9. DODATAK A: SAVETI ZA OPIS DIGITALNIH SISTEMA U VHDL-U

U ovom poglavlju je data kratka istorija VHDL jezika za opis fizičke arhitekture, kao i nekoliko praktičnih saveta za opis digitalnih sistema sa ciljem da se izbegnu greške koje se često ponavljaju.

9.1 KRATKA ISTORIJA I NAMENA VHDL-A

Veliki razvoj tehnologije integrisanih kola početkom 80-tih godina 20-tog veka doneo je potrebu za standardizacijom metodologije testiranja digitalnih sistema. VHDL jezik za opis fizičke arhitekture je nastao kao posledica ove potrebe. U međuvremenu, VHDL je postao industrijski standardni jezik za opis digitalnih sistema zahvaljujući tome što je postao zvanični IEEE standard. Prva verzija standarda je usvojena 1987. godine i nazvana je IEEE 1076. Tokom godina VHDL se razvijao i 1993. godine je donesena revizija standarda pod oznakom IEEE 1164. U inženjerskom svetu ova dva standarda se kratko zovu VHDL-87 i VHDL-93 prema godinama kada su usvojeni.

U početku, VHDL je bio zamišljen da služi za dve namene. Prva od njih je jezik za dokumentovanje opisa strukture složenih digitalnih sistema. Dokumentovanje složenih sistema isprojektovanih od strane više inženjera je bilo moguće obaviti na jedinstven način uz pomoć VHDL-a zahvaljujući tome što je VHDL postao IEEE zvanični standard. Pored toga VHDL poseduje mogućnost za modelovanje funkcionalnosti digitalnog sistema. To je omogućilo da se uz pomoć namenskih programskih alata simulira rad digitalnih sistema, gde je polazna tačka VHDL opis sistema.

Pored ove dve namene, VHDL opis digitalnog sistema se u međuvremenu počeo koristiti kao ulaz za CAD sisteme. CAD programski alati su korišćeni da sintetizuju VHDL kod u direktnu implementaciju fizičke arhitekture opisanog digitalnog sistema. Danas je ovakva upotreba VHDL-a standard u proizvodnji integrisanih kola.

9.2 KAKO NE PISATI VHDL KOD

Tokom učenja jezika za opis fizičke arhitekture kao što je VHDL, ili bilo koji drugi sličan jezik, početnici pokušavaju da pišu kod koji ima karakteristike računarskog programa, npr. C++ ili Java, sa mnogo promenljivih i programskih petlji. U tom slučaju je jako teško odrediti kakvu logiku će alati za sintezu da proizvedu. Generalno pravilo je da ako dizajner nije u stanju da čitajući VHDL kod odredi koju logiku on opisuje, tada alati za sintezu neće moći da generišu logiku koju dizajner pokušava da opiše.

Kada je kompletan VHDL kod određenog digitalnog sistema kreiran, zgodno je da se izanalizira rezultujući sistem proizveden sa alatima za sintezu. Mnogo toga se može naučiti o VHDL-u, opisu i sintezi digitalnih sistema analizom digitalnih sistema automatski generisanih uz pomoć alata za sintezu VHDL koda.

9.3 ČESTE GREŠKE U VHDL KODOVIMA

Nazivi entiteta i arhitektura:

Nazivi korišćeni u deklaraciji entiteta i njemu odgovarajuće arhitekture moraju biti identični. Kod prikazan u tekstu ispod sadrži grešku jer je u deklaraciji entiteta korišćen nativ `adder` dok je u zaglavlju arhitekture korišćen naziv `adder4`.

```
ENTITY adder IS

...

END adder;

ARCHITECTURE adder_arh OF adder4 IS

…

END adder_arh;
```

Nedostajući znak tačka-zarez:

Svaki VHDL iskaz se mora završiti sa znakom `;` .

Znakovi navoda:

Jednostruki znaci navoda (`' ... '`) se koriste za jednobitne podatke dok se za višebitne podatke koriste dvostruki znaci navoda (`" ... "`).

