

■ Verilog: moduli i portovi – nastavak

■ Primjer – SR latch

// naziv modula i lista portova

```
module SR_latch(Q, Qbar, Sbar, Rbar);
```

// deklaracija portova

```
output Q, Qbar;
```

```
input Sbar, Rbar;
```

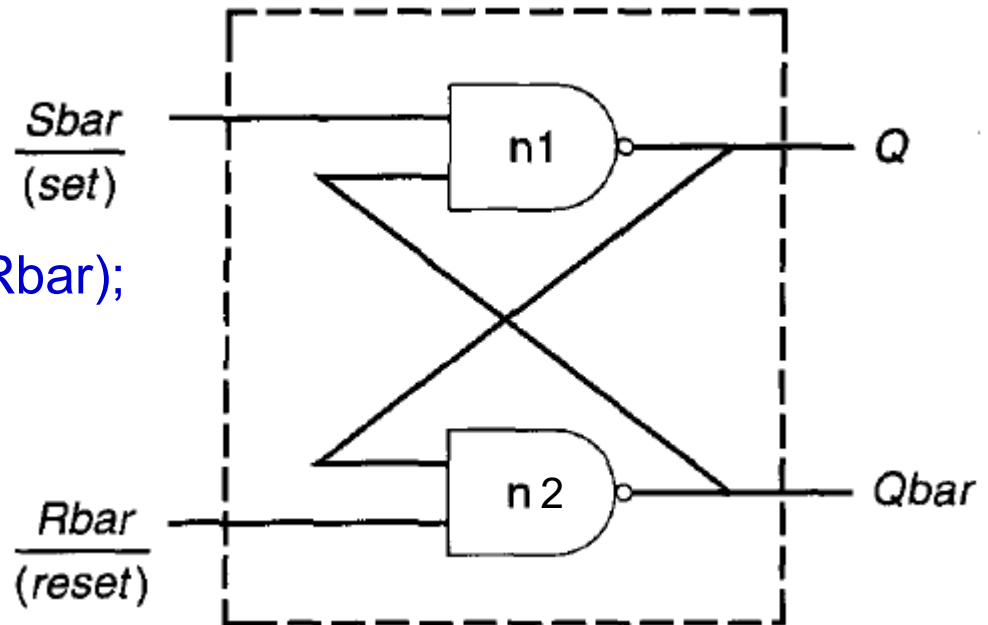
// instanciranje modula nižeg nivoa

// ovdje su to Verilog elementi – nand kola

```
nand n1(Q, Sbar, Qbar);
```

```
nand n2(Qbar, Rbar, Q);
```

```
endmodule
```



■ Verilog: moduli i portovi – nastavak

■ Primjer – SR latch (stimulus modul)

// Stimulus modul – on će biti modul najvećeg nivoa

```
module Top;
```

// deklaracija promjenljivih

```
wire q, qbar;
```

```
reg set, reset;
```

// instanciranje modula nižeg nivoa – u našem slučaju SR latch

// na SR latch dovodimo invertovane signale *set* i *reset*

```
SR_latch m1(q, qbar, ~set, ~reset);
```

```
initial
```

```
begin
```

```
$monitor($time, " set = %b, reset= %b, q= %b\n",set,reset,q);
```

```
set = 0; reset = 0;
```

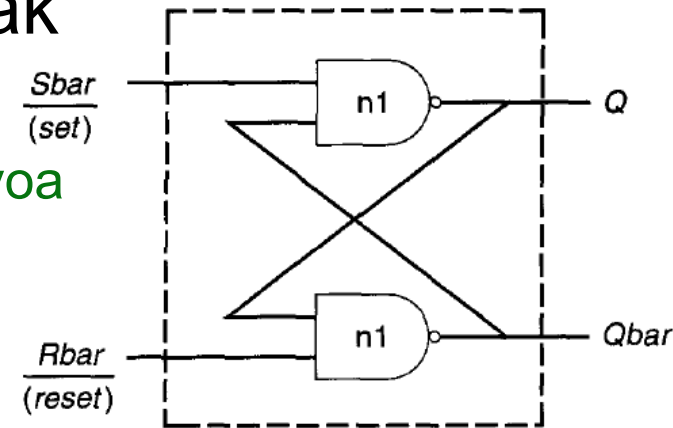
```
#5 reset = 1;
```

```
#5 reset = 0;
```

```
#5 set = 1;
```

```
end
```

```
endmodule
```



Izrazi kao $\sim set$ ili $\sim reset$ su dozvoljeni u port listi prilikom instanciranja. Ipak, složenije izraze treba izbjegavati radi lakše čitljivosti koda.

IZLAZ:

0 set = 0, reset= 0, q= x

5 set = 0, reset= 1, q= 0

10 set = 0, reset= 0, q= 0

15 set = 1, reset= 0, q= 1



Projektovanje digitalnih sistema

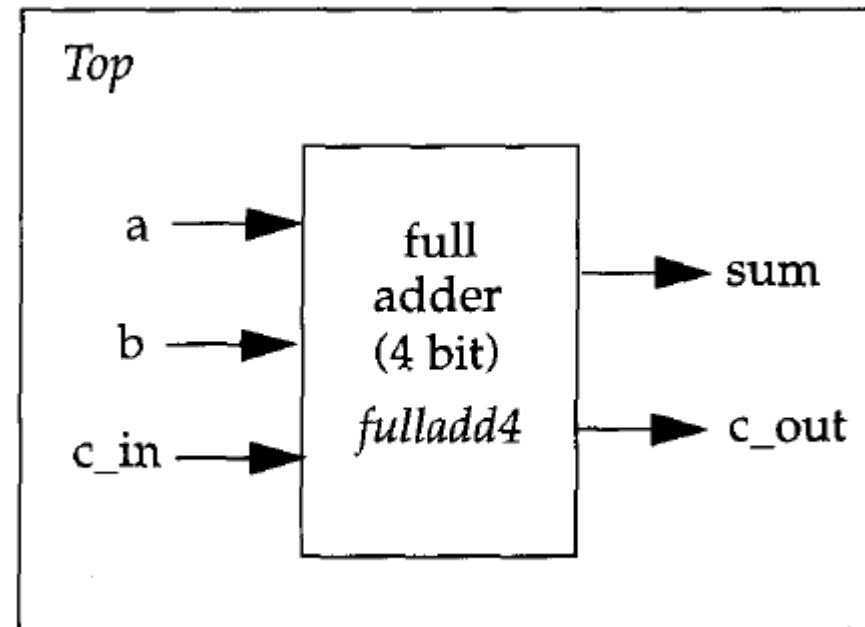
Portovi

■ Verilog: portovi

- Portovi su interfejs preko kojeg modul komunicira sa okruženjem
- Unutrašnjost modula nije “vidljiva” okruženju – interakcija je isključivo preko portova

- Primjer – 4-bitni potpuni sabirač

- Modul *Top* je najvišeg nivoa i on nema potrebu za portovima
- Sabirač ima 3 ulazna i dva izlazna porta
- Deklaracije modula (imena i lista portova) bi zato izgledale ovako:



```
module fulladd4(sum, c_out, a, b, c_in); // modul sa listom portova  
module Top; // nema liste portova, modul najvišeg nivoa u simulaciji
```

■ Verilog: deklaracija portova

- Portovi se mogu deklarirati kao ulazni (**input**), izlazni (**output**) ili bidirekcionni (**inout**)
- Deklaracija portova za modul *fulladd4* iz prethodnog primjera:

```
module fulladd4(sum, c_out, a, b, c_in);  
    // početak bloka za deklaraciju portova  
    output [3:0] sum;  
    output c_out;  
    input [3:0] a, b;  
    input c_in;  
    // kraj bloka za deklaraciju portova  
    ...  
    <sadržaj modula>  
    ...  
endmodule
```

■ Verilog: deklaracija portova – nastavak

- U Verilogu se portovi implicitno deklarišu kao **wire**
- Za **input** i **inout** portove to je očekivano i uobičajeno
- Kod **output** portova je nekad potrebno da zadrže svoju vrijednost i oni se tada moraju deklarirati kao **reg**
- Na primjer, D flip flop mora čuvati vrijednost na izlazu (**q**) do nailaska sljedećeg taktnog impulsa:

```
module DFF(q, d, clk, reset);  
    output q;  
    reg q; // izlaz q mora zadržati vrijednost  
    input d, clk, reset;  
    ...  
    <sadržaj modula>  
    ...  
endmodule
```

