



Projektovanje digitalnih sistema

Dataflow modelovanje



■ Verilog: *dataflow*

- U kompleksnijem dizajnu broj logičkih kapija može biti jako veliki i tada nije pogodno koristiti modelovanje na nivou kapija (*gate-level*)
- Implementira se **funkcija** na nivou apstrakcije višem od nivoa logičkih kola
- Uređaj se dizajnira na osnovu opisa **toka** podataka između registara ili na osnovu opisa načina kako se podaci **obrađuju**, umjesto instanciranja logičkih kola
- Automatizovani alati kreiraju kolo na nivou logičkih kapija, na osnovu opisa dizajna: **logička sinteza**
- Često se koristi kombinacija: nivo logičkih kapija, *dataflow* i *behavioral* dizajn
- Termin **RTL** (*Register Transfer Level*) se koristi za kombinaciju *dataflow* i *behavioral* dizajna



■ Verilog: ***continuous assignment***

- Najosnovniji izraz u *dataflow* modelovanju: koristi se za postavljanje određene vrijednosti na *net*
- Zamjenjuje logičke kapije u opisu kola – opisuje ga na višem nivou apstrakcije
- Počinje sa ključnom riječi **assign**:

`assign <jačina_signala> <kašnjenje> <lista_pridruživanja>;`

- `<jačina_signala>` je opcionalna; *default* je `strong1/ strong0`
- `<kašnjenje>` je takođe opcionalno i koristi se na isti način kao kod log. kapija
- Primjer (zanemariti operacije – fokus je na specifikaciji *assign*):

`assign izlaz = ul1 & ul2; // izlaz je tipa net`

`assign addr[15:0] = addr1[15:0] ^ addr2[15:0]; // vektori tipa net`



■ Verilog: *continuous assignment* – karakteristike

- Sa lijeve strane u listi pridruživanja mora biti *net* (skalar ili vektor); ne može biti registarska promjenljiva
- Izrazi u listi se proračunavaju čim dođe do promjene nekog od operandi sa desne strane znaka jednakosti i dobijena vrijednost se dodjeljuje promjenljivoj sa lijeve strane znaka jednakosti
- Operandi sa desne strane znaka jednakosti mogu biti
 - Registarske promjenljive (skalari ili vektori)
 - Net promjenljive (skalari ili vektori)
 - Pozivi funkcija
- Kašnjenje se specificira u vremenskim jedinicama i izražava vrijeme u kojem se izračunata vrijednost dodjeljuje promjenljivoj
- Kašnjenje se koristi da modeluje propagaciju u realnim kolima



■ Verilog: *continuous assignment* – implicitni zapis

- Može se specificirati za vrijeme deklaracije

// Standardni zapis

```
wire izlaz;
```

```
assign izlaz = ulaz1 & ulaz2;
```

// Implicitni zapis

```
wire izlaz = ulaz1 & ulaz2;
```



■ Verilog: *continuous assignment* – kašnjenje

- Može se specificirati na tri načina: regularno uvođenje kašnjenja, implicitno uvođenje kašnjenja i prilikom deklaracije
- Regularno uvođenje kašnjenja

assign #10 izlaz = ulaz1 & ulaz2;

- Kašnjenje se navodi nakon *assign*
- Kad se promijeni *ulaz1* ili *ulaz2* proći će 10 vremenskih jedinica prije nego se izvrši računanje i rezultat operacije pridruži *izlazu*
- Ako *ulaz1* ili *ulaz2* ponovo promijene vrijednosti prije isteka 10 vremenskih jedinica, uzeće se vrijednosti u trenutku računanja – *inertial delay*
- => ulazni impuls koji je kraći od vremena kašnjenja neće se propagirati na izlaz



■ Verilog: *continuous assignment* – kašnjenje

- Primjer za assign #10 izlaz = ulaz1 & ulaz2;

```
`timescale 1ns / 1ps
```

```
module kasnjenje_assign;
```

```
reg ulaz1, ulaz2;
```

```
wire izlaz;
```

```
assign #10 izlaz = ulaz1 & ulaz2;
```

```
initial
```

```
begin
```

```
    $monitor("ulaz1=%b ulaz2=%b izlaz=%b", ulaz1, ulaz2, izlaz);
```

```
    ulaz1=0; ulaz2=0;
```

```
    #20 ulaz1=1; ulaz2=1;
```

```
    #40 ulaz1=0;
```

```
    #20 ulaz1=1;
```

```
    #5 ulaz1=0;
```

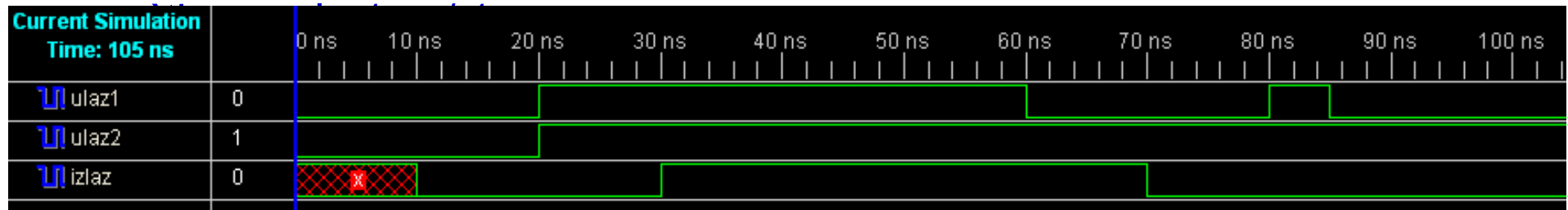
```
    #20 $finish;
```

```
end
```

```
endmodule
```

