

UVM
2. dan
Osnovi Verilog-a
podsetnik?

Verilog HDL jezik

- Originalno jezik za efikasno modelovanje event-driven digitalnih logičkih simulacija
- Naknadno prilagođen upotrebi u logičkoj sintezi
- Aktuelno sada uz VHDL predstavlja najzastupljenij HDL
- Praktično svako integrisano kolo u sebi sadrži više ili manje delova dizajniranih u Verilog-u.
- Kombinuje strukturalno i behavioral opisivanje sistema

Strukturalno opisivanje sistema

- Kada je Verilog inicijalno razvijen (1984) većina logičkih simulatora bazirala je svoj rad na netlistama.
- Netlista: lista logičkih kapija i njihovih međusobnih veza
- Prirodna prezentacija digitalnih logičkih kola
- Ujedno ne predstavlja najpodesniji način modelovanja test benča.

Behavioral Modeling

- Znatno jednostavniji način opisivanja testbenčeva
- Ujedno podesan za formiranje apstraktnih modela logičkih kola
 - Jednostavno se opisuje
 - Brže se simulira
- Fleksibilno

Korišćenje Verilog-a

- Praktično svaki ASIC danas je dizajniran u Verilog-u ili VHDL-u
- Upotreba Behavioral modeling-a sa strukturalnim elementima
- “Sintetizibilni podskup”
 - Može se prevesti korišćenjem “kompajlera” u netlistu
- Dizajn napisan u Verilog-u može se
 - Simulirati u cilju provere funkcionalnosti
 - Sintetisati (generiše se sintetizibilna netlista)
 - Uraditi Static Timing Analysis (STA) radi provere timing-a.

Dve osnovne komponente Verilog-a

- Konkurentni, event-triggered procesi (behavioral)
 - *Initial* i *Always* blokovi
 - Kod koji realizuje standardne taskove za manipulaciju sa podacima (dodele, if-then, case)
 - Proces se izvršava na zahtevani triggering event ili je u stanju čekanja tog eventa.
- Strukturni Verilog kod
 - Verilog dizajn sastavljen od modula sa I/O portovima
 - Moduli mogu sadržati instance drugih modula
 - Moduli sadrže lokalne signale itd.
 - Konfiguracija modula je statička i sve se izvršava konkurentno

Dve osnovne vrste podataka

- Čvorovi (Nets) predstavljaju konekcije između različitih modula
 - Ne čuvaju vrednosti
 - Preuzimaju vrednosti od driver-a koji može biti log. Kapija ili registar
 - Ne može biti dodeljen unutar *initial* ili *always* bloka
- Registri predstavljaju osnovne memorijske elemente - DFF
 - Ponašaju se kao memorije
 - Čuvaju vrednost sve dok im se ne dodeli nova vrednostu u okviru *initial* ili *always* bloka

Discrete-event Simulacija

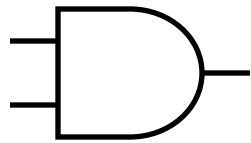
- Osnovna ideja: izvrši akciju kada se nešto promeni
- Centrirano oko event queue-a
 - Sadrži događaje (events) labelirane simuliranim vremenom po kome će oni biti izvršavani
- Osnovni simulacioni zahtev
 - Izvrši svaki događaj (event) u aktuelnom simulacionom vremanskom trenutku
 - Izvršavajući ovaj zadatak, stanje sistema se menja i dodaju se novi eventi u budućnosti.
 - Kada više nema event-a preostalih u aktuelnom simulacionom vremenskom trenutku, inkrementiraj simulaciono vreme prema narednom event-u u queue

Podaci se modeluju sa 4 vrednosti

- Verilog čvorovi i registri čuvaju 4 vrednosti podataka
- 0, 1
 - Osnovne binarne vrednosti
- Z
 - Izlaz neaktivnog tri-state driver-a
 - Modeluje slučaj kada ništa ne goni wire signal (čvor)
- X
 - Modeluje slučaj u kome simulator ne može da odluči vrednost
 - Inicijalno stanje registra
 - Modeluje koliziju na čvoru gonjenom sa dve strane 0 i 1 simultano
 - Izlaz kapije gonjene sa Z na ulazu.