■ Verilog: deklaracija portova – nastavak

- **input** i **inout** portovi se ne mogu deklarirati kao **reg** – oni ne treba da čuvaju vrijednost već da reflektuju promjene spoljašnjih signala na koje su povezani

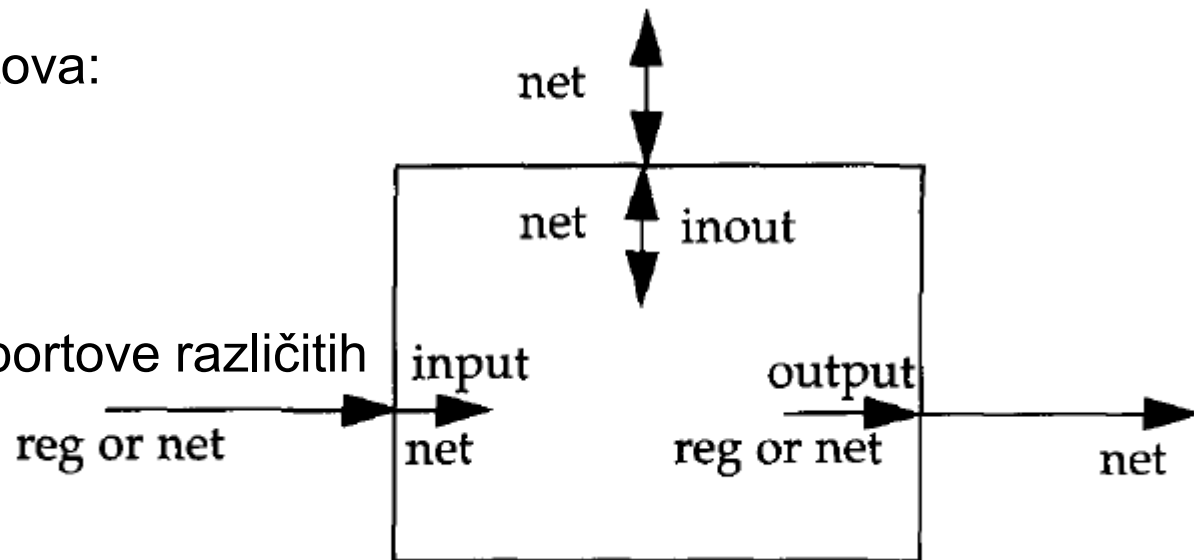
- Pravila povezivanja portova:

- Dozvoljeno je povezati portove različitih dužina (**warning**)

- Dozvoljeno je da portovi ostanu nepovezani

- Npr. neki izlazni port može da služi za potrebe kontrole rada (*debugging*) pa ga nije potrebno povezati na spoljašnji signal:

```
fulladd4 fa0(ZBIR, , A, B, C_UL); // izlazni port c_out nije povezan
```



■ Verilog: primjer nedozvoljenog povezivanja portova

```
module Top;  
    // deklaracija promjenljivih za konekciju na sabirač  
    reg [3:0] A, B;  
    reg C_UL;  
    reg [3:0] ZBIR;  
    wire C_IZ;  
  
    // instanciranje fulladd4, instanca se zove fa0  
    fulladd4 fa0(ZBIR, C_IZ, A, B, C_UL);  
    // nedozvoljena konekcija jer je izlazni port sum u modulu fulladd4  
    // povezan na registarsku promjenljivu ZBIR  
    ...  
    <stimulus>  
    ...  
endmodule
```


■ Verilog: povezivanje portova na spoljašnje signale

- Dva metoda povezivanja signala (specificiranih prilikom instanciranja) i portova (u definiciji modula):

- Uređena lista
- Po imenu

- Ova dva metoda se ne mogu kombinovati u istoj instanci

■ Uređena lista

- Signali se prilikom instanciranja navode u istom redoslijedu kao portovi u definiciji modula
- Intuitivan metod
- Pogodan kada nema veliki broj portova

■ Verilog: povezivanje portova na spoljašnje signale

■ Uređena lista – nastavak

```
module Top;  
    reg [3:0] A, B; // deklaracija promjenljivih za povezivanje  
    reg C_UL;  
    wire [3:0] ZBIR;  
    wire C_IZ;  
    // instanciranje - signali su redom povezani na portove (po poziciji)  
    fulladd4 fa_uredjena_lista(ZBIR, C_IZ, A, B, C_UL);  
    ... <stimulus>...  
endmodule
```

```
module fulladd4(sum, c_out, a, b, c_in);  
    output [3:0] sum;  
    output c_cout;  
    input [3:0] a, b;  
    input c_in;  
    ...<unutrašnjost modula>...  
endmodule
```

■ Verilog: povezivanje portova na spoljašnje signale

■ Povezivanje po imenu

- Kada modul ima veliki broj portova nije praktično pamtit i redoslijed
- Veze se mogu specificirati u proizvoljnom redoslijedu, ako se navedu imena portova:

```
fulladd4 fa_imenom(.c_out(C_IZ), .sum (ZBIR), .b(B), .c_in(C_UL), .a(A));
```

- Specificiraju se samo oni portovi koje treba povezati na spoljašnje signale – ostali se ne pominju:

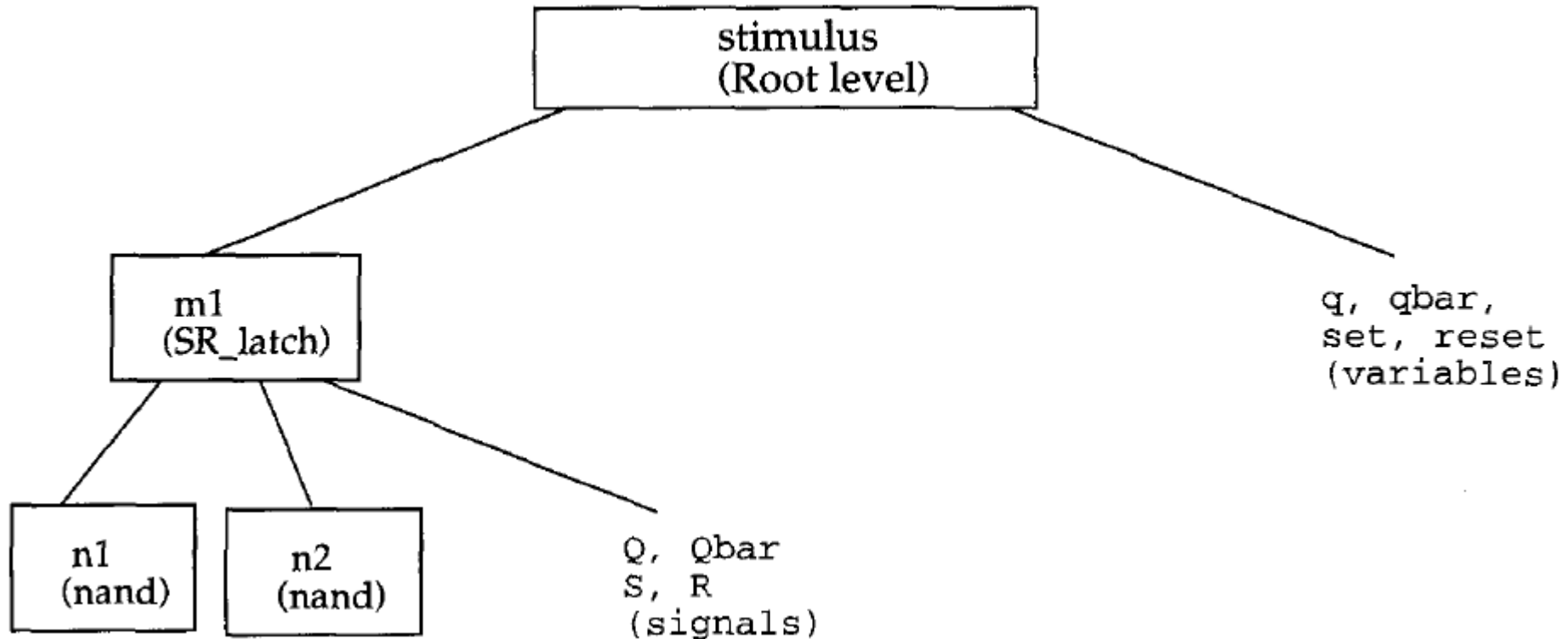
```
fulladd4 fa_byname(.sum(ZBIR), .b(B), .c_in(C_UL), .a(A)); // c_out "visi"
```