■ Verilog: *continuous assignment* – kašnjenje

- Primjer za `assign #10 izlaz = ulaz1 & ulaz2;`



```
assign #10 izlaz = ulaz1 & ulaz2;
initial
begin
    $monitor("ulaz1=%b ulaz2=%b izlaz=%b", ulaz1, ulaz2, izlaz);
    ulaz1=0; ulaz2=0;
    #20 ulaz1=1; ulaz2=1;
    #40 ulaz1=0;
    #20 ulaz1=1;
    #5 ulaz1=0;
    #20 $finish;
end
endmodule
```



■ Verilog: *continuous assignment* – kašnjenje


- Implicitno uvođenje kašnjenja (prilikom deklaracije):

```
wire #10 izlaz = ulaz1 & ulaz2;
```

- Isto je što i:

```
wire izlaz;
```

```
assign #10 izlaz = ulaz1 & ulaz2;
```

- 
- Verilog: **kašnjenje** prilikom deklaracije *net*-a
 - Kašnjenje se može specificirati na promjenljivoj tipa *net* prilikom deklaracije, bez stavljanja *continuous assignment*-a
 - U tom slučaju svaka promjena na promjenljivoj će biti adekvatno odložena

// Net kašnjenje

wire # 10 izlaz;

assign izlaz = ulaz1 & ulaz2;

// Gornji izraz ima isti efekat kao i sledeće:

wire izlaz;

assign #10 izlaz = ulaz1 & ulaz2;



■ Verilog: izrazi, operatori i operandi

- *Dataflow* modelovanje opisuje dizajn preko izraza umjesto preko logičkih kapija
- Izrazi, operatori i operandi čine osnovu *dataflow* modelovanja
- **Izrazi** su konstrukcije koje kombinuju operande i operatore da proizvedu rezultat

// Primjeri izraza – kombinuju operande i operatore

$a \wedge b$

$\text{addr1}[20:17] + \text{addr2}[20:17]$

$\text{ulaz1} \mid \text{ulaz2}$

- **Operandi** mogu biti konstante, cijeli brojevi, realni brojevi, net, reg, time, bitovi (jedan ili više bitova vektor net-a ili vektor reg-a), memorije ili funkcijski pozivi (o funkcijama kasnije)



■ Verilog: izrazi, operatori i **operandi**

■ Primjeri:

```
integer brojac, ukupni_brojac;  
ukupni_brojac = brojac + 1; // brojac je cjelobrojni operand
```

```
real a, b , c;  
c = a - b; // a i b su realni operandi
```

```
reg [15:0] reg1, reg2;  
reg [3:0] reg_3;  
reg_3 = reg1[3:0] ^ reg2[3:0]; // reg1[3:0] i reg2[3:0] su reg operandi
```

```
reg povratna_vrijednost;  
povratna_vrijednost = izracunati_parnost(A, B);  
// izracunati_parnost je operand funkcijskog tipa
```



■ Verilog: izrazi, **operatori** i operandi

- Operatori djeluju na operande u cilju dobijanja potrebnog rezultata

- Primjeri:

d1 && d2 // && je operator nad operandima d1 i d2

!a[0] // ! je operator nad operandom a[0]

b >> 1 // >> je operator nad operandima b i 1

- Tipovi operatora:

- Aritmetički
- Logički
- Relacioni
- Jednakosti
- Nad bitovima
- Redukcije
- Pomjeranja
- Konkatencije – replikacije
- Uslovni operator

■ Verilog: aritmetički operatori

- Aritmetički operatori su binarni
- + i – mogu biti unarni (znak broja)

```
module aritmeticki_operatori();  
initial begin  
    $display (" 5 + 10 = %d", 5 + 10);  
    $display (" 5 - 10 = %d", 5 - 10);  
    $display (" 10 - 5 = %d", 10 - 5);  
    $display (" 10 * 5 = %d", 10 * 5);  
    $display (" 10 / 5 = %d", 10 / 5);  
    $display (" 10 / -5 = %d", 10 / -5);  
    $display (" 10 %s 3 = %d","%", 10 % 3);  
    $display (" +5      = %d", +5);  
    $display (" -5      = %d", -5);  
    #10 $finish;  
end  
endmodule
```

Operator	Opis
*	množenje
/	dijeljenje
+	sabiranje
–	oduzimanje
%	moduo

Rezultat:

```
5 + 10 = 15  
5 - 10 = -5  
10 - 5 = 5  
10 * 5 = 50  
10 / 5 = 2  
10 / -5 = -2  
10 % 3 = 1  
+5      = 5  
-5      = -5
```

■ Verilog: aritmetički operatori – nastavak

■ Primjeri:

`A = 4'b0011; B = 4'b0100; // A i B su vektori reg`

`D = 6; E = 4; // D i E su cijeli brojevi`

`A * B // Proizvod A i B. Rezultat je 4'b1100`

`D / E // Dijeljenje D sa E. Rezultat je 1. Odbacuje se decimalni dio.`

`A + B // Sabiranje A i B. Rezultat je 4'b0111`

`B - A // Oduzimanje A od B. Rezultat je 4'b0001`

`13 % 3 // Dijeljenje po modulu. Rezultat je 1`

`16 % 4 // Dijeljenje po modulu. Rezultat je 0`

`-7 % 2 // Rezultat je -1 => uzima znak prvog operanda (samo kod %)`

`7 % -2 // Rezultat je +1 => uzima znak prvog operanda (samo kod %)`

- Ako je vrijednost nekog bita kod operandada jednak **x**, čitav izraz ima vrijednost **x**:

`ulaz1 = 4'b101x;`

`ulaz2 = 4'b1010;`

`zbir = ulaz1 + ulaz2; // zbir će imati vrijednost 4'bx`



■ Verilog: aritmetički operatori – nastavak

- Unarni aritmetički operatori imaju viši prioritet od binarnih
- Negativni brojevi se reprezentuju u dvojnem komplementu i zato ih nije preporučljivo koristiti u zapisu sa bazom brojnog sistema
- Koriste se kod cjelobrojnih i realnih promjenljivih

`-10 / 5 // Rezultat je -2`

`// ne koristiti brojeve tipa <sss> ‘<osnova> <nnn>`

`-’d10 / 5 // je ekvivalentno sa (dvojni komplement od 10)/5 =`

`// = $(2^{32} - 10) / 5$`

`// gdje je 32 širina riječi. Ovo vodi do neočekivanog rezultata.`



■ Verilog: logički operatori

Operator	Opis
!	Logička negacija
&&	Logičko I (AND)
	Logičko ILI (OR)

- Daju uvijek jednobitni rezultat:
1 (tačno), 0 (netačno), x (neodređeno)
- Ako operand nije jednak nuli, ekvivalentan je logičkoj 1 (tačno)
- Ako je operand jednak nuli, ekvivalentan je logičkoj 0 (netačno)
- Ako je bilo koji bit operanda jednak **x** ili **z**, operand je ekvivalentan **x**
- Operandi logičkih operatora mogu biti promjenljive ili izrazi

■ Verilog: logički operatori – primjer

```
module logicki_operatori;
initial begin
    // Logičko AND
    $display ("1'b1 && 1'b1 = %b", (1'b1 && 1'b1));
    $display ("1'b1 && 1'b0 = %b", (1'b1 && 1'b0));
    $display ("1'b1 && 1'bx = %b", (1'b1 && 1'bx));
    // Logičko OR
    $display ("1'b1 || 1'b0 = %b", (1'b1 || 1'b0));
    $display ("1'b0 || 1'b0 = %b", (1'b0 || 1'b0));
    $display ("1'b0 || 1'bx = %b", (1'b0 || 1'bx));
    // Logičko NE
    $display ("! 1'b1      = %b", (! 1'b1));
    $display ("! 1'b0      = %b", (! 1'b0));
    #10 $finish;
end
endmodule
```

Rezultat:

```
1'b1 && 1'b1 = 1
1'b1 && 1'b0 = 0
1'b1 && 1'bx = x
1'b1 || 1'b0 = 1
1'b0 || 1'b0 = 0
1'b0 || 1'bx = x
! 1'b1      = 0
! 1'b0      = 1
```



■ Verilog: logički operatori – primjer

A = 3; B = 0;

A && B // rezultat je 0 (ekvivalentno sa log.1 && log.0)

A || B // rezultat je 1 (ekvivalentno sa log.1 || log.0)

!A // rezultat je 0

!B // rezultat je 1

A = 2'b0x; B = 2'b10;

A && B // rezultat je x (ekvivalentno sa x && log.1)

// izrazi

(A == 2) && (B == 3) // rezultat je 1 ako su oba izraza istinita
// rezultat je 0 ako je bilo koji izraz netačan



■ Verilog: relacioni operatori

Operator	Opis
$a < b$	a manje od b
$a > b$	a veće od b
$a \leq b$	a manje ili jednako b
$a \geq b$	a veće ili jednako b

- Izraz daje vrijednost logičke 1 ako je tačan, a logičke 0 ako je netačan
- Ako postoji bar jedan **x** ili **z** bit u operandima, izraz daje vrijednost **x**

■ Verilog: relaciji operatori – primjer

```
module relaciji_operatori;
integer A = 4, B = 3;
reg [3:0] X=4'b1010, Y = 4'b1101, Z = 4'b0xxx;
initial begin
    $display (" 5    <=  10 = %b", 5 <= 10);
    $display (" 5    >=  10 = %b", 5 >= 10);
    $display (" 1'bx  <=  10 = %b", 1'bx <= 10);
    $display (" 1'bz  <=  10 = %b", 1'bz <= 10);
    $display (" A <= B (%0d <= %0d) = %b", A, B);
    $display (" A > B (%0d > %0d) = %b", A, B);
    $display (" Y >= X (%b >= %b) = %b", Y, X);
    $display (" Y < Z (%b < %b) = %b", Y, Z, Y);
    #10 $finish;
end
endmodule
```

Rezultat:

```
5    <=  10 = 1
5    >=  10 = 0
1'bx  <=  10 = x
1'bz  <=  10 = x
A <= B (4 <= 3) = 0
A > B (4 > 3) = 1
Y >= X (1101 > 1010) = 1
Y < Z (1101 < 0xxx) = 1
```

■ Verilog: operatori jednakosti

Operator	Opis
<code>a === b</code>	a jednako b, uključujući x i z (potpuna jednakost)
<code>a !== b</code>	a nije jednako b, uključujući x i z (nije potpuna jednakost)
<code>a == b</code>	a jednako b, rezultat može biti nepoznat (logička jednakost)
<code>a != b</code>	a nije jednako b, rezultat može biti nepoznat (logička nejednakost)

- Upoređuju dva operanda bit po bit
- Ako su operandi nejednake dužine, kraći se dopunjava nulama
- Vraćaju logičku 1 ako je izraz tačan, a logičku 0 ako je netačan
- Operatori logičke (ne)jednakosti (`==` i `!=`) daju rezultat **x**, ako u nekom operandu ima bitova **x** i/ili **z**
- Operatori `===` i `!==` (*case equality*) daju kao rezultat logičku 1 ili logičku 0, zavisno da li se na svim mjestima nalaze iste vrijednosti, uključujući **x** i **z**
- Oni nikada ne daju rezultat **x**

