PROJEKTOVANJE INTEGRISANIH KOLA SA MEŠOVITIM SIGNALIMA

5

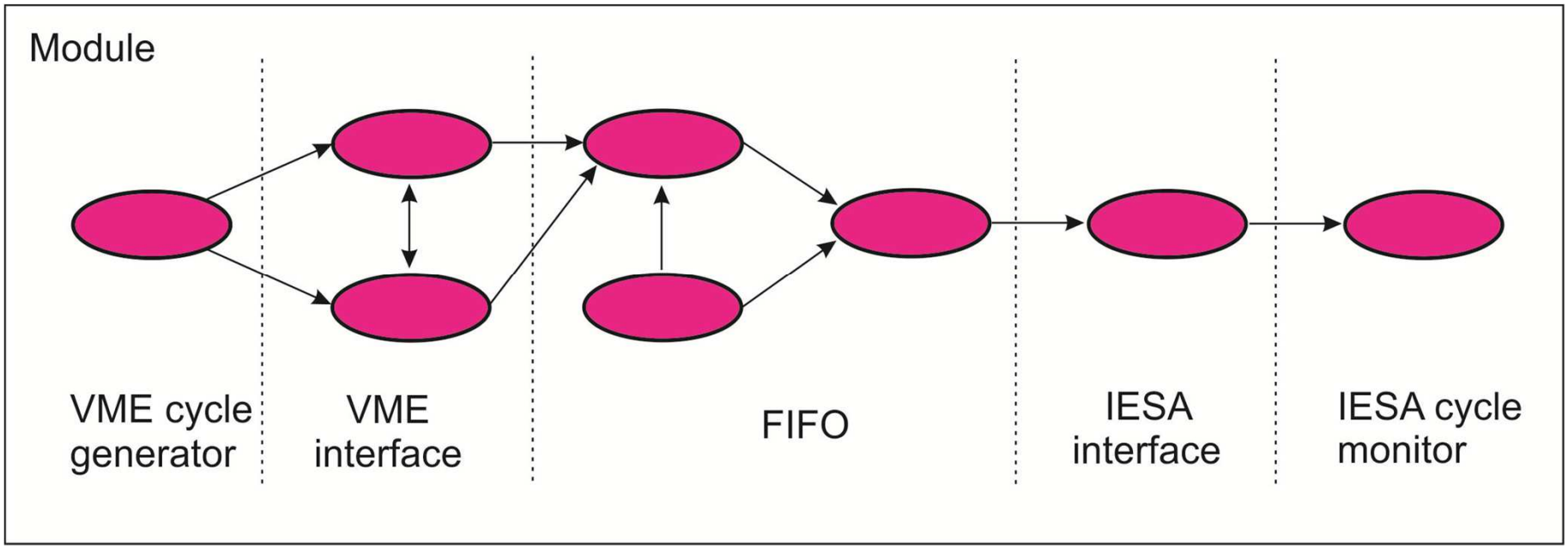
Zašto je Verilog dobar?

- Sličan je programskim jezicima
  - koristi sekvencijalne iskaze
- Šta je potrebno još naučiti:
  - Hijerarhijska dekompozicija
  - Povezanost - connectivity

