



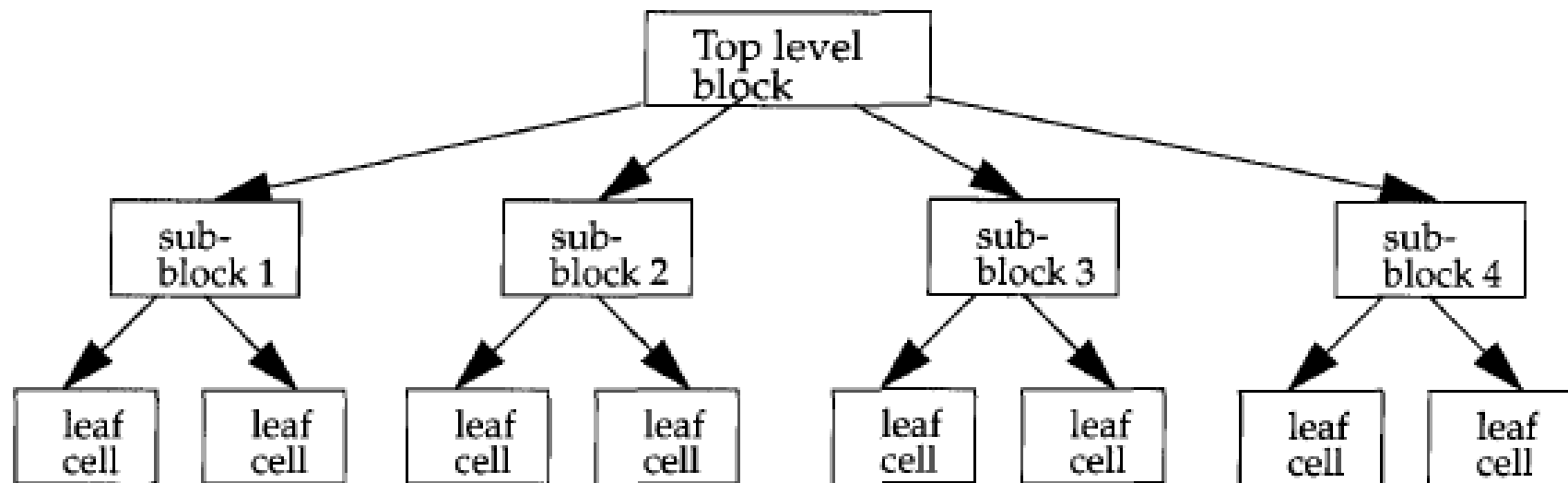
Projektovanje digitalnih sistema

Metodologije dizajna

■ Metodologije dizajna

■ Odozgo ka dolje (top-down)

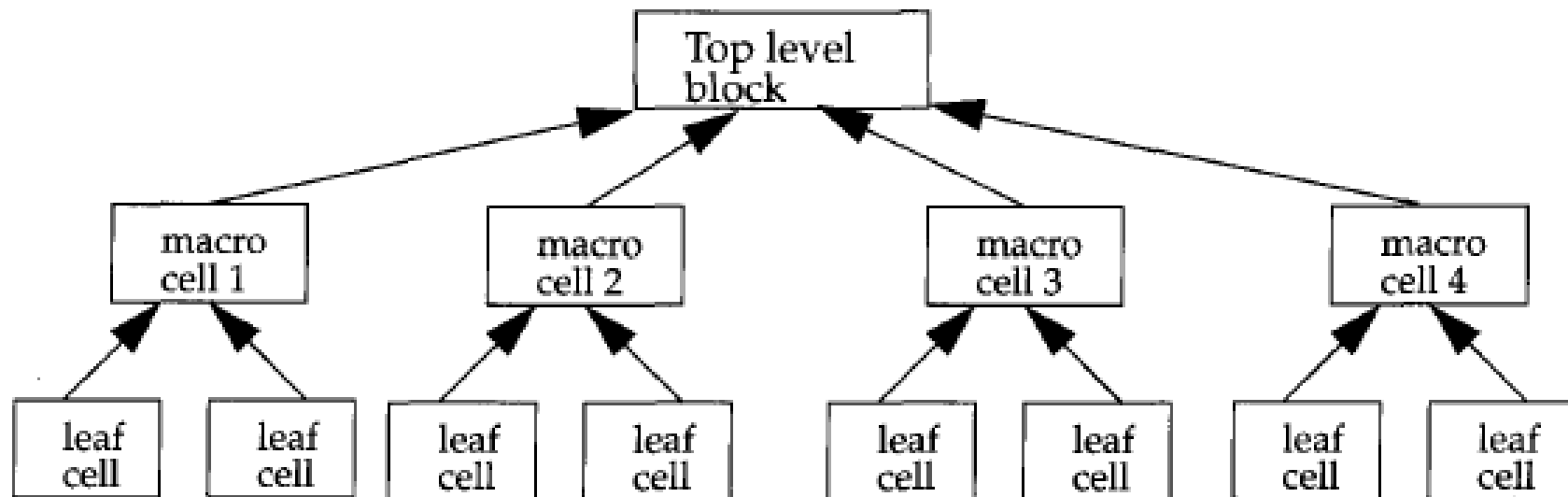
- Definiše se blok najvišeg nivoa i identifikuju se manji blokovi neophodni za njegovu implementaciju
- Manji blokovi se dijele na još manje blokove dok se ne dođe do ćelija koje se ne mogu više dijeliti