Logika sa 4-vrednosti

- Logički operatori barataju sa 3 vrednosti



	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

← Izlaz 0 ako je jedan ulaz 0

← Izlaz X ako su oba ulaza X/Z

Strukturalno Modelovanje

Čvorovi i Registri

- Wires i registers mogu biti biti, vektori i nizovi

```
wire a; // Simple wire
tri [15:0] dbus; // 16-bit tristate bus
tri #(5,4,8) b; // Wire with delay
reg [-1:4] vec; // Six-bit register
triereg (small) q; // Wire stores a small charge
integer imem[0:1023]; // Array of 1024 integers
reg [31:0] dcache[0:63]; // A 32-bit memory
```

Moduli i instance

- Osnovna struktura Verilog modula:

```
module mymod(output1, output2,... input1, input2);  
output output1;  
output [3:0] output2;  
input input1;  
input [2:0] input2;  
...  
endmodule
```



Verilog convention
lists outputs first

Instanciranje Modula

- Instances of

```
module mymod(y, a, b);
```

- look like

```
mymod mm1(y1, a1, b1);           // Connect-by-position  
mymod (y2, a1, b1),  
      (y3, a2, b2);              // Instance names omitted  
mymod mm2(.a(a2), .b(b2), .y(c2)); // Connect-by-name
```

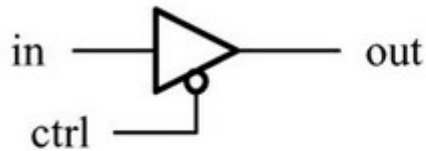
Gate-level Primitive

- Verilog obezbeđuje sledeće:

and	nand	logical AND/NAND
or	nor	logical OR/NOR
xor	xnor	logical XOR/XNOR
buf	not	buffer/inverter
Bufif0	notif0	Tristate with low enable
bifif1	notif1	Tristate with high enable

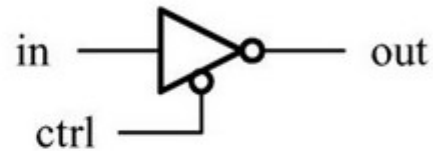
bufif0/notif0

bufif0/notif0 Gates



bufif0		ctrl			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	x	z	x	x

(a) bufif0

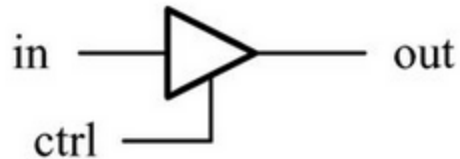


notif0		ctrl			
		0	1	x	z
in	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	x	x

(b) notif0

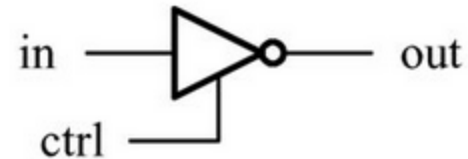
bufif1/notif1

bufif1/notif1 Gates



bufif1		ctrl			
		0	1	x	z
in	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

(c) bufif1



notif1		ctrl			
		0	1	x	z
in	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

(d) notif1

Kašnjenja uključena u instance Primitiva

- Instance primitiva mogu da uključe kašnjenje

buf	b1(a, b);	// Zero delay
buf #3	b2(c, d);	// Delay of 3
buf #(4,5)	b3(e, f);	// Rise=4, fall=5
buf #(3:4:5)	b4(g, h);	// Min-typ-max

Korisnički generisane Primitive

- Način da se formiraju logičke kapije i sekvencijalni elementi korišćenjem istinitosne tablice
- Brže se simuliraju nego modeli zadati izrazima, skupovima izraza, skupovima primitiva...
- Daje bolju kontrolu ponašanja
- Često se koristi za formiranje custom gate libraries

Primer - Carry Primitives punog sabirača

```
primitive carry(out, a, b, c);
```

```
output out;
```

```
input a, b, c;
```

```
table
```

```
00? : 0;
```

```
0?0 : 0;
```

```
?00 : 0;
```

```
11? : 1;
```

```
1?1 : 1;
```

```
?11 : 1;
```

```
endtable
```

```
endprimitive
```