■ Verilog: operatori jednakosti – primjer

```
module operatori_jednakosti();
```

```
initial begin
```

```
// Case Equality
```

```
$display (" 4'bx001 === 4'bx001 = %b", (4'bx001 === 4'bx001));
```

```
$display (" 4'bx0x1 === 4'bx001 = %b", (4'bx0x1 === 4'bx001));
```

```
$display (" 4'bz0x1 === 4'bz0x1 = %b", (4'bz0x1 === 4'bz0x1));
```

```
$display (" 4'bz0x1 === 4'bz001 = %b", (4
```

```
// Case Inequality
```

```
$display (" 4'bx0x1 !== 4'bx001 = %b", (4'
```

```
$display (" 4'bz0x1 !== 4'bz001 = %b", (4'
```

```
// Logical Equality
```

```
$display (" 5      == 10      = %b", (5 == 10));
```

```
$display (" 5      == 5       = %b", (5 == 5));
```

```
$display (" 4'b0001 == 4'bx001 = %b", (4'b0001 == 4'bx001));
```

```
$display (" 4'bx001 == 4'bx001 = %b", (4'bx001 == 4'bx001));
```

```
// Logical Inequality
```

```
$display (" 5      != 5       = %b", (5 != 5));
```

```
$display (" 5      != 6       = %b", (5 != 6));
```

```
#10 $finish;
```

```
end
```

```
endmodule
```

Rezultat:

```
4'bx001 === 4'bx001 = 1
```

```
4'bx0x1 === 4'bx001 = 0
```

```
4'bz0x1 === 4'bz0x1 = 1
```

```
4'bz0x1 === 4'bz001 = 0
```

```
4'bx0x1 !== 4'bx001 = 1
```

```
4'bz0x1 !== 4'bz001 = 1
```

```
5      == 10      = 0
```

```
5      == 5       = 1
```

```
4'b0001 == 4'bx001 = x
```

```
4'bx001 == 4'bx001 = x
```

```
5      != 5       = 0
```

```
5      != 6       = 1
```

■ Verilog: operatori nad bitovima (*bitwise*)

Operator	Opis
~	negacija (NOT)
&	I (AND)
	ILI (OR)
^	ekskluzivno ILI
^~ ili ~^	ekskluzivno NILI

- Bit po bit: uzimaju jedan bit operanda i izvršavaju operaciju sa odgovarajućim bitom drugog operanda
- Ako operandi nisu iste dužine, kraći se dopunjava nulama
- **z** se tretira kao **x**
- Negacija je unarna operacija – nema drugog operanda

■ Verilog: operatori nad bitovima (*bitwise*) – primjer

```
module bitwise_operatori;  
initial begin
```

```
    // Bit Wise NOT
```

```
    $display (" ~4'b0001      = %b", (~4'b0001));
```

```
    $display (" ~4'bx001      = %b", (~4'bx001));
```

```
    $display (" ~4'bz001      = %b", (~4'bz001));
```

```
    // Bit Wise AND
```

```
    $display (" 4'b0001 & 4'b1001 = %b", (
```

```
    $display (" 4'b1001 & 4'bx001 = %b", (
```

```
    $display (" 4'b1001 & 4'bz001 = %b", (
```

```
    // Bit Wise OR
```

```
    $display (" 4'b0001 | 4'b1001 = %b", (
```

```
    $display (" 4'b0001 | 4'bx001 = %b", (
```

```
    $display (" 4'b0001 | 4'bz001 = %b", (
```

```
    #10 $finish;
```

```
end
```

```
endmodule
```

Rezultat:

```
~4'b0001      = 1110
```

```
~4'bx001      = x110
```

```
~4'bz001      = x110
```

```
4'b0001 & 4'b1001 = 0001
```

```
4'b1001 & 4'bx001 = x001
```

```
4'b1001 & 4'bz001 = x001
```

```
4'b0001 | 4'b1001 = 1001
```

```
4'b0001 | 4'bx001 = x001
```

```
4'b0001 | 4'bz001 = x001
```

■ Verilog: operatori nad bitovima (*bitwise*) – primjer

```
module bitwise_operatori;
```

```
initial begin
```

```
// Bit Wise XOR
```

```
$display (" 4'b0001 ^ 4'b1001 = %b", (4'b0001 ^ 4'b1001));
```

```
$display (" 4'b0001 ^ 4'bx001 = %b", (4'b0001 ^ 4'bx001));
```

```
$display (" 4'b0001 ^ 4'bz001 = %b", (4'b0001 ^ 4'bz001));
```

```
// Bit Wise XNOR
```

```
$display (" 4'b0001 ~^ 4'b1001 = %b", (4'b0001 ~^ 4'b1001));
```

```
$display (" 4'b0001 ~^ 4'bx001 = %b", (4'b0001 ~^ 4'bx001));
```

```
$display (" 4'b0001 ~^ 4'bz001 = %b", (4'b0001 ~^ 4'bz001));
```

```
#10 $finish;
```

```
end
```

```
endmodule
```

Rezultat:

```
4'b0001 ^ 4'b1001 = 1000
```

```
4'b0001 ^ 4'bx001 = x000
```

```
4'b0001 ^ 4'bz001 = x000
```

```
4'b0001 ~^ 4'b1001 = 0111
```

```
4'b0001 ~^ 4'bx001 = x111
```

```
4'b0001 ~^ 4'bz001 = x111
```



