pričetek uspešne uporabe morajo obstajati ustrezna programska orodja in interes načrtovalcev za uporabo. Javna last VHDL-a in standardiziranost so prispevali k temu, da se VHDL pričenja množično uporabljati.

3. Hierarhični opis vezja

Z VHDL-om lahko opisujemo vezje na različnih abstraktnih nivojih. Pri opisu zasnove digitalnega vezja imamo dve domeni: domeno obnašanja in domeno strukture. Z opisom obnašanja vezja definiramo, *kaj* vezje dela. Obnašanje opišemo s podatkovnimi strukturami in funkcijskimi preslikavami med vhodnimi in izhodnimi vrednostmi na priključkih vezja. Z opisom strukture vezja definiramo komponente vezja in njihove medsebojne povezave, torej kako vezje opravlja zahtevano funkcijo. Domeni opisa vezja in nivoji abstrakcije v VHDL-u so prikazani na sliki 1.

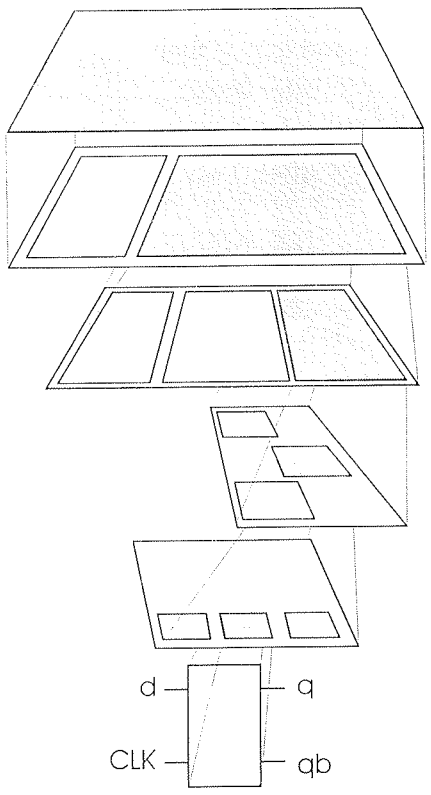
nivo	domena obnašanja	domena strukture
arhitektura	specifikacija zmogljivosti	logične povezave med procesorji, pomnilniki, krmilniki, vodili
algoritem	proceduralno obnašanje, manipulacije s podatkovnimi strukturami	podatkovne strukture, dekompozicija procedur
funkcije	sočasne operacije, prenosi med registri, prehajanje med stanji	fizične povezave med funkcionalnimi enotami (aritmetične in logične enote, multipleksorji in registri)
logika	Boolove enačbe	fizične povezave med logičnimi vrati in zadrževalniki

Slika 1: Domeni opisa vezja in nivoji abstrakcije v VHDL-u

Z opisom obnašanja na nivoju arhitekture specificiramo zmogljivosti celotnega digitalnega sistema, na nivoju algoritma opišemo proceduralno obnašanje in manipulacije nad podatkovnimi strukturami, na funkcionalnem nivoju sočasne operacije, prenose med registri in preha-

janje med stanji. Obnašanje na najnižjem nivoju, nivoju logike, opišemo z Boolovimi enačbami. Tudi strukturo vezja lahko opišemo na istih abstraktnih nivojih. Na nivoju arhitekture opišemo logične povezave med procesorji, pomnilniki, krmilniki, vodili in drugimi večjimi logičnimi enotami, ki predstavljajo neko zaključeno celoto. Na nivoju algoritma opišemo podatkovne strukture in dekomponiramo opise procedur. Na funkcionalnem nivoju opišemo fizične povezave med funkcijskimi enotami, kot so npr. aritmetične in logične enote, multipleksorji in registri. Strukturo na nivoju logike opišemo s fizičnimi povezavami med logičnimi vrati in zadrževalniki.

