Parametrizovano projektovanje

- Design reuse - projektovanje za ponovno korišćenje

  Svuda tamo gde je to moguće, koristi ranije projektovane module

  Module koje se projektuju realizovati na takav način da se lako mogu ponovno iskoristiti
- Pretpostavka projektovanja za ponovno korišćenje jeste u mogućnosti parametrizacije projektnih modula
- Umesto modula fiksnih karakteristika I funkcionalnosti, treba projektovati ˝univerzalne˝ module - pri ugradnji u novi sistem mogu se u izvesnoj meri I na lak način prilagoditi specifičnim zahtevima I nove primene.

Podrška za parametrizovano projektovanje u VHDL-u

- GENERIC
- Atributi i konstante
- Konkurentna naredba GENERATE
- Sekencijalna naredba LOOP

Vrste parametara

- Dimenzioni

  Definišu veličinu, odnosno broj bita višebitnih portova i signala koji se koriste za razmenu podataka sa sistemom, odnosno za prenos I čuvanje podataka unutar sistema

- Funkcionalni

  Uključivanje/isključivanje pojedinih funkcija ili izbor jedne od nekoliko raspoloživih varijanti implementacije sistema

Specifikacija parametara

- Generički parametri
- Atributi vektora
- Vektori s nedefinisanim opsegom

Generički parametri

```
ENTITY binarni_brojac IS
  GENERIC(N :  NATURAL);
  PORT(clk, rst : IN STD_LOGIC;
          q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
END bin_brojac;
```

Deklaracija generičkog entiteta

```
COMPONENT binarni_brojac
  GENERIC(N NATURAL) : NATURAL);
  PORT(clk, rst : IN STD_LOGIC;
          q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
END COMPONENT;
```

Deklaracija generičke komponente

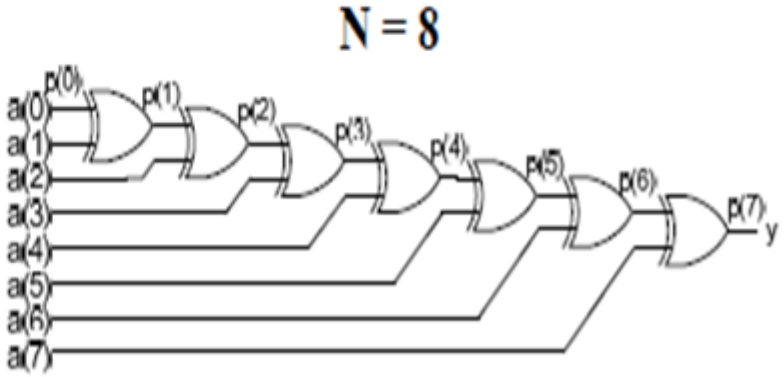
```
dvo_bit_brojac: binarni_brojac
  GENERIC MAP(N => 2);
  PORT MAP(clk => clock, rst => reset, q => q_out);
```

Instancira generičke komponente

Parametrizovan opis generatora bita parnosti

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY pargen IS
    GENERIC(N: INTEGER := 8);
    PORT (a : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
          y : OUT STD_LOGIC);
END pargen;
ARCHITECTURE param_arch OF pargen IS
BEGIN
    PROCESS(a)
        VARIABLE p : STD_LOGIC;
    BEGIN
        p := a(0);
        FOR i IN 1 TO (N-1) LOOP N
            p := p XOR a(i);
        END LOOP;
        y <= p;
    END PROCESS;
END param_arch;
```

Podrazumevana vrednost generičkog parametra



Atributi vektora

- Podsećanje:

Za signale:

SIGNAL s1 : STD_LOGIC_VECTOR(15 DOWNTO 0);

SIGNAL s2 : STD_LOGIC_VECTOR(8 TO 15);

važi:

s1'LEFT = 15; s1'RIGHT = 0;

s1'LOW = 0; s1'HIGH = 15;

s1'LENGTH = 32;

s1'RANGE = 15 DOWNTO 0;

S1'REVERSE RANGDE = 0 TO 15;

S2'LEFT = 8; s2'RIGHT = 15;

s2'low = 8; s2'HIGH = 15;

s2'LENGTH = 8;

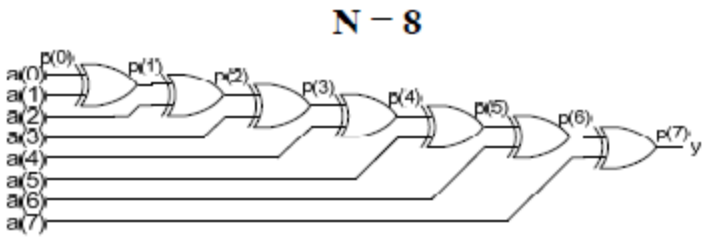
s2'RANGE = 8 TO 15;

s2'REVERSE_RANGDE = 15 DOWNTO 8;

Parametrizovan opis generatora parnosti korišćenjem atributa vektora

```
ENTITY pargen IS
    GENERIC(N: INTEGER := 8);
    PORT (a : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
          y : OUT STD_LOGIC);
END pargen;
ARCHITECTURE param_arch OF pargen IS
BEGIN
  PROCESS(a)
     VARIABLE p : STD LOGIC;
  BEGIN
     p := a(0);
     FOR i IN 1 TO (a'length-1) LOOP
        p := p XOR a(i);
     END LOOP;
     y <= p;
  END PROCESS;
END param_arch;
```

A moglo je i ovako:
FOR i IN (a'low + 1) TO a'high LOOP ili
FOR i IN (a'right + 1) TO a'left LOOP



Vektori s nedefinisanim opsegom

- Npr. u paketu *std_logic_1164, tip podataka std_logic_vector definisan je na sledeći način:*

TYPE STD_LOGIC_VECTOR IS ARRAY(NATURAL RANGE <>) OF STD_LOGIC;

- Tj. kao vektor s nedefinisanim opsegom indeksa.
- Opseg se specificira prilikom deklaracije signala:

SIGNAL s1 : STD_LOGIC_VECTOR(15 DOWNTO 0);

Port, za razliku od signala, može biti deklarisan bez navođenja opsega sledećem opsega, kao u primeru:

Sabirač bez granica

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_ALL;
ENTITY unconstrain_adder IS
    PORT(a : IN STD_LOGIC_VECTOR;
         b : IN STD_LOGIC_VECTOR;
         c : OUT STD_LOGIC_VECTOR);
END unconstrain_adder;
ARCHITECTURE arch OF unconstrain_adder IS
BEGIN
    c <= STD_LOGIC_VECTOR(UNSIGNED(a) + UNSIGNED(b));
END arch;
```

Opseg indeksa nije definisan !