- Verilog: operatori nad bitovima (*bitwise*)

- Razlikovati od logičkih operatora:

// X = 4'b1010, Y = 4'b0000

X I Y // bitwise operacija: rezultat je 4'b1010

X II Y // logička operacija: ekvivalentno sa 1 II 0, rezultat je 1 (tačno)

■ Verilog: operatori redukcije

Operator	Opis
&	AND
~&	NAND
	OR
~	NOR
^	XOR
^~ ili ~^	XNOR

- Imaju samo **jedan operand** (vektor)
- Izvršavaju operaciju nad bitovima vektora i daju **jednobitni** rezultat
- Operacija se izvršava bit po bit, sa desna na lijevo
- Redukcioni NAND, NOR i XNOR se izračunavaju invertovanjem rezultata redukcionog AND, OR i XOR, respektivno

■ Verilog: operatori redukcije – primjer

```
module redukcioni_operatori;
```

```
initial begin
```

```
    // Bit Wise AND redukcija
```

```
    $display (" & 4'b1001 = %b", (& 4'b1001));
```

```
    $display (" & 4'bx111 = %b", (& 4'bx111));
```

```
    $display (" & 4'bx101 = %b", (& 4'bx101));
```

```
    $display (" & 4'bz111 = %b", (& 4'bz111));
```

```
    // Bit Wise NAND redukcija
```

```
    $display (" ~& 4'b1001 = %b", (~& 4'b1001));
```

```
    $display (" ~& 4'bx001 = %b", (~& 4'bx001));
```

```
    $display (" ~& 4'bx111 = %b", (~& 4'bx111));
```

```
    $display (" ~& 4'bz001 = %b", (~& 4'bz001));
```

```
    // Bit Wise OR redukcija
```

```
    $display (" | 4'b1001 = %b", (| 4'b1001));
```

```
    $display (" | 4'bx000 = %b", (| 4'bx000));
```

```
    $display (" | 4'bz000 = %b", (| 4'bz000));
```

```
    #10 $finish;
```

```
end
```

```
endmodule
```

Rezultat:

& 4'b1001 = 0

& 4'bx111 = x

& 4'bx101 = 0

& 4'bz111 = x

~& 4'b1001 = 1

~& 4'bx001 = 1

~& 4'bx111 = x

~& 4'bz001 = 1

| 4'b1001 = 1

| 4'bx000 = x

| 4'bz000 = x

■ Verilog: operatori redukcije – primjer

```
module redukcioni_operatori();
initial begin
    // Bit Wise OR redukcija
    $display (" ~| 4'b1001 = %b", (~| 4'b1001));
    $display (" ~| 4'bx001 = %b", (~| 4'bx001));
    $display (" ~| 4'bz001 = %b", (~| 4'bz001));
    // Bit Wise XOR redukcija
    $display (" ^ 4'b1001 = %b", (^ 4'b1001));
    $display (" ^ 4'bx001 = %b", (^ 4'bx001));
    $display (" ^ 4'bz001 = %b", (^ 4'bz001));
    // Bit Wise XNOR redukcija
    $display (" ~^ 4'b1001 = %b", (~^ 4'b1001));
    $display (" ~^ 4'bx001 = %b", (~^ 4'bx001));
    $display (" ~^ 4'bz001 = %b", (~^ 4'bz001));
    #10 $finish;
end
endmodule
```

Rezultat:

```
~| 4'b1001 = 0
~| 4'bx001 = 0
~| 4'bz001 = 0
^ 4'b1001 = 0
^ 4'bx001 = x
^ 4'bz001 = x
~^ 4'b1001 = 1
~^ 4'bx001 = x
~^ 4'bz001 = x
```



■ Verilog: operatori redukcije – primjer upotrebe

■ Napraviti osmo-ulazno AND kolo

```
module and8 (y, a);  
output y;  
input [7:0] a;
```

```
assign y = &a; // mnogo je lakše napisati y = &a nego  
// assign y = a[7] & a[6] & a[5] & a[4] & a[3] & a[2] & a[1] & a[0];  
endmodule
```



■ Verilog: operatori pomjeranja

Operator	Opis
<<	pomjeranje ulijevo
>>	pomjeranje udesno

- Pomjeraju bitove vektorskog operanda za specificirani broj bitova
- Operandi su vektor i broj bitova za koje treba pomjeriti sadržaj vektora
- Upraznjena mjesta se popunjavaju nulama
- Korisno za implementaciju pomjeračkih registara, množenje sa stepenom dvojke, množenje sabiranjem,...

■ Verilog: operatori pomjeranja – primjer

```
module shift_operatori;  
initial begin  
    // Pomjeranje u lijevu stranu  
    $display (" 4'b1001 << 1 = %b", (4'b1001 << 1));  
    $display (" 4'b10x1 << 2 = %b", (4'b10x1 << 2));  
    $display (" 4'b10z1 << 1 = %b", (4'b10z1 << 1));  
    // Pomjeranje u desnu stranu  
    $display (" 4'b1001 >> 1 = %b", (4'b1001 >> 1));  
    $display (" 4'b10x1 >> 1 = %b", (4'b10x1 >> 1));  
    $display (" 4'b10z1 >> 1 = %b", (4'b10z1 >> 1));  
    #10 $finish;  
end  
endmodule
```

Rezultat:

```
4'b1001 << 1 = 0010  
4'b10x1 << 2 = x100  
4'b10z1 << 1 = 0z10  
4'b1001 >> 1 = 0100  
4'b10x1 >> 1 = 010x  
4'b10z1 >> 1 = 010z
```

■ Verilog: operator konkatencije (nastavljanja)

