OSNOVNI TIPOVI PODATAKA U VERILOGU

- Podaci u verilogu mogu uzeti i četri vrednosti
- 0: logicka nula
- 1: logicka jedinica
- x: nedefinisana vrednost (koristi se kod simulatora)
- z: vrednost visoke impedanse

OSNOVNI TIPOVI PODATAKA U VERILOGU

- Vrednosti 0 i 1 su najcesce rezultat toga sto je

  signal povezan sa odgovarajucim izvorom niskog, odnosno visokog napona.
- Vrednost x znači da signal nije definisan tj. inicijalizovan i uglavnom se koristi kod simulatora.
- Vrednost z visoka impedansa označava prekinutu žicu tj. u pitanju je signal sa nekog isključenog kola.

Promenljive

- U Verilogiu postoje dva tipa premenljivih promeljive tipa wire (žice) i registrarske promenljive
- reg.
- Promenljiva tipa wire služi da poveže izlaz iz nekog logičkog kola ili da se pomoću nje zada ulazna vrednost u neko logičko kolo.
- Svaka žica samim tim predstavlja tačku u kolu u kojoj se može izmeriti vrednost signala.
- Ono što je karakteristično za žice je da one same po sebi nemaju mogućnost da čuvaju neku vrednost, tj. nemaju memorijsku sposobnost.

Promenljive

Primer promenjivih tipa wire

- wire x; // Deifnisemo jednobitni podatak tipa wire
- wire [7:0] y; // 8-bitni podatak tipa wire
- Signalima se može pristupati celovito x, ali I pojedinačnim bitovima y[5], kao i grupama bitova y[4:2] daje trobitni signal koji se sastoji iz bitova y[4], y[3] i y[2]).

Promenljive

Registarski tip podataka ima memorijsko svojstvo tj. u njega se može upisati vrednost.

- reg x; // deklariše jednobitni podatak tipa reg
- reg [7:0] y; // deklariše 8-bitni podatak tipa reg
- x=1;
- x=0;
- y=8’b00000011; isto što i y=3;

If, while, for, case

Verilog takođe ima podršku za if, while, for , case strukture stim što treba voditi racuna da while petlja neće raditi na realnom hardveru jer nije unapred poznat broj iteracija.

primer if

```
if (a == 5)
    b = 15;
else
    b = 25;
```

primer case

```
reg [1:0] address;
case (address)
    2'b00 : statement1;
    2'b01, 2'b10 : statement2;
    default : statement3;
endcase
```

If, while, for, case

Petlje:

```
for (index=0; index < 10; index = index + 2)
    mem[index] = index;
end
```