Ne može se sintetizovati, ali se može koristiti kao komponenta

Prilikom instanciranja portovi preuzimaju instanciranja, sabirača opsege stvarnih signala:

```
. . .
SIGNAL a1, b1, c1: STD_LOGIC_VECTOR(7 DOWNTO 0);
. . .
add8 : unconstrain_adder
   PORT MAP(a=>a1, b=>b1, c=>c1);
```

Generator bita parnosti s portovima bez definisanog opsega

```
ENTITY unconstrain_pargen IS
    PORT (a : IN STD_LOGIC_VECTOR;
          y : OUT STD_LOGIC);
ND unconstrain_pargen;
ARCHITECTURE arch OF unconstrain_pargen IS
  CONSTANT N : NATURAL := a'LENGTH;
  VARIABLE p : STD_LOGIC;
BEGIN
  PROCESS(a)
  BEGIN
    p := a(0);
    FOR N 1 i IN 1 TO (N-LOOP
        p(i) := p XOR a(i);
    END LOOP;
     y <=p;
   END PROCESS;
END arch;
```

N dobija vrednost dužine signala koji se prilikom instanciranja gen. par. vezuje na port *a*.

Oprez!

```
. . .
SIGNAL a1 : STD_LOGIC_VECTOR(15 DOWNTO
0);
SIGNAL y1 : STD_LOGIC;
. . .
pargen8 : unconstrain_pargen
PORT MAP(a=>a1(15 DOWNTO 8), y=>y1);
. . .
```

Izaziva grešku

```
ARCHITECTURE arch OF
unconstrain_pargen IS
  CONSTANT N : NATURAL := a'LENGTH;
  VARIABLE p : STD_LOGIC;
BEGIN
  PROCESS(a)
  BEGIN
    p := a(0);
    FOR i IN 1 TO (N-1) LOOP
        p := p XOR a (i);
    END LOOP;
    y <= p;
  END PROCESS;
END arch;
```

Ovi indeksi ne postoje u stvarnom signalu

Bolji način

```
ARCHITECTURE arch OF unconstrain_pargen IS
        VARIABLE p : STD_LOGIC;
BEGIN
   PROCESS(a)
   BEGIN
        p := a(a‘LOW);
        FOR i IN (a'LOW+1) TO (a'HIGH-1) LOOP
             p := p XOR a(i);
         END LOOP;
         y <= p;
     END PROCESS;
END arch;
```

Ili ovako:

```
ARCHITECTURE arch OF unconstrain_pargen IS
    CONSTANT N : NATURAL := a'LENGTH;
    VARIABLE p : STD_LOGIC;
    SIGNAL aa : STD_LOGIC_VECTOR(N-1
DOWNTO 0);
BEGIN
    aa <= a;
    PROCESS(aa)
    BEGIN
        p := aa(0);
        FOR i IN 1 TO (N-1) LOOP
            p := p XOR aa(i);
        END LOOP;
        y <= p;
    END PROCESS;
END arch;
```

Generički parametri vs. vektori bez definisanog opsega

- Parametrizovan opis koji se oslanja na vektore bez definisanog opsega, iako fleksibilan za korišćenje, mora biti pažljivo pisan i analiziran kako bi se predupredile eventualne greške koje mogu nastati zbog neusklađenosti formata signala.
- Iz tog razloga, za pisanje parametrizovanog kôda preporučuju se generički parametri, osim ukoliko se

ne radi o krajnje generalnim strukturama, poput sabirača

Manipulacije s vektorima - agregacije

- Umesto

  q <= “00000000”; bolje je

  q <= (OTHERS => ‘0’);

- Sve jedinice

  q <= (OTHERS => ‘1’);

˝00…..01˝

  q <= (0 => ‘1’, OTHERS => ‘0’);

Manipulacije s vektorima - agregacije

```
SIGNAL a : STD LOGIC VECTOR(N–1 DOWNTO 0);
. . .
y <= ‘1’ WHEN (a = (OTHERS => ‘0’)) ELSE . . .
```

Nije ispravno!

```
y <= ‘1’ WHEN (a = (a’RANGE => ‘0’)) ELSE . . .
```

Ispravno

Ili ovako:

```
CONSTANT zero : STD_LOGIC_VECTOR(N–1 DOWNTO 0) := (OTHERS => ‘0’);
SIGNAL a : STD_LOGIC_VECTOR(N–1 DOWNTO 0);
. . .
y <= ‘1’ WHEN (a = zero) ELSE . . .
```

Manipulacije s vektorima – korišćenje integer-a i funkcija za konverziju

y <= ‘1’ WHEN (a = “00000101”) ELSE . . .

a je tipa unsigned ili signed

Bolje je ovako:

y <= ‘1’ WHEN (a = 5) ELSE . .

5 se ne menja ako se promeni dužina signala *a*

y <= TO_UNSIGNED(5, N);

y je tipa unsigned ili signed

Ne može ovako:

y <= 5;

Ako želimo da vektoru tipa *std_logic_vector dodelimo integer konstantu:*

y<=STD_LOGIC_VECTOR(TO_UNSIGNED(5, N));

Manipulacije s vektorima – Referenciranje delova i elemenata vektora korišćenjem alijasa

Alias definiše alternativno (obično skraćeno) ime za neki objekat ili deo objekta:

SIGNAL a : SIGNED(7 DOWNTO 0)

ALIAS sign : STD_LOGIC IS a(7);

a(7) je bit znaka

sign je drugo ime za bit a(7)

Može i ovako:

ALIAS sign : STD_LOGIC IS a(a'LEFT);

Ili ovako:

ALIAS sign : STD_LOGIC IS a(N-1);

SIGNED(N-1 DOWNTO 0)

Manipulacije s vektorima – Referenciranje delova i elemenata vektora korišćenjem atributa

Rotacija za 1 poziciju ulevo

q <= q(0) & q(7 DOWNTO 0);

Na opštiji način:

q <= q(q'RIGHT) & q(q'LEFT DOWNTO q'RIGHT+1);

Ispravno ukoliko je dužina signala *q* *veća od* dva i njen opseg indeksa je opadajući (*downto)*

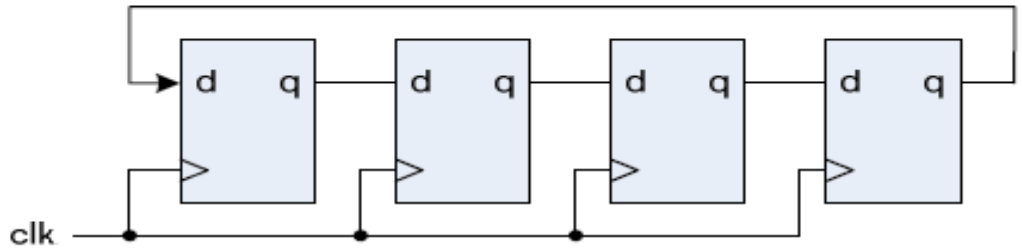
Ako je *q već parametrizovan:*

SIGNAL q : STD_LOGIC_VECTOR(N-1 DOWNTO 0)

q <= q(0) & q(N-1 DOWNTO 0);

Kružni brojač

- Kružni brojač sadrži samo jedno 1 koje se pomera duž registra sve do poslednjeg flip-flopa, a onda se vraća na početak.