Instanciranje

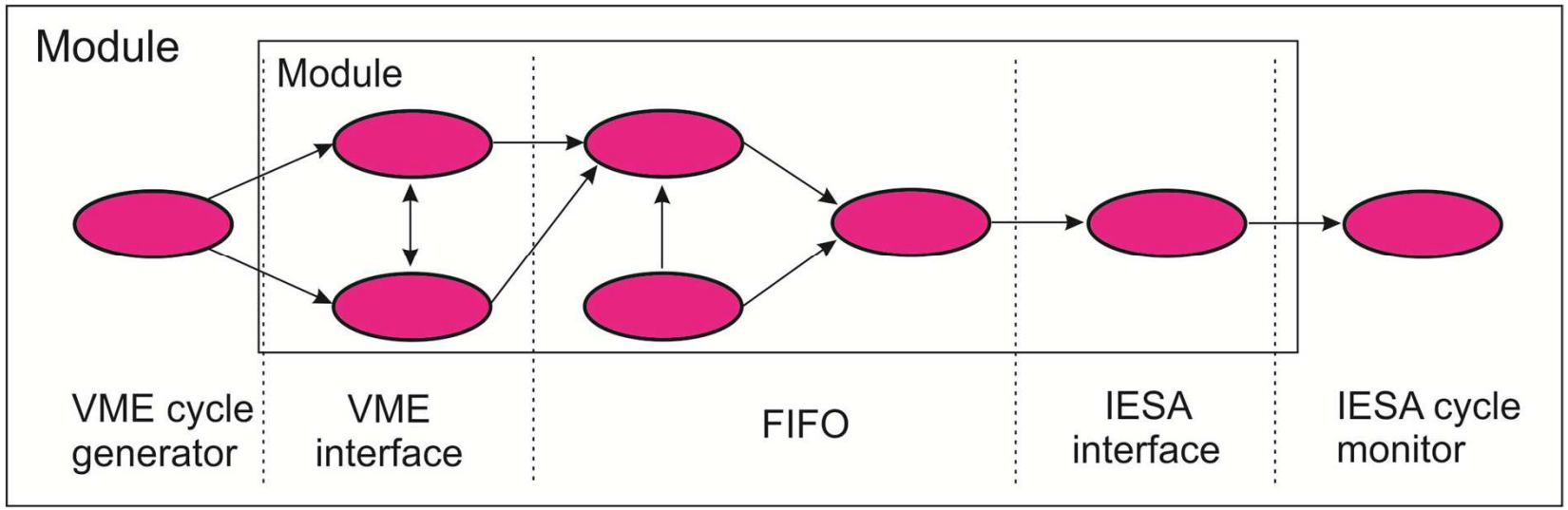
Hijerarhijska dekompozicija

- Flat dizajni mogu biti glomazni



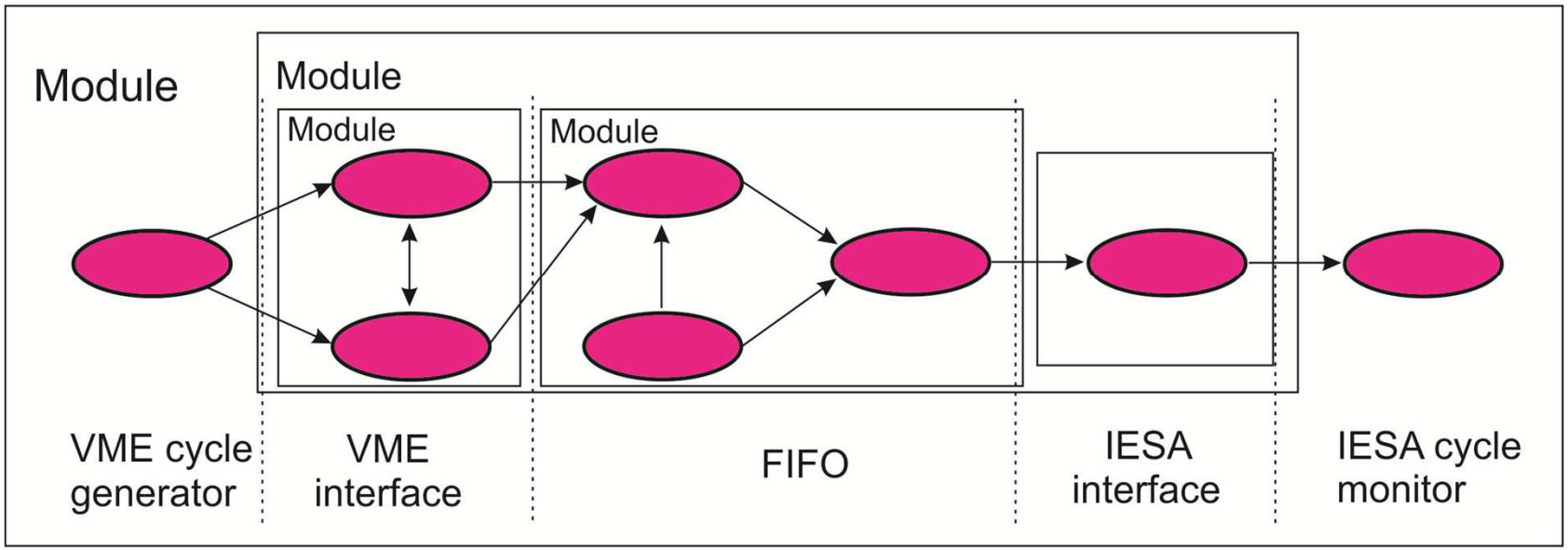
Hijerarhijska dekompozicija

- Dizajni mogu biti razloženi na podmodule



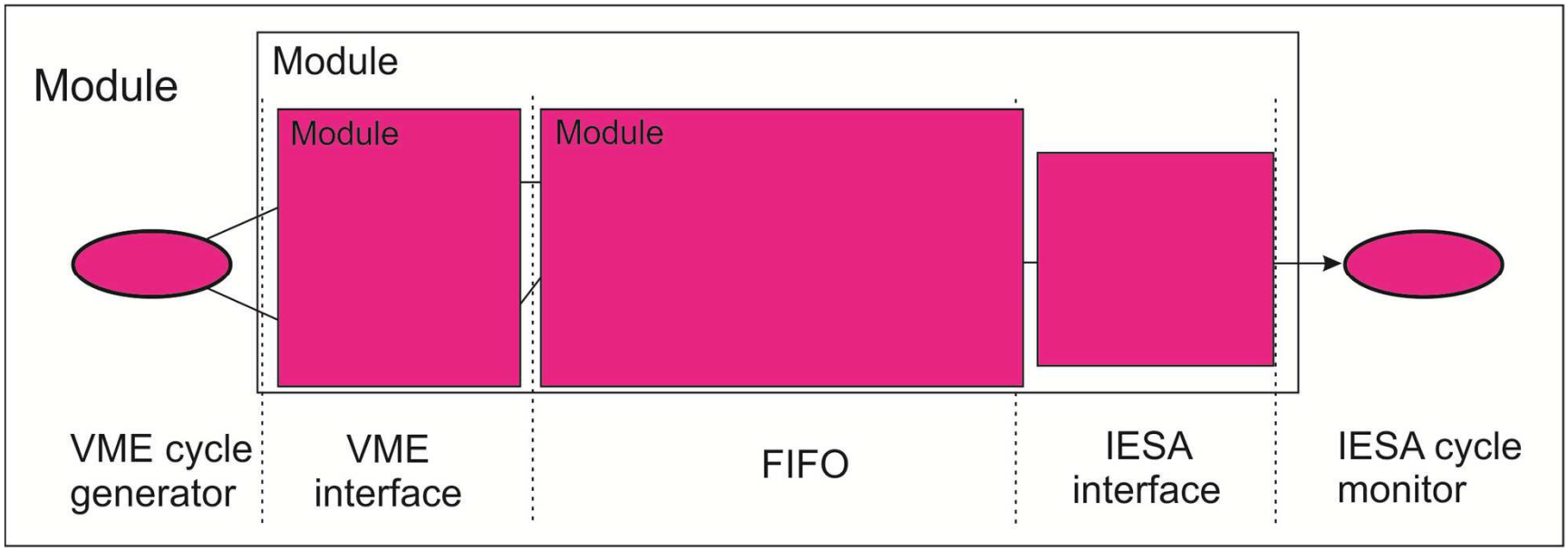
Hijerarhijska dekompozicija

- Podmoduli također mogu sadržati podmodule



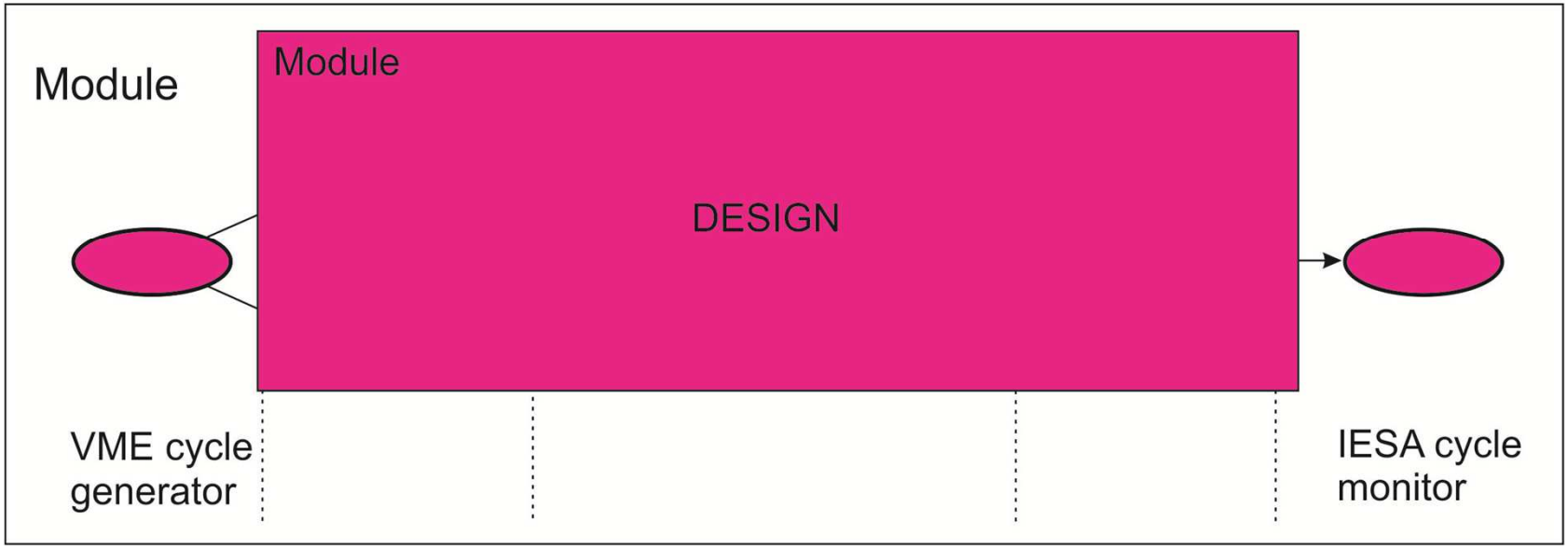
Hijerarhijska dekompozicija

- Moduli postaju crne kutije



Hijerarhijska dekompozicija

- Moduli postaju crne kutije



Pinovi modula

- Informacija ulazi i izlazi iz modula putem *pinova*
- *Pinovi* mogu biti samo bitovi ili vektori
  - Ne integer, real, ni time

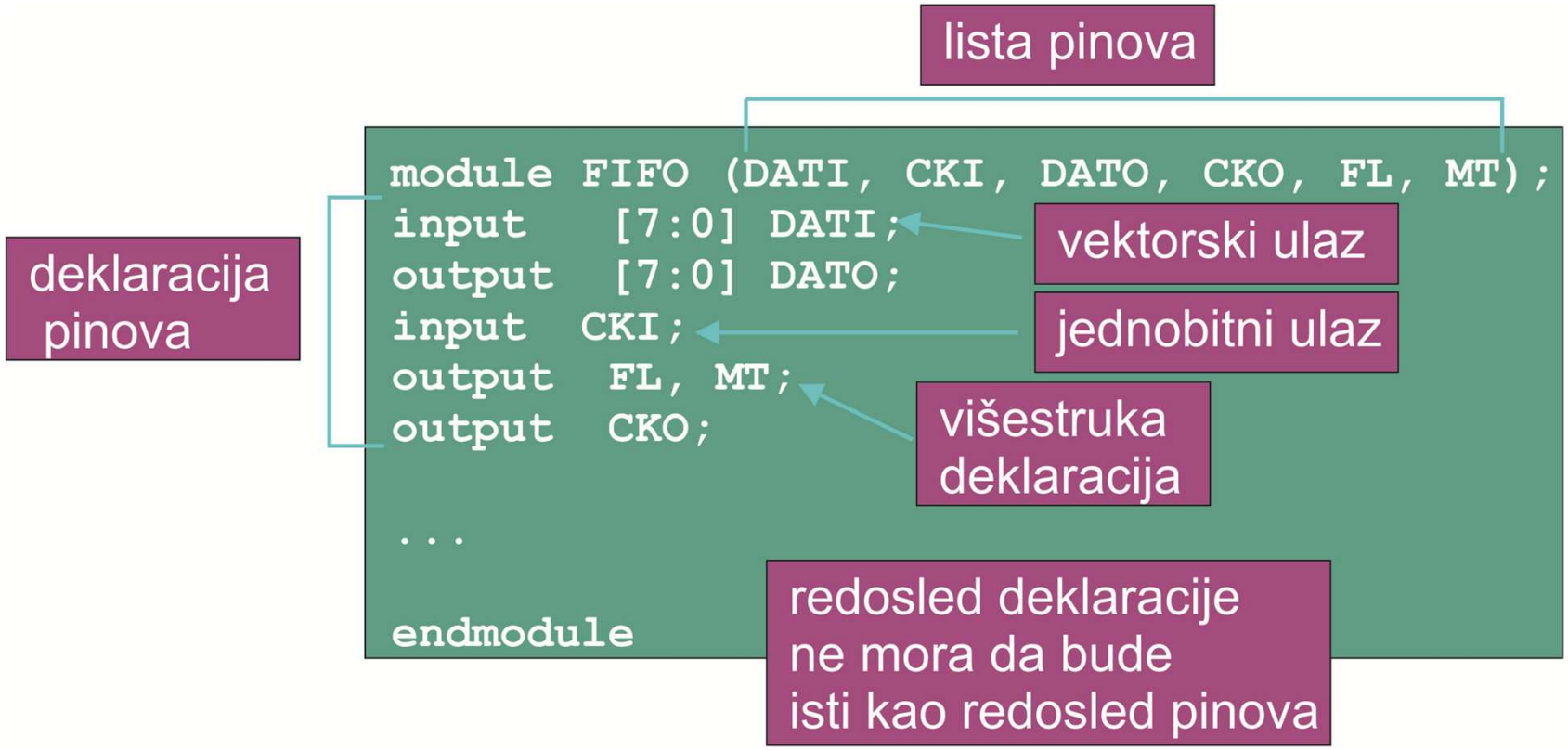
- Sintaksa:

```
module <name>({<name>,});
[input  [<size>] {<name>,};]
[output [<size>] {<name>,};]
[inout  [<size>] {<name>,};]

{<module_items>}

endmodule
```