- Ako se ne mijenjaju imena portova, može se mijenjati njihov redoslijed u definiciji modula, a da se to ne reflektuje na instance

■ Verilog: hijerarhijsko imenovanje

- Verilog podržava hijerarhijsku metodologiju dizajna
- Svaka *instanca modula*, *signal* ili *promjenljiva* se definiše pomoću identifikatora
- Pojedini identifikator ima jedinstveno mjesto u hijerarhiji dizajna
- Referenciranje preko hijerarhijskog imena omogućava da označimo svaki identifikator unutar dizajna sa **jedinstvenim** imenom
- Hijerarhijsko ime je lista identifikatora rastavljenih tačkom, za svaki nivo hijerarhije
- Dakle, bilo koji identifikator se može referencirati sa bilo kojeg mjesta unutar dizajna navodeći kompletno hijerarhijsko ime
- Modul najvišeg nivoa se nigdje ne referencira i naziva se **korijen** (*root*)
- Dodjeljivanje jedinstvenog imena identifikatoru: kreće se od modula najvišeg nivoa i prati se “put” duž hijerarhije dizajna do željenog identifikatora

■ Verilog: hijerarhijsko imenovanje- primjer SR latch



stimulus
stimulus.q
stimulus.qbar
stimulus.set
stimulus.reset
stimulus.m1

stimulus.m1.Q
stimulus.m1.Qbar
stimulus.m1.S
stimulus.m1.R
stimulus.m1.n1
stimulus.m1.n2

Podsjećanje: `%m` u funkciji
`$display`

■ Verilog: hijerarhijsko imenovanje – nastavak

■ Prednost:

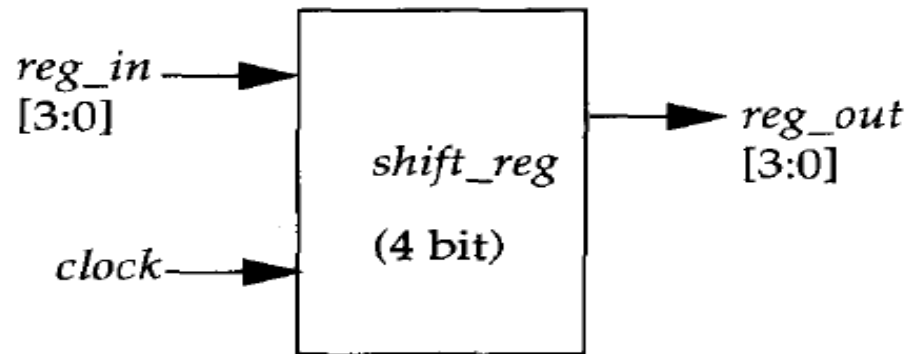
- olakšava traženje grešaka kod internih signala dizajna, naročito ako ti signali nisu priključni signali najvišeg nivoa (pinovi uređaja)

■ Mana:

- nekada se tokom sintetizacije dizajna, saglasno ugrađenim strategijama u sintetizator, vrši preimenovanje hijerarhijskih imena, pa originalna prestanu da postoje (ovo se može riješiti kod nekih alata zadavanjem posebnih opcija)
- ako Verilog kod treba translirati u neki drugi HDL, hijerarhijska imena nisu uvijek prenosiva

■ Vježba (test):

- Četvorobitni pomjerački registar sa paralelnim upisom je prikazan na slici. Napisati definiciju za modul *shift_reg* koji definiše ovaj registar. Uključiti listu portova i njihovih deklaracija. Ne uključivati funkcionalni sadržaj modula.



- Deklarisati modul najvišeg nivoa po imenu *stimulus*. Definirati *reg* promjenljive REG_IN (4 bita) i CLK (1 bit), kao i *wire* promjenljivu REG_OUT (4 bita). Instancirati modul *shift_reg* po imenu *sr1*. Povezati portove koristeći metod *uređene liste*.
- Povezati portove *po imenu*.
- Napisati hijerarhijska imena za: promjenljive REG_IN, CLK, REG_OUT, instancu *sr1* i njene portove *reg_in* i *clock*

■ Verilog: (skoro) najniži nivo abstrakcije – gate level

- (najniži je *switch* odnosno *transistor* level, ali se on rijetko koristi)
- Kolo se opisuje preko logičkih kapija (*gate-ova*: *and*, *or*, *nand*, *nor*, ...)
- Odnos logičkog dijagrama i opisa u Verilogu je 1:1
- Verilog podržava osnovna logička kola kao predefinisane *primitive*
- Instanciraju se kao i moduli samo što nije potrebno definisati module
- Dvije klase osnovnih logičkih kapija:
 - *and/or*
 - ❖ imaju više (skalarnih) ulaza i jedan (skalarni) izlaz
 - ❖ prvi u listi portova je izlaz, a ostali su ulazi
 - *buf/not*
 - ❖ imaju jedan (skalarni) ulaz i jedan ili više (skalarnih) izlaza
 - ❖ posljednji u listi portova je ulaz, a ostali su izlazi

■ Verilog: and/or klasa logičkih kapija

- Izlaz se određuje čim dođe do promjene na nekom od ulaza
- Naziv instance može ali ne mora da se daje ugrađenim primitivama

```
wire IZLAZ, ULAZ1, ULAZ2, ULAZ3;
```

```
// osnovna logička kola
```

```
and a1(IZLAZ, ULAZ1, ULAZ2);
```

```
nand na1(IZLAZ, ULAZ1, ULAZ2);
```

```
or or1(IZLAZ, ULAZ1, ULAZ2);
```

```
nor nor1(IZLAZ, ULAZ1, ULAZ2);
```

```
xor x1(IZLAZ, ULAZ1, ULAZ2);
```

```
xnor nx1(IZLAZ, ULAZ1, ULAZ2);
```

```
// više od dva ulaza: troulazno nand kolo
```

```
nand na_3inp(IZLAZ, ULAZ1, ULAZ2, ULAZ3);
```

```
// instanciranje logičke kapije bez naziva instance
```

```
and (IZLAZ, ULAZ1, ULAZ2); // dozvoljeno u Verilogu
```

■ Verilog: and/or klasa logičkih kapija – nastavak

		i1			
i2	and	0	1	x	z
	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

		i1			
i2	or	0	1	x	z
	0	0	1	x	x
	1	1	1	1	1
	x	x	1	x	x
	z	x	1	x	x

		i1			
i2	xor	0	1	x	z
	0	0	1	x	x
	1	1	0	x	x
	x	x	x	x	x
	z	x	x	x	x

		i1			
i2	nand	0	1	x	z
	0	1	1	1	1
	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x

		i1			
i2	nor	0	1	x	z
	0	1	0	x	x
	1	0	0	0	0
	x	x	0	x	x
	z	x	0	x	x

		i1			
i2	xnor	0	1	x	z
	0	1	0	x	x
	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