″0001″,
″1000″,
″0100″
″0010″.

```
ENTITY para_ring_counter IS
  GENERIC(N : NATURAL);
  PORT (q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        clk, rst : IN STD_LOGIC);
END para_ring_counter;
```

Postoji problem inicijalizacije, koji se rešava:

-Reset signalom koji postavlja registar u stanje ″0001″ (a ne u ″0000″)

-Kolom za samo-korekciju

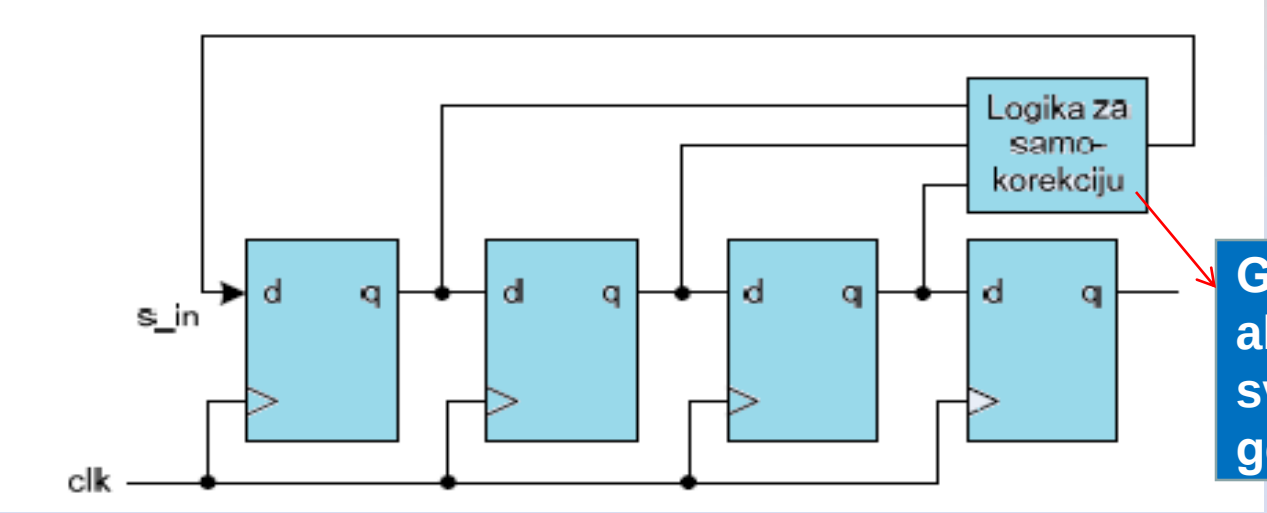
Kružni brojač – inicijalizacija reset signalom

```
ARCHITECTURE rst_arch OF para_ring_counter IS
    SIGNAL r_reg : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
    SIGNAL r_next : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
BEGIN
-- registar ----------------------------------
     PROCESS(clk, rst)
     BEGIN
          IF(rst = '1') THEN
           r reg <= (0=>'1', OTHERS =>' 0' );
           ELSIF(clk'EVENT AND clk='1') THEN
               r_reg <= r_next;
        END IF; Rotiranje
   END PROCESS;
-- logika sledeceg stanja/izlaza ---------------
  r next <= r reg(& r reg(N-1 DOWNTO 1);
  q <= r_reg;
END rst_arch;
```

Inicijalizacija

Rotiranje

Samo-korigujući kružni brojač

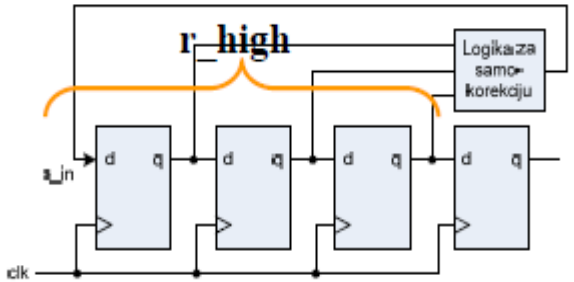


Generiše 1 samo ako na ulazu ima sve 0,inače generiše 0.

Bez obzira na početno stanje, kružni brojač ulazi u regularan način
rada nakon konačnog broja taktnih ciklusa
Mogućnost oporavka nakon greške (ako se stanje nekog flip-flopa
promeni pod uticajem smentnji)

Samokorigujući kružni brojač

```
ARCHITECTURE self_corect_arch OF para_ring_counter IS
 SIGNAL r_reg, r_next : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
 SIGNAL s_in : STD_LOGIC;
 ALIAS r_high : STD_LOGIC_VECTOR(N-2 DOWNTO 0) IS r_reg(N-1 DOWNTO 1);
BEGIN
PROCESS(clk, rst)
 BEGIN
   IF(rst = '1') THEN
      r_reg <= (OTHERS => '0');
   ELSIF(clk'EVENT AND clk='1') THEN
      r_reg <= r_next;
   END IF;
END PROCESS;
s_in <= '1' WHEN r_high = (r_high'range => '0') ELSE
        '0';
r_next <= s_in & r_high;
q <= r_reg;
END self_corect_arch;
```



GENERATE

- Konkurentna naredba
- Koristi se za kontrolisano ″generisanje″ konkurentnog kôda
- Ekvivalent sekvencijalnih naredbi *loop i if*
- Sva oblika:

FOR GENERATE (ekv. Loop) - Omogućava da se deo konkurentnog kôda ponovi više puta i na taj način kreira više instanci istog skupa naredbi dodele

IF GENERATE (ekv. If) - Omogućava uslovno kreiranje obuhvaćenog konkurentnog kôda

FOR GENERATE

- Sintaksa

labela: FOR indeks IN opseg GENERATE
(konkurentne naredbe;)
END GENERATE;