Možnost opisovanja vezja na različnih abstraktnih nivojih omogoča, da opis vezja hierarhično dekomponiramo. Vezje najprej opišemo na nivoju arhitekture. Takšen opis lahko hitro simuliramo in ugotovimo morebitne napake pri zasnovi vezja. Če vezje deluje po zahtevani specifikaciji, lahko posamezne enote vezja dekomponiramo in jih opišemo na nižjih nivojih. Dekompozicija pa ni nujno potrebna za vse enote vezja. Na nižjih nivojih lahko opišemo samo posamezne enote vezja, druge pa pustimo na višjem nivoju. Tako lahko imamo v opisu vezja različne nivoje abstrakcije.




```
signal CLK, q, qb, d : bit;
D_FF : block (CLK = '1')
    signal s : bit;
begin
    s <= guarded d;
    q <= s;
    qb <= not s;
end block D_FF;
```

Slika 2: Hierarhični pristop k načrtovanju vezja

Hierarhični pristop k načrtovanju vezja je ilustriran s primerom na sliki 2 /13/. S takšnim pristopom dobimo hierarhičen opis vezja, v katerem je na nivoju i opisana specifikacija komponente, na sosednjem nižjem nivoju i-1 pa implementacija komponente.

4. Glavni konstrukti jezika VHDL

Opis celotnega vezja ali njegovega dela je sestavljen iz:

- *osebka* in
- ene ali več *arhitektur.*

V osebku opišemo zunanji vmesnik vezja in deklaracije, ki se nanašajo na vse arhitekture, ki pripadajo temu osebku. Z arhitekturami opisujemo obnašanje ali strukturo vezja.

Splošni zapis osebka ima naslednjo obliko:

entity *identifikator* is
    deklaracije
begin
    stavki
end *identifikator*

V deklaracijah navedemo ime, način in podatkovni tip posameznega priključka vezja, po potrebi pa deklariramo še podatkovne tipe, podtipe, konstante, notranje signale, podprograme, ... Dovoljeni načini priključkov so vhodni, izhodni, vhodno-izhodni in vmesniški. Vmesniški način se uporablja za izhodne priključke pomnilniških elementov. VHDL pozna naslednje podatkovne tipe: skalarne (celoštevilčni, realni, naštevalni in fizični), sestavljene (zapis in polje), kazalčne in datotečne. Uporabnik lahko sam deklarira nove tipe in podtipe. Podtip je podmnožica elementov nekega tipa. Posebej uporaben podatkovni tip je polje, saj ga lahko uporabljamo za krajši in preglednejši prikaz vrednosti signalov na vodilih.

Stavki v deklaraciji osebka niso namenjeni opisu dinamičnega obnašanja komponente, ampak preverjanju zunanjega vmesnika komponente. Z njimi lahko npr. preverjamo, če sta oba vhoda pomnilniške celice SR na logični enici.

Deklaracije osebkov so obvezne, saj specificirajo *zunanji pogled* na vezje in nekatere skupne lastnosti več različnih opisov iste komponente vezja. Zunanji pogled pomeni, da opazujemo komponento kot *črno škatlo* in vidimo samo njene priključke.

Primer deklaracije osebka *vrata_XOR*:

```
entity vrata_XOR is
    port (
        In1, In2 : in Bit;
        Out1 : out Bit );
    constant Delay : Time := 5 ns;
end vrata_XOR;
```

V zgornji deklaraciji deklariramo osebek *vrata_XOR*, ki ima vhodna priključka *In1* in *In2* tipa Bit in izhodni priključek *Out1* tipa Bit. Poleg priključkov je v zgornjem primeru deklarirana še konstanta *Delay* tipa Time z vrednostjo 5 ns. To konstanto lahko uporabljamo v vseh opisih arhitektur *vrata_XOR*.

Splošni zapis arhitekture ima naslednjo obliko:

architecture *identifikator* **of** *ime_osebka* **is**
    deklaracije
begin
    stavki
end *identifikator*;

V glavi arhitekture deklariramo spremenljivke, konstante, signale in druge konstrukte, ki veljajo samo znotraj arhitekture. Notranji signali predstavljajo povezave med posameznimi elementi opisane arhitekture. Pri deklaraciji signalov moramo navesti tip podatka, ki ga prenašajo.