■ Metodologije dizajna - nastavak

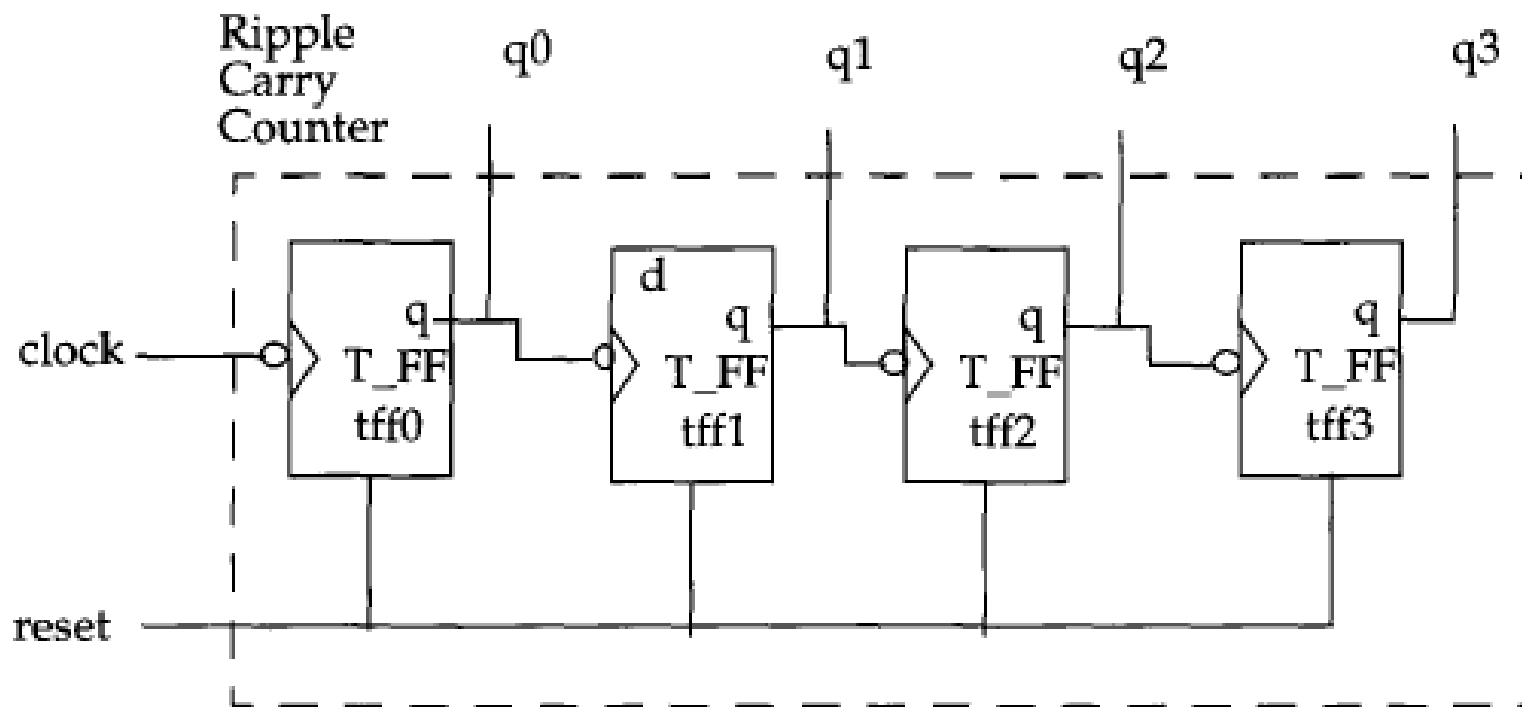
■ Odozdo ka gore (bottom-up)

- Prvo se identifikuju blokovi (ćelije) koji su dostupni za implementaciju
- Formiraju se veći blokovi koristeći dostupne ćelije
- Ovi se blokovi koriste za realizaciju blokova višeg nivoa, sve dok se ne dođe do bloka najvišeg nivoa



- Tipično se koristi kombinacija ove dvije metodologije

■ Primjer: 4-bitni *ripple carry* brojač

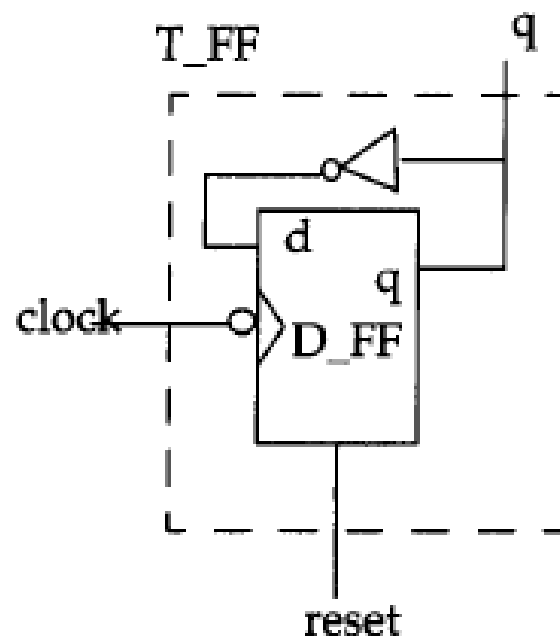


- Sastoji se od *toggle* flip flopova (T_FF) koji reaguju na silaznu ivicu
- Kako napraviti T_FF?