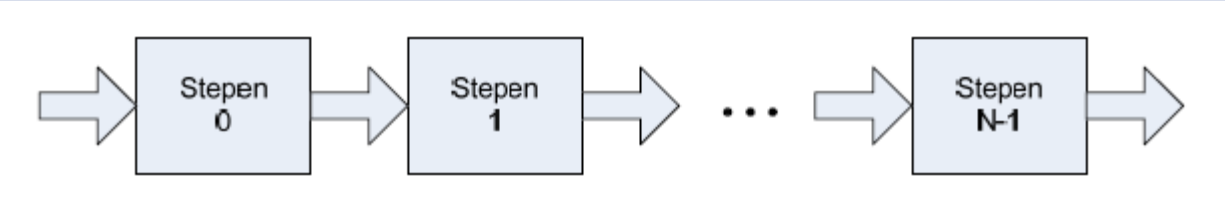
- Ponavlja obuhvaćene konkurentne naredbe zadati broj puta-jedanput za svaku vrednost indeksa iz zadatog opsega.

```
SIGNAL x: BIT_VECTOR(7 DOWNTO 0);
SIGNAL y: BIT_VECTOR(15 DOWNTO 0);
z: BIT_VECTOR(7 DOWNTO 0);
. . .
G1: FOR i IN x'RANGE GENERATE
z(i) <= x(i) z(4) <= x(4) AND y(4+8);
END GENERATE;
```

```
z(7) <= x(7) AND y(7+8);
z(6) <= x(6) AND y(6+8);
z(5) <= x(5) AND y(5+8);
z(4) <= x(4) AND y(4+8);
z(3) <= x(3) AND y(3+8);
z(2) <= x(2) AND y(2+8);
z(1) <= x(1) AND y(1+8);
z(0) <= x(0) AND y(0+8);
```

FOR GENERATE

•Tipična primena: modeliranje iterativnih hardverskih struktura

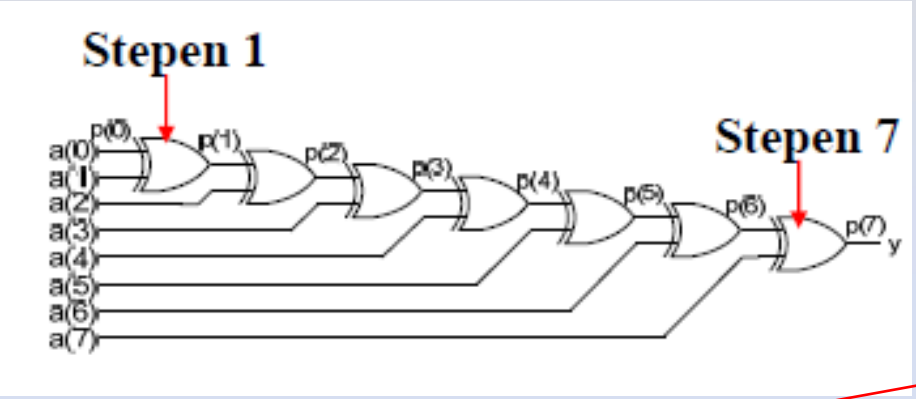


G1 : FOR I IN 0 TO N-1 GENERATE
(konkurentne naredbe;)
END GENERATE;

I - brojač stepena

Opisuje jedan stepen + način povezivanja stepena

Generator bita parnosti – opisan kao iterativno kolo



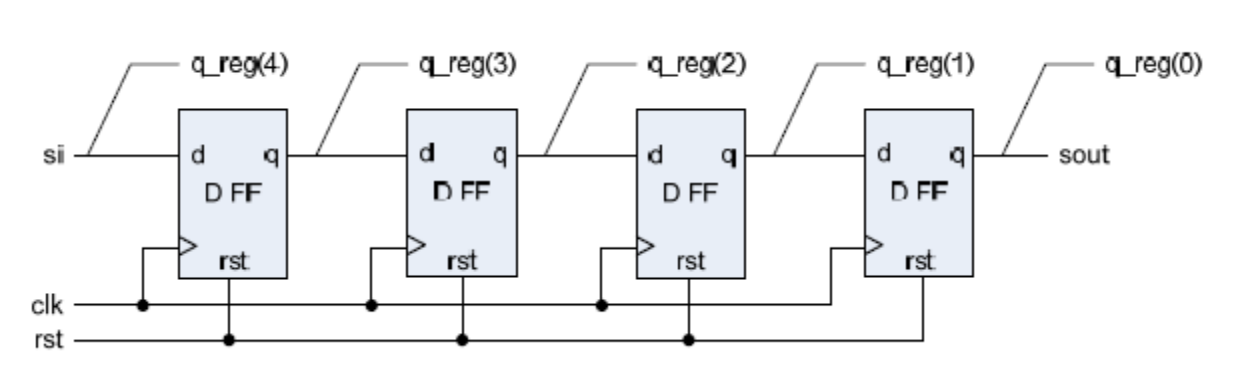
Za i-ti stepen važi:
p(i) <= a(i) XOR pi 1);

p - signal za povezivanje stepena

```
ARCHITECTURE generate_arch OF pargen IS
    SIGNAL p : STD_LOGIC_VECTOR(N-DOWNTO 0);
BEGIN
    p(0) <= a(0);
    FOR N 1 GENERATE
        p(I) <= a(I) XOR p(I-1);
END GENERATE;
y <= p (N-1);
END generate_arch;
```

˝Obod˝ iterativnog kola se obično izdvojeno tretira

Parametrizovan opis pomeračkog registra



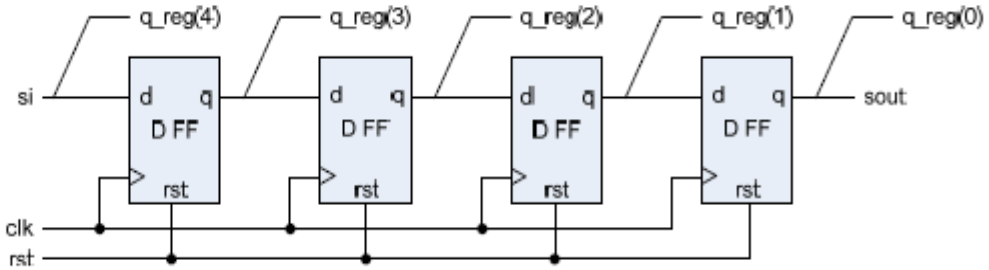
Iterativno kolo- svaki D flip-flop je jedan stepen

```
ENTITY pomreg IS
    GENERIC(N : NATURAL);
    PORT(si : IN STD_LOGIC;
         sout : OUT STD_LOGIC;
         clk rst : IN STD clk, STD_LOGIC);
END pomreg;
```