Za zgled si oglejmo primer deklaracij v arhitekturi *opis1* za vezje z imenom *vezje*:

```
architecture opis1 of vezje is
    variable temp1, temp2 : Bit;
    constant Delay : Time := 7 ns;
    signal select : Bit_vector (2 downto 0);
    signal input, output :
        Bit_vector (7 downto 0);
begin
    .
    .
    .
end opis1;
```

V deklaracijah smo navedli, da v arhitekturi nastopata spremenljivki *temp1* in *temp2* tipa Bit, konstanta *Delay* tipa Time z vrednostjo 7 ns, signal *select*, ki predstavlja polje treh bitov, ter signala *input* in *output*, ki predstavljata polje osmih bitov.

S stavki v telesu arhitekture opisujemo obnašanje ali strukturo vezja. V zgornjem primeru bi lahko arhitektura *opis1* pomenila ali opis obnašanja ali opis strukture vezja *vezje*. Če bi želeli, bi lahko vezje opisali še z alternativnimi arhitekturami za obnašanje in/ali strukturo.

4.1 Opis obnašanja vezja

Obnašanje vezja opišemo s funkcijsko odvisnostjo vrednosti na izhodnih priključkih od vrednosti na vhodnih priključkih vezja. Pri opisu se ne spuščamo v podrobnosti o tem, kako je vezje sestavljeno, ampak gledamo na posamezne module vezja kot na črne škatle in opišemo samo akcije, ki se dogajajo na izhodnih priključkih vezja.

Obnašanje vezja lahko opišemo na dva načina: na *sekvenčni* ali *sočasni* način /11/. Pri sekvenčnem načinu opišemo obnašanje vezja s procesi, pri sočasnem načinu pa s sočasnimi prireditvami izhodnih vrednosti.

Proces je opis obnašanja dela vezja, običajno neke logične enote, ki ga pri opisu nima smisla deliti v manjše enote. Proces se izvede, ko postane aktiven, aktiven pa postane, ko se izpolnijo določeni pogoji. Pogoji so lahko spremembe vhodnih signalov ali ustrezni časovni pogoji. Več procesov se lahko izvaja sočasno in neodvisno drug od drugega, če le izpolnjujejo pogoje za svojo aktivnost. S procesi prirejamo vrednosti signalom na izhodnih priključkih vezja. VHDL omogoča specifikacijo časovnih zakasnitev pri prirejanju izhodnih signalov. Kot pri programskih jezikih lahko tudi v VHDL-u uporabimo krmilne stavke, kot sta npr. stavka *if-then-else* in *case* ter zanki *loop* in *for*.

Za zgled opišimo obnašanje vrat *vrata_XOR* s procesom:

```
proces_XOR : process
begin
    if In1='0' and In2='0' then
        Out1 <='0' after Delay;
    elsif In1='1' and In2='1' then
        Out1 <='0' after Delay;
    elsif In1='X' or In2='X' then
        Out1 <='X' after Delay;
    else
        Out1 <='1' after Delay;
    end if;
    wait on In1, In2;
end process;
```

Če sta na obeh vhodih enaki logični vrednosti, '0' ali '1', bo izhodu prirejena vrednost '0'. Če je na katerem vhodu vrednost 'X' (nedoločena vrednost), vrednosti izhoda ni mogoče določiti (priredimo mu vrednost 'X'), v vseh drugih primerih pa je vrednost izhoda '1'. Vrednosti se izhodom priredijo po preteku zakasnitve *Delay*. Opis procesa je zaključen s stavkom **wait on**, ki pomeni, da se bo proces ponovno aktiviral šele, ko se bo spremenila vrednost vsaj enega od naštetih vhodov.