■ Primjer: 4-bitni *ripple carry* brojač - nastavak

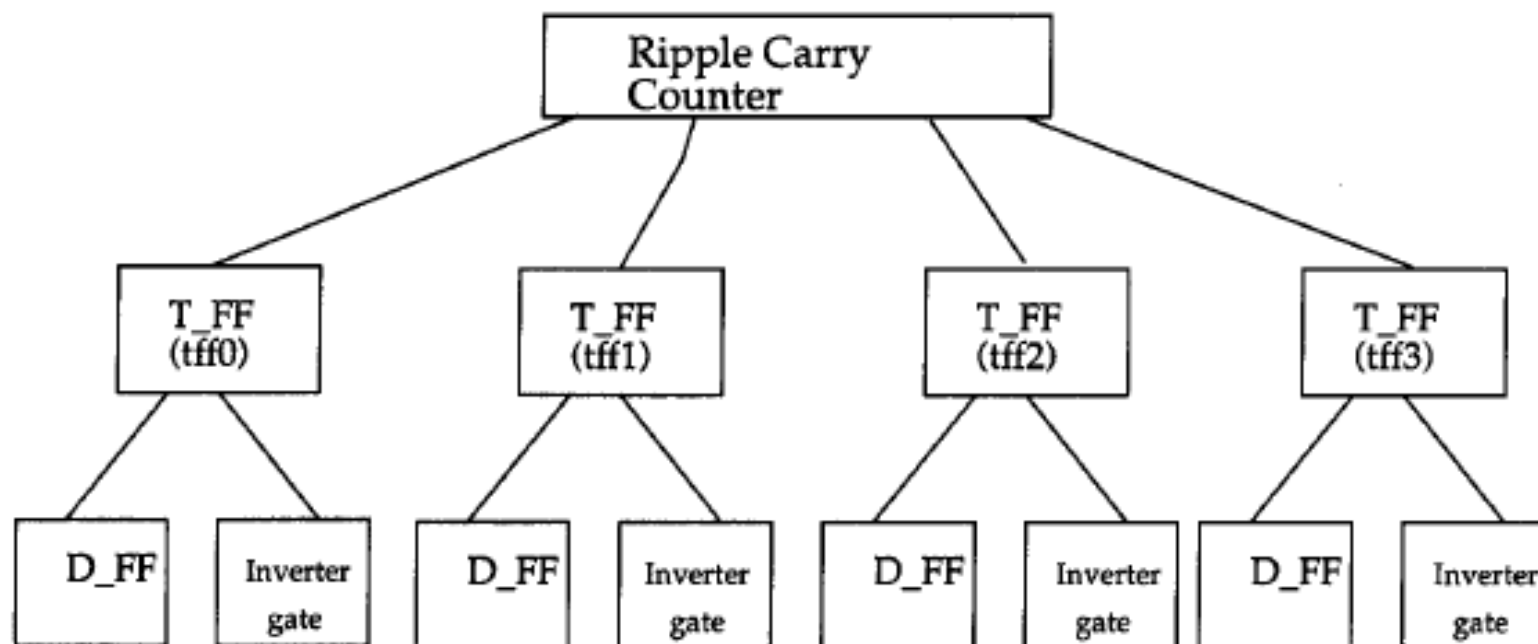
- T_FF se može napraviti od D flip flopa koji reaguje na silaznu ivicu i invertora (pod pretpostavkom da D flip flop nema invertovani izlaz):

reset	q_n	q_{n+1}
1	1	0
1	0	0
0	0	1
0	1	0



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- Brojač je izrađen u hijerarhijskom obliku, korišćenjem manjih blokova:





■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- Kod *top-down* metodologije prvo bi specificirali funkcionalnost brojača (on je blok najvišeg nivoa)
 - Nakon toga se brojač implementira pomoću blokova T_FF
 - T_FF se implementira pomoću D flip flopa i invertora
 - Dakle, veće blokove dijelimo u manje dok ne odlučimo da dalje dijeljenje nema smisla
- Kod *bottom-up* metodologije išlo bi se u suprotnom smjeru: kombinuju se mali blokovi za realizaciju većih
 - Napravili bi D_FF koristeći logička kola (AND, OR, NOT, ...) ili čak pomoću tranzistora
 - Kod D flip flopova bi se susreli *top-down* i *bottom-up* tokovi



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- **Modul**– osnovni gradivni blok u Verilogu
- Modul može biti osnovni element ili kombinacija blokova nižeg nivoa
- Elementi se grupišu u module da obezbijede funkcionalnost koja bi se koristila na više mjesta unutar dizajna
- Modul obezbjeđuje potrebnu funkcionalnost prema blokovima višeg nivoa preko portova (input, output), ali od njih “sakriva” unutrašnju implementaciju
- Na prethodnoj slici, *Ripple Carry counter*, *T_FF* i *D_FF* su primjeri modula
- Modul se deklariše pomoću ključne riječi **module**
- Na kraju definicije modula stavlja se ključna riječ **endmodule**



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

■ Svaki modul mora imati

- svoje ime (*module_name*) koje identifikuje taj modul
- *module_terminal_list* koji opisuje ulazne i izlazne priključke modula

■ Opšti slučaj:

```
module <module_name> <module_terminal_list>;
```

```
...
```

```
<sadržaj modula>
```

```
...
```

```
endmodule
```

■ T_FF bi se mogao definisati kao modul na ovaj način:

```
module T_FF (q, clock, reset);
```

```
...
```

```
<funkcionalnost toggle flip flopa>
```

```
...
```

```
endmodule
```



- Primjer: 4-bitni *ripple carry* brojač - nastavak
- Modul predstavlja **šablon** od kojeg se kreira stvarni objekat
- Svaki **objekat** ima svoje vlastito ime, promjenljive, parametre i I/O interfejs
- Proces kreiranja objekta koristeći šablon modula naziva se *instantiation* (instanciranje), a objekti se nazivaju *instances* (instance)
- Blok najvišeg nivoa u primjeru brojača kreira 4 instance T_FF šablona
- Svaki T_FF instancira D_FF i invertor
- Svaka od ovih instanci mora imati jedinstveno ime

■ Primjer: 4-bitni *ripple carry* brojač - nastavak

■ Instanciranje modula (Napomena: // - predstavlja komentar)

// Definirati modul najvišeg nivoa sa imenom ripple_carry_counter

module ripple_carry_counter(q, clk, reset);

output [3:0] q; // deklaracija vektora i I/O signala (objašnjenje kasnije)

input clk, reset;

