Radionica digitalnog dizajna III

dr. sc. Jurica Kundrata

Centar za svemirsku i inovativnu tehnologiju

2. - 3. ožujka 2024.

wlan: kpckat
pass:kpc123456

Raspored - Dan I

2. ožujka 2024., subota	
9.00 - 12.00	• Slanje dizajna IC-a na izradu pomoću TinyTapeout Verilog predloška
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	• Pregled prošlih TinyTapeout projekata i razvoj testbencha za odabrane primjere

Raspored - Dan II

3. ožujak 2024., nedjelja	
9.00 - 12.00	● Specifikacija dizajna IC-a po timovima
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	● Razvoj high-level prototipa i testbencha po timovima

TinyTapeout 6

Postupak prijave dizajna na TinyTapeout

Što je Verilog Testbench?

- Virtualno okruženje za testiranje Verilog dizajna
- Ne sintetizira se u stvarni hardver
- Omogućuje simulaciju rada digitalnog sklopa
- Provjera funkcionalnosti i otkrivanje grešaka prije implementacije

Zašto je Testbench važan?

- Omogućava ranu detekciju i ispravljanje grešaka
- Smanjuje troškove i vrijeme razvoja
- Povećava pouzdanost i performanse konačnog proizvoda
- Neophodan za kompleksne dizajne gdje je ručno testiranje nepraktično

Osnovni elementi Testbencha

- Modul Testbench-a: Glavni okvir koji ne sadrži ulaze ili izlaze kao tradicionalni Verilog moduli.
- Stimulus: Skup ulaznih signala koji se primjenjuju na DUT (Device Under Test) za simulaciju različitih scenarija.
- Instance DUT-a: Uključivanje modula koji se testira unutar Testbench-a kako bi se na njega mogli primijeniti stimulansi.
- Monitore i Provjere: Skripte ili blokovi koda unutar Testbench-a koji nadziru izlaze DUT-a i provjeravaju jesu li u skladu s očekivanjima.
- Vremenska kontrola: Upotreba vremenskih funkcija (#(time)) za simuliranje prolaska vremena i sinkronizaciju stimulansa.

Modul Testbench-a

- Definicija: Kao i svaki Verilog modul, Testbench se definira ključnom riječju module, ali bez portova.
- Svrha: Služi za simulaciju okruženja u kojem će se testirati DUT (Device Under Test).
- Primjer: `module testbench();` - Osnovna struktura koja ne uključuje ulazne i izlazne porte.

Inicijalizacija i završetak simulacije

- Inicijalizacija: Korištenje initial bloka za postavljanje početnih uvjeta simulacije, kao što su početne vrijednosti signala.
- Simulacija: Unutar initial bloka, simulacija se pokreće primjenom različitih ulaznih signala na DUT.
- Završetak: Simulacija se može završiti naredbom $finish; za zaustavljanje simulacije u određenom trenutku.
- Primjer:

```
initial begin
  // Postavljanje početnih uvjeta
  // Primjena testnih signala
  // Završetak simulacije
  $finish;
end
```

Deklaracija i inicijalizacija varijabli

- Deklaracija: Varijable se deklariraju unutar Testbench modula za predstavljanje ulaznih i izlaznih signala DUT-a, kao i za interne varijable potrebne za testiranje.
- Tipovi varijabli: Mogu uključivati reg za ulazne signale koji se mijenjaju tijekom simulacije i wire za izlazne signale DUT-a.
- Inicijalizacija: Početne vrijednosti varijabli postavljaju se unutar initial bloka kako bi se definiralo početno stanje simulacije.
- Primjer:

```
reg [3:0] input_a, input_b; // 4-bitni ulazni signali

wire [4:0] output_sum; // 5-bitni izlazni signal za sumu
```

Opis jednostavnog DUT-a: 4-bitno zbrajalo

- Funkcionalnost: Zbraja dva 4-bitna broja te generira 4-bitni zbroj i bit prijenosa (carry out).
- Primjena: Osnovna komponenta u digitalnim sustavima za izvođenje aritmetičkih operacija.
- Bit prijenosa: Dodatni izlaz koji ukazuje na prekoračenje kapaciteta 4-bitnog zbroja.