Pinovi modula

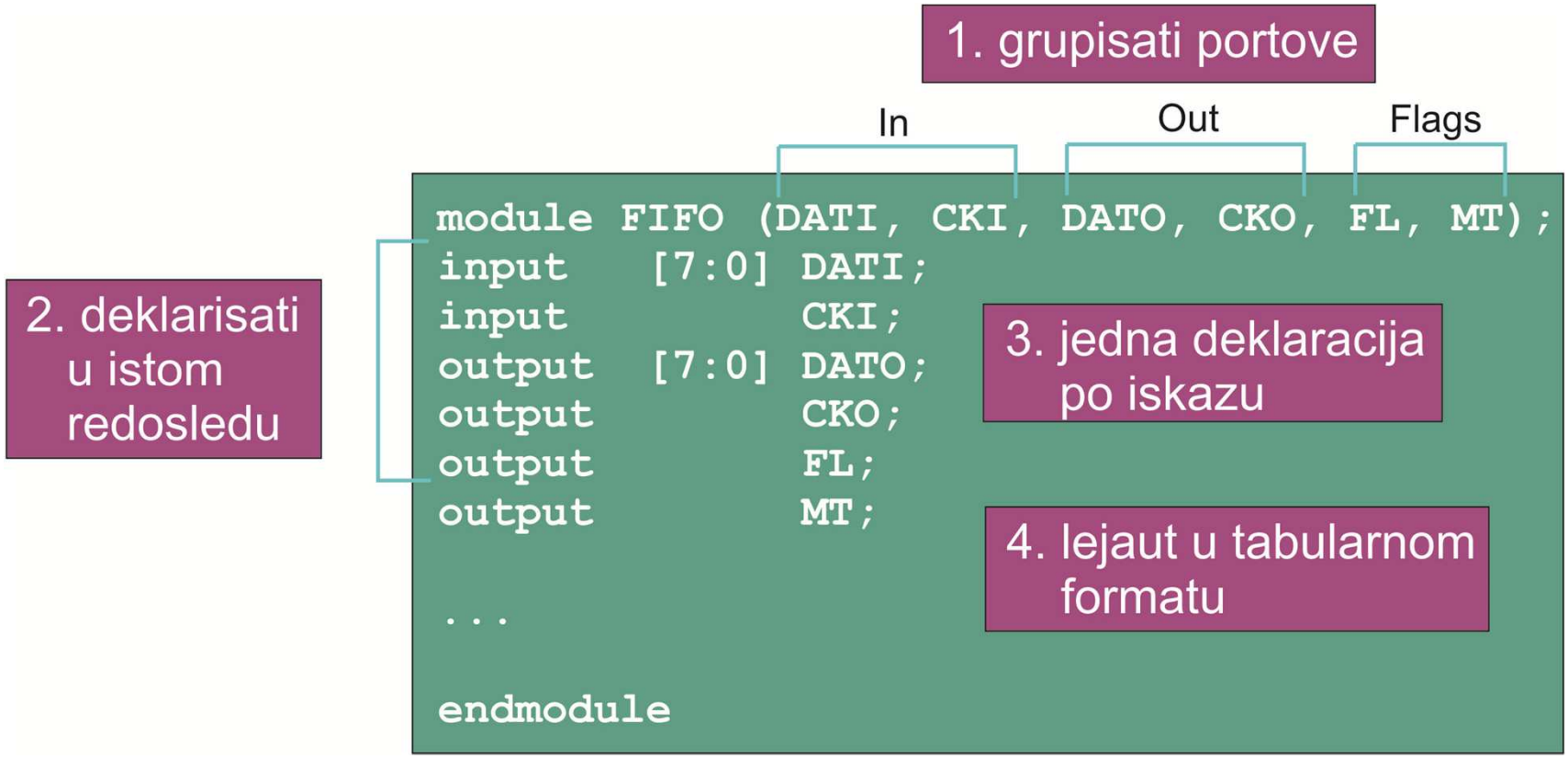
- Primer:

in, out i inout su u proizvoljnom redosledu



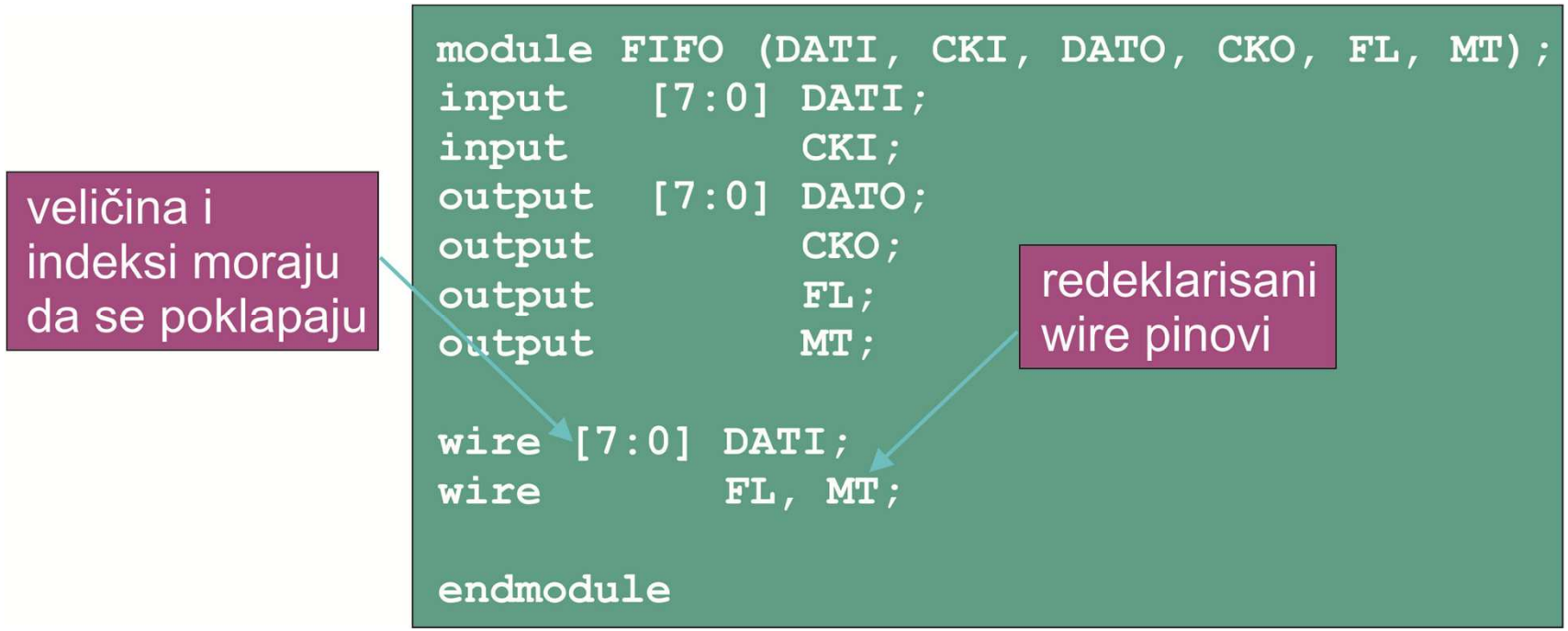
Pinovi modula

- Uputstvo:



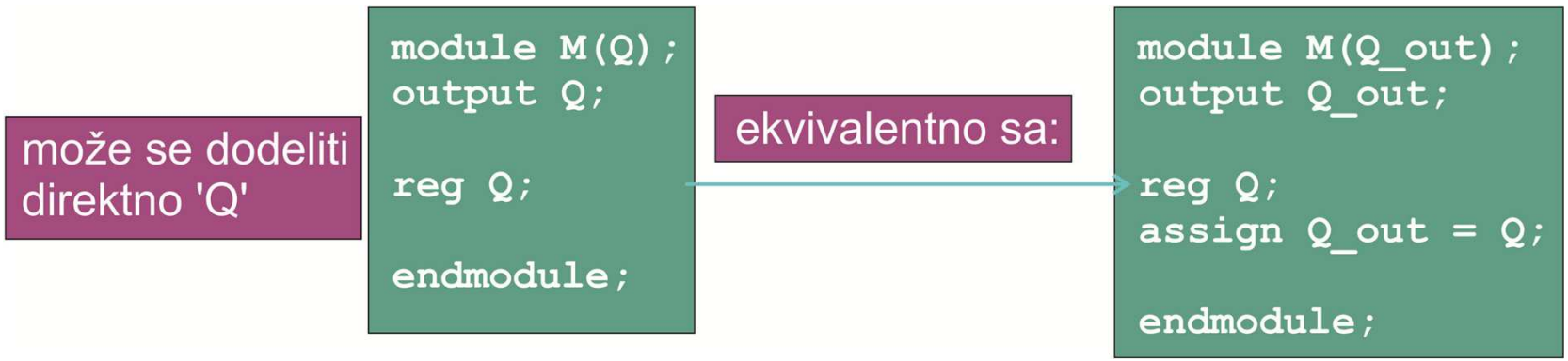
Pinovi modula

- Pinovi su wires
- Mogu da se redeklarišu
- Primer:



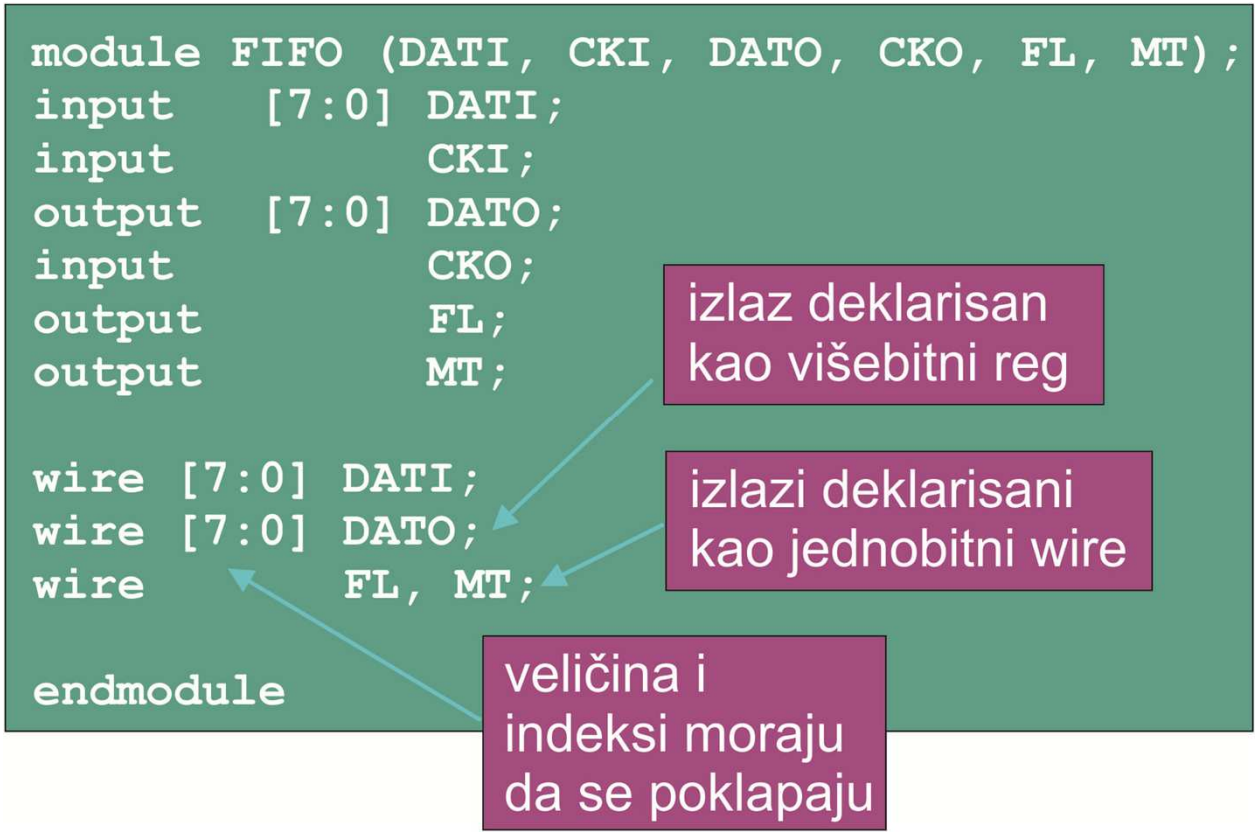
Pinovi modula

- Output pinovi mogu da se deklarišu kao *reg*
- Kreira se implicitni continuous assignment sa nultim kašnjenjem između reg i output pina
- Omogućava dodeljivanje output pinovima
- Primer:



Pinovi modula

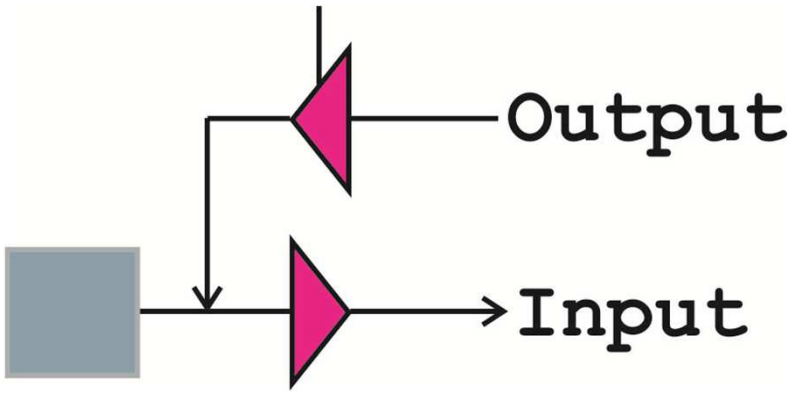
- Primer:



u novijim standardima je moguće odmah deklarisati wire i reg

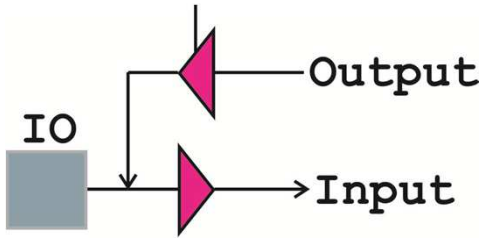
Inout

- Inout pinovi su wires
  - ne može im se direktno dodeliti
- Mora se koristiti continuous assignment da bi se pobudila ‘out’ komponenta signala
- Analogno sa I/O pad strukturom



Inout

- Može se isključiti ‘out’ komponenta signala dodeljivanjem ‘bz u:
  - continuous assignment-u
  - u reg assignment-u
- Primeri:




```
assign IO = (CS == 1'b1) ?
            Q : 1'bz;

always @(...)
begin
  Q <= <expr>;
end
```

```
assign IO = Q;

always @(...)
begin
  if (CS == 1'b1) Q <= <expr>;
  else            Q <= 1'bz;
end
```

Instanciranje

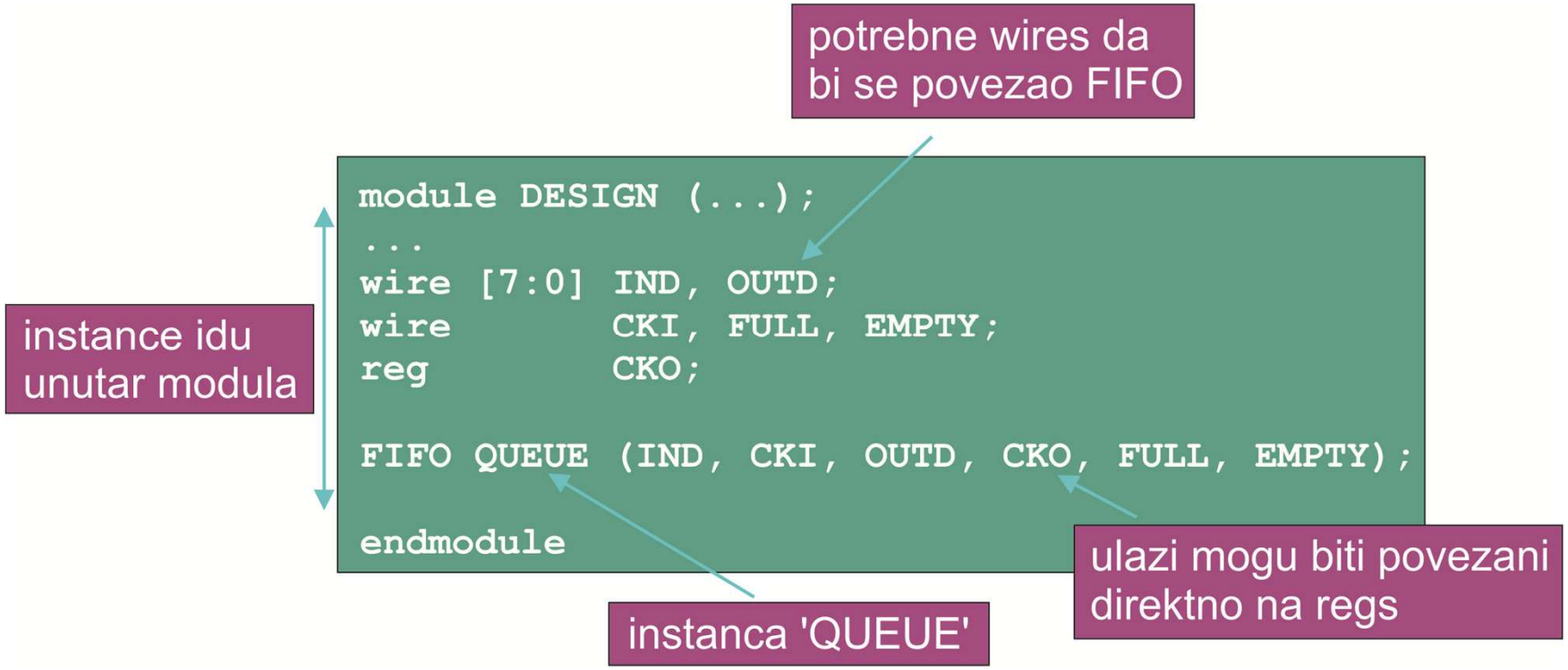
- Podmoduli se koriste tako što se instanciraju u modul višeg nivoa
- Pinovi se povezuju za wires
- Ulazni pinovi mogu biti povezani direktno na reg
  - kreira se implicitni continuous assignment sa nultim kašnjenjem