// Kreiraju se 4 instance modula T_FF. Svaka ima jedinstveno ime

// Svakoj instanci se prosleđuje set signala. Primjetiti da je svaka
instanca kopija modula T_FF

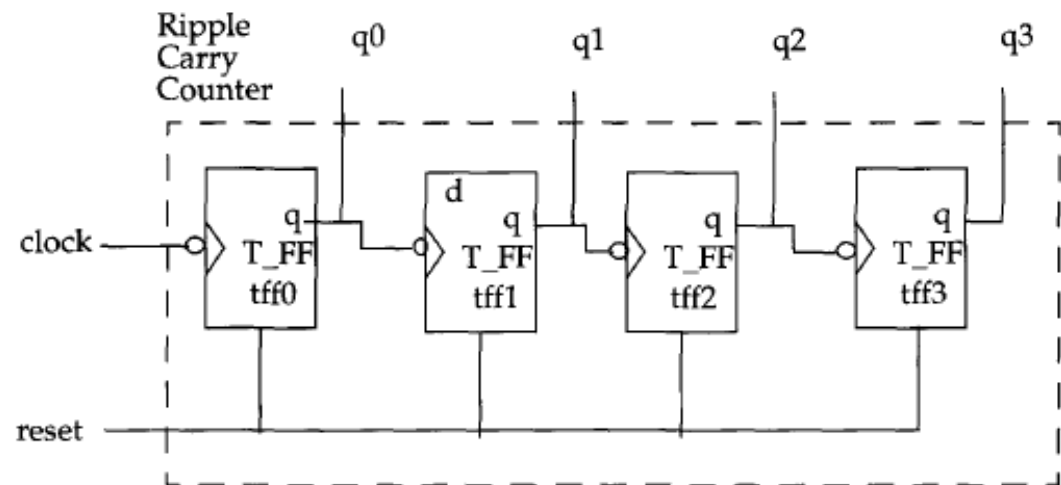
T_FF tff0(q[0], clk, reset);

T_FF tff1(q[1], q[0], reset);

T_FF tff2(q[2], q[1], reset);

T_FF tff3(q[3], q[2], reset);

endmodule



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

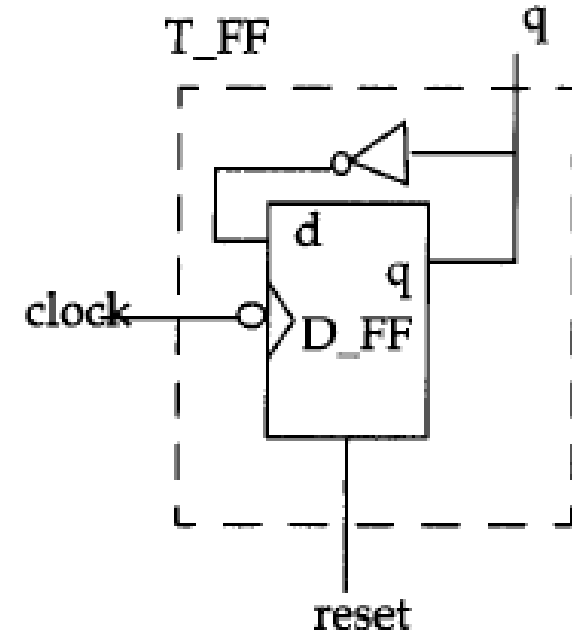
■ Instanciranje modula

// Definirati modul T_FF. On instancira D_FF. Pretpostavljamo da je
// modul D_FF definisan negdje drugo unutar dizajna

```
module T_FF(q, clk, reset);  
output q;  
input clk, reset;  
wire d;
```

```
D_FF dff0(q, d, clk, reset);  
not n1(d,q); // not kolo je Verilog element
```

```
endmodule
```





■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- U Verilogu nije dozvoljeno ugnijezditi module!
- Unutar jedne definicije modula ne smije se naći druga definicija modula
- Dozvoljeno je da se instancira kopija drugog modula
- **Praviti razliku između definicije i instance modula:**
 - definicija modula specificira kako će modul raditi, šta se u njemu nalazi i njegov interfejs
 - modul mora biti instanciran da bi se koristio u dizajnu

```
module ripple_carry_counter(q, clk, reset);  
output [3:0] q;  
input clk, reset;  
    module T_FF(q, clk, reset); // nedozvoljeno!!!  
    ...  
    <unutrašnjost modula T_FF  
    ...  
endmodule  
endmodule
```