Always struktura

Blok oblika

```
always @(p)
begin
end
```

se aktivira na svaku promenu p

```
always @(*)
begin
end
```

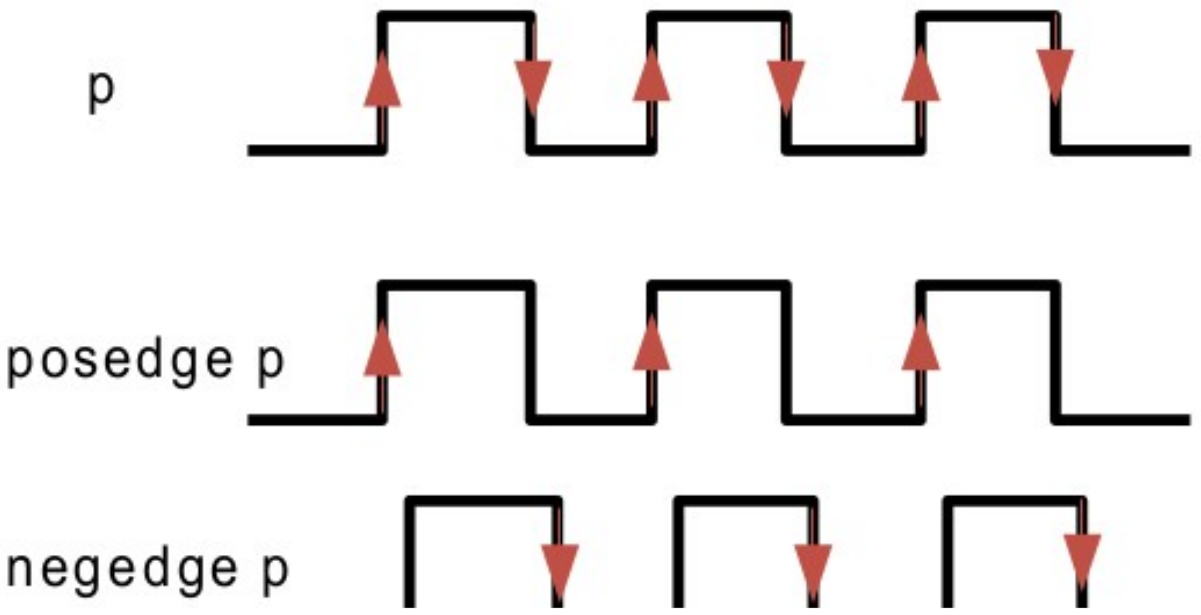
uvek se izvrsava

Always struktura

Postoji više varijanti

- always @(p) // na svaku promenu p
- always @(posedge p) // na uzlaznu ivicu od p
- always @(negedge p) // na silaznu ivicu od p
- Nakon always naredbe sledi blok begin end ukoliko imamo više od jedne naredbe

Always struktura



Always struktura

Razlika između operatora = i <=

always @(posedge p or negedge q)

begin

p <= q;

q <= p;

end

- Kod operatora <= se najpre izračunava vrednost izraza na desnoj strani, a sama dodela se vrši tek na kraju bloka kada se dođe do end.
- Gornji primer razmenjuje vrednost između p i q kada bilo koji od njih promeni vrednost.

Always struktura

Kada bi se koristio operator obične dodele = primer bi morao da izgleda ovako:

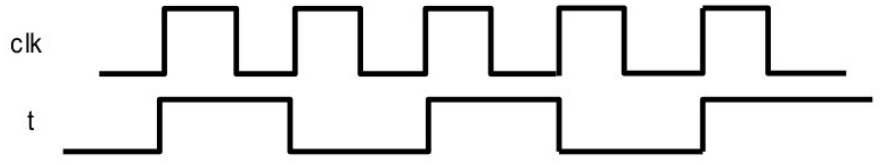
```
always @(posedge p or negedge q)
begin
t = p;
p= q;
q=t;
end
```

- Kod operatora = najpre se izračunava vrednost izraza na desnoj strani nakon čega se ta vrednost dodeli operandu.
- Tako da gornji primer razmenjuje vrednost između p i q kad bilo koji od njih promeni vrednost stim što je neophondo korišćenje i pomoćne promenljive t .

Deljenje takta

Ukoliko imamo signal clk koji se menja u vremenu prema slici:

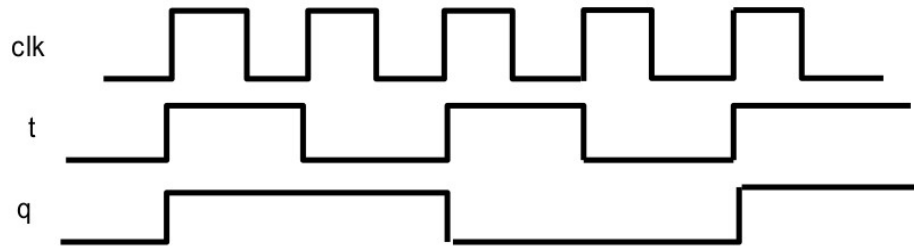
```
always @( posedge clk)
begin
t<=~t;
end
```