- Wires koje se ne poklapaju se odbacuju ili popunjavaju nulama
- Sintaksa:

```
<module-name> <instance-name>
    ({[.<pin-name>(]<wire-name>[)],});
```

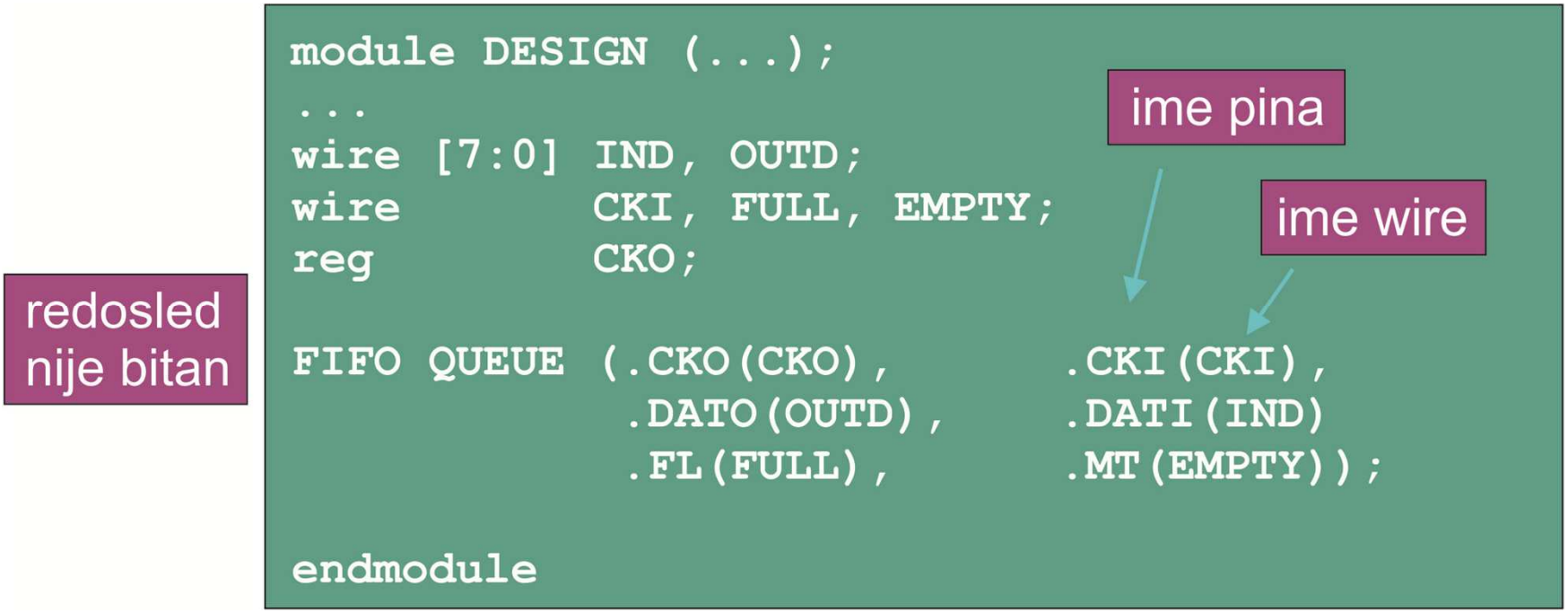
Instanciranje

- Veze se prave u skladu sa redosledom pinova
- Primer:



Instanciranje

- Povezivanje može da se vrši u odnosu na ime pina
- Primer:



Instanciranje

- Pinovi mogu da budu ostavljeni nepovezani
- Ulazi i inouts se onda pobuđuju visokom impedansom
- .CKI() ne povezujemo pin jer nećemo da ga koristimo.

Možemo i da ga potpuno izostavimo (kao MT)

- Primer:

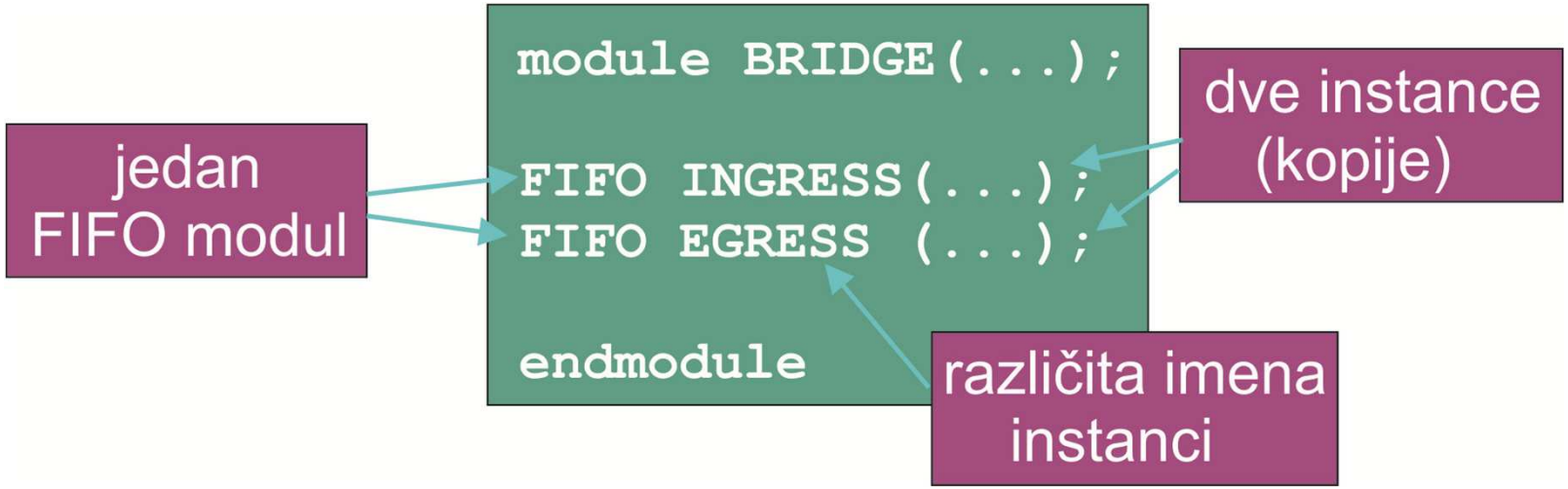
'CKI' i 'MT' su nepovezani

```
FIFO POSITION(IND, , OUTD, CKO, FULL);

FIFO NAMED (.CKO(CKO),          .CKI(),
            .DATO(OUTD),        .DATI(IND)
            .FL(FULL));
```

Instanciranje

- Modul može biti instanciran više nego jednom
- čuva se samo jedna source kopija
- koristi se onoliko kopija koliko je potrebno u simulaciji