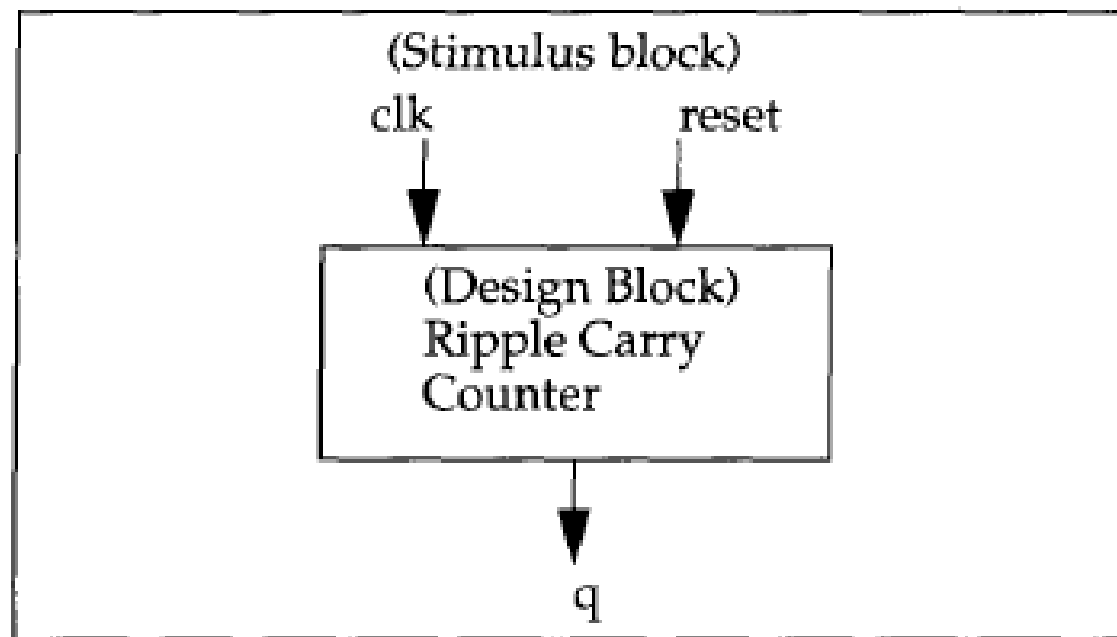


■ Komponente simulacije

- Kada se završi dizajn bloka, potrebno ga je testirati
- Funkcionalnost se testira tako što se dovedu različite kombinacije ulaznih signala (tzv. *stimulus*) i provjere se rezultati
- Blok sa ulaznim signalima se naziva *stimulus* blok (*test bench*) i poželjno ga je držati odvojeno od dizajna
- Dva su moguća stila primjene *stimulusa*:
 1. stimulus blok instancira blok dizajna
 2. i stimulus i blok dizajna se instanciraju u bloku najvišeg nivoa (top-level *dummy module*)

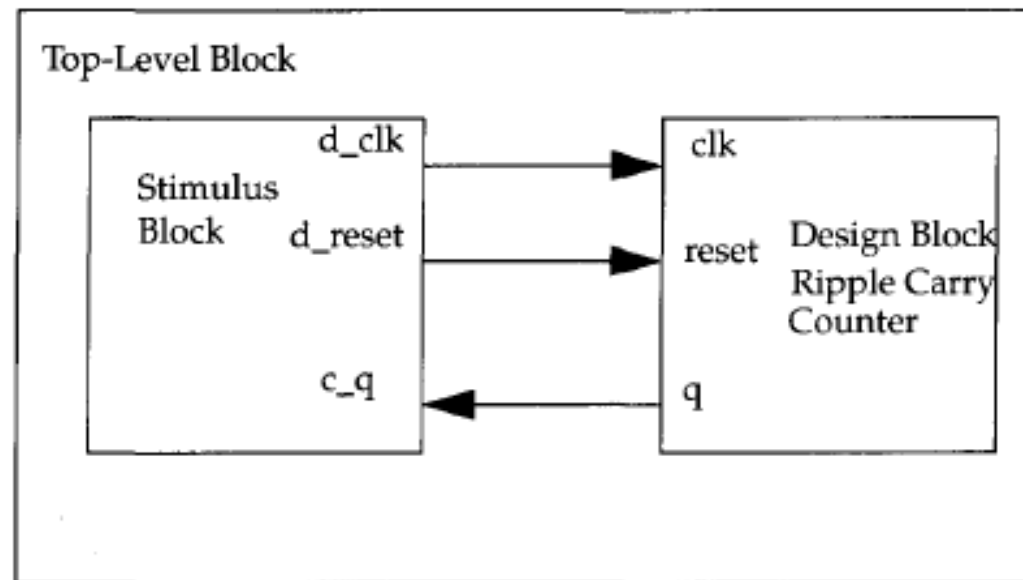
■ Komponente simulacije – nastavak

1. - stimulus blok direktno upravlja signalima u bloku dizajna
 - stimulus blok postaje blok najvišeg nivoa
 - on upravlja signalima *clk* i *reset* i provjerava i prikazuje izlazni signal *q*



■ Komponente simulacije – nastavak

- 2.
- stimulus blok vrši interakciju sa blokom dizajna jedino preko interfejsa
 - upravlja signalima *d_clk* i *d_reset* koji su povezani na signale *clk* i *reset* u bloku dizajna
 - provjerava i prikazuje signal *c_q*, koji je povezan sa signalom *q* u bloku dizajna
 - funkcija bloka najvišeg nivoa je samo da instancira blok dizajna i stimulus blok



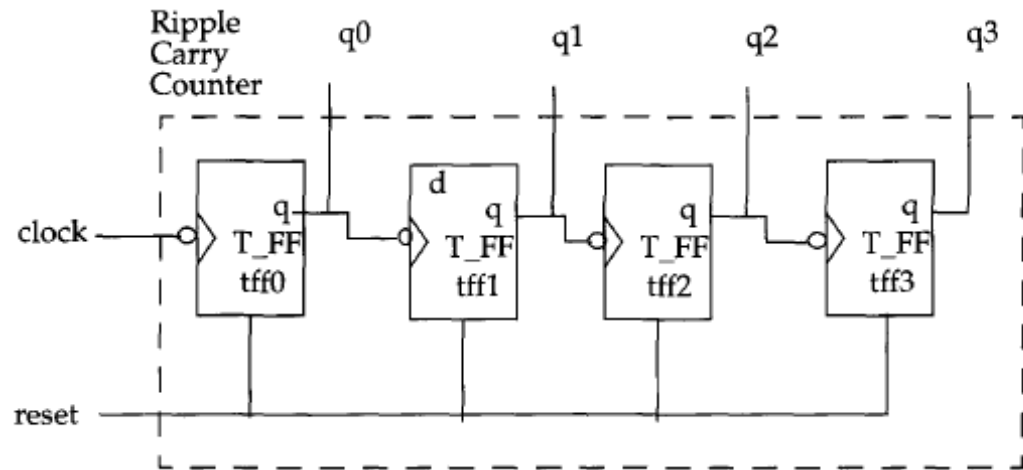
■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- Koristimo *top-down* metodologiju: prvo definišemo blok najvišeg nivoa

```
module ripple_carry_counter(q, clk, reset);  
  output [3:0] q;  
  input clk, reset;
```

```
  T_FF tff0(q[0], clk, reset);  
  T_FF tff1(q[1], q[0], reset);  
  T_FF tff2(q[2], q[1], reset);  
  T_FF tff3(q[3], q[2], reset);
```

```
endmodule
```