Parametrizovan opis pomeračkog registra -DFF kao proces

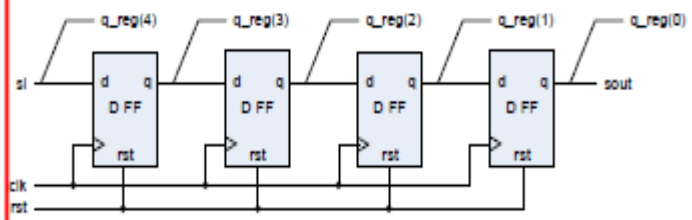
```
ARCHITECTURE gen_proc_arch OF pomreg IS
  SIGNAL q_reg : STD_LOGIC_VECTOR(N DOWNTO 0);
BEGIN
  q_reg(N) <= si;
  dff_gen: FOR I IN (N-1) DOWNTO 0 GENERATE
  -- D FF
     PROCESS(clk, rst)
     BEGIN
       IF(rst = '1') THEN
         q_reg(I) <= '0';
       ELSIF(clk'EVENT AND clk = '1') THEN
         q_reg(I) <= q_reg(I-1);
       END IF;
     END PROCESS;
  END GENERATE;
-- izlaz
 sout <= q_reg(0);
END gen_proc_arch;
```

Naredba GENERATE se obrađuje (´razmotava´) u fazi eleboracije - pre sinteze i zato opseg indeksa mora biti statički

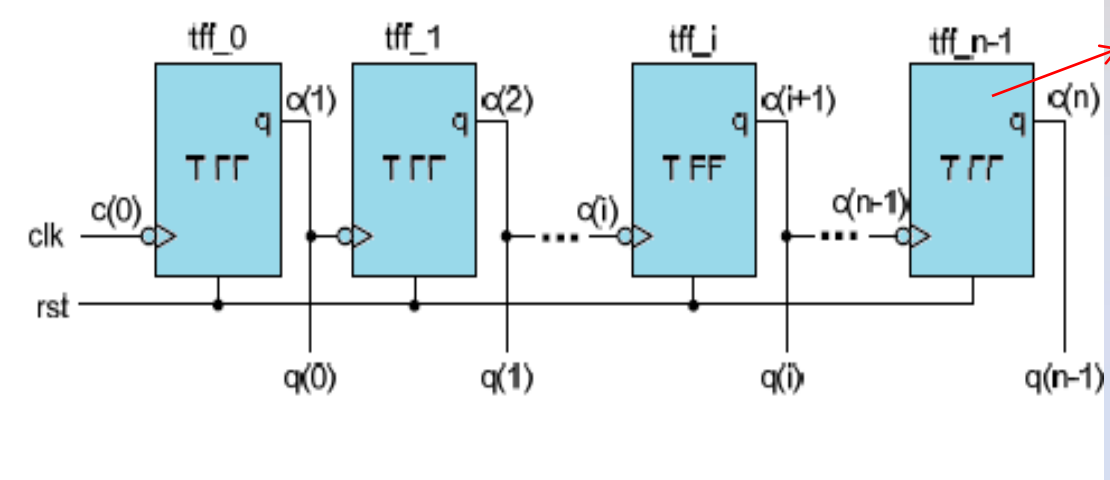


Parametrizovan opis pomeračkog registra -DFF kao komponenta

```
ARCHITECTURE gen comp arch OF pomreg IS
  COMPONENT DFF
    PORT(d : IN STD_LOGIC;
         q : OUT STD_LOGIC;
         clk, rst : IN STD_LOGIC);
  END COMPONENT;
  SIGNAL q_reg : STD_LOGIC_VECTOR(N DOWNTO 0);
BEGIN
  q_reg(N) <= si;
  dff_gen: FOR I IN (N-1) DOWNTO 0 GENERATE
    dff_I: DFF
    PORT MAP(d=>q_reg(I+1), q=>q_reg(I), clk=>clk, rst=>rst);
  END GENERATE;
sout <= q_reg(0);
END gen_proc_arch;
```



Asinhroni brojač



T flip-flop s okidanjem na opadajuću ivicu

Binarni brojač osnove 2N

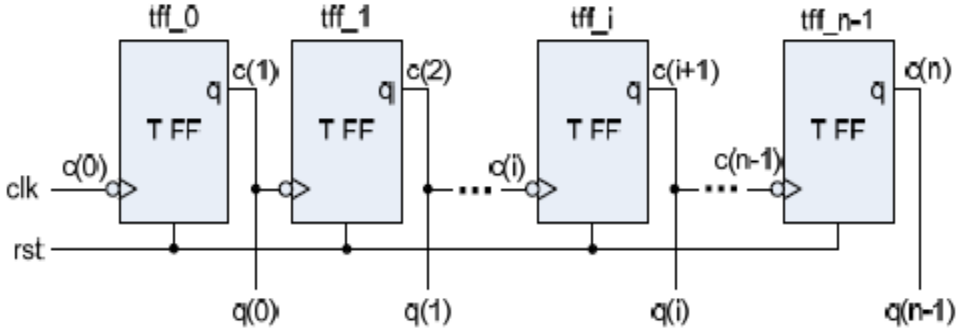
```
ENTITY tff IS
    GENERIC(N : INTEGER);
    PORT (clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END tff;
```

Asinhroni brojač - T flip-flop

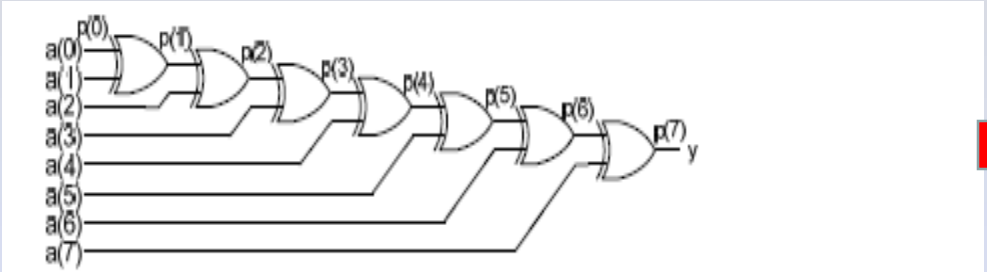
```
ENTITY tff IS
    PORT (clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END tff;
ARCHITECTURE behavior OF tff IS
  SIGNAL ff : STD_LOGIC;
BEGIN
  PROCESS (clk, rst)
  BEGIN
     IF(rst='1') THEN
        ff <= '0';
     ELSIF (clk'EVENT AND clk='0') THEN
        ff <= not ff;
     END IF;
  END PROCESS;
  q <= ff;
END behavior;
```