Iskazi koji se generišu izvan i u procesu:

Dodela vrednosti signalu i dodela vrednosti odredjenoj grupi bita jednog signala može biti izvršena i izvan procesa i u procesu. Isto važi i za osnovne logičke operacije tipa `AND`, `OR` i `NOT`.

Iskazi koji sadrže `IF`, `CASE` i `LOOP` ključne reči mogu se generisati samo u okviru procesa.

Instanciranje komponenti se izvršava isključivo izvan procesa.

Instanciranje komponenti:

Sledeći VHDL iskaz sadrži dve greške:

```
control: shiftr GENERIC MAP(K => 3);
                PORT MAP('1', Clock, w, Q);
```

Prva greška se nalazi na kraju prvog reda gde ne sme da se nalazi znak `;` iz razloga što je ove dve linije koda predstavljaju jedan VHDL iskaz. Takođe, nije dozvoljeno dodeljivati konstantnu vrednost bilo kom portu komponente koja se instancira.
U nastavku je prikazano kako se ove dve greške mogu ispraviti.

```
SIGNAL HIGH;

HIGH <= '1';
control: shiftr GENERIC MAP(K => 3)
                PORT MAP(HIGH, Clock, w, Q);
```

Nazivi labela, signala i promenljivih:

Nije dozvoljeno koristiti bilo koju VHDL ključnu reč kao naziv labele, signala ili promenljive. Na primer, pogrešno je koristiti signal sa imenom `In` ili `Out`. Takođe, pogrešno je koristi isti naziv više puta za bilo koju labelu, signal ili promenljivu u jednom VHDL dizajnu. Česta greška je da se upotrebljava isti naziv za signal i promenljivu koja se koristi za indeksiranje u okviru `GENERATE` ili `LOOP` iskaza. Na primer, ako je u kodu sledeći iskaz

```
Generate_label:
FOR i IN 0 TO 3 GENERATE
  bit: fa PORT MAP(C(i),X(i),Y(i),S(i),C(i+1));
END GENERATE;
```

tada nije dozvoljeno definisati signal sa imenom `i` (ili `I`, jer VHDL ne razlikuje mala i velika slova).

Implicitna memorija:

Implicitne memorijske lokacije se koriste kao kola za memorisanje vrednosti signala. Pri tome, ova kola su aktivna na nivo kontrolog (takt) signala a ne na ivicu signala takta kao što je slučaj sa flip-flopovima. Precizno rečeno, implicitne memorijske lokacije su lečevi (engl. *latch*).
Tokom pisanja VHDL koda mora se voditi računa da se ne opisuju neželjeni implicitni memorijski elementi. Tako na primer sledeći kod

```
IF LA = '1' THEN
  EA <= '1';
END IF;
```

rezultuje implicitnim memorijskim elementom na signalu `EA`. U slučaju da to nije potrebno, ispravka se može uraditi na sledeći način:

```
IF LA = '1' THEN
  EA <= '1';
ELSE
  EA <= '0';
END IF;
```

Implicitne morije se takođe javljaju kod CASE iskaza.

```
CASE y IS
  WHEN S1 => EA <= '1';
  WHEN S2 => EB <= '1';
END CASE;
```

Prikazani kod ne definiše vrednost EA signala kada y nije jednak S1. Takođe, nije definisana vrednost EB signala kada y nije jednak S2. Da bi se zaobišlo generisanje implicitne memorije za oba signala, EA i EB, potrebno im je dodeliti početne vrednosti. To je prikazano u narednom kodu.

```
EA <= '0'; EB <= '0';
CASE y IS
  WHEN S1 => EA <= '1';
  WHEN S2 => EB <= '1';
END CASE;
```

Generalno pravilo je da se implicitna memorija na nekom signalu javlja u slučaju kada su navedeni nepotpuni uslovni iskazi za taj signal. Jednostavan recept za zaobilaženje implicitne memorije je da se uvek pišu potpuni uslovni iskazi (IF i CASE).