Operator	Opis
{ }	konkatencija

- Obezbjeđuje mehanizam za “spajanje” više operanada
- Operandi **moraju** imati naznačenu veličinu
- Konkatencija se označava vitičastim zagradama pri čemu se operandi razdvajaju zarezima
- Operandi mogu biti skalarni *net* ili *reg*, vektorski *net* ili *reg*, selektovani bit, selektovana sekvenca bitova ili konstanta sa naznačenom dužinom

// A = 1'b1, B = 2'b00, C = 2'b10, D = 3'b110

Y = {B , C} // Rezultat Y je 4'b0010

Y = {A, B, C, D, 3'b001} // Rezultat Y je 11'b10010110001

Y = {A, B[0], C[1]} // Rezultat Y je 3'b101

■ Verilog: operator replikacije

Operator	Opis
{ { } }	replikacija

- Uzastopno nastavljajanje istog broja
- Konstanta replikacije specificira koliko puta se ponavlja broj unutar vitičastih zagrada

```
reg A;  
reg [1:0] B, C;  
A = 1'b1; B = 2'b00; C = 2'b10;  
Y = { 4{A} } // Rezultat Y je 4'b1111  
Y = { 4{A}, 2{B} } // Rezultat Y je 8'b11110000  
Y = { 4{A}, 2{B} , C } // Rezultat Y je 10'b1111000010
```

■ Verilog: operatori konkatencije i replikacije – primjer

```
module operatori_konk_repl;
```

```
initial begin
```

```
    // konkatencija
```

```
    $display ("{4'b1001,4'b10x1} = %b", {4'b1001,4'b10x1});
```

```
    // replikacija
```

```
    $display ("{4{4'b1001}} = %b", {4{4'b1001}});
```

```
    // replikacija i konkatencija
```

```
    $display ("{4{4'b1001,1'bz}} = %b", {4{4'b1001,1'bz}});
```

```
    #10 $finish;
```

```
end
```

```
endmodule
```

Rezultat:

```
{4'b1001,4'b10x1} = 100110x1
```

```
{4{4'b1001}} = 1001100110011001
```

```
{4{4'b1001,1'bz}} = 1001z1001z1001z1001z
```

■ Verilog: uslovni operator

Operator	Opis
? :	uslovni operator

- Uzima tri operanda: *izraz_uslova* ? *izraz_tačan* : *izraz_netačan* ;
- Prvo se evaluira *izraz_uslova*
 - Ako je rezultat tačan (log. 1) evaluira se dio *izraz_tačan*
 - Ako je rezultat netačan (log. 0) evaluira se dio *izraz_netačan*
 - Ako je rezultat neodređen (x) evaluiraju se i *izraz_tačan* i *izraz_netačan* i njihovi rezultati se porede bit po bit: na svakoj poziciji se dobija **x** ako su bitovi različiti, a vrijednost tih bitova ako su isti
- Ponašanje uslovnog operatora je slično radu multipleksera
- Može se uočiti sličnost sa *if – else* izrazom



■ Verilog: uslovni operator – nastavak

// modelovanje funkcionalnosti *tristate* bafera

```
assign addr_bus = drive_enable ? addr_out : 32'bz;
```

// modelovanje funkcionalnosti multipleksora 2/1

```
assign izlaz = select ? ulaz1 : ulaz0;
```

- Uslovni operatori se mogu ugnijezditi: i *izraz_tačan* i *izraz_netačan* mogu biti uslovni operatori

```
assign izlaz = (A == 3) ? ( select ? x : y ) : ( select ? m : n ) ;
```

■ Verilog: uslovni operator – primjer

```
module uslovni_operator;
wire out;
reg enable, data;
assign out = (enable) ? data : 1'bz; // Tri-state bafer
initial begin
    $display ("time\t enable data out");
    $monitor ("%g\t %b    %b    %b", $time, enable, data, out);
    enable = 0;
    data = 0;
    #1 data = 1;
    #1 data = 0;
    #1 enable = 1;
    #1 data = 1;
    #1 data = 0;
    #1 enable = 0;
    #10 $finish;
end
endmodule
```

Rezultat:

time	enable	data	out
0	0	0	z
1	0	1	z
2	0	0	z
3	1	0	0
4	1	1	1
5	1	0	0
6	0	0	z

■ Verilog: prioritet operatora

- Koristiti zagrade
- U suprotnom ugrađeni prioritet:

Operator	Simboli	Prioritet
Unarni, Množenje, Dijeljenje, Moduo	!, ~, *, /, %	Najviši ↑ ↓ Najniži
Sabiranje, Oduzimanje, Pomjeranje	+, - , <<, >>	
Relacioni, Jednakosti	<, >, <=, >=, ==, !=, ===, !==	
Redukcioni	&, !&, ^, ^~, , ~	
Logički	&&,	
Uslovni	? :	Najniži



■ Verilog: skraćivanje evaluacije izraza (*short-circuit*)

- Poštujući pravila prioriteta operatora i njihove asocijativnosti, nekada je moguće izračunati rezultat prije nego se u obzir uzmu svi članovi izraza
- U tom slučaju nema potrebe za evaluacijom čitavog izraza
- Na primjer:

```
assign out = ((a > b) & (c | d));
```