Pri sočasnem modeliranju obnašanja opišemo sočasne prireditve izhodnih vrednosti. V tem primeru ni nobenih pogojev, ki bi prožili določeno akcijo. Tudi pri sočasnem modeliranju obnašanja imamo podobne prireditvene in krmilne stavke kot pri sekvenčnem modeliranju.

Za zgled opišimo obnašanje vrat *vrata_XOR* še s sočasnimi prireditvami:

```
Out1 <=
    '0' after Delay when
        In1='0' and In2='0' else
```

```
    '0' after Delay when
        In1='1' and In2='1' else
    'X' after Delay when
        In1='X' or In2='X' else
    '1' after Delay;
```

Zgornji opis je funkcionalno ekvivalenten opisu obnašanja na sekvenčni način s procesom. Iz primera je razvidno, da je opis pri sočasnem modeliranju obnašanja krajši in tudi preglednejši.

Za konec si oglejmo še, kako bi v VHDL-u opisali obnašanje preprostega sekvenčnega vezja. Prikazana je definicija zunanjega vmesnika in opis obnašanja pomnilne celice T s procesom:

```
entity T_FF is
    port(CLK : Bit; T : Bit;
        QOut :buffer Bit);
end T_FF;

architecture obnasanje of T_FF is
    signal s : Bit;
begin
    proces_T : process(CLK)
    begin
        if not CLK'Stable and CLK='1' then
            s <=T xor s;
            QOut <=s;
        end if;
    end process;
end obnasanje;
```

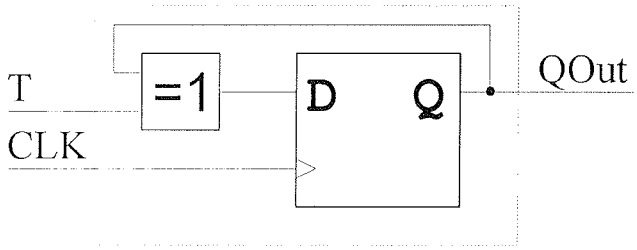
V opisanem primeru se proces aktivira ob vsakem prehodu ure CLK, signalu s pa se priredi nova vrednost le ob pozitivnem prehodu ure.

4.2 Opis strukture vezja

Strukturo vezja opišemo s komponentami vezja in njihovimi medsebojnimi povezavami. Iz strukture je razvidna notranja zgradba vezja.

Pri opisu strukture vezja je osnovni stavek stavek o deklaraciji in povezovanju komponent. Komponente so tisti osnovni gradniki vezja, ki jih ne razbijamo v bolj podrobne opise ali pa so podrobnejši opisi v ustreznih knjižnicah. Imena komponent in njihove zunanje vmesnike opišemo s stavki **component** med seboj pa jih povezujemo s stavki **portmap**.

V razdelku 4.1 smo že definirali zunanji vmesnik in opisali obnašanje pomnilniške celice T. Zdaj opišimo še njeno strukturo. Pomnilniško celico T lahko zgradimo npr. iz pomnilniške celice D in vrat XOR, kot je to prikazano na sliki 3.



Slika 3: Struktura pomnilniške celice T

Strukturo opisuje naslednja arhitektura:

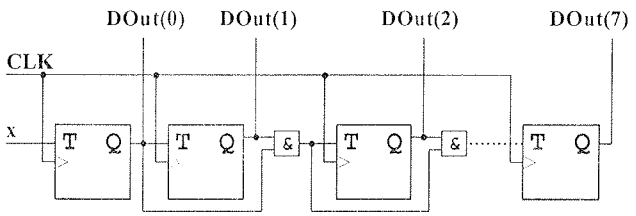
```
architecture struktura of T_FF is
    component vrata_XOR
        port (In1, In2 : Bit; Out1 : out Bit);
    end component;
    component D_FF
        port (CLK : Bit; D : Bit;
            Q : buffer Bit);
    end component;
    signal S1 : Bit;
begin
    vrata : vrata_XOR port map (
        In1 => T,
        In2 => QOut,
        Out1 => S1);
    FF : D_FF port map (
        CLK => CLK,
        D => S1,
        Q => QOut);
end struktura;
```