- Ovdje su korišćene 4 instance T_FF modula => moramo definisati modul T_FF

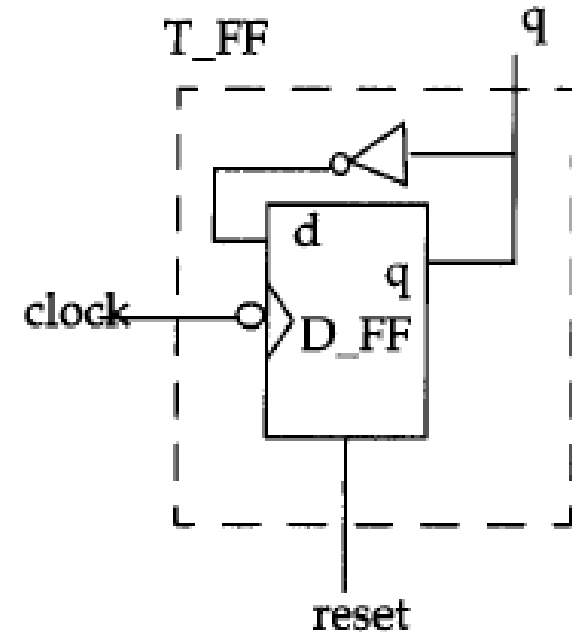
■ Primjer: 4-bitni *ripple carry* brojač - nastavak

■ Definicija bloka T_FF

```
module T_FF(q, clk, reset);  
  output q;  
  input clk, reset;  
  wire d;
```

```
  D_FF dff0(q, d, clk, reset);  
  not n1(d,q);
```

```
endmodule
```



■ Ovaj modul instancira D_FF => moramo definisati modul D_FF



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- Definicija bloka D_FF (uzećemo da se vrši asinhroni reset)

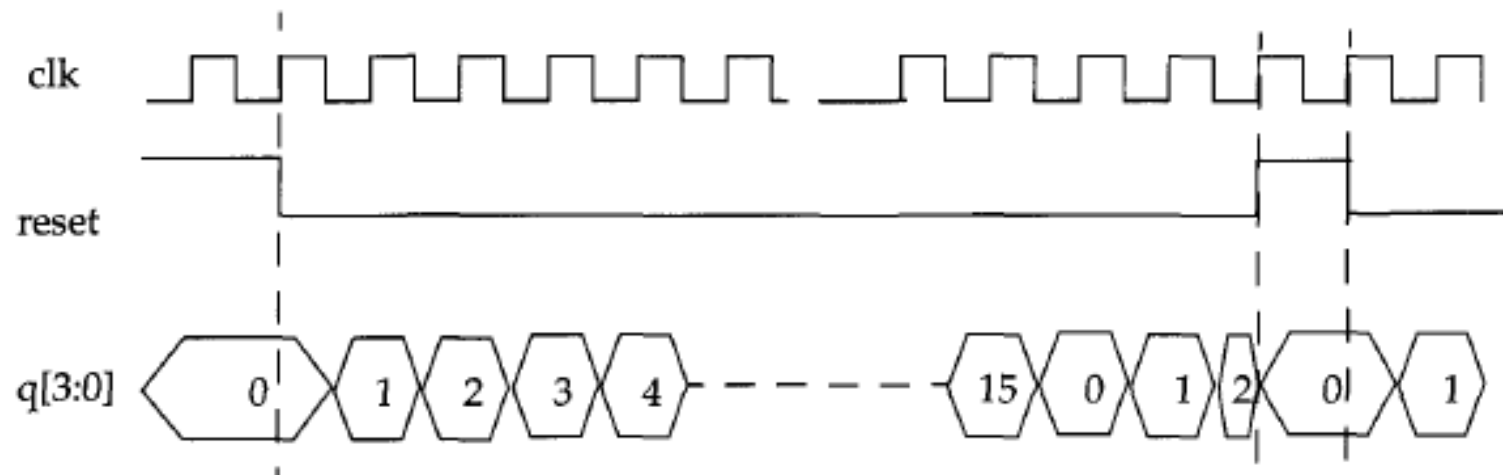
```
module D_FF(q, d, clk, reset);  
    output q;  
    input d, clk, reset;  
    reg q;  
    // Ne obraćati pažnju na sintaksu. Koncentrisati se na način dizajna  
    // u top-down metodologiji  
    always @(posedge reset or negedge clk)  
        if (reset)  
            q = 1'b0;  
        else  
            q = d;  
  
endmodule
```

- Svi moduli su definisani => blok dizajna je kompletiran

■ Primjer: 4-bitni *ripple carry* brojač - nastavak

■ Stimulus blok

- moramo napraviti stimulus blok kako bi provjerili ispravnost funkcionisanja dizajna brojača
- kontrolišemo signale *clk* i *reset*, da provjerimo i brojanje i resetovanje
- koristićemo sljedeći talasni oblik za testiranje dizajna:



- perioda *clk* signala je 10 jedinica vremena
- *reset* je aktivan od vremenskog trenutka 0 do 15 i od 195 do 205



■ Primjer: 4-bitni *ripple carry* brojač - nastavak

- Stimulus blok – primijenit ćemo stil direktnog upravljanja signalima u bloku dizajna

```
module stimulus;  
  reg clk;  
  reg reset;  
  wire[3:0] q;  
  ripple_carry_counter r1(q, clk, reset); // instancira blok dizajna  
  initial // Kontrola clk signala koji upravlja blokom dizajna. Perioda = 10  
    clk = 1'b0; // postavi clk na 0  
  always  
    #5 clk = ~clk; // invertuj clk svakih 5 vremenskih jedinica  
  ...
```