Deljenje takta

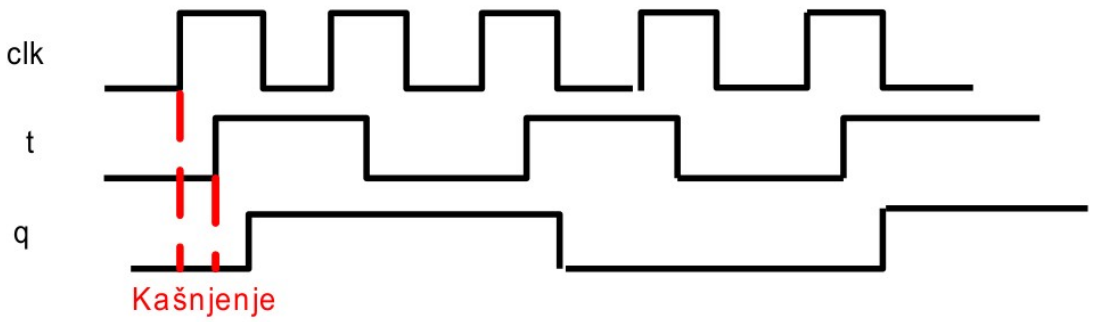
Ukoliko imamo signal clk koji se menja u vremenu prema slici možemo dobiti signale cije su frekvencije ½, ¼. …$1/2^n$

```
always @( posedge clk)
t<=~t;
always @( posedge t)
q<=~q;
```



Deljenje takta

- Ako imamo veći broj always blokova u nizu kod realnih logickih kola problem je sinhronizacija, odnosno kasnjenje.



Deljenje takta

Iz tog razloga je bolje koristiti sledeće

```
reg [1:0] count;
always@(posedge clk)
count <= count + 1; // counts 0..15
wire t = (count[0] == 1’b1);
wire q = (count[1] == 1’b1);
```

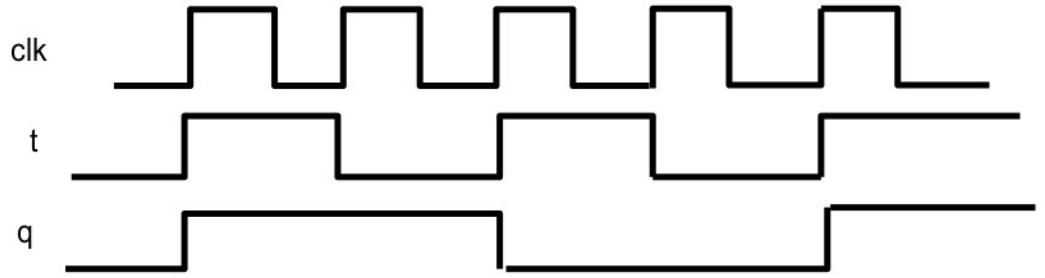
Na ovaj način se postiže bolja sinhronizacija

Deljenje takta

Iz tog razloga je bolje koristiti sledeće

```
reg [1:0] count;
always@(posedge clk)
count <= count + 1;   // counts 0..15
wire t = (count[0] == 1'b1);
wire q = (count[1] == 1'b1);
```