Stavek **architecture** pomeni, da opisujemo arhitekturo vezja. S prvim stavkom **component** smo deklarirali komponento, v našem primeru vrata *vrata_XOR* z vhodnima priključkoma *In1* in *In2* tipa Bit ter izhodnim priključkom *Out1* tipa Bit. Z drugim stavkom **component** je deklarirana še pomnilniška celica *D_FF*. S signalom S1 je predstavljena povezava med izhodom vrat in vhodom pomnilne celice. Povezave signalov s priključki komponente se opišejo s stavkom **port map**, ki mu sledijo pari

ime priključka komponente => ime signala.

Oznaki *vrata* in *FF* predstavljata simbolični imeni za komponenti *vrata_XOR* in *D_FF*.

Pri opisu strukture vezja imamo na voljo še dodatne stavke, ki nam olajšajo opis zahtevnejših vezij s *pravilnimi strukturami*. Pravilne strukture imenujemo tiste dele vezja, ki se v celotnem vezju večkrat ponovijo. To so lahko registri, večbitni števniki, ... Uporabnost stavkov, ki opisujejo pravilne strukture, je prikazana na primeru 8-bitnega števnika s slike 4:



Slika 4: 8-bitni števnik

```
entity stevnik is
    port (CLK : Bit; x : Bit;
        DOut : buffer
            Bit_vector (7 downto 0));
end stevnik;

architecture struktura of stevnik is
    component TFF
        port (CLK : Bit; T : Bit;
            Q : buffer Bit);
    end component;
    component And2
        port (I1, I2 : Bit; O1 : out Bit);
    end component;
    signal S : Bit_vector(7 downto 0) ;
    signal x : Bit := '1';
begin
    G1 : for I in 7 downto 0 generate
        G2 : if I=7 generate
            TFF_7 : TFF port map (
            CLK => CLK,
            T => S(I-1),
            Q => DOut(I));
        end generate;
        G3 : if I=0 generate
            TFF_0 : TFF port map (
            CLK => CLK,
            T => x,
            Q => DOut(I));
            S(I) <= DOut(I);
        end generate;
        G4 : if I>0 and I<7 generate
            And_1 : And2 port map (
            I1 => S(I-1),
            I2 => DOut(I),
            O1 => S(I));
            TFF_1 : TFF port map (
            CLK => CLK,
            T => S(I-1),
            Q => DOut(I));
        end generate;
    end generate;
end struktura;
```

Poleg prikazanih možnosti opisovanja strukture in obnašanja ima VHDL še številne druge konstrukte, kot so funkcije in procedure, bloki, knjižnice, pretvorba podatkov med različnimi tipi, itd.

5. Zakaj izbrati VHDL?

VHDL ima pred drugimi jeziki za opis vezij številne prednosti /9/:

- *Javna last:* VHDL je bil razvit s podporo vlade ZDA in je sedaj standard organizacije IEEE. Pri razvoju je imela vlada ZDA interes, da je VHDL v javni lasti in zanj ni potrebno plačevati licenc.

- *Podpora različnih metodologij načrtovanja in načinov implementacije vezja:* VHDL podpira več različnih metodologij načrtovanja (načrtovanje od zgoraj navzdol, načrtovanje od spodaj navzgor, uporaba knjižnic, ...) in različnih načinov implementacije (sinhrona vezja, asinhrona vezja, PLA, FPGA, ...). Opisni jezik VHDL je primeren za uporabo tako pri načrtovalnih orodjih, ki temeljijo na uporabi knjižnic, kot tudi pri orodjih za načrtovanje vezij ASIC.

- *Neodvisnost od tehnološkega procesa*: Opis vezja v VHDL-u je neodvisen od tehnološkega procesa izdelave integriranih vezij in zato nove tehnologije ne omejujejo njegove uporabe. Pri opisu ne navajamo, v kateri logični družini (npr. CMOS, NMOS, GaAs) bo vezje implementirano.

