Osnove Verilog HDL jezika

Moduli u Verilog-u

Moduli u Verilog-u

- Modul u verilog-u predstavlja osnovnu jedinicu dizajna i tipicno predstavlja hardversku komponentu koju zelimo da napravimo.
- Svaki modul moze imati ulaze i izlaze koji predstavljaju signale koji respektivno, ulaze i izlaze iz komponente koju dizajniramo (ulaze I izlaze drugacije zovemo portovi).

Moduli u Verilog-u

- Manji moduli se mogu instancirati u okviru vecih modula (kao sto se jednostavnija logicka kola koriste u izgradnji slozenijih).
- Takodje, u verilog-u postoje i moduli za koje nije cilj da se sintetizuju, vec se koriste iskljucivo za testiranje drugih modula.

Moduli u Verilog-u

- Ovi moduli najcesce nemaju portove, a sintaksa veriloga koja se u njima koristi je obicno znatno slobodnija, jer ne moramo da vodimo racuna o tome da li se nesto moze ili ne moze sintetizovati u stvarnom hardveru.

Moduli u Verilog-u

Sintaksa kreiranja modula :

```
module <ime_modula>(<lista_portova>);
<deklaracije_portova>;

// kod

endmodule // kraj modula
```

Moduli u Verilog-u

- U istom fajlu veriloga se obicno moze definisati veci broj modula,mada je cesto dobra praksa da se za svaki modul kreira poseban fajl sa kodom koji se zove isto kao i modul (sa ekstenzijom .v).

Moduli u Verilog-u

- Portovi modula se navode svojim identifikatorima koji su razdvojeni zarezima.
- Nakon zaglavlja slede deklaracije portova.
- Za svaki port treba definisati da li je ulazni, izlazni, ili ulazno-izlazni.

Moduli u Verilog-u

- Za portove se podrazumeva da su tipa wire.
- Ako zelimo da ne budu tipa wire, onda to moramo eksplicitno navesti u odgovarajucoj redekleraciji, npr:

```
reg x[3:0];
```

Moduli u Verilog-u

- Modul se moze instancirati u okviru drugog modula.
- Tom prilikom senavodi sledeca deklaracija:

<ime_modula> <ime_instance>(<signali_koji_se_povezuju_sa_portovima>);

Moduli u Verilog-u

- Signali koji se povezuju sa portovima su zapravo neki signali koji postoje u modulu u okviru koga instanciramo nas modul.
- Pri navodjenju, oni se razdvajaju zarezima (kao argumenti funkcija u C-u).

Moduli u Verilog-u

- Ponekad je moguce da se na neki port ne dovede ni jedan signal (npr.neki izlaz nas ne zanima, neki ulaz nije bitan, i sl.).
- U tom slucaju se prosto ostavi prazan prostor u listi signala, npr:

- moj_modul _mm1(x, y, , z,); // ovde su treci i peti port ostali nepovezani

Moduli u Verilog-u

- Drugi nacin da se ovo uradi je eksplicitno navodjenje signala:

  moj_modul _mm1(.x(x), .y(y), .z(z), .u(), .w());

- Ovde se iza tacke navodi naziv porta u modulu moj_modul, a u zagradama se navodi naziv signala iz spoljnog modula koji se povezuje na ovaj port (ne moraju se zvati isto, ali cesto to jeste slucaj).
- Za one portove koji ostaju nepovezani prosto se stave prazne zagrade.
- U ovom slucaju nije bitan redosled navodjenja portova.

GATE-LEVEL MODELIRANJE:

- Ovo je najnizi nivo modelovanja, a sastoji se u dizajnu kola na nivou osnovnih logickih kola koja se onda eksplicitno povezuju u slozenije kolo.
- Na primer:

GATE-LEVEL MODELIRANJE:

```
wire x, y;
wire z;
and(z, x, y);  // ovim se kreira AND kolo ciji je izlaz z,
               //  a ulazi su x i y

not(z, x);      // ovim se kreira NOT kolo ciji je izlaz z,
                //  a ulaz x
```

GATE-LEVEL MODELIRANJE:

- Generalno, uvek je prvi argument izlaz, a ostalo su ulazi (kojih u slucaju and, or, nor, xor, nand, xnor kola moze biti dva ili vise).
- Svi ulazi (kao i izlaz) moraju biti jednobitni.

GATE-LEVEL MODELIRANJE:

- Pored pomenutih standardnih kola, definisana su i sledeca kola:

- -- buf(x, y) -- uzima vrednost y i salje na izlaz x.
- Baferi se tipicno koriste za simulaciju kasnjenja, jer inace ne menjaju vrednost ulaznog signala, vec ga samo prosledjuju na izlaz.

GATE-LEVEL MODELIRANJE:

- -- bufif0(x, y, e)
- vrednost ulaza y se prosledjuje na izlaz x ako je ulaz e jednak 0.
- U suprotnom, vrednost izlaza je z.