Always have exactly
one output

Truth table may
include don't-care (?)
entries

Primer Sekvencijalne Primitive

```
Primitive dff( q, clk, data);
```

```
output q;
```

```
input clk, data;
```

```
reg q;
```

```
table
```

```
// clk data q  new-q
```

```
(01) 0 : ? : 0;
```

```
// Latch a 0
```

```
(01) 1 : ? : 1;
```

```
// Latch a 1
```

```
(0x) 1 : 1 : 1;
```

```
// Hold when d and q both 1
```

```
(0x) 0 : 0 : 0;
```

```
// Hold when d and q both 0
```

```
(?0) ? : ? : -;
```

```
// Hold when clk falls
```

```
?  (??) : ? : -;
```

```
// Hold when clk stable
```

```
endtable
```

```
endprimitive
```

Simboli dozvoljeni u istinitosnoj tablici za modelovanje primitiva

Symbol	Definition
0	Logic 0
1	Logic 1
x or X	Unknown
?	Don't care if input is 0, 1 or X
b or B	Don't care if input is 0 or 1
-	Output does not change (sequential UDP only)
(vw)	Input transition from logic v to logic w
r or R	Rising input transition: (01)
f or F	Falling input transition: (10)
p or P	Positive input transition: (01), (0x) or (x1)
n or N	Negative input transition: (10), (1x) or (x0)
*	Any possible input transition: (??)

Kontinualna dodela

- Način da se zada kombinaciona funkcija

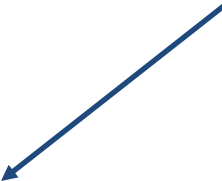
```
wire [8:0] sum;  
wire [7:0] a, b;  
wire carryin;
```

Define bus widths



```
assign sum = a + b + carryin;
```

Continuous
assignment:
permanently sets the
value of sum to be
 $a+b+carryin$



Recomputed when a,
b, or carryin changes

Behavioral Modelovanje

Initial i Always Blokovi

- Osnovne komponente za behavioral modeling

initial

begin

...statements ...

end

always

begin

...statements ...

end

**Izvršava se pri pokretanju
simulacije**

**Završava se kada kontrola
stigne do kraja**

**Pogodno za formiranje
stimulusa**

**Pokreće se pri pokretanju
simulacije**

**Restartuje se čim kontrola
stigne do kraja**

**Pogodno za
modelovanje/simulaciju
hardvera**

Initial i Always Blokovi

- Izvršava se sve do zahtevanog kašnjenja

```
initial begin  
    #10 a = 1; b = 0;  
    #10 a = 0; b = 1;  
end
```

- Ili čekanja na određeni event

```
always @(posedge clk) q = d;  
always begin wait(i); a = 0; wait(~i); a = 1; end
```

Proceduralna dodela vrednosti

- Unutar initial ili always bloka:

`sum = a + b + cin;`

- Kao u C-u: RHS sračunata i dodeljena LHS pre izvršavanja naredne dodele
- RHS može da sadrži wires i regs
- LHS mora biti reg

Uslovna dodela

```
if (select == 1)    y = a;  
else                y = b;
```

```
case (op)  
  2'b00: y = a + b;  
  2'b01: y = a - b;  
  2'b10: y = a ^ b;  
  default: y = 'hxxxx;  
endcase
```

For Loop

- Rastuća sekvenca vrednosti generiše se na izlazu

```
reg [3:0] i, output;
```

```
for ( i = 0 ; i <= 15 ; i = i + 1 ) begin
```

```
    output = i;
```

```
    #10;
```

```
end
```

While Loop

- Rastuća sekvenca vrednosti generiše se na izlazu
reg [3:0] i, output;

```
i = 0;
```

```
while (i <= 15) begin
```

```
    output = i;
```

```
    #10 i = i + 1;
```

```
end
```

Modelovanje Flip-Flopa korišćenjem Always konstrukcije

- Ivično okidani flip-flop

```
reg q;
```

```
always @(posedge clk)
```

```
    q = d;
```

- q = d assignment se realizuje na rastuću ivicu takta:upravo očekivano ponašanje

Blocking i Nonblocking dodela

- Verilog podržava dve vrste proceduralne dodele vrednosti
- Fundamentalni problem:
 - Unutar synchronog sistema, svi flip-flopovi sempluju simultano
 - U Verilog-u, always @(posedge clk) blok izvršava se unutar nedefinisane sekvence (ko će opredeliti kojim redom se ažurira koji flip-flop)?