- *Opis na različnih abstraktnih nivojih*: VHDL omogoča opisovanje digitalnega vezja na različnih abstraktnih nivojih, od arhitekturnega nivoja do nivoja logičnih vrat. Ena od glavnih prednosti VHDL-a je možnost hkratnega opisa vezja na različnih nivojih.

- *Prenosljivost opisov*: Ker je VHDL standardiziran, je mogoče opise vezij v VHDL-u prenašati med različnimi sistemi in orodji, ki standard podpirajo.

- *Modularnost in ponovna uporabljivost*: VHDL je zasnovan podobno kot sodobni programski jeziki in omogoča dekompozicijo podatkovnih struktur in uporabo knjižnic z že izdelanimi komponentami vezja. Njegova modularnost dopušča skupinsko delo, kjer vsak načrtovalec razvija del vezja. Na koncu se opisi vseh načrtovalcev združijo v en sam opis.

- *Podpora vlade ZDA*: VHDL je nastal pod okriljem obrambnega ministrstva ZDA in zato morajo biti vsa integrirana vezja, ki so namenjena za potrebe vlade ZDA, opisana v VHDL-u. Posledica tega je, da se vedno več načrtovalcev odloča za VHDL in ne za kateri drugi jezik.

Ker VHDL ni edini jezik za opis vezij, se postavlja vprašanje, zakaj uporabljati VHDL in ne katerega drugega jezika. To vprašanje je bilo posebej očitno pred nekaj leti, ko na tržišču še ni bilo orodij, ki bi sprejemala opis vezja v VHDL-u in so ga uporabljali le v akademskih krogih. Vzrok za zakasnitev nastanka ustreznih orodij izhaja iz dejstva, da je VHDL standard za opisni jezik in ne za produkt katerega proizvajalca programskega orodja. Do nastanka VHDL-a je praviloma imel vsak proizvajalec programske opreme svoj opisni jezik, ki v večini primerov ni bil kompatibilen z drugimi. S povečevanjem ponudbe je na tržišču nastala zmeda, saj uporabnik ni vedel, za katero orodje in s tem opisni jezik naj se odloči. Tako se je pojavila potreba po standardizaciji in VHDL je bil sprejet kot standard IEEE-1076. Počasi so nastajala tudi programska orodja za VHDL, ki pa niso v celoti podpirala standarda. Iz standarda je bila izbrana samo določena podmnožica lastnosti. S tem je sicer bila možna njegova delna uporaba, vendar ni bil odpravljen eden od glavnih razlogov za njegov nastanek, to je prenosljivost opisov vezij med programsko opremo različnih proizvajalcev. Šele sedaj se na tržišču pojavljajo orodja, ki standard v celoti podpirajo.

Raziskava med 435 načrtovalci integriranih vezij v začetku leta 1992 je pokazala, da je tedaj 46% načrtovalcev uporabljalo opisni jezik Verilog in 39% VHDL /8/. Ista raziskava je pokazala, da kar 69% načrtovalcev namerava v prihodnosti uporabljati orodja na osnovi VHDL in 37% Verilog. To pa vsekakor pomeni, da bo v prihodnjih letih VHDL prevladal na tržišču.

6. Omejitve pri načrtovanju z jezikom VHDL

Čeprav ima opis vezja z jezikom VHDL številne prednosti pred drugimi načini opisa vezij, ima tudi nekatere slabosti in omejitve. Osnova jezika je bila postavljena s standardom IEEE-1076, ki pa je dokaj ohlapen in dopušča načrtovalcu, da jezik dograjuje s svojimi komponentami, funkcijami, procedurami in tipi podatkov. Takšni dodatki se običajno ne podajajo v samem opisu vezja, ampak so v knjižnicah. To pa pomeni, da se zmanjša prenosljivost opisov, saj morata imeti izvorno in ciljno orodje ekvivalentni knjižnici.