Asinhroni brojač - strukturni opis

```
ARCHITECTURE structural OF asyn_count IS
  COMPONENT tff IS
  PORT (clk, rst: IN STD_LOGIC;
        q: OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL c : STD_LOGIC_VECTOR(N DOWNTO 0);
BEGIN
  c(0) <= clk;
  G1:FOR i IN 0 TO N-1 GENERATE
     tffx : tff PORT MAP (clk=>c(i),rst=>rst,q=>c(i+1));
  END GENERATE;
  q <= c(N DOWNTO 1);
END structural;
```

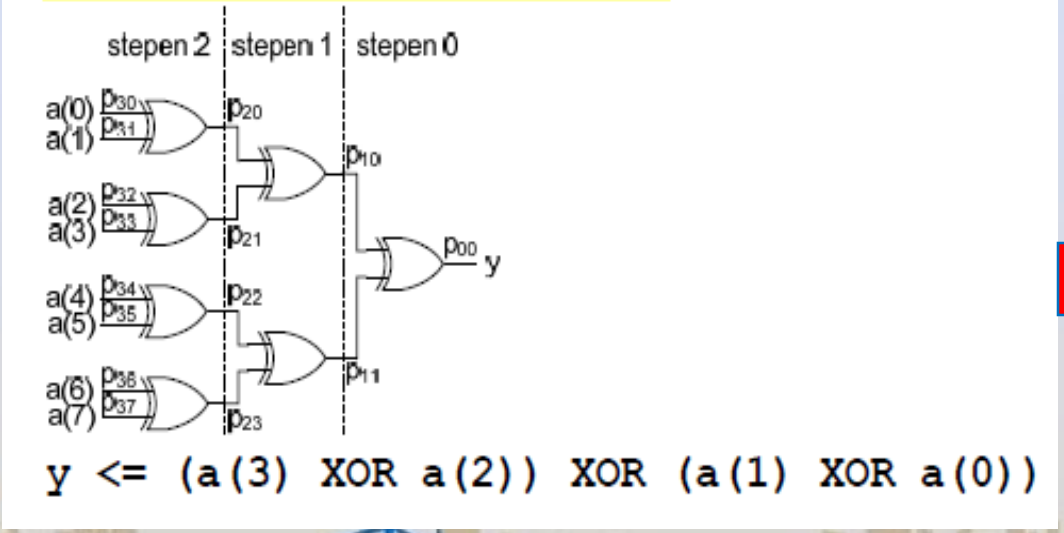


Generator bita parnosti – hijerarhijska struktura




```
y <= a(3) XOR (a(2) XOR (a(1) XOR a(0))
```

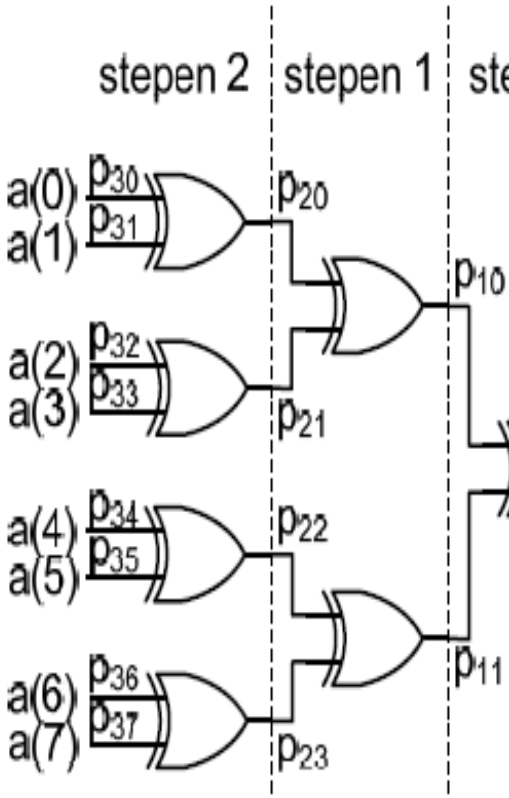
Redna realizacija




```
y <= (a(3) XOR a(2)) XOR (a(1) XOR a(0))
```

Hijerarhijska realizacija

Generator bita parnosti – hijerarhijska struktura - identifikacija stepena



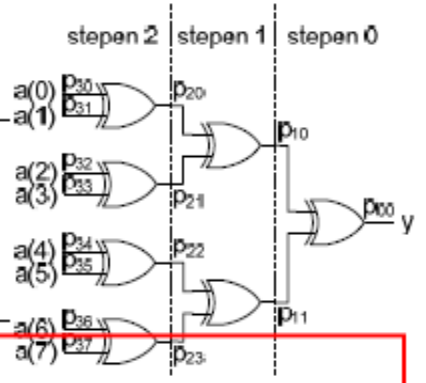
$$p_{s,r} = p_{s+1,2r} \oplus p_{s+1,2r+1}$$

```
ENTITY pargen IS
  GENERIC(N: INTEGER);
  PORT (a : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        y : OUT STD_LOGIC);
END pargen;
```

N je stepen dvojke (4, 8, 16 ,…)

Generator bita parnosti – hijerarhijska struktura

```
ARCHITECTURE tree_arch OF pargen IS
  CONSTANT STAGE : NATURAL := log2c(N);
  TYPE d2sig IS ARRAY(STAGE DOWNTO 0,N-1 DOWNTO 0) OF STD_LOGIC;
  SIGNAL p : d2signal;
BEGIN
-- preklapanje ulaznih signala ----------------
  in_gen: FOR I IN 0 TO (N-1) GENERATE
            p(STAGE,I) <= a(I);
          END GENERATE;
-- generisanje XOR polja ----------------------
  st_gen: FOR s IN (STAGE-1) DOWNTO 0 GENERATE
            xor_gen: FOR r IN 0 TO (2**s - 1) GENERATE
                       p(s,r) <= p(s+1, 2*r) XOR p(s+1, 2*r+1);
                     END GENERATE;
          END GENERATE;
 - izlaz ----------------------------------------
  y <= p(0,0);
END tree_arch;
```