Problematičan Shift Registar

- Kako se ponaša ovako implementiran shift reg?:

```
reg d1, d2, d3, d4;
```

```
always @(posedge clk) d2 = d1;
```

```
always @(posedge clk) d3 = d2;
```

```
always @(posedge clk) d4 = d3;
```

- Po kom kriterijumu simulator odlučuje koji od ova tri SR se prvi ažurira?

Non-blocking dodela

Nonblocking rule:

RHS evaluated when
assignment runs



- Ovakav shift reg radi:

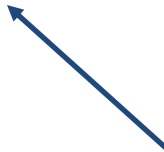
```
reg d1, d2, d3, d4;
```

```
always @(posedge clk) d2 <= d1;
```

```
always @(posedge clk) d3 <= d2;
```

```
always @(posedge clk) d4 <= d3;
```

LHS updated only after
all events for the current
instant have run



Nonblocking način razmišljanja

- Sekvenca nonblocking dodela ne komunicira direktno međusobno

a = 1;

b = a;

c = b;

Blocking assignment:

c = b = a = 1

a <= 1;

b <= a;

c <= b;

Nonblocking assignment:

a = 1

b = old value of a

c = old value of b

Nonblocking se ponaša kao dff

- RHS nonblockinga kao da se preuzima sa dff
- RHS blockinga kao da se preuzima sa wires

a = 1;

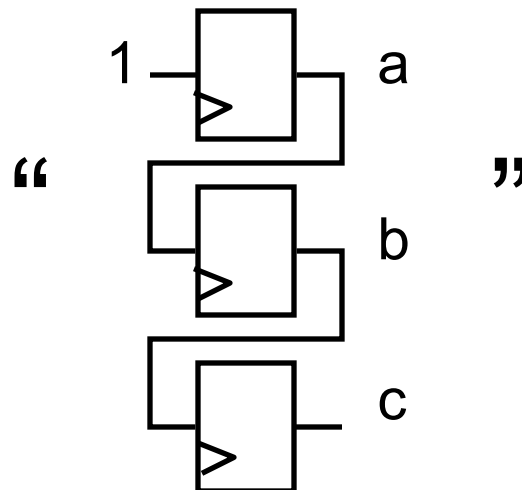
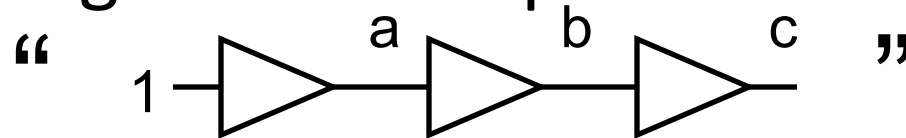
b = a;

c = b;

a <= 1;

b <= a;

c <= b;



Kreiranje Behavioral Modela

Behavioral Modelovanje FSM-ova

- Može se uraditi na više načina:
- Definišemo next-state logic kombinaciono, dok formiramo state-holding registre eksplicitno (dvoprocesni FSM)
- Definisanje behavior-a unutar jedinstvenog always @(posedge clk) blok-a (jednoprocesni FSM)
- Kombinacija...