Druga slabost je ta, da lahko imajo opisi na različnih abstraktnih nivojih različno abstrakcijo podatkov. Primer za to je opis seštevalnika, kjer lahko obnašanje opišemo s stavkom

```
Z <= X + Y;
```

in pri tem sploh ne določimo števila bitov v sumandih, pri opisu strukture pa moramo imeti tudi to informacijo, ki je zelo pomembna pri uporabi orodij za avtomatsko sintezo vezja iz podanega obnašanja vezja. Pri reševanju tega problema so izdelovalci orodij izbrali dve poti. Prva je ta, da moramo pri opisu obnašanja uporabljati samo logične operatorje. S tem standard ni v celoti implementiran. Po drugem načinu se v takšnih primerih doda informacija o številu bitov, nad katerimi deluje operator, kot komentar. S tem se lahko izgubi prenosljivost, saj mora ustrezno orodje to informacijo znati uporabiti in je ne sme prezreti kot komentar.

Naslednji problem je časovno zaporedje prirejanja vrednosti signalom. Problem je razviden iz naslednjega primera:

```
P1 : process (x,y,w)
begin
    z <=x + y;
    q <=z + w;
end process P1;

P2 : process (x,y,w)
begin
    wait until clock'event and clock = '1';
    z <=x + y;
    wait until clock'event and clock = '1';
    q <=z + w;
end process P2;
```

V procesu P1 se izhodoma *z* in *q* priredi vrednost hkrati, v procesu P2 pa dobi *z* končno vrednost po prvem urnem impulzu, *q* pa šele po drugem urnem impulzu. Čeprav je končni rezultat v obeh primerih enak, pa proces P2 opisuje sekvenčno vezje, iz opisa procesa P1 pa lahko sklepamo, da gre za kombinacijsko vezje. Nekatera orodja imajo dodane *predprocesorje RTL (register-transfer level)*, ki iz opisa vezja razpoznajo, ali gre za opis sekvenčnega vezja, in nato v primerih, kot je zgornji, dodajo ustrezne pomnilniške elemente (registre in zadrževalnike) /12/.

Nekatera orodja ne dopuščajo specifikacije zakasnitve izhodne vrednosti s stavkom **after**. Vzrok za to omejitev je v dejstvu, da orodja za sintezo ne morejo zagotoviti, da se bo izhodna vrednost spremenila natanko po specificirani zakasnitvi.

7. Analogni VHDL

S sedaj veljavno definicijo VHDL-a je mogoče opisovati samo digitalna vezja. Želja načrtovalcev integriranih vezij pa je univerzalno orodje za načrtovanje tako digitalnih kot tudi analognih vezij in kombinacije obeh. Zaradi tega se v zadnjem času posveča precej pozornosti razširitvi jezika VHDL, ki bi omogočala opis analognih vezij in tudi nekaterih elektromehanskih sistemov. Takšna razširitev jezika bi nudila načrtovalcu enotno orodje za načrtovanje in opis kateregakoli električnega sistema. Z nadgradnjo VHDL-a se ukvarja več skupin, vendar njihovo delo ni skupno in je tako predstavljenih več različic analognega VHDL-a. Ena od njih je VHDL-A /7/. Značilnosti tega jezika so:

- enoten opis analogno-digitalnih in neelektričnih sistemov,
- močan jezik za modeliranje časa,
- popolna podpora hierarhičnemu načrtovanju,
- transparentna delitev na digitalni in analogni del sistema in
- možnost snovanja v frekvenčnem prostoru.

Odprta vprašanja, ki pri tej razširitvi še ostajajo, so:

- ni upoštevanja medsebojnega elektromagnetnega vpliva elementov,
- ne moremo modelirati ultravisokih frekvenc in mikrovalovnih sistemov in
- ne moremo modelirati in simulirati šuma.