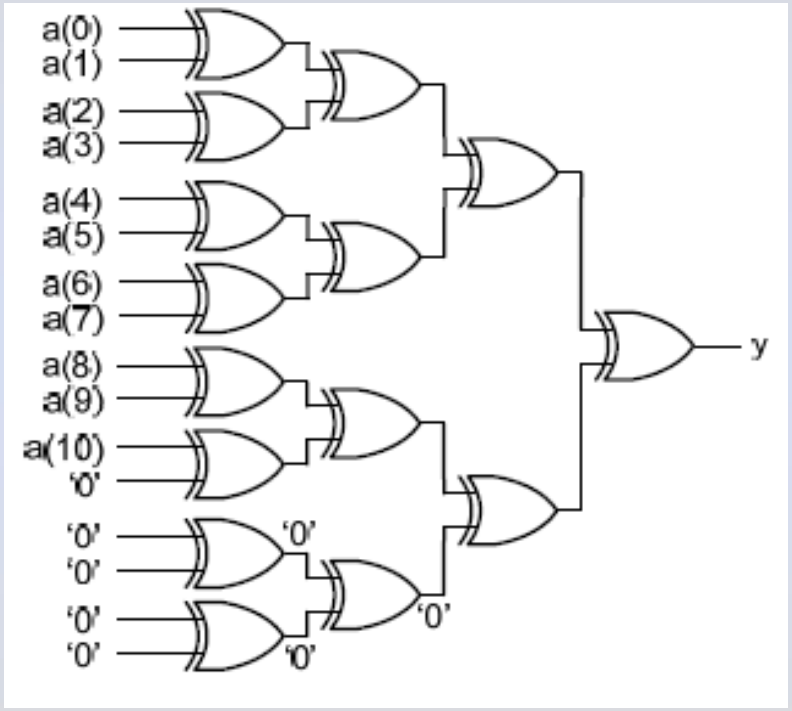


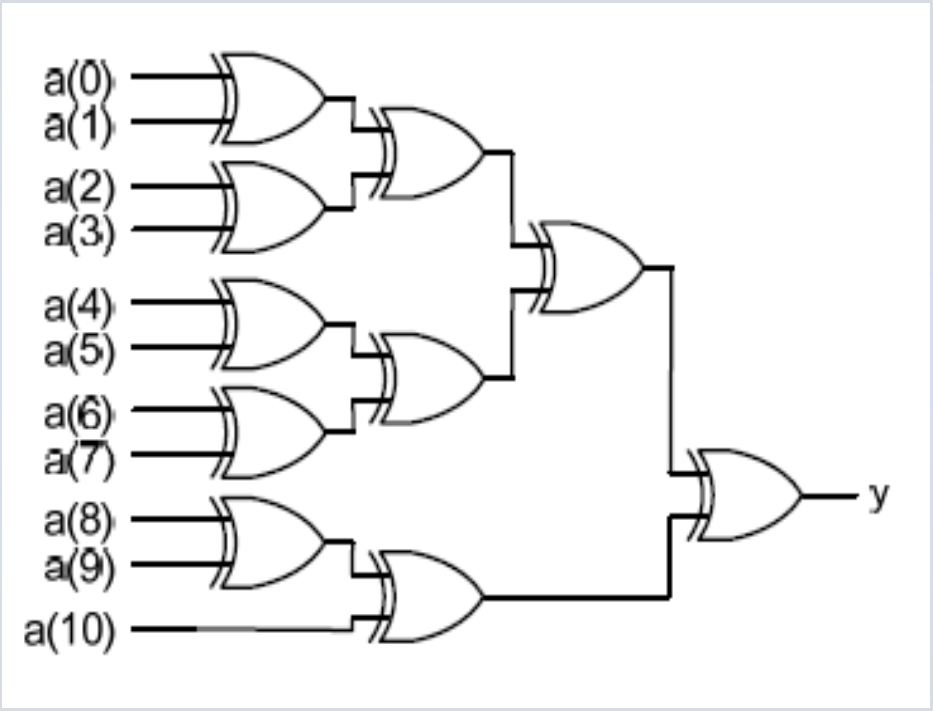
Ugnježdene FOR GENERATE naredbe

Generator bita parnosti – hijerarhijska struktura – za proizvoljan broj ulaza

- Šta će se desiti ako N nije stepen dvojke, npr. N=11?
- Generisaće se *stage = 4 stepena, a ulazni* stepen imaće *2stage-1 = 8 XOR kola sa ukupno* 16 ulaza, od kojih će 11 biti povezani s ulaznim signalima, dok će preostalih 5 ostati nepovezani.
- Problem se može rešiti tako što će se nepovezani ulazi fiksno postaviti na ´0´

Generator bita parnosti – hijerarhijska struktura - za proizvoljan broj ulaza





´0´ na ulazu ne utiču na rezultat.

Prilikom sinteze, sva redundantna XOR kola biće automatski eliminisana.

```
ARCHITECTURE tree_arch_1 OF pargen IS
  CONSTANT STAGE : NATURAL := log2c(N);
  TYPE d2signal IS ARRAY(STAGE DOWNTO 0, 2**STAGE-1 DOWNTO 0)
    OF STD_LOGIC;
  SIGNAL p : d2signal;
BEGIN
-- preklapanje ulaznih signala -------------------------
  in_gen: FOR I IN 0 TO (N-1) GENERATE
     p(STAGE,I) <= a(i);
  END GENERATE;
-- postavljanje nula -----------------------------------
  zero_gen: FOR I IN N TO (2**STAGE-1) GENERATE
     p(STAGE,I) <= '0';
  END GENERATE;
-- generisanje XOR polja -------------------------------
  st_gen: FOR s IN (STAGE-1) DOWNTO 0 GENERATE
     xor_gen: FOR r IN 0 TO (2**s - 1) GENERATE
        p(s,r) <= p(s+1, 2*r) XOR p(s+1, 2*r+1);
     END GENERATE;
  END GENERATE;
-- izlaz -----------------------------------------------
  y <= p(0,0);
END tree_arch_1;
```

IF GENERATE

- Sintaksa

labela: IF uslov GENERATE

(konkurentne naredbe;)

END GENERATE;

- **Ako je uslov tačan, hardver opisan konkurentnim naredbama se uključuje u implementaciju, inače se izostavlja**

Koristi se za realizaciju funkcionalnih parametara.

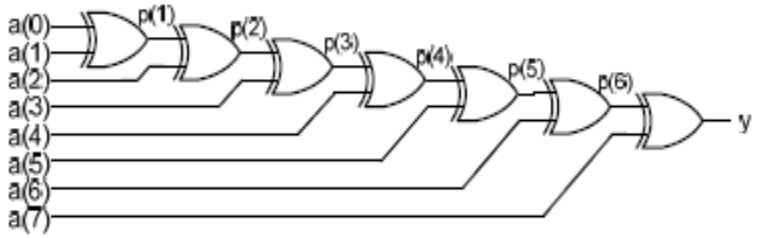
U kodu za sintezu, uslov mora biti statički

Uslov se najčešće izražava posredstvom generičkih parametara.