FSM sa kombinacionom logikom

```
module FSM(o, a, b, reset);  
output o;  
reg o;  
input a, b, reset;  
reg [1:0] state, nextState;  
  
always @(a or b or state)  
case (state)  
  2'b00: begin  
    nextState = a ? 2'b00 : 2'b01;  
    o = a & b;  
  end  
  2'b01: begin nextState = 2'b10; o = 0; end  
endcase
```

Output o is declared
a reg because it is
assigned
procedurally, not
because it holds state

Combinational block
must be sensitive to
any change on any of
its inputs

(Implies state-holding
elements otherwise)

FSM sekvencijalni deo

```
module FSM(o, a, b, reset);
```

```
...
```

```
always @(posedge clk or reset)
```

```
  if (reset)
```

```
    state <= 2'b00;
```

```
  else
```

```
    state <= nextState;
```


Celokupni FSM

```
always @(a or b or state)
case (state)
  2'b00: begin
    nextState = a ? 2'b00 : 2'b01;
    o = a & b;
  end
  2'b01: begin nextState = 2'b10; o = 0; end
endcase
```

```
always @(posedge clk or reset)
if (reset)
  state <= 2'b00;
else
  state <= nextState;
```

This is a Mealy machine because the output is directly affected by any change on the input

Jednoprocesni FSM

```
module FSM(o, a, b);  
output o; reg o;  
input a, b;  
reg [1:0] state;
```

```
always @(posedge clk or reset)  
if (reset) state <= 2'b00;  
else case (state)  
2'b00: begin  
state <= a ? 2'b00 : 2'b01;  
o <= a & b;  
end  
2'b01: begin state <= 2'b10; o <= 0; end  
endcase
```

Expresses Moore
machine behavior:

Outputs are latched

Inputs only sampled
at clock edges

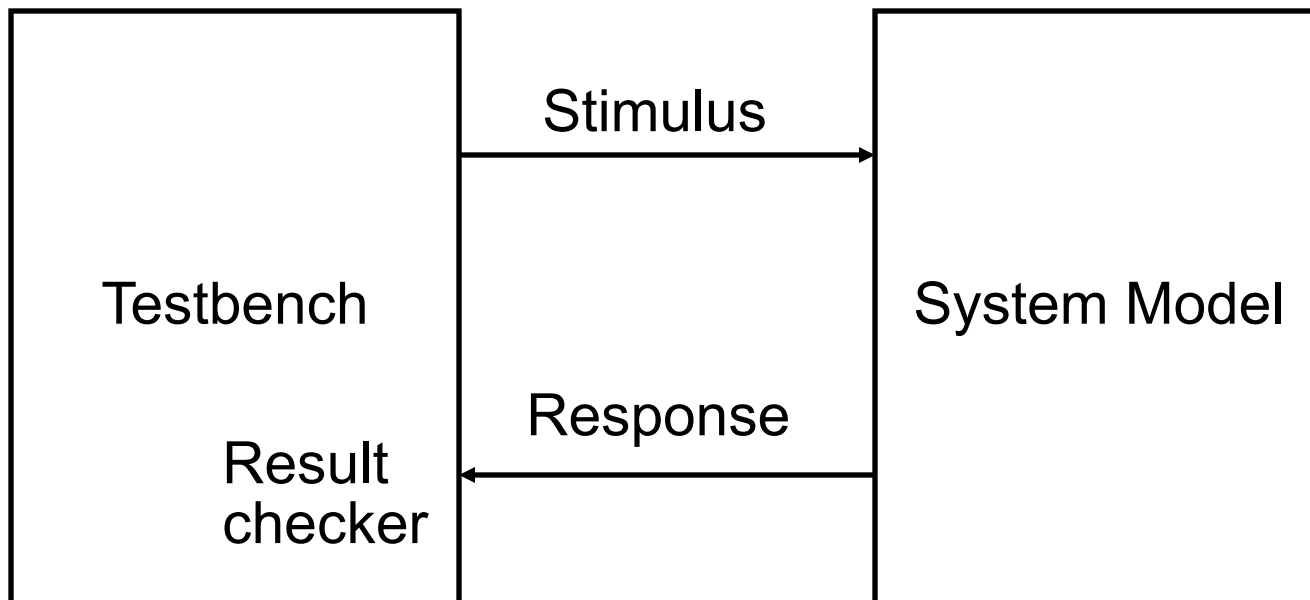
Nonblocking
assignments used
throughout to ensure
coherency.

RHS refers to values
calculated in previous
clock cycle

Verilog simulacije

Kako se koriste simulatori?