■ Primjer: 4-bitni *ripple carry* brojač - nastavak

■ ... nastavak

initial

begin

reset = 1'b1;

#15 reset = 1'b0;

#180 reset = 1'b1;

#10 reset = 1'b0;

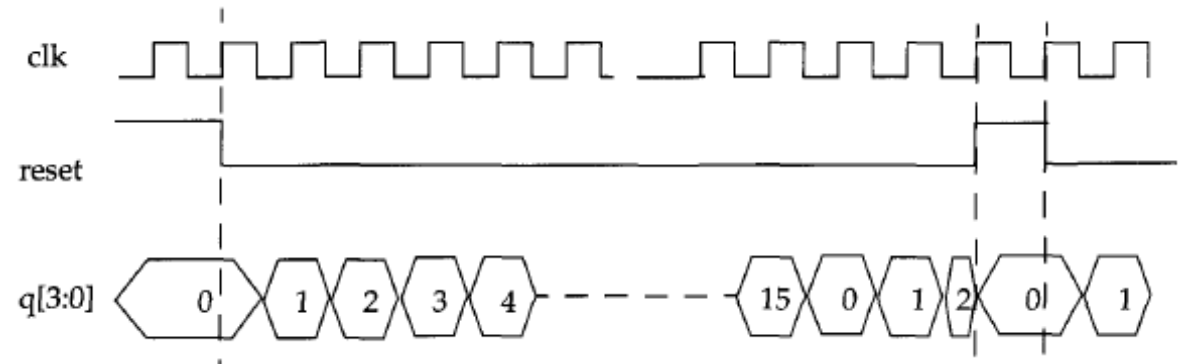
#20 \$finish; // prekini simulaciju

end

initial

\$monitor(\$time, " Izlaz q = %d ", q); // nadgledanje izlaza

endmodule





■ Primjer: 4-bitni *ripple carry* brojač - nastavak

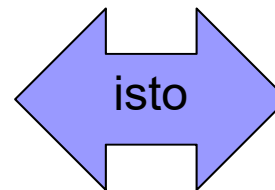
■ Izlaz simulacije

0 Izlaz q = 0
20 Izlaz q = 1
30 Izlaz q = 2
40 Izlaz q = 3
50 Izlaz q = 4
60 Izlaz q = 5
70 Izlaz q = 6
80 Izlaz q = 7
90 Izlaz q = 8
100 Izlaz q = 9
110 Izlaz q = 10
120 Izlaz q = 11
130 Izlaz q = 12
140 Izlaz q = 13
150 Izlaz q = 14
160 Izlaz q = 15
170 Izlaz q = 0
180 Izlaz q = 1
190 Izlaz q = 2
195 Izlaz q = 0
210 Izlaz q = 1
220 Izlaz q = 2

■ Verilog: leksičke konvencije

- Verilog je jezik u kome se pravi razlika između malih i velikih slova (case-sensitive)
- Sve ključne riječi se pišu malim slovima
- Prazno mjesto (\b), tab (\t), vraćanje na početak linije (\r) i prelazak u novi red (\n) predstavljaju tzv. “razmak”. Razmaci se ignorišu, osim kad su unutar stringova.

```
module addbit(a,b,ci,sum,co);  
input a,b,ci;output sum, co;  
wire a,b,ci,sum,co;endmodule
```



```
module addbit (  
    a,  
    b,  
    ci,  
    sum,  
    co);  
input      a;  
input      b;  
input      ci;  
output     sum;  
output     co;  
wire       a;  
wire       b;  
wire       ci;  
wire       sum;  
wire       co;  
endmodule
```



■ Verilog: leksičke konvencije – nastavak

■ Komentari se mogu unijeti na dva načina:

- Linijski komentar (počinje sa //) – preskače se sve do kraja linije
- Višelinijski komentar (počinje sa /* a završava sa */) – ne smiju biti ugniježdeni

`a = b && c; // Ovo je komentar u jednoj liniji`

`/* Ovo je komentar`

`u više linija */`

`/* ovo je /* nepravilni */ komentar */`



■ Verilog: leksičke konvencije – nastavak

■ Operatori mogu biti *unarni*, *binarni* i *trojni*:

- Unarni operatori se pišu ispred operanda
- Binarni operandi se pišu između operanada
- Trojni operandi se označavaju sa dva različita simbola koji razdvajaju tri operanda

$a = \sim b;$ // \sim je unarni operator. b je operand

$a = b \&\& c;$ // $\&\&$ je binarni operator. b i c su operandi

$a = b ? c : d;$ // $? :$ je trojni operator. b , c i d su operandi

■ Verilog: predstavljanje brojeva

- Brojevi sa specificiranom i nespecificiranom veličinom
- Format brojeva sa specificiranom veličinom:

`<veličina>'<brojna osnova><broj>`

- `<veličina>` se piše u dekadnom sistemu i govori koliko ima bitova u predstavljanom broju – **opciono**
- `<brojna osnova>` može biti decimalna ('d ili 'D), heksadecimalna ('h ili 'H), binarna ('b ili 'B) ili oktalna ('o ili 'O) - **opciono**
- `<broj>` se predstavlja neprekidnim nizom cifara iz skupa {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f}. U pojedinim brojnim sistemima se koriste samo podskupovi ovog skupa cifara. Mogu se koristiti i velika slova za predstavljanje cifara.