GATE-LEVEL MODELIRANJE:

- -- bufif1(x, y, e) -- slicno, samo e treba da bude 1

-- notif0(x, y, e) -- isto, samo sto ce x biti negacija od y
-- notif1(x, y, e) -- slicno

GATE-LEVEL MODELIRANJE:

- Logickim kolima se moze zadati kasnjenje po zelji. Na primer:
- and #(5) and(z, x, y); // rezultat se propagira na izlaz
  // sa kasnjenjem koje iznosi 5
  // vremenskih jedinica

GATE-LEVEL MODELIRANJE:

- Sintaksa zadavanja kasnjenja

#n
#(n)
#(n,m)
#(n,m,k)

GATE-LEVEL MODELIRANJE:

- #n
- #(n)

Ova dva slucaja su ekvivalentna -- uvodi se kasnjenje od n vremenskih jedinica.

GATE-LEVEL MODELIRANJE:

- #(n,m)
- U ovom slucaju, kasnjenje n se odnosi na prelaz sa 0 na 1, dok se kasnjenje m odnosi na prelaz sa 1 na 0.

GATE-LEVEL MODELIRANJE:

#(n,m,k)

- U ovom slucaju, kasnjenje n se odnosi na prelaz sa 0 na 1, m na prelaz sa 1 na 0, dok se k odnosi na kasnjenje prilikom prelaska u stanje visoke impedanse (z).
-

GATE-LEVEL MODELIRANJE:

- Kasnjenja se koriste pri simulaciji, medjutim, ona se ni na koji nacin ne mogu sintetizovati.
- Prosto, hardverska kasnjenja zavise od konkretne tehnologije izrade i to je nesto sto ne moze da se programira.

DATA-FLOW MODELIRANJE:

- Na data-flow nivou, modeliranje se sastoji u formiranju izraza (pomocu ranije opisanih operatora) i pridruzivanja tih izraza odgovarajucim signalima.
- Ovi izrazi ce u fazi sinteze biti zamenjeni odgovarajucim kombinatornim kolom koje se sastoji iz odgovarajucih osnovnih logickih kola, ali se na ovaj nacin dizajner oslobadja mukotrpnog posla rucnog povezivanja odgovarajucih primitiva.
- Osnovni mehanizam data-flow modeliranja je assign naredba:

DATA-FLOW MODELIRANJE:

```
wire [3:0] x;
wire y, z;
wire [2: 0] p;

assign x = { y | ~z, { y, {2{z}} } ^ ~p };
```

DATA-FLOW MODELIRANJE:

```
wire [3:0] x;
wire y, z;
wire [2: 0] p;

assign x = { y | ~z, { y, {2{z}} } ^ ~p };
```

- Sa desne strane se formira 4-bitni signal. Ovaj signal se pridruzuje podatku x.
- Ova naredba se zove naredba KONTINUIRANE DODELE. Ova dodela se prilicno razlikuje od koncepta dodele u proceduralnim programskim jezicima.

DATA-FLOW MODELIRANJE:

- Kontinuirana dodela znaci da se signal (zica) x trajno povezuje na izlaz kola koje izracunava signal na desnoj strani.
- Ovim signal na desnoj strani postaje drajver za zicu x.
- Svaki put kada se bilo koji od signala koji ucestvuju u izrazu promeni, vrsi se ponovno izracunavanje vrednosti izraza, cime se odmah menja i vrednost signala x (u proceduralnim jezicima poput C-a, dodela se izvrsi u jednom trenutku, nakon cega promenljiva sa leve strane vise nije ni na koji nacin povezana sa izrazom na desnoj strani; ako zelimo da joj ponovo promenimo vrednost, moramo ponovo da izvrsimo dodelu).

DATA-FLOW MODELIRANJE:

- Modelovanje na DATA-FLOW nivou je znacajno jednostavnije, jer sada mozemo na intuitivniji nacin da zapisemo zeljeno izracunavanje.
- Na primer, gornji izraz bi se na gate-level nivou morao opisati sledecim kodom:

DATA-FLOW MODELIRANJE:

```
wire z1;
wire p1[2:0];
not(z1, z);
or(x[3], y, z1);
not(p1[2], p[2]);
xor(x[2], y, p1[2]);
not(p1[1], p[1]);
xor(x[1], z, p1[1]);
not(p1[0], p[0]);
xor(x[0], z, p1[0]);
```

DATA-FLOW MODELIRANJE:

- Naredba assign takodje dozvoljava definisanje kasnjenja, radi vernije simulacije.
- Kasnjenje se navodi iza assign kljucne reci:
- assign #5 x = y;
- Ova naredba je ekvivalentna sa:
- buf #5 (x, y); na gate-level nivou.

DATA-FLOW MODELIRANJE:

- Umesto naredbe assign, moguce je koristiti inicijalizaciju prilikom deklaracije wire podatka:

```
wire x = y | z;
```

- Ovo je ekvivalentno sa:

```
wire x;
assign x = y | z;
```