8. Nadaljnji razvoj jezika

Nadaljnji razvoj jezika VHDL skuša odpraviti nekatere slabosti, ki so se pojavile v dosedanji praksi. Prvi korak je dopolnitev obstoječega standarda, ki poenoti oznake za logične nivoje. Pred tem je lahko vsak načrtovalec uporabljal nabor logičnih nivojev, ki mu je v danem problemu najbolj ustrezal. Tako je bila osnova dvonivojska logika (nivoja '0' in '1'), ki se je razširila v trinivojsko ('0', '1' in 'X'), štirinivojsko ('0', '1', 'X' in 'Z') in večnivojsko. To je onemogočalo združitev celotnega opisa vezja v primerih, kjer je del vezja opisan z eno logiko, del pa z drugo. Marca 1993 je bil postavljen standard IEEE-1164, ki za opise vezja definira devet logičnih nivojev ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H' in '-') /2/. 'U' predstavlja neinicializirano stanje, 'X' neznano vrednost, '0' logično ničlo, '1' logično enico, 'Z' stanje visoke impedance, 'W' šibko nepoznano vrednost, 'L' šibko logično ničlo, 'H' šibko logično enico in '-' nepomembno vrednost.

Leta 1992, pet let po sprejemu prvega standarda, se je pričela revizija jezika. Dodana so bila nekatera dopolnila glede prirejanja vrednosti signalom in podrobneje obdelan časovni model opisa, ki je potreben za pravilno delovanje simulacijskega cikla. Dopolnila so bila sprejeta kot standard IEEE-1076-1993. Za ta standard se pogosto uporablja oznaka VHDL'93 ali VHDL'92 /3/.

Razvoj poteka tudi v smeri določitve standardnih knjižnic /4/. S tem bi se dejansko poenotili opisi vezij različnih načrtovalcev in tako postali resnično popolnoma prenosljivi med načrtovalskimi orodji različnih proizvajalcev.

Literatura

/1/ Ashendern P. J., The VHDL Cookbook, 1st ed., University of Adelaide, July 1990.

/2/ Billowitch W. D., IEEE 1164: helping designers share VHDL models. *IEEE Spectrum*, 37, June 1993.

/3/ Börger E., Glässer U., Müller W., The Semantics of Behavioral VHDL'93 Descriptions, *Proceedings of EURO-DAC'94/EURO-VHDL'94*, Grenoble, September 1994.

/4/ Carroll M., VHDL - panacea or Hype? *IEEE Spectrum*, 34-37, June 1993.

/5/ Chu Y., Three Decades of HDLs. *IEEE Design & Test of Computers*, 69-81, June 1992.

/6/ Dewey A., De Geus A. J., VHDL: Toward a Unified View of Design. *IEEE Design & Test of Computers*, 8-12, June 1992.

/7/ Fisher-Binder J.-O., VHDL Analog Extensions, 1993.

/8/ Jain M., The VHDL forecast. IEEE Spectrum, 36, June 1993.

/9/ Lipsett R., Schaefer C., Ussery C., VHDL: *Hardware Description and Design*. Kluwer Academic Publishers, 1989.

/10/ Marschner F. E., Evolutionary Processes in Language, Software, and System Design. *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, Kluwer Academic Publishers, 1992.

/11/ Mazor S., Langstraat P., A Guide to VHDL, 2nd ed., Kluwer Academic Publishers, 1993.

/12/ Nagasamy V., Berry N., Dangero C., Specifications, Planning, and Synthesis in a VHDL Design Environment. *IEEE Design & Test of Computers*, 58-67, June 1992.

/13/ Waxman R., Saunders L., Carter H., VHDL links design, test, and maintenance. *IEEE Spectrum*, 40-44, May 1989.

Marjan Štrakl, dipl.ing.,
Doc.dr. Zmago Brezočnik, dipl.ing.,
Prof. dr. Bogomir Horvat, dipl.ing.,
Doc. dr. Tatjana Kapus, dipl.ing.,
Tehniška fakulteta Maribor
Elektrotehnika, računalništvo in informatika,
Smetanova 17, 62000 Maribor, Slovenija,
tel. +386 62 25 461
fax + 386 62 225 013

Prispelo (Arrived): 31.01.95 ***Sprejeto (Accepted): 06.02.95***