- Testbenč generiše stimulus i proverava odziv
- Povezan je na model sistema
- Upareni su i izvršavaju se simultano



Pisanje testbenčeva

```
module test;
reg a, b, sel;

mux m(y, a, b, sel);

initial begin
    $monitor($time,, "a = %b b=%b sel=%b y=%b",
              a, b, sel, y);
    a = 0; b= 0; sel = 0;
    #10 a = 1;
    #10 sel = 1;
    #10 b = 1;
end
```

Inputs to device under test

Device under test

\$monitor is a built-in event driven "printf"

Stimulus generated by sequence of assignments and delays

Ponašanje Simulacije

- Scheduled korišćenjem event queue
- Non-preemptive, no priorities
- Eventi u istom vremenskom trenutku nemaju opredeljen poredak izvršavanja – simulator odlučuje

Dve vrste događaja (Events)

- Evaluation events (sračunavaju vrednost funkcije na osnovu ulaza)
- Update events (ažuriraju izlaznu vrednost)
- Njihovo razdvajanje je neophono zbog vremenskog kašnjenja, nonblocking dodela...

Update event
writes new value
of a and
schedules any
evaluation events
that are sensitive
to a change on a

$$a \leq b + c$$

Evaluation event
reads values of b and
c, adds them, and
schedules an update
event

Ponašanje simulacije

- Konkurentni procesi (initial, always) trče, sve dok se ne zaustave na:
- #42
 - Schedule proces da se razreši nakon 42 vremenske jedinice
- wait(cf & of)
 - razreši kada izraz “cf & of” postane true
- @(a or b or y)
 - razreši kada se a, b, ili y promene
- @(posedge clk)
 - Razreši kada se clk promeni sa 0 na 1

Ponašanje simulacije

- Beskonačne petlje su moguće u Verilog-u simulator ih regularno izvršava
- Ukoliko u beskonačnoj petlji izostane vremensko kašnjenje simulator će je izvršavati beskonačno, neće moći da uradi ništa drugo.

```
while (1)  
    count = count + 1;
```

Ponašanje simulacije

- Race conditions u Verilog-u
- Ovo se može izvršiti u bilo kom poretku:
finalna vrednost signala a, nedefinisana:

```
always @(posedge clk) a = 0;
```

```
always @(posedge clk) a = 1;
```

Ponašanje simulacije

- Semantika jezika je izvedena iz njegove simulacione prirode
- Context switching ponašanje adekvatno za simulaciju, ali ne uvek i za modelovanje
- Nedefinisan redosled izvršavanja adekvatan za implementaciju event queue

Verilog i logička sinteza

Logička sinteza

- Verilog se koristi u dva scenarija
 - Modelovanje za simulaciju sa diskretnim event-ima
 - Modelovanje za logičku sintezu
- Logička sinteza konvertuje Verilog kod u netlistu za implementaciju
- To predstavlja jedan od značajnijih proboja u dizajnu digitalnih kola u poslednjih 30 godina

Logička sinteza

- Realizuje se u dva koraka:
- Prevođenje Verilog (ili VHDL) koda u netlistu
 - Uz prepoznavanje i implementaciju registara
- Optimizacija rezultujuće netliste u cilju optimizacije brzine i/ili količine HW resursa

Prevođenje Veriloga u Log. kapije

- Deo jezika koji se lako prevodi:
 - Strukturalni opisi sa primitivama
 - Već izvorno sistemski predstavljaju netlistu
 - Continuous dodela (kombinaciona)
 - Izrazi se pretvaraju u Bulove funkcije
- Behavioral izrazi predstavljaju veći izazov u procesuprevođenja

Šta se može prevesti

- Structuralne definicije
 - sve
- Behavioral blokovi
 - Zavisno od liste osetljivosti
 - Samo kada se mogu interpretirati kao kombinaciona logika, ivično ili nivoom okidani lečevi / dff
 - Blokovi osetljivi na obe ivice takta, na promenu nepovezanih signala itd ne mogu biti sintetizovani.
- Korisnički definisane primitive
 - Primitive definisane istinitosnim tablicama
 - Određene sekvencijalne korisnički definisane primitive se ne mogu prevesti (niti lečevi, niti flip-flopovi)