Blok dijagram DUT-a

- Ulazi: Dva 4-bitna ulaza označena kao A[3:0] i B[3:0].
- Izlazi: Jedan 4-bitni izlaz zbroja označen kao SUM[3:0] i jedan bit prijenosa (COUT).
- Komponente: Četiri puna zbrajala (full adder) povezana u kaskadu.
- Prikaz veza: Svaki puni zbrajalo prima dva bita od A i B, te bit prijenosa od prethodnog zbrajala (osim prvog koji prima ulazni bit prijenosa).

Verilog kod za DUT - definicija

```
module four_bit_adder(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
```

Verilog kod za DUT - Implementacija

- Korištenje četiri instance punog zbrajala.
- Povezivanje bita prenosa između zbrajala.

```
wire c1, c2, c3; // Unutarnji bitovi prijenosa
full_adder fa0(A[0], B[0], Cin, Sum[0], c1);
full_adder fa1(A[1], B[1], c1, Sum[1], c2);
full_adder fa2(A[2], B[2], c2, Sum[2], c3);
full_adder fa3(A[3], B[3], c3, Sum[3], Cout);
```

- Završetak modula:

```
endmodule
```

Implementacija modula full_adder

```
module full_adder(
    input A,
    input B,
    input Cin,
    output Sum,
    output Cout
);
    // Izračun sume
    assign Sum = A ^ B ^ Cin;
    // Izračun izlaznog bita prijenosa
    assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule
```

Kako definirati Testbench modul

- Bez Portova: Testbench modul se definira bez ulaznih i izlaznih portova jer ne komunicira direktno s vanjskim svijetom na isti način kao drugi moduli.

```
module testbench;
```

- Initial Blok: Koristi initial blokove za simulaciju testnih slučajeva, generiranje ulaznih signala i kontrolu vremena izvođenja.
- Završetak Simulacije: Implementira $finish za završetak simulacije nakon izvođenja testnih slučajeva.

Instanciranje DUT-a unutar Testbench-a

- Kreiranje Instance: Unutar testbench modula stvorite instancu DUT-a (Device Under Test), u ovom slučaju 4-bitnog zbrajala.

```
four_bit_adder adder_instance (.A(A), .B(B), .Cin(Cin), .Sum(Sum), .Cout(Cout));
```

- Povezivanje Signala: Signali definirani unutar testbench-a se koriste za povezivanje s DUT-om, omogućavajući stimulaciju DUT-a i promatranje izlaza.

Definiranje ulaznih i izlaznih signala

- Ulazni Signali: Definirajte ulazne signale koji će se koristiti za testiranje DUT-a kao reg tipove podataka, omogućavajući im da se mijenjaju tijekom simulacije.

```
reg [3:0] A, B;
reg Cin;
```

- Izlazni Signali: Definirajte izlazne signale koji će prikazivati rezultate DUT-a kao wire tipove podataka, jer se njihove vrijednosti automatski ažuriraju ovisno o ulazima i logici unutar DUT-a.

```
wire [3:0] Sum;
wire Cout;
```

- Inicijalizacija i Test Slučajevi: Koristite initial blok za postavljanje vrijednosti ulaznih signala na željene testne slučajeve, čime se simulira različite radne uvjete DUT-a.

```
initial begin
  A = 4'b0001; B = 4'b0010; Cin = 0; // Primjer testnog slučaja
  #10; // Čeka 10 jedinica vremena prije sljedećeg testa
end
```

- Promatranje i Provjera: Koristite initial blok sa $monitor ili $display funkcijama za praćenje promjena na signalima tijekom simulacije, omogućavajući efikasno debugiranje i validaciju funkcionalnosti DUT-a.