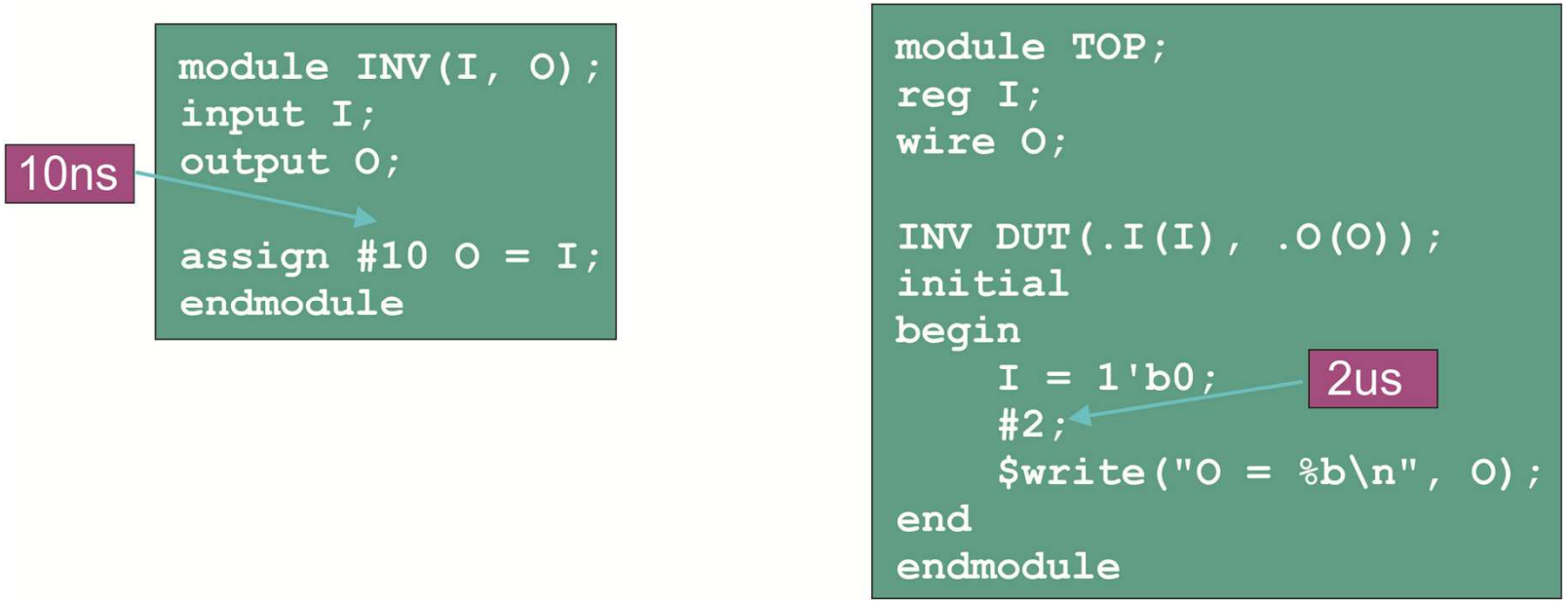
- Primer:



Vremenska skala (Timescale)

Timescale

- Do sada smo kašnjenja specificirali kao vremenske jedinice bez oznake za jedinicu
- Šta ako različiti moduli imaju različit pojam o tome šta je vremenska jedinica?
- Primer:



Timescale

- Rešenje: *`timescale* direktiva
- Utiče na interpretaciju vremenskih jedinica u modulima koji idu posle nje
- Mora biti van modula

- Sintaksa:

```
`timescale <scale> / <precision>
```

- Primer:

```
`timescale 1 ns / 100 ps
```

1, 10 ili 100

s, ms, us, ns, ps ili fs

Timescale

- Primer:

```
`timescale 1ns/1ns
module INV(I, O);
input I;
output O;

assign #10 O = I;
endmodule
```

10ns

```
`timescale 1us/1us
module TOP;
reg I;
wire O;

INV DUT(.I(I), .O(O));
initial
begin
    I = 1'b0;
    #2;
    $write("O = %b\n", O);
end
endmodule
```

2us

Timescale

- Simulacija koristi najmanju od specificiranih preciznosti u svim `timescale direktivama
- Vrednosti kašnjenja su zaokružena na preciznost vremenske skale modula
- Neki simulatori se usporavaju sa povećavanjem odnosa scale i precision
  - izbegavati preveliku preciznost

Timescale

- Pitanje: koje su apsolutne vrednosti vremena?

Timescale:	1ns/10ps	100ps/100ps
#1	1.00ns	100ps
#1.5	1.50ns	200ps
#1.555	1.56ns	200ps
#0	0.00ns	0ps
#(1.0/3)	0.33ns	0ps

Timescale

- Pitanje: Kako modelovati takt generator sa faktorom ispune od 50% i periodom od 15ns?

Prvo rešenje:

```
`timescale 1ns/1ns
always
begin
#(15/2);
CK <= ~CK;
end
```

integer se deli kašnjenje 7ns. perioda takta je 14ns

Timescale

- Pitanje: Kako modelovati takt generator sa faktorom ispune od 50% i periodom od 15ns?

Drugo rešenje:

```
`timescale 1ns/1ns
always
begin
#(15.0/2);
CK <= ~CK;
end
```

kašnjenje zaokruženo na 8ns. perioda takta je 16ns

Timescale

- Pitanje: Kako modelovati takt generator sa faktorom ispune od 50% i periodom od 15ns?

Treće rešenje:

```
`timescale 1ns/100ps
always
begin
#(15.0/2);
CK <= ~CK;
end
```

povećana preciznost
simulacija se usporava

Najtačnije rešenje

Timescale

- Pitanje: Kako modelovati takt generator sa faktorom ispune od 50% i periodom od 15ns?

Četvrto rešenje:

```
`timescale 1ns/1ns
always
begin
#7;
CK <= 1'b1;
#8;
CK <= 1'b0;
end
```

ukoliko nije neophodan faktor ispune od baš 50%

Brzina simulacije se ne smanjuje

Timescale

- Direktive kompajlera deluju na source fajlove
  - moduli mogu da naslede neočekivani timescale
- Uputstvo: koristiti `timescale direktivu pre svakog modula

Timescale za M2 zavisi od redosleda kompajliranja

```
`timescale 1ns/1ns
module M1;
endmodule;
```

```
module M2;
endmodule;
```

1ns

```
`timescale 1ps/1ps
module M3;
endmodule;
```

```
`timescale 1ns/1ns
module M1;
endmodule;
```

```
`timescale 1ps/1ps
module M3;
endmodule;
```

1ps

```
module M2;
endmodule;
```

Predefinisane vremenske funkcije

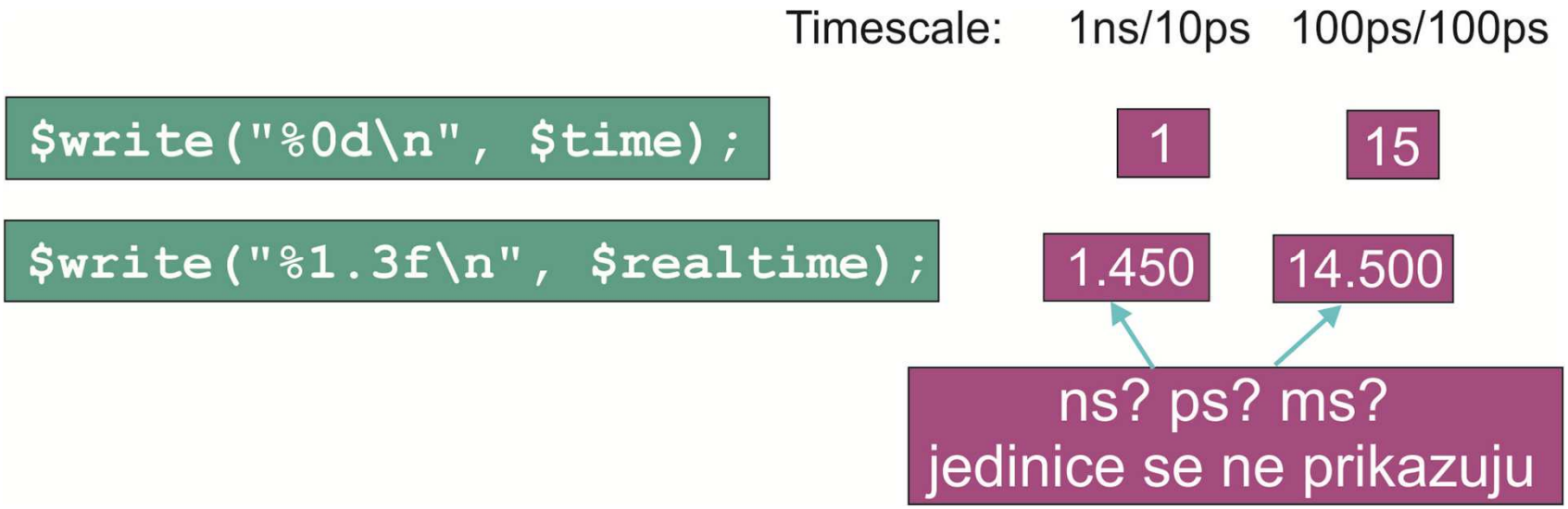
- **$time**
  - vraća trenutno vreme simulacije kao integer broj u tekućoj vremenskoj skali
- **$realtime**
  - vraća trenutno vreme simulacije kao realni broj u tekućoj vremenskoj skali i punoj preciznosti

- Primer:

```
THEN = $time;
...
if ($time - THEN < SETUP) begin
    $write("Setup violation\n");
end
```

Prikazivanje vremena

- Primer: Vreme simulacije je 1450ps



- Mogla bi da se doda jedinica kao string

Prikazivanje vremena

- Moguće je globalno podesiti kako se vreme prikazuje
  - Koristi se '%t' marker u formatu za prikazivanje
- Sintaksa:

```
$timeformat(<magnitude>, <precision>,
            <suffix>, <width>);
```