Šta se ne može prevesti

- Initial blokovi
 - Koriste se da podese inicijalno stanje ili opišu testbenč stimuli
 - Nemaju svoj prirodni hardverski pandan
- Delays
 - Mogu se naći u Verilog kodu, ali se pri sintezi ignorišu
- Različite druge jezičke konstrukcije:
 - Generalno konstrukcije zavisne od diskretne event bazirane simulacione semantike
 - “disable” izrazi
 - čisti events

Implementacija reg tipa podatka

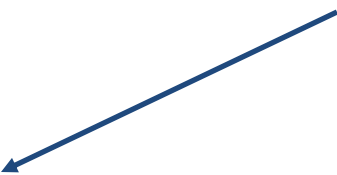
- Reg ne podrazumeva uvek registar u sintezi!
- Pravilo: Implementiraće se kombinaciono ukoliko izlazi uvek zavise isključivo od liste osetljivosti
- Implementiraće se sekvencijalno (kao registar) ako izlazi zavise i od prethodnog (memorisanog) stanja

Primer sa reg tipom

- Kombinacioni:

```
reg y;  
always @(a or b or sel)  
  if (sel) y = a;  
  else y = b;
```

Sensitive to changes
on all of the variables
it reads



Y is always assigned



- Sekvencijalni:

```
reg q;  
always @(d or clk)  
  if (clk) q = d;
```

q only assigned when
clk is 1



Primer sa reg tipom

- Tipična greška koja se sastoji u nekompletiranoj uslovnoj dodeli
- To kreira leč u sintezi:

always @(a or b)

case ({a, b})

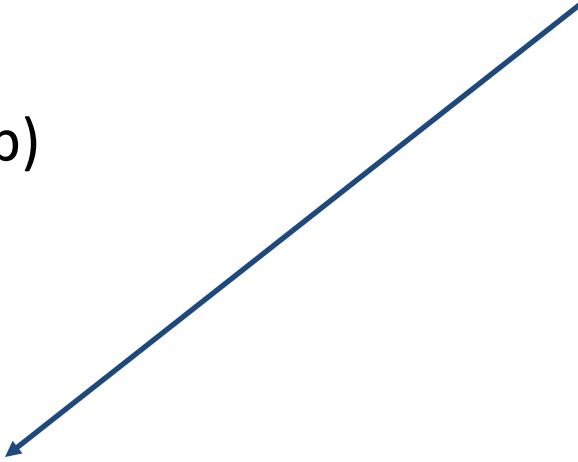
2'b00 : f = 0;

2'b01 : f = 1;

2'b10 : f = 1;

endcase

f is not assigned
when {a,b} = 2b'11



Primer sa reg tipom

- Rešenje je da se uvek kompletira uslovna dodela

always @(a or b)

case ({a, b})

2'b00: f = 0;

2'b01: f = 1;

2'b10: f = 1;

default: f = 0;

endcase

f is always assigned



Klasični registri sa resetom

- Mogu biti sinhroni i asinhroni
- Asinhroni dff - reset pozitivnim nivoom:

```
always @(posedge clk or posedge reset)
  if (reset)
    q <= 0;
  else q <= d;
```

Nedoslednosti između simulacije i sinteze

- Više mogućih izvora neslaganja
- Sinteza ignoriše kašnjenja(#10...), nasuprot tome, simulacija ih uvažava
- Simulator modeluje vrednost X eksplicitno, sinteza ne.
- Ponašanje izvedeno iz bloking dodele u okviru sekvencijalnih procesa ne može se sintetisati.
 - always @(posedge clk) a = 1;

Poređenje sa VHDL-om

- Verilog i VHDL su comparabilni jezici
- VHDL je strongly typed jezik
- Verilog je loosely typed
- Verilog danas ima svoje OOP proširenje- to je System Verilog, na njemu je baziran UVM
- Oba jezika se mogu kombinovati u savremenim simulacionim i sintetizibilnim alatima