Najčešće greške pri pisanju Testbench-a

- Neodređeni ili neinicijalizirani signali: Zaborav na inicijalizaciju svih ulaznih signala može dovesti do neodređenih rezultata u simulaciji.
- Nedovoljno pokrivanje testnih slučajeva: Ignoriranje rubnih uvjeta ili rijetkih slučajeva može rezultirati neotkrivenim bugovima.
- Pretpostavka o vremenskom ponašanju: Pretpostavke o sinkronizaciji bez eksplicitnog testiranja mogu dovesti do problema koji se neće manifestirati do implementacije na stvarnom hardveru.

Savjeti za efikasno debugiranje

- Koristite display naredbe: display naredbe u Verilogu omogućavaju ispis vrijednosti varijabli tijekom simulacije, što može pomoći u identifikaciji problema.
- Postepeno testiranje: Testirajte svaki dio koda zasebno prije nego što ga integrirate s ostatkom Testbench-a.
- Vizualizacija valnih oblika: Korištenje alata za vizualizaciju valnih oblika može pomoći u razumijevanju interakcija između signala i pronalaženju grešaka.
- Komentari i dokumentacija: Dobro dokumentirajte svoj Testbench i korišteni testni plan kako biste olakšali razumijevanje koda i proces debugiranja.

Korištenje assert izraza za provjeru ispravnosti

- Assert naredbe: assert naredbe se koriste za provjeru da li su određeni uvjeti zadovoljeni tijekom simulacije. Ako uvjet nije zadovoljen, simulacija može ispisati grešku ili upozorenje.
- Primjena: Koristite assert za automatsku provjeru ispravnosti izlaza DUT-a na temelju poznatih ulaznih signala. Na primjer, provjerite ispravnost rezultata aritmetičke operacije.
- Primjer koda:

```
always @(posedge clk) begin

    assert (sum == expected_sum) else $error("Suma nije ispravna.
Očekivano: %d, Dobiveno: %d", expected_sum, sum);
```

- Prednosti: Automatizacija testiranja i rano otkrivanje grešaka. assert naredbe pomažu u održavanju integriteta dizajna tijekom cijelog razvojnog ciklusa.

Važnost Testbench-a u razvoju digitalnih sklopova

- Testbench je neophodan za validaciju i verifikaciju digitalnih dizajna, osiguravajući da sklop radi kako je zamišljeno prije fizičke implementacije.
- Omogućava detaljno testiranje svih aspekata sklopa, uključujući rubne slučajeve i neočekivane uvjete, čime se smanjuje rizik od grešaka u kasnijim fazama razvoja.
- Smanjuje troškove razvoja i vremenski okvir projekta eliminacijom potrebe za čestim revidiranjem i ispravkom fizičkih prototipova.
- Može biti alat za razvoj specifikacije sustava ili modula

Specifikacija digitalnog sustava ili modula

Sekcija	Opis	Detalji
1. Općenito	Naziv modula/sustava	Kratak opis funkcije modula/sustava
	Verzija dokumenta	Datum i verzija specifikacije
	Autor(i)	Imena i kontakti autora specifikacije
2. Ulazi	Signali ulaza	Ime, tip (npr. digitalni, analogni), raspon vrijednosti, opis funkcije
	Kontrolni signali	Detalji o svakom kontrolnom signalu, uključujući funkciju i ponašanje

Specifikacija digitalnog sustava ili modula

Sekcija	Opis	Detalji
3. Izlazi	Signali izlaza	Ime, tip, raspon vrijednosti, opis funkcije
	Statusni signali	Informacije o statusnim signalima koje modul/sustav generira
4. Funkcionalnost	Glavne funkcije	Detaljan opis glavnih funkcionalnosti koje modul/sustav nudi
	Pomoćne funkcije	Opis dodatnih, sekundarnih funkcija koje sustav podržava
5. Zahtjevi PPA	Performanse	Ciljane performanse sustava, uključujući brzinu obrade, propusnost, vrijeme odziva, itd.
	Potrošnja (Power)	Maksimalna potrošnja energije, ciljana potrošnja u mirovanju i pod opterećenjem, strategije upravljanja energijom
	Površina (Area)	Procjena veličine čipa ili modula, zahtjevi za minimalnu površinu, ograničenja vezana uz integraciju