Magnitude	Scale
0	seconds
-3	milliseconds
-6	microseconds
-9	nanoseconds
-12	picoseconds
-15	femtoseconds

Prikazivanje vremena

- Primer: Vreme simulacije je 1450ps

```
`timescale 1ns/100ps
$write("%t\n", $realtime);
```

`$timeformat(-9, 3, " ns", 10);`	1.450ns
`$timeformat(-9, 3, " ps", 10);`	1.450ps
`$timeformat(-12, 3, " ps", 10);`	1450.000ps
`$timeformat(-9, 0, " ns", 10);`	1ns

Parametri

Parametri

- Zadatak: napisati Verilog module za 1-bitni sabirač, 2-bitni sabirač,..., i 32-bitni sabirač, i još sabirača koji budu potrebni
- Jedan pristup:

add1.v

```
module ADD1(A, B, S);
input A, B;
output S;

assign S = A + B;

endmodule
```

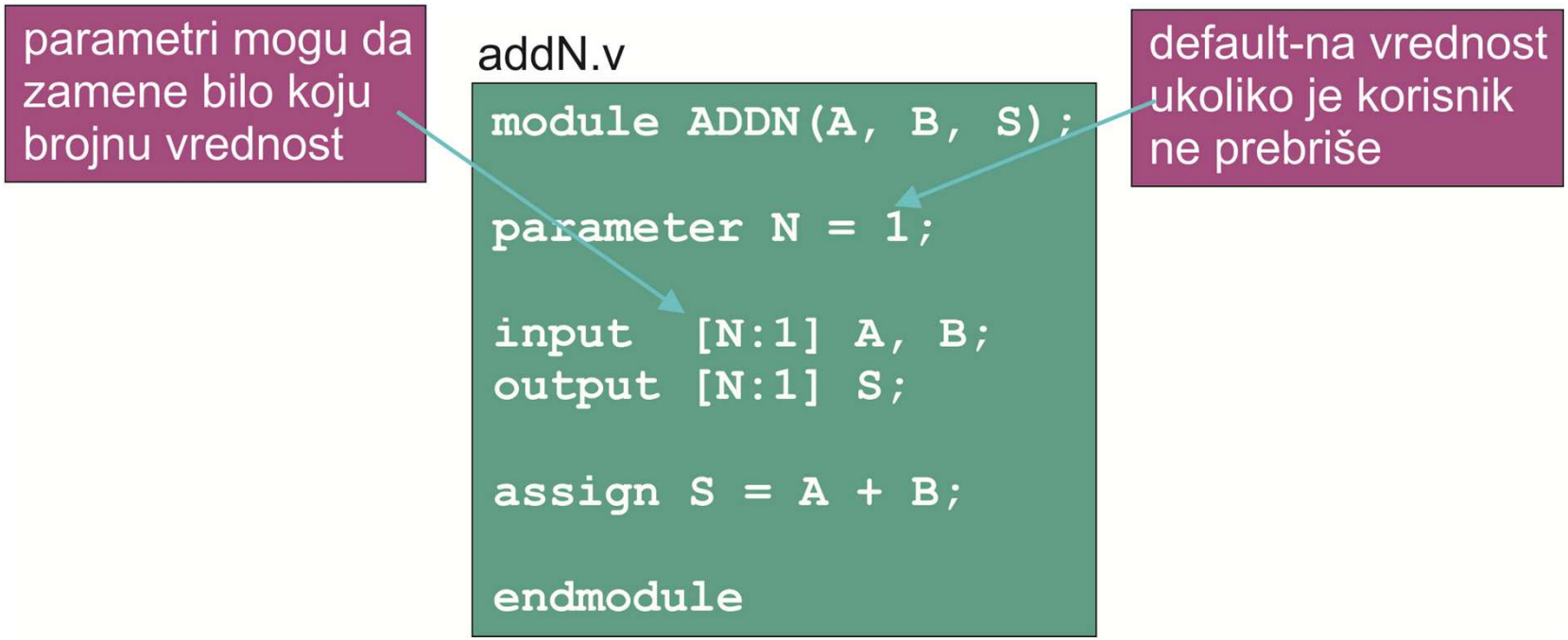
add2.v

```
module ADD2(A, B, S);
input [1:0] A, B;
output [1:0] S;

assign S = A + B;

endmodule
```

ponoviti za ADD3,..., ADD32

Parametri

- Bolji pristup: modelovati *N*-bitni sabirač
  - definisati *N* kao parametar



Parametri

- Parametri mogu biti prebrisani različitim vrednostima za svaku instancu
  - jedna fizička kopija modula u fajl sistemu
  - mnogo logičkih kopija u simulaciji
    - po jedna za svaku instancu

- Sintaksa #1:

```
<module-name> #({param-value,})
    <instance-name> ({connection,});
```

- Sintaksa #2:

```
defparam
   {<instance-name>.<param-name>
        = <param-value>,};
```

Parametri

- Primer:

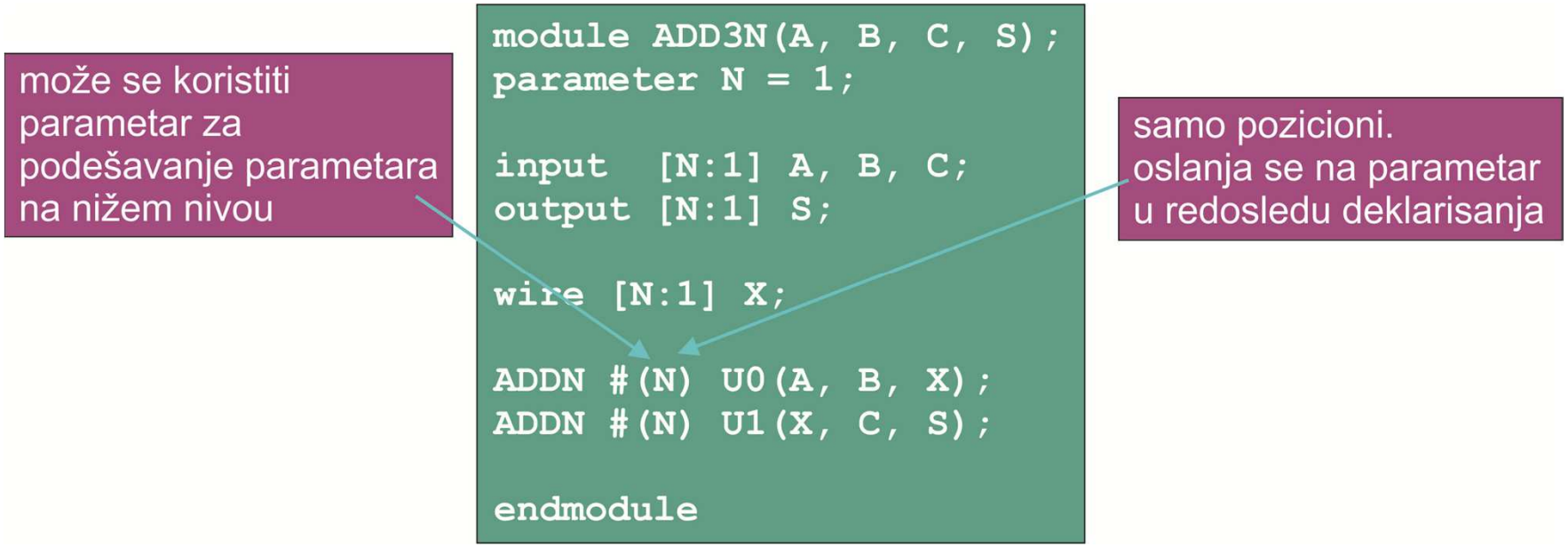
```
ADDN #(3)  U0(A, B, X);
ADDN #(32) U1(OP1, OP2, ACC);
```

```
ADDN U0(A, B, X);
ADDN U1(OP1, OP2, ACC);

defparam U0.N = 3;
defparam U1.N = 32;
```