Na ovaj način se postiže bolja sinhronizacija

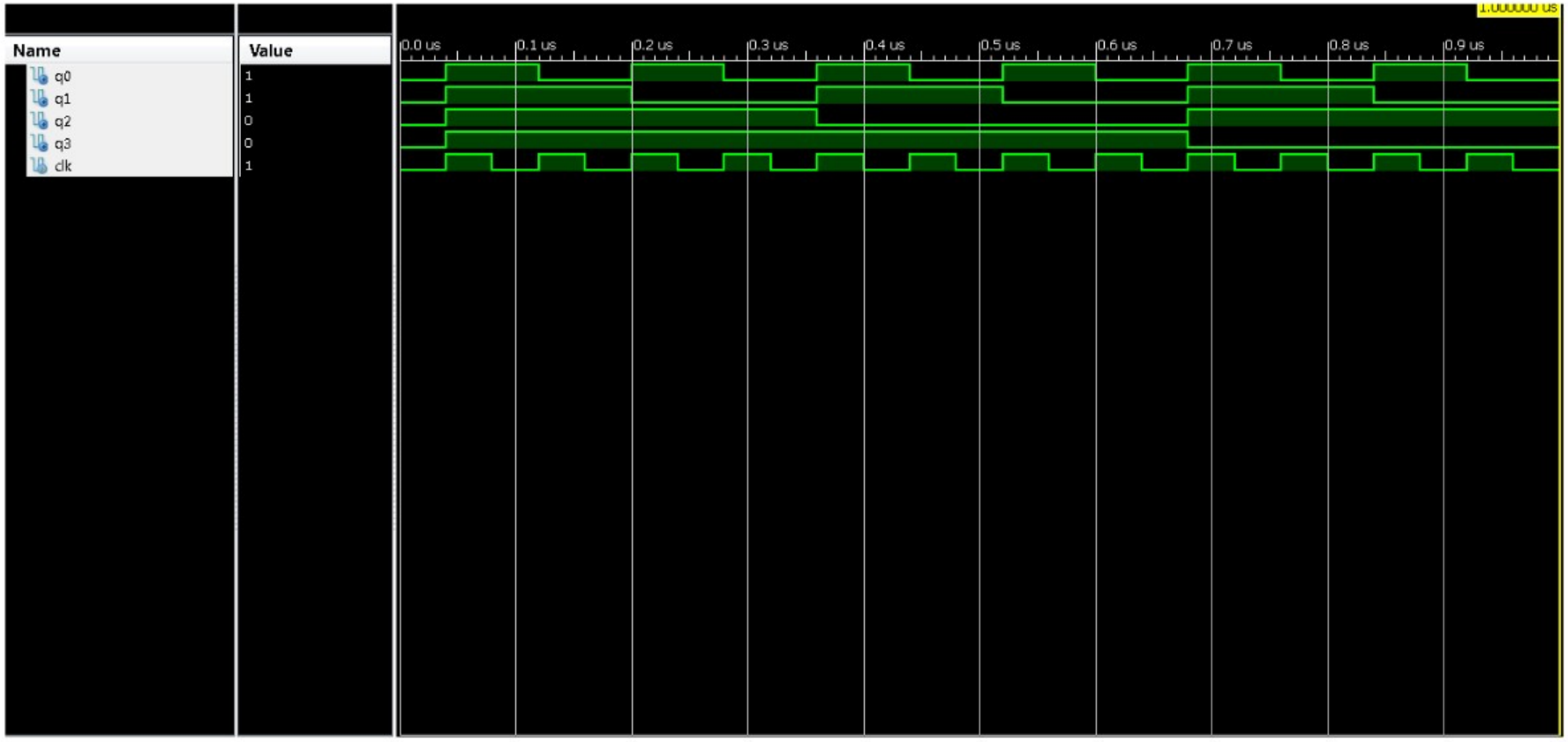


Primer

Realizovati 4-bitni brojač.

```
module  counter(input clk, output q);
reg q=0;
                always @(posedge clk)
                q<=~q;
endmodule

module counter_top(input clk, output q0,q1,q2,a3
   );
counter  c1 (.clk(clk),  .q(q0));
counter  c2 ( .clk(q0),  .q(q1));
counter  c3 ( .clk(q1),  .q(q2));
counter  c4 ( .clk(q2),  .q(q3));
endmodule
```


Name
Value
q0
q1
q2
q3
clk
1
1
0
0
1
0.0 us
0.1 us
0.2 us
0.3 us
0.4 us
0.5 us
0.6 us
0.7 us
0.8 us
0.9 us