`4'b1111` // ovo je 4-bitni binarni broj: $1111_{(2)} = 15_{(10)}$

`12'habc` // ovo je 12-tobitni heksad. broj: $ABC_{(16)} = 2748_{(10)} = 101010111100_{(2)}$

`16'd255` // ovo je 16-tobitni decimalni broj: $255_{(10)} = 0000000011111111_{(2)}^{27}$



■ Verilog: predstavljanje brojeva – nastavak

- Format brojeva sa nespecificiranom veličinom:

`'<brojna osnova><broj>`

- `<brojna osnova>` može biti izostavljena => podrazumijeva se da je broj u dekadnom brojnom sistemu
- Veličina nespecificiranih brojeva zavisi od korišćenog simulatora/uređaja i iznosi najmanje 32 bita

`23456 // ovo je 32-bitni dekadni broj`

`'hc3 // ovo je 32-bitni heksadekadni broj`

`'o21 // ovo je 32-bitni oktalni broj`



■ Verilog: predstavljanje brojeva – nastavak

- Nepoznata ili neispravna vrijednost (često potrebno kod realnih kola) se označava sa **x**
- Stanje visoke impedanse (*plivajuće* stanje) se označava sa **z** – pogodno za opisivanje *tristate* bafera

12'h13x // ovo je 12-bitni broj; 4 najniža bita su nepoznata

6'hx // ovo je 6-bitni heksadekadni broj

32'bz // ovo je 32-bitni “broj” kod kojeg svih 32 bita imaju visoku impedansu

- **x** i **z** predstavljaju grupu od 4 bita u hex sistemu, grupu od 3 bita u oct sistemu i 1 bit u binarnom sistemu
- Ako je bit najviše težine u broju 0, **x** ili **z**, broj se automatski **proširuje** sa 0, **x** ili **z**, respektivno
- Na ovaj način se olakšava zadavanje vektora – vidi gornji primjer
- Ako je bit najviše težine u broju 1, broj se **proširuje** nulama!



- [illegible]



■ Verilog: predstavljanje realnih brojeva

- Verilog podržava realne konstante i promjenljive
- Realni brojevi ne mogu sadržati 'Z' i 'X'
- Realni brojevi se mogu specificirati u decimalnoj ili naučnoj notaciji:
 - Decimalna notacija: <vrijednost>.<vrijednost>
 - Naučna notacija: <mantisa>E<eksponent>
- Realni brojevi se zaokružuju na najbližu cijelu vrijednost, kada se dodjeljuju cjelobrojnoj promjenljivoj

1.4 // decimalni broj: 1.4

0.8 // decimalni broj: 0.8

5.7E4 // decimalni broj: 5.7×10^4 , odnosno 57000

■ Verilog: predstavljanje brojeva - primjer

```
module signed_number;
```

```
reg [31:0] a;
```

```
initial begin
```

```
    a = 14'h1234;
```

```
    $display ("Trenutna vrijednost a = %h", a);
```

```
    a = -14'h1234;
```

```
    $display ("Trenutna vrijednost a = %h", a);
```

```
    a = 32'hDEAD_BEEF;
```

```
    $display ("Trenutna vrijednost a = %h", a);
```

```
    a = -32'hDEAD_BEEF;
```

```
    $display ("Trenutna vrijednost a = %h", a);
```

```
    #10 $finish;
```

```
end
```

```
endmodule
```

IZLAZ:

Trenutna vrijednost a = 00001234

Trenutna vrijednost a = ffffedcc

Trenutna vrijednost a = deadbeef

Trenutna vrijednost a = 21524111



■ Verilog: predstavljanje stringova

- String je sekvenca karaktera koji se nalaze unutar znakova navoda
- String se mora nalaziti u okviru jedne linije
- String se tretira kao sekvenca jednobajtnih ASCII vrijednosti: jedna ASCII – jedan karakter

“Hello Verilog World” // ovo je string

“a / b” // ovo je string

- Verilog ne smješta terminacioni karakter
- Kada je promjenljiva veća nego što je potrebno da se smjesti string, dopunjava se nulama sa lijeve strane
- Neki karakteri se mogu koristiti u stringu samo ako se ispred njih nalazi *escape* karakter: `\n` – novi red, `\t` – tab, `\\` – backslash (`\`), `\"` – navodnici, `%%` – procenat



■ Verilog: identifikatori i ključne riječi

- Identifikator je naziv nekog objekta preko kojeg se taj objekat referencira
- Mora počinjati (malim ili velikim) slovom ili donjom crtom ('_')
- Identifikator može sadržati slova, cifre, donju crtu i znak za dolar (a-z A-Z 0-9 _ \$)
- Identifikator ne može počinjati sa \$ jer su takve konstrukcije rezervisane za sistemske funkcije
- Dužina naziva može biti do 1024 karaktera
- Ključne riječi su specijalni identifikatori rezervisani za definisanje jezičkih konstrukcija

reg value; // **reg** je ključna riječ; **value** je identifikator

input clk; // **input** je ključna riječ; **clk** je identifikator



■ Verilog: tipovi podataka

- Verilog podržava 4 vrijednosti i 8 *jačina* signala da bi se modelovao realni hardver
- Vrijednosti signala su:
 - 0 => logička nula
 - 1 => logička jedinica
 - x => nepoznata ili neispravna vrijednost
 - z => visoka impedansa (*plivajuće* stanje)
- 0 i 1 mogu imati neki od sledećih nivoa *jačine* (*strenth*):
 - supply
 - strong
 - pull
 - large
 - weak
 - medium
 - small
 - highz
- Poređani su od najjačeg prema najslabijem



■ Verilog: tipovi podataka – nastavak

- Ako se dva signala nejednake jačine nađu na istom vodu, preovlađaće jači signal
- Ako se dva signala jednake jačine nadmeću na istom vodu – rezultat je nepoznat (x)
- Ako su svi signali na istom vodu u stanju visoke impedanse (isključeni *tristate* baferi) – signal na vodu će biti u *plivajućem* stanju (z)