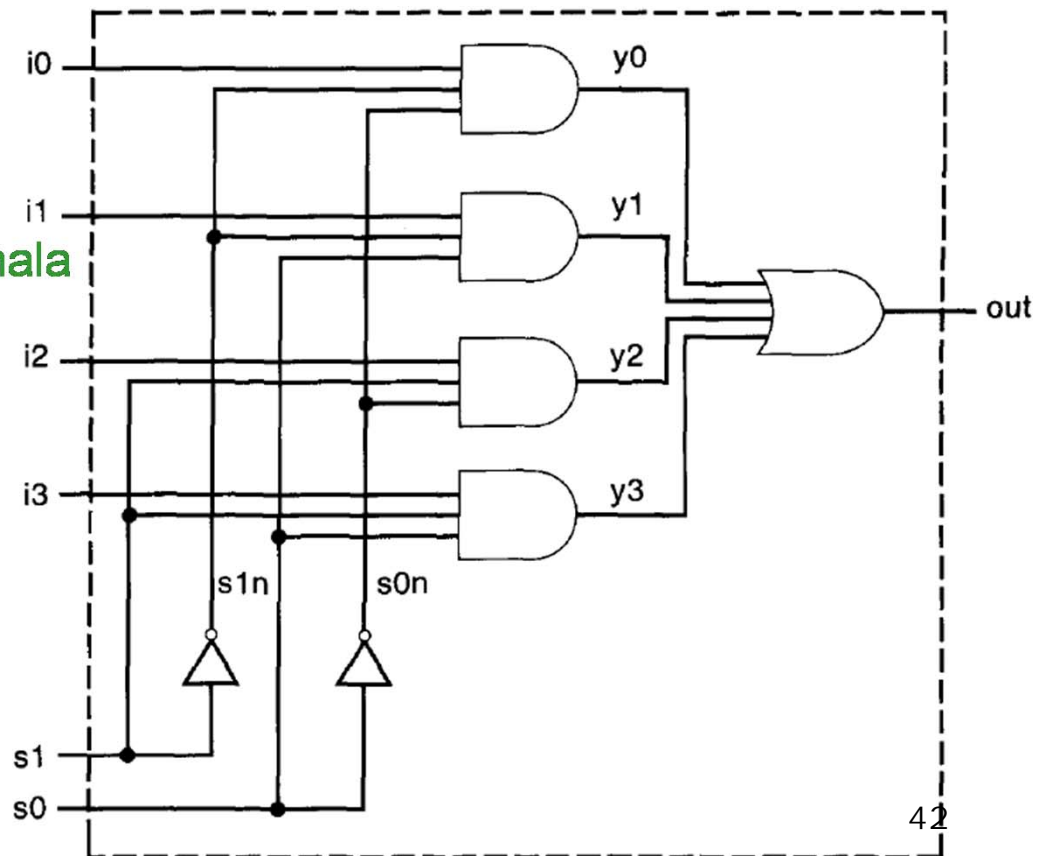
- Ako je rezultat `(a > b)` netačan (1'b0), rezultat AND operacije će biti 0 bez obzira na vrijednost preostalog dijela izraza
- Dakle, tada nema potrebe vršiti evaluaciju izraza `(c | d)`

■ Verilog: primjeri *dataflow* modelovanja

■ Multiplekser 4/1

- Podsjećanje na nivo logičkih kapija:

```
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
```





■ Verilog: primjeri *dataflow* modelovanja

■ Multiplekser 4/1 (prvi način – logička jednačina)

$$out = I_0 \overline{S_1} \overline{S_0} + I_1 \overline{S_1} S_0 + I_2 S_1 \overline{S_0} + I_3 S_1 S_0$$

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

```
    assign out = (~s1 & ~s0 & i0) |  
                  (~s1 &  s0 & i1) |  
                  (  s1 & ~s0 & i2) |  
                  (  s1 &  s0 & i3);  
endmodule
```

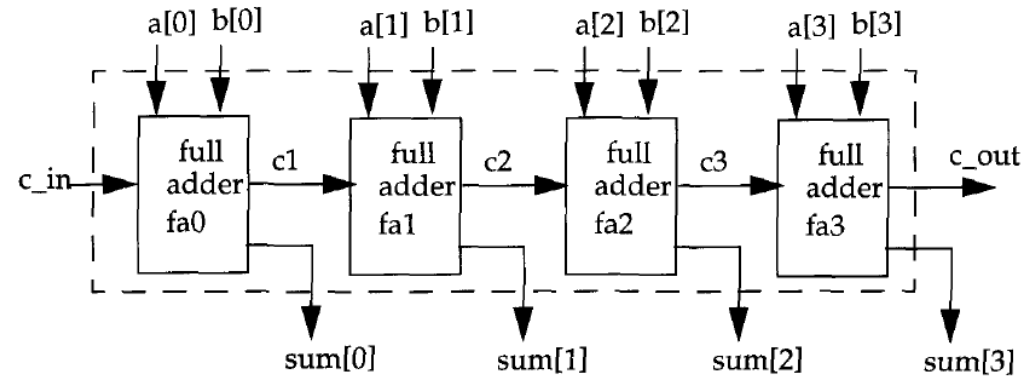
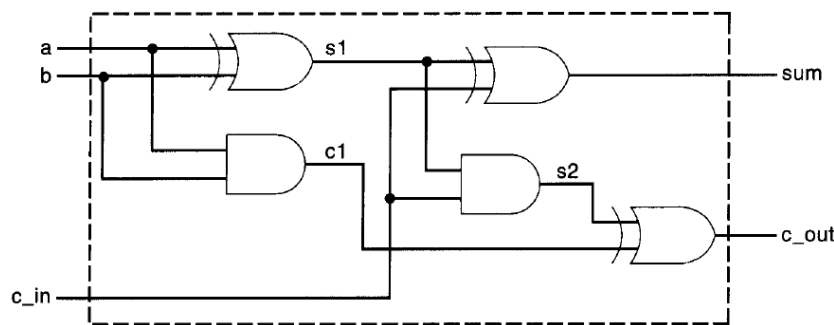
- 
- Verilog: primjeri *dataflow* modelovanja
 - **Multiplexer 4/1 (drugi način – uslovni operator)**