■ Verilog: buf/not klasa logičkih kapija

```
wire IZLAZ1, IZLAZ2, ULAZ;
```

```
// osnovna logička kola
```

```
buf b1(IZLAZ1, ULAZ);
```

```
not n1(IZLAZ1, ULAZ);
```

```
// više od jednog izlaza
```

```
buf b1in_2out(IZLAZ1, IZLAZ2, ULAZ);
```

```
// instanciranje logičke kapije bez naziva instance
```

```
not (IZLAZ1, ULAZ); // dozvoljeno u Verilogu
```

buf	in	out
	0	0
	1	1
	x	x
	z	x

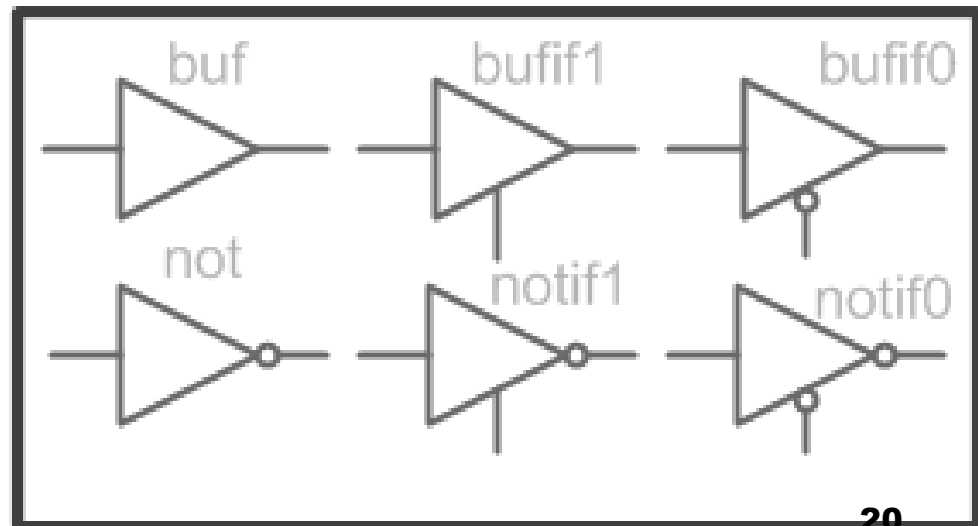
not	in	out
	0	1
	1	0
	x	x
	z	x

■ Verilog: buf/not klasa logičkih kapija – nastavak

- Postoje kapije sa dodatnim kontrolnim signalom (*three-state*):
 - Ako je kontrolni signal aktivan, rade svoju funkciju
 - Ako je kontrolni signal neaktivan, na izlazu je **z**
- Označavaju se sa **bufif** i **notif** (sa dodatkom **0** ili **1** koji definiše aktivnu vrijednost kontrolnog signala)
- Redoslijed portova je: izlaz, ulaz, kontrola

```
bufif1 b1 (out, in, ctrl) ;  
bufif0 b0 (out, in, ctrl) ;
```

```
notif1 n1 (out, in, ctrl) ;  
notif0 n0 (out, in, ctrl) ;
```



■ Verilog: buf/not klasa logičkih kapija – nastavak

		control			
		bufif1	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

		control			
		bufif0	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

		control			
		notif1	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

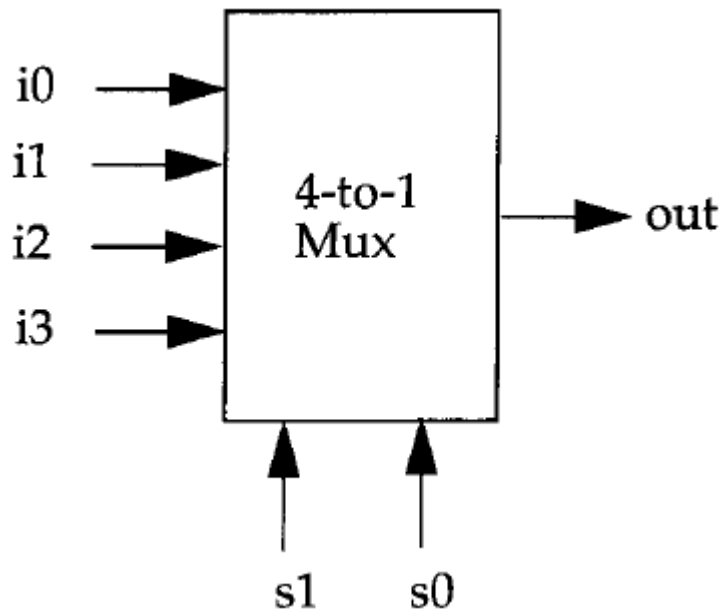
		control			
		notif0	0	1	x z
in	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	x	x

■ L=> 0 ili Z

H=> 1 ili Z

■ Primjer gate-level modelovanja: multiplekser 4/1

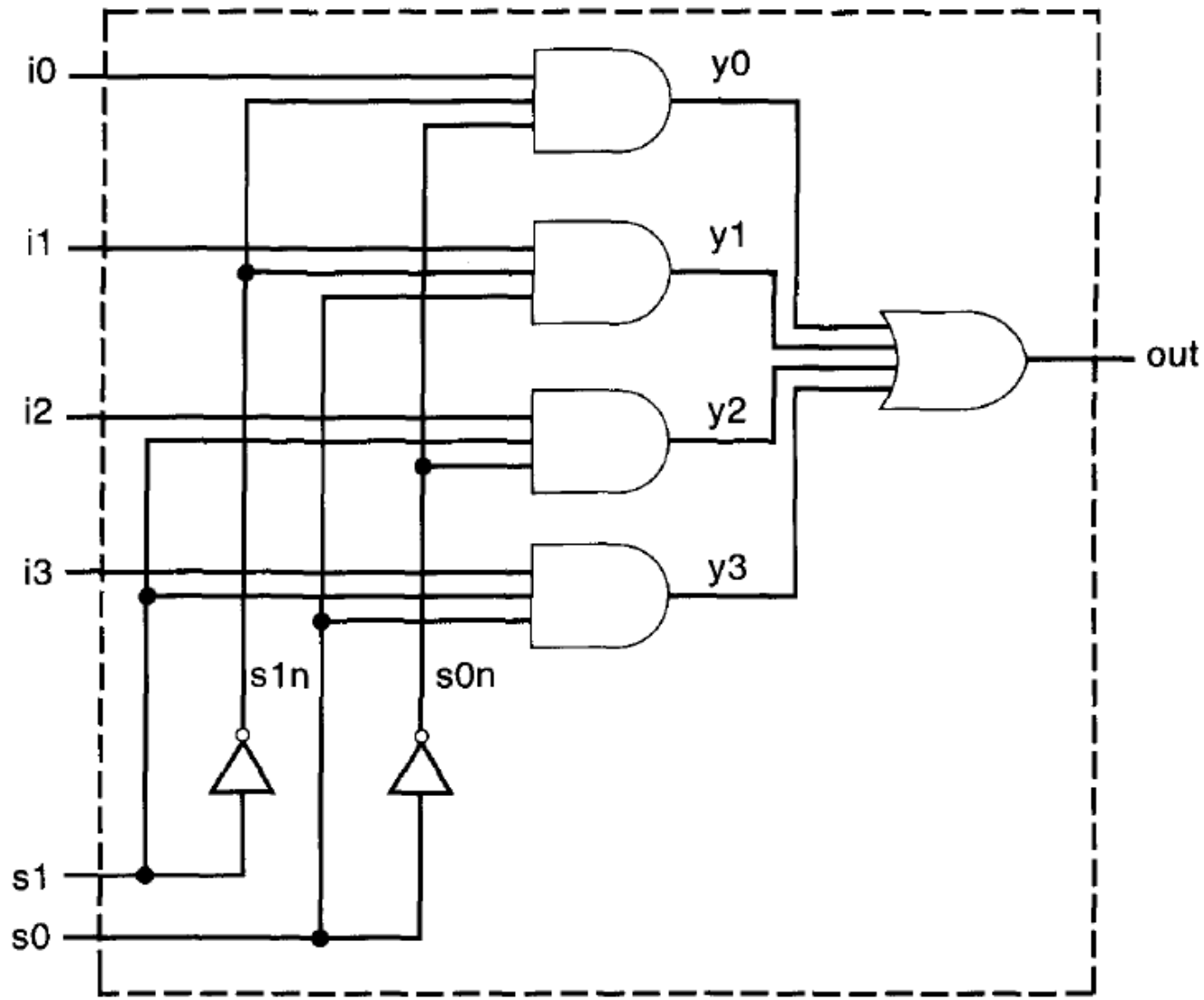
■ Način rada MUX 4/1:



s_1	s_0	out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

■ Primjer gate-level modelovanja: multiplekser 4/1

■ Logički dijagram MUX 4/1:



■ Primjer gate-level modelovanja: multiplexer 4/1

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
```

```
    output out;
```

```
    input i0, i1, i2, i3, s1, s0;
```

```
    // unutrašnje linije
```

```
    wire s1n, s0n;
```

```
    wire y0, y1, y2, y3;
```

```
    // kreiranje s1n i s0n signala
```

```
    not (s1n, s1);
```

```
    not (s0n, s0);
```

```
    // 3-ulazna and kola
```

```
    and (y0, i0, s1n, s0n);
```

```
    and (y1, i1, s1n, s0);
```

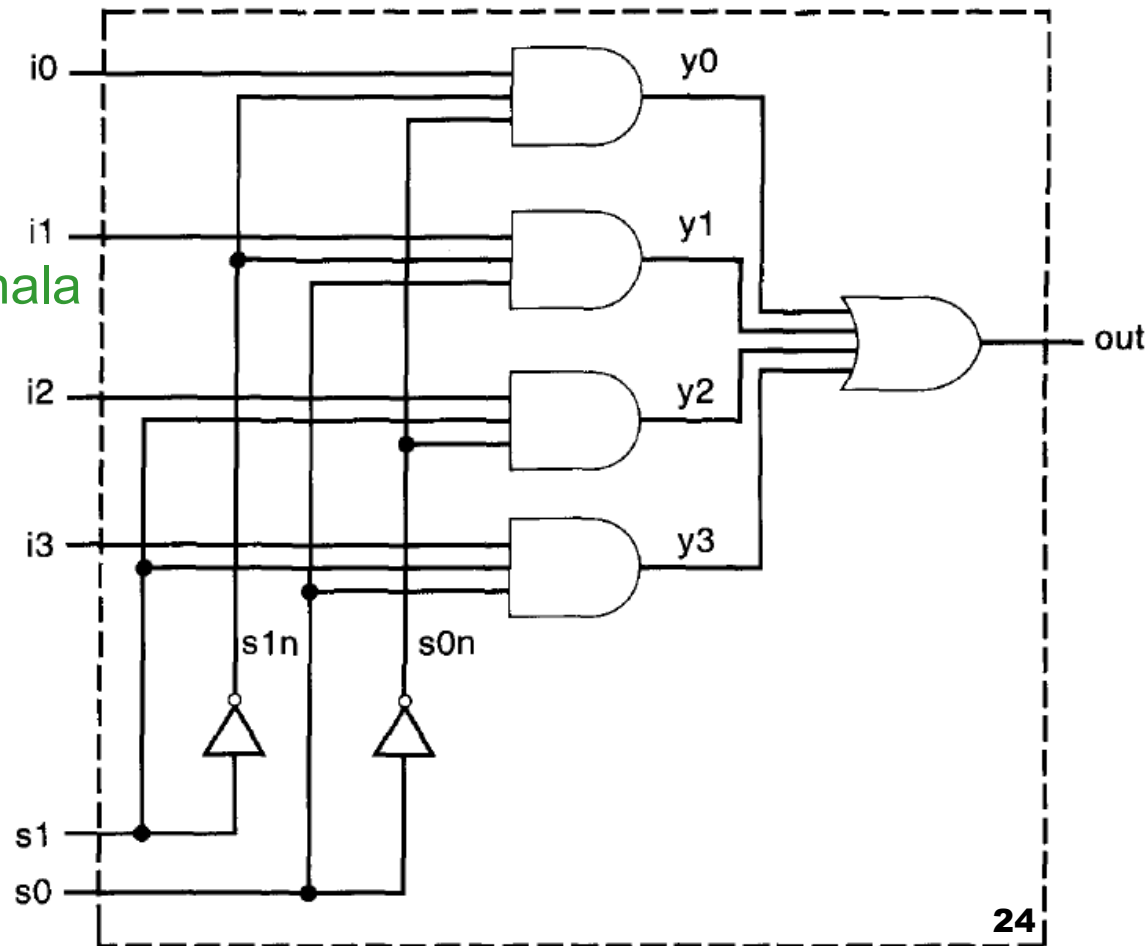
```
    and (y2, i2, s1, s0n);
```

```
    and (y3, i3, s1, s0);
```

```
    // 4-ulazno or kolo
```

```
    or (out, y0, y1, y2, y3);
```

```
endmodule
```



■ Primjer gate-level modelovanja: multiplekser 4/1

module stimulus;

// deklaracija promjenljivih za ulaze

reg IN0, IN1, IN2, IN3;

reg S1, S0;

// deklaracija izlazne linije

wire OUTPUT;

// instanciranje multipleksera

mux4_to_1 m1(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);

initial

begin

IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0; // postavljanje ulaznih linija

\$display("IN0=%b, IN1=%b, IN2=%b, IN3=%b\n", IN0, IN1, IN2, IN3);

\$monitor("S1 %b, S0 %b, IZLAZ %b \n", S1, S0, OUTPUT);

S1 = 0; S0 = 0; // izabrati IN0

#1 S1 = 0; S0 = 1; // izabrati IN1

#1 S1 = 1; S0 = 0; // izabrati IN2

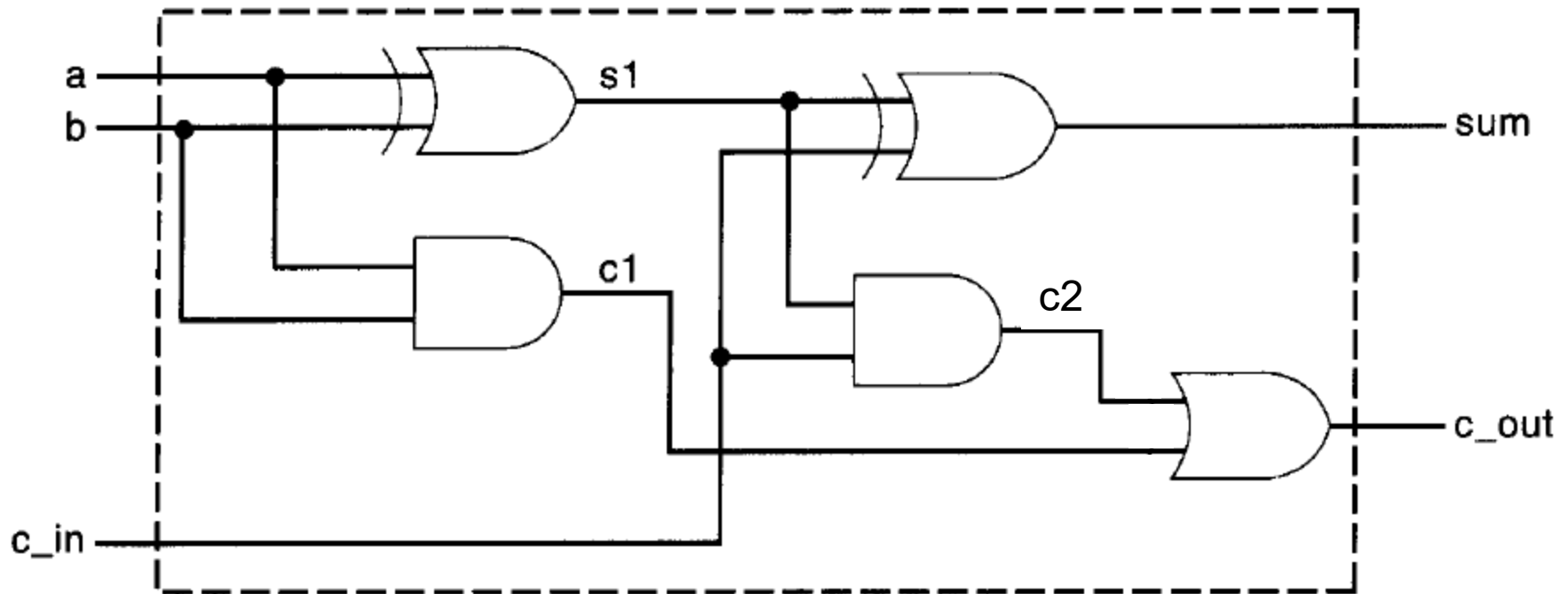
#1 S1 = 1; S0 = 1; // izabrati IN3

end

endmodule

■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

- Osnovni gradivni blok je jednobitni potpuni sabirač:



■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

■ Jednobitni potpuni sabirač:

// definicija jednobitnog potpunog sabirača