ekvivalentni modeli

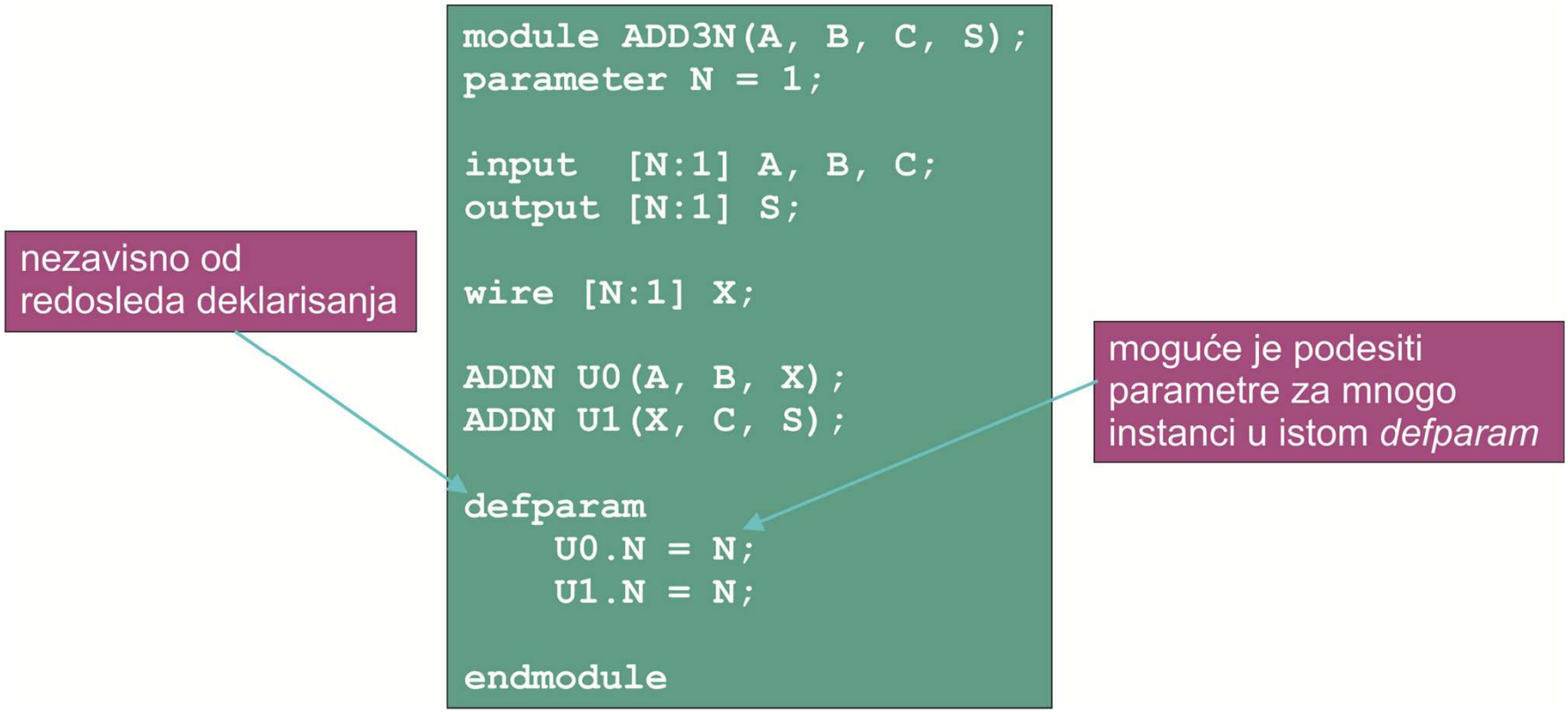
Parametri

- Primer:



Parametri

- Primer:



Primitive

Primitive

- Primitive su ugrađene u predefinisane module
- Mogu da imaju bolje performanse u simulaciji nego model na nivou gejtova koji koristi modele gejtova
- Mogu da koriste ime instance (preporučeno)
- Koriste jednobitne pinove
- Koriste samo poziciono povezivanje
  - redosled portova je uvek output(s), input(s), i enable

Primitive

- Primitive sa N ulaza
  - and, nand, or, nor, xor, xnor
- Sintaksa:

```
<primitive> [<instance-name>]
    (<output>, {<input>,});
```

- Primeri:

```
and U0(Z, A, B);
xor(EVEN, DATA[0], DATA[1], DATA[2], DATA[3],
          DATA[4], DATA[5], DATA[6], DATA[7]);
```

```
xor U1(EVEN, DATA);
```

greška ako je DATA vektor (odseca na DATA[0])

Primitive

- Primitive sa N izlaza
  - buf, not
- Sintaksa:

```
<primitive> [<instance-name>]
    ({<output>,} <input>);
```

- Primeri:

```
buf(OUT, IN);
not N0(OUT1, OUT2, OUT3, IN);
```

raspodeliti opterećenje na izlaze invertora

- Napomena: koristiti samo jedan izlaz

Primitive

- 3-state primitive
  - bufif0, bufif1 neinvertujući enabled low/high
  - notif0, notif1 invertujući enabled low/high

- Sintaksa:

```
<primitive> [<instance-name>]
    (<output>, <input>,<enable>);
```

- Primeri:

```
bufif1 PAD(O, I, EN);
```

```
assign O = (EN) ? I : 'bz;
```

bolje
radi sa 32 bita

Primitive

- Pull tranzistorske primitive
  - pullup, pulldown
- Sintaksa:

```
<primitive> [<instance-name>](<output>);
```

- Primeri:

```
pullup(BUS);
```

```
assign (pull1, pull0) BUS = -1;
```

identična funkcionalnost
radi sa do 32 bita

Primitive

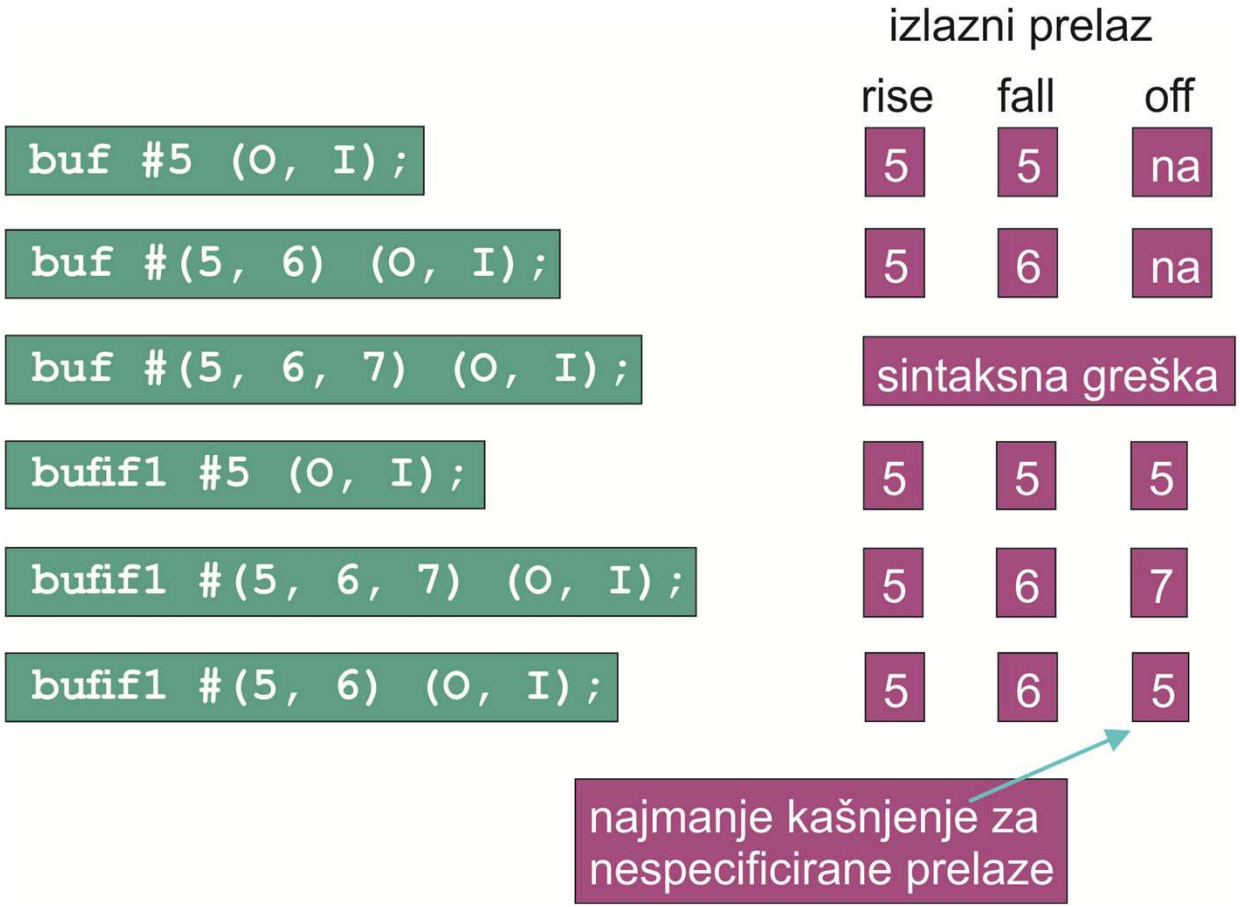
- Po difoltu, primitive imaju nulto kašnjenje
- Moguće je specificirati rise, fall i turn-off kašnjenje
- Inercioni model kašnjenja
- koristi se vremenska skala instanciranog modula

- Sintaksa:

```
<primitive>
    [#(<rise>[, <fall>[, <off>]])]
    [<instance-name>]
    ({<output>,} {<input>,} [<enable>]);
```

Primitive

- Primeri:



Primitive

- Moguće je definisati svoju primitivu
- Zove se „User Defined Primitive“ (UDP)
- Korisno je za modelovanje ćelija u biblioteci
  - npr. definisati primitivu D FF