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);  
output out;  
input i0, i1, i2, i3, s1, s0;  
  
assign out = s1 ? ( s0 ? i3 : i2) : (s0 ? i1 : i0);  
  
endmodule
```

■ Verilog: primjeri *dataflow* modelovanja

■ 4-bitni potpuni binarni sabirač

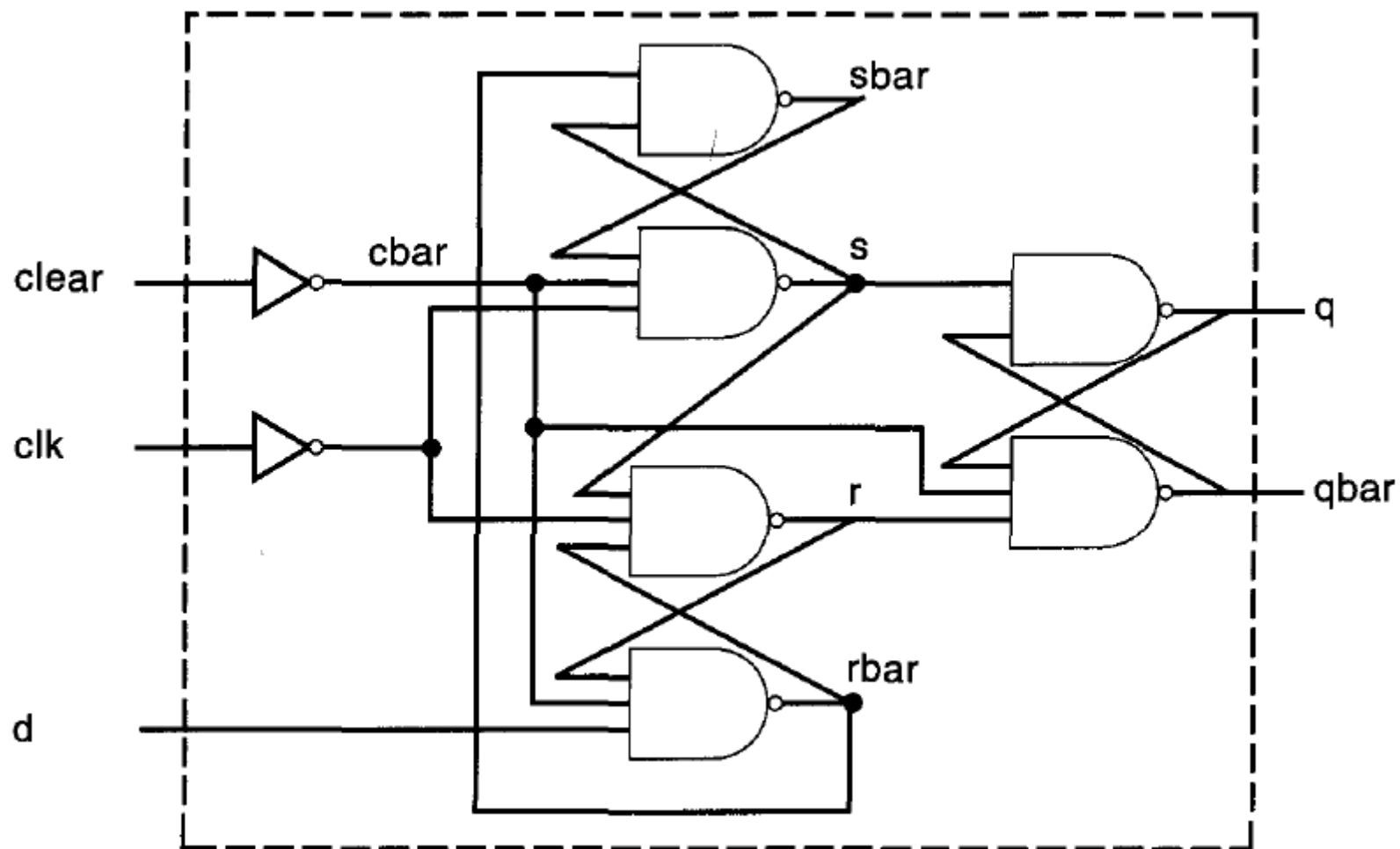
- Podsjećanje – projektovali smo jednobitni FA, pa pomoću njega 4-bitni



- Dataflow:

```
module fulladd4(sum, c_out, a, b, c_in);  
  output [3:0] sum;  
  output c_out;  
  input[3:0] a, b;  
  input c_in;  
  assign {c_out, sum} = a + b + c_in;  
endmodule
```

- Verilog: primjeri *dataflow* modelovanja
- D flip flop koji reaguje na silaznu ivicu i ima *clear* signal



■ Verilog: primjeri *dataflow* modelovanja

■ D flip flop koji reaguje na silaznu ivicu i ima *clear* signal

```
module nedge_dff(q, qbar, d, clk, clear);  
output q,qbar;  
input d, clk, clear;
```

```
wire s, sbar, r, rbar, cbar;
```

```
assign cbar = ~clear;
```

```
assign sbar = ~(rbar & s),  
       s = ~(sbar & cbar & ~clk),  
       r = ~(rbar & ~clk & s),  
       rbar = ~(r & cbar & d);
```

```
assign q = ~(s & qbar),  
       qbar = ~(q & r & cbar);
```

```
endmodule
```