```
module fulladd(sum, c_out, a, b, c_in);
```

```
    output sum, c_out;
```

```
    input a, b, c_in;
```

// unutrašnje linije

```
    wire s1, c1, c2;
```

// instanciranje logičkih primitiva

```
    xor(s1, a, b);
```

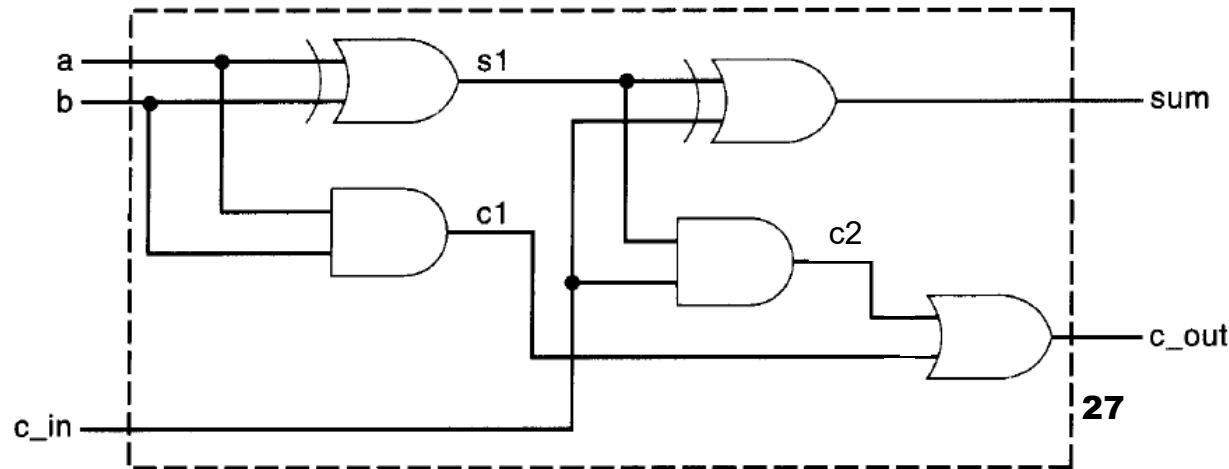
```
    and(c1, a, b);
```

```
    xor(sum, s1, c_in);
```

```
    and(c2, s1, c_in);
```

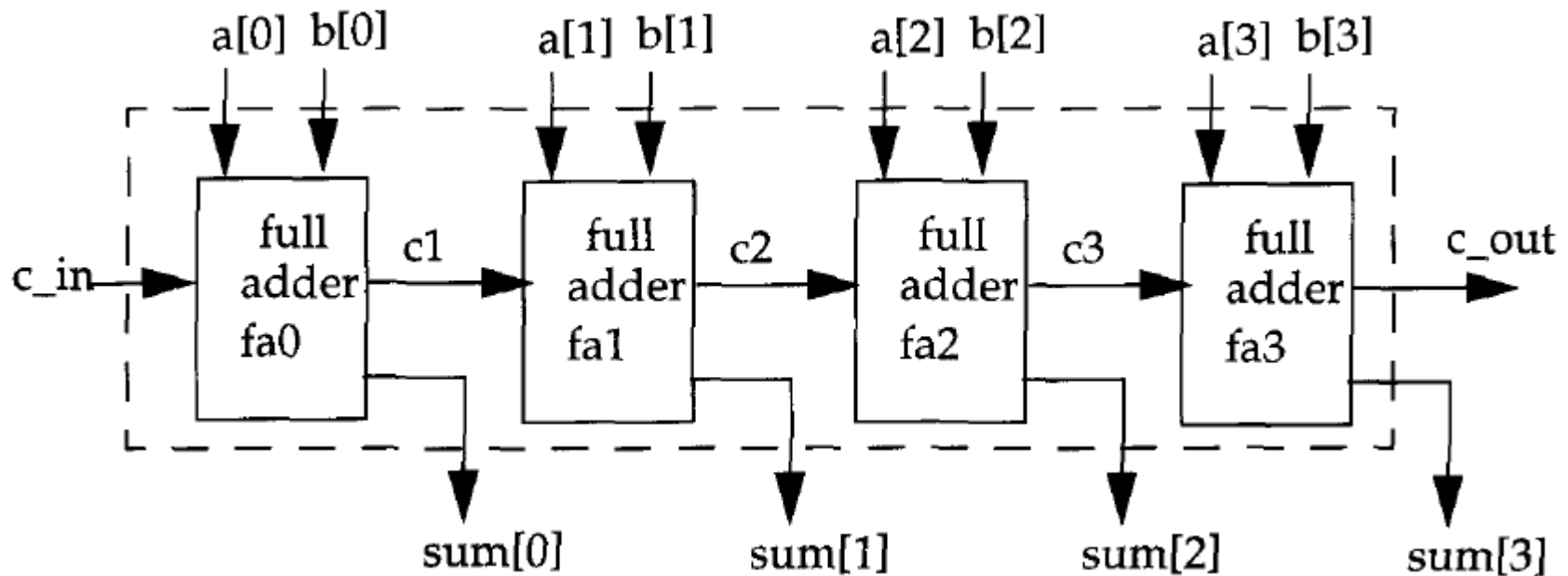
```
    or(c_out, c2, c1);
```

```
endmodule
```



■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

- Četvorobitni *ripple carry* potpuni sabirač:



- *fa0*, *fa1*, *fa2* i *fa3* su instance jednobitnog potpunog sabirača
- Izlazi su skalar ***c_out*** i vektor ***sum***

■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

■ Definicija četvorobitnog potpunog sabirača

```
module fulladd4(sum, c_out, a, b, c_in);
```

```
    output [3:0] sum;
```

```
    output c_out;
```

```
    input[3:0] a, b;
```

```
    input c_in;
```

```
// unutrašnje linije
```

```
wire c1, c2, c3;
```

```
// instanciranje 4 jednobitna sabirača
```

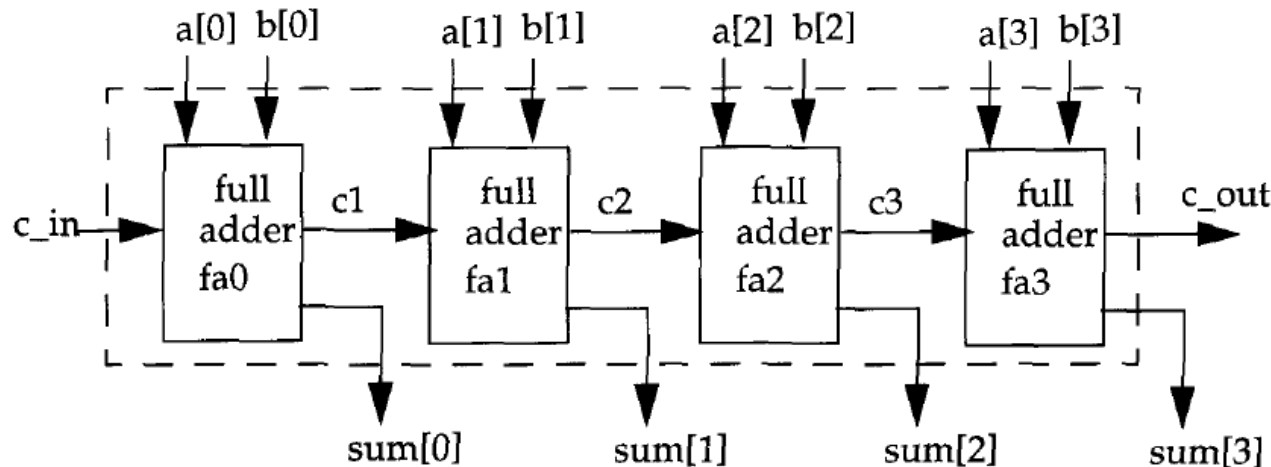
```
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
```

```
fulladd fa1(sum[1], c2, a[1], b[1], c1);
```

```
fulladd fa2(sum[2], c3, a[2], b[2], c2);
```

```
fulladd fa3(sum[3], c_out, a[3], b[3], c3);
```

```
endmodule
```



■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

- Primjetiti da su korišćeni isti nazivi za portove u jednobitnom i u četvorobitnom sabiraču iako predstavljaju različite elemente
- Element **sum** u jednobitnom sabiraču je skalarna vrijednost, a u četvorobitnom sabiraču je četvorobitni vektor
- U Verilogu su nazivi **lokalni** u okviru modula
- Nazivi nisu vidljivi van modula ukoliko se ne koristi hijerarhijski naziv
- **Podsjećanje**: nazivi instanci su obavezni kada se instanciraju definisani moduli, a nisu obavezni kada se instanciraju Verilog primitive

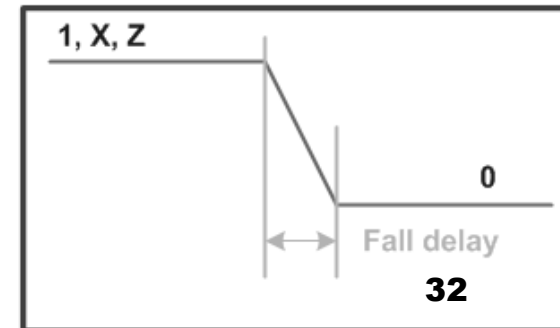
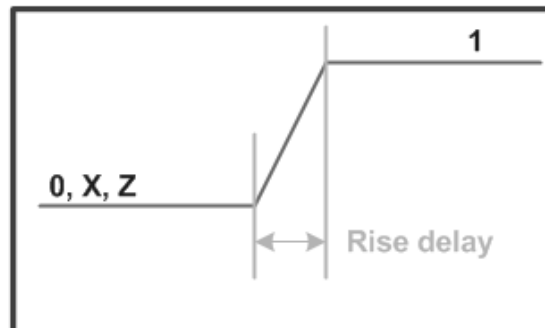
■ Primjer gate-level modelovanja: potpuni sabirač (4 b)

```
module stimulus;
    reg [3:0] A, B;
    reg C_IN;
    wire [3:0] SUM;
    wire C_OUT;
    fulladd4 FAI_4(SUM, C_OUT, A, B, C_IN);
    initial
    begin
        $monitor($time," A=%b, B=%b, C_IN= %b, --- C_OUT=%b,
            SUM=%b\n", A, B, C_IN, C_OUT, SUM);
        A = 4'd0; B = 4'd0; C_IN = 1'b0;
        #5 A = 4'd3; B = 4'd4;
        #5 A = 4'd2; B = 4'd5;
        #5 A = 4'd9; B = 4'd1;
        #5 A = 4'd10; B = 4'd0;
        #5 A = 4'd10; B = 4'd1;
    end
endmodule
```

Time	A	B	C_IN	C_OUT	SUM
0	0000	0000	0	0	0000
5	0011	0100	0	0	0111
10	0010	0101	0	0	0111
15	1001	1001	0	1	0010
20	1010	1111	0	1	1001
25	1010	0101	1	1	0000

■ Verilog: kašnjenje kod logičkih kapija

- U dosadašnjim primjerima nije tretirano kašnjenje koje unosi logičko kolo
- U stvarnosti kašnjenje postoji
- Verilog omogućava da se specificiraju kašnjenja u logičkim šemama
- Postoje tri vrste kašnjenja od ulaza do izlaza Verilog logičke primitive:
 - *Rise delay* – vezano za tranziciju izlaza sa neke vrijednosti na logičku jedinicu
 - *Fall delay* – vezano za tranziciju izlaza sa neke vrijednosti na logičku nulu
 - *Turn-off delay* – vezano za tranziciju izlaza sa neke vrijednosti na visoku impedansu (z)




■ Verilog: kašnjenje kod logičkih kapija – nastavak

- U slučaju kada se vrijednost mijenja na **x** uzima se najmanja od ovih vrijednosti
- Moguće je specificirati jednu, dvije ili sve tri vrijednosti kašnjenja:
 - Jedna – koristi se za sve tri tranzicije
 - Dvije – odnose se na *rise delay* i *fall delay*; *turn-off delay* je manji od ova dva
 - Tri – odnose se na *rise delay*, *fall delay* i *turn-off delay*
- Ako se ne specificira kašnjenje, uzima se da je nula (nema kašnjenja)

and #(5) a1(out, i1, i2); // kašnjenje za sve tri tranzicije

and #(4,6) a2(out, i1, i2); // Rise = 4, Fall = 6

bufif0 # (3,4,5) b1(out, in, control); // Rise = 3, Fall = 4, Turn-off = 5

- 
- Verilog: kašnjenje kod logičkih kapija – nastavak
 - Moguće je dodatno kontrolisati svaki od navedenih tipova kašnjenja:
 - Minimalno – najmanja očekivana vrijednost kašnjenja logičkog kola
 - Tipično – tipična vrijednost koja se očekuje od logičkog kola
 - Maksimalno – najveća očekivana vrijednost kašnjenja logičkog kola
 - Tip kašnjenja se bira na početku simulacije, a način izbora zavisi od simulatora
 - Ukoliko se ne specificira, uzima se tipično kašnjenje
 - Na ovaj način se omogućava dizajneru da eksperimentiše sa različitim vrijednostima za kašnjenje **bez izmjena u kodu**

■ Verilog: kašnjenje kod logičkih kapija – nastavak

■ Primjer:

```
and #(2:3:4) a1(out, i1,i2);
```

```
// Min: 2, Typ: 3, Max:4
```

```
and #(2:3:4, 3:4:5, 4:5:6) a1(out, i1,i2);
```

```
// Min: rise= 2  fall= 3  turn-off= 4
```

```
// Typ: rise= 3  fall= 4  turn-off= 5
```

```
// Max:rise= 4  fall= 5  turn-off= 6
```

■ Verilog: kašnjenje kod logičkih kapija – primjer

```
module buf_gate ();  
    reg in;  
    wire out;  
    // jedno kašnjenje  
    buf #(5) (out,in);
```

```
    initial begin
```

```
        $monitor ("Time = %g in = %b out=%b", $time, in, out);
```

```
        in = 0;
```

```
        #10 in = 1;
```

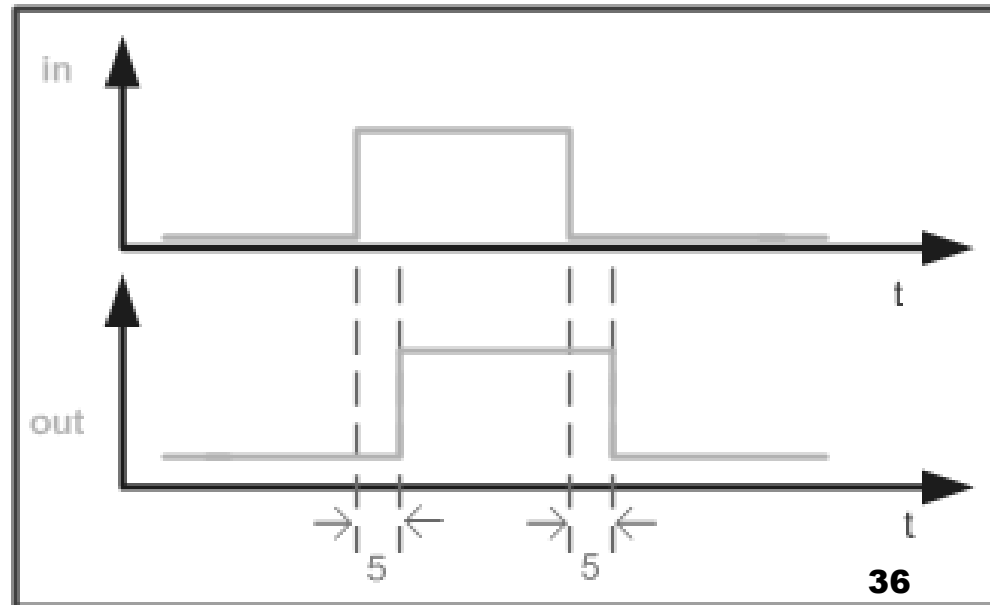
```
        #10 in = 0;
```

```
        #10 $finish;
```

```
    end
```

```
endmodule
```

Time = 0 in = 0 out=x
Time = 5 in = 0 out=0
Time = 10 in = 1 out=0
Time = 15 in = 1 out=1
Time = 20 in = 0 out=1
Time = 25 in = 0 out=0



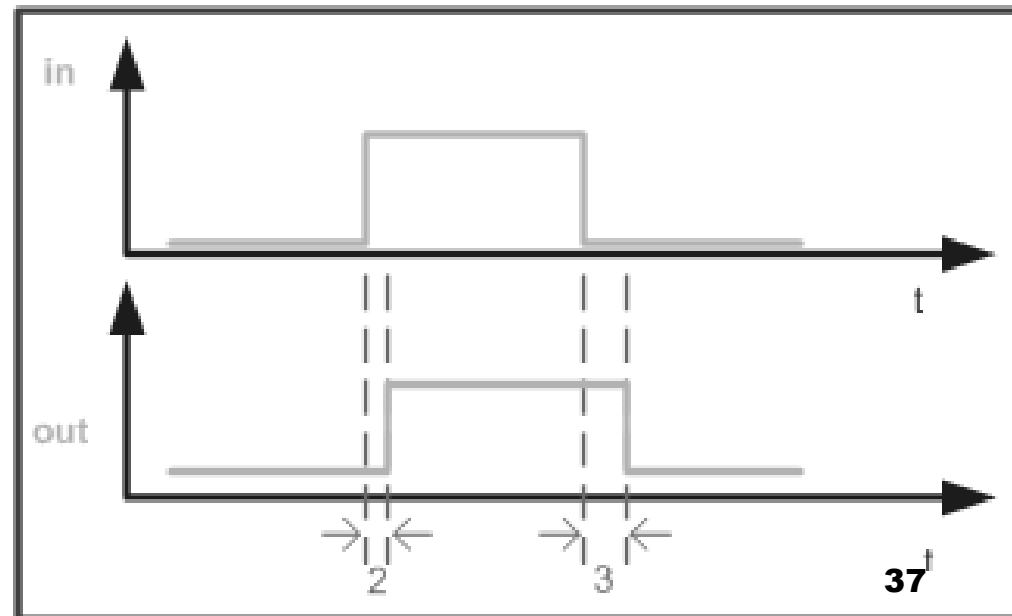
■ Verilog: kašnjenje kod logičkih kapija – primjer 2

```
module buf_gate1 ();  
    reg in;  
    wire out;  
    // dva kašnjenja  
    buf #(2,3) (out,in);
```

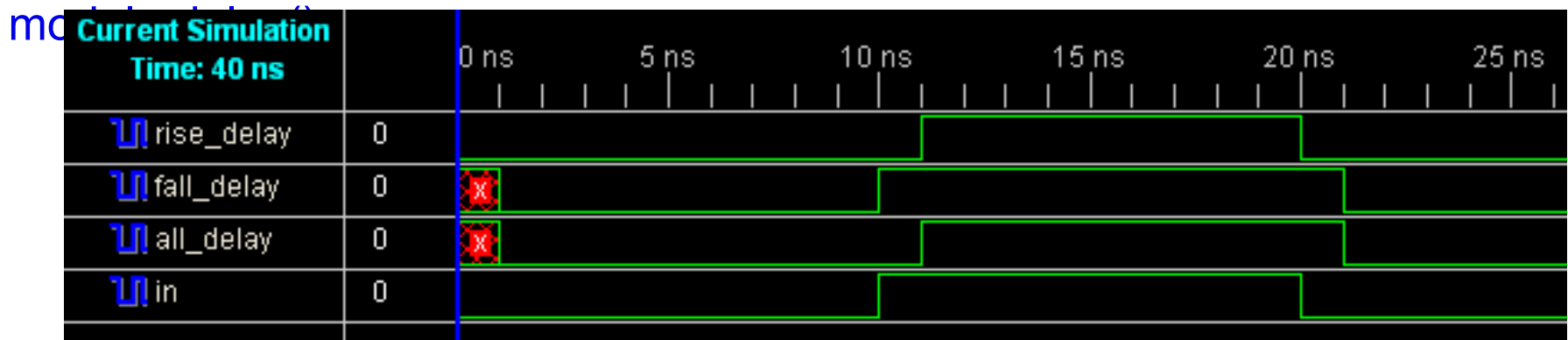
Time = 0 in = 0 out=x
Time = 3 in = 0 out=0
Time = 10 in = 1 out=0
Time = 12 in = 1 out=1
Time = 20 in = 0 out=1
Time = 23 in = 0 out=0

```
    initial begin  
        $monitor ("Time = %g in = %b out=%b", $time, in, out);  
        in = 0;  
        #10 in = 1;  
        #10 in = 0;  
        #10 $finish;  
    end
```

```
endmodule
```



■ Verilog: kašnjenje kod logičkih kapija – primjer 3



```
$time, in, rise_delay, fall_delay, all_delay);
```

```
in = 0;
```

```
#10 in = 1;
```

```
#10 in = 0;
```

```
#20 $finish;
```

```
end
```

```
buf #(1,0) U_rise (rise_delay,in);
```

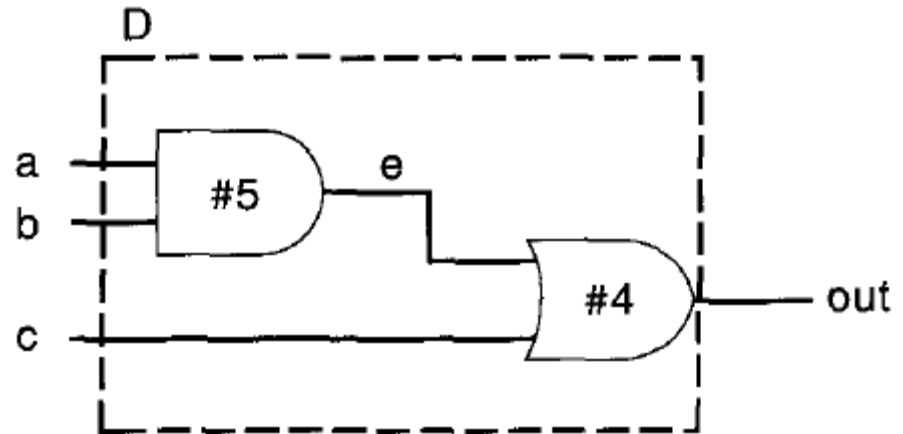
```
buf #(0,1) U_fall (fall_delay,in);
```

```
buf #1 U_all (all_delay,in);
```

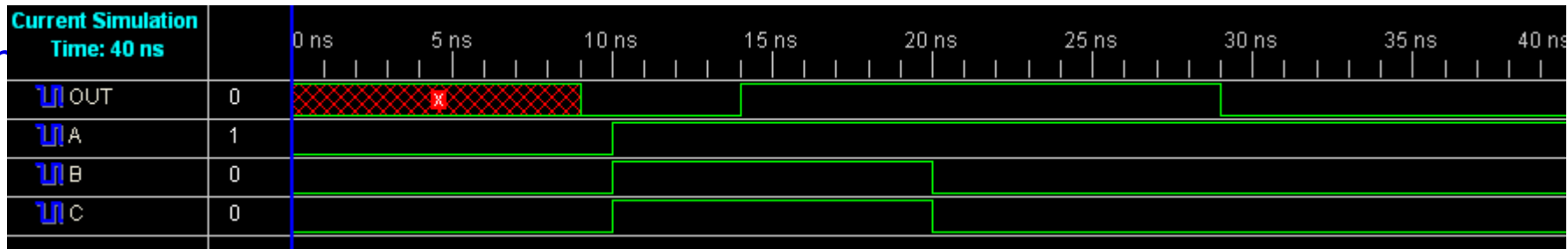
```
endmodule
```

■ Verilog: kašnjenje kod logičkih kapija – primjer 4

```
module D(out, a, b, c);  
    output out;  
    input a,b,c;  
  
    wire e;  
  
    and #(5) a1(e, a, b);  
    or #(4) o1(out, e,c);  
  
endmodule
```



■ Verilog: kašnjenje kod logičkih kapija – primjer 4



D d1(OUT, A, B, C);

initial

begin

A= 1'b0; B= 1'b0; C= 1'b0;

#10 A= 1'b1; B= 1'b1; C= 1'b1;

#10 A= 1'b1; B= 1'b0; C= 1'b0;

#20 \$finish;

end

endmodule