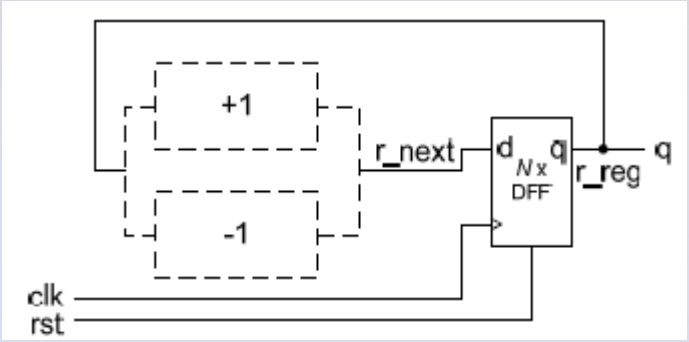
Obrađuje se tokom eleboracije tako što se, pre sinteze, odstranjuje kod iz IF GENERATE naredbi s netačnim uslovima.

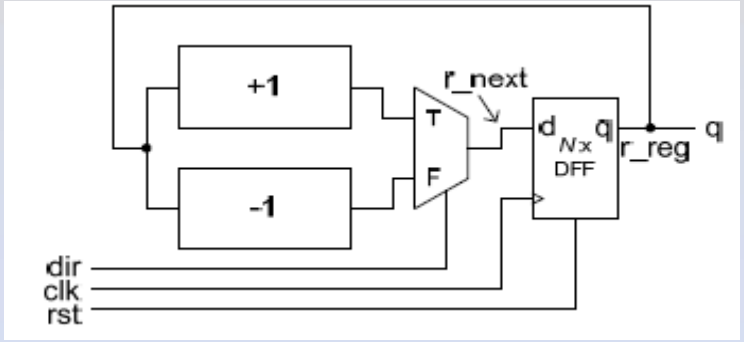
Generator bita parnosti

```
ARCHITECTURE gen_if_arch OF pargen IS
3     SIGNAL p : STD_LOGIC_VECTOR(N-2 DOWNTO 1);
4   BEGIN
5     xor_gen: FOR I IN 1 TO (N-1) GENERATE
6        -- krajnji levi stepen
7        left_gen: IF I = 1 GENERATE
8           p(I) <= a(I) XOR a(0);
9        END GENERATE;
10       -- sredisnji stepeni
11       middle_gen: IF (I > 1) AND (I < (N-1) GENERATE
12          p(I) <= a(I) XOR p(I-1);
13       END GENERATE;
14       -- krajnji desni stepen
15       right_gen: IF I = (N-1) GENERATE
16          y <= a(I) XOR p(I-1);
17       END GENERATE;
18     END GENERATE;
19 END gen_if_arch;
```



Up-or-Down brojač



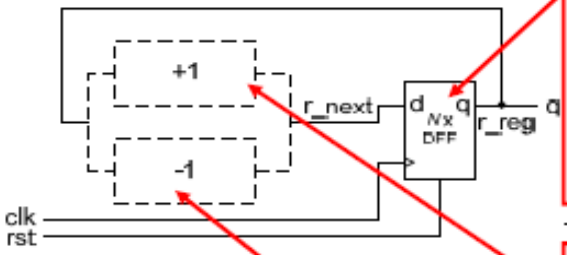


Komponenta koja može da se ***instancirati za*** rad u jednom od dva režima: brojanje naviše (*Up) ili brojanje naniže (Down)*

Obostrani brojač - mogućnost izbora smera brojanja (ulaz *dir)*

```
ENTITY up_or_down_counter IS
  GENERIC(N : NATURAL;
          UP : NATURAL);
  PORT (q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        clk, rst : IN STD_LOGIC);
END up_or_down_counter;
```

UP = 1 bira brojač naviše
UP = 0 bira brojač naniže




```
ARCHITECTURE arch OF up_or_down_counter IS
  SIGNAL r_reg : UNSIGNED(N-1 DOWNTO 0);
  SIGNAL r_next : UNSIGNED(N-1 DOWNTO 0);
BEGIN
- registar ------------------------------
PROCESS(clk, rst)
BEGIN
  IF(rst = '1') THEN
     r_reg <= (OTHERS => '0');
  ELSIF(clk'EVENT AND clk='1') THEN
     r_reg <= r_next;
  END IF;
END PROCESS;
-- logika sledeceg stanja - inkrementer ---
inc_gen: IF UP = 1 GENERATE
  r_next <= r_reg + 1;
END GENERATE;
 -- logika sledeceg stanja - dekremeter ---
dec_gen: IF UP /= 1 GENERATE
  r_next <= r_reg - 1;
END GENERATE;
 -- logika izlaza ------------------------
 q <= STD_LOGIC_VECTOR(r_reg);
END arch;
```

Up-or-Down brojač

- **Instanciranje 8-bitnog Up brojača**

```
count8up: up_or_down_counter
    GENERIC MAP(N=>8, UP=>1);
    PORT MAP(clk=>clk, rst=>rst, q=>q);
```

Logika koja se odnosi na biće Down eliminisana tokom faze eleboracije

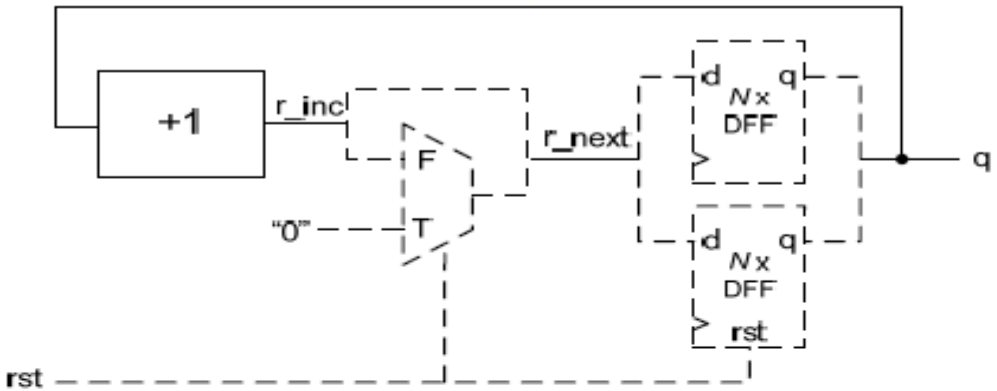
- **Isti efekat kao da se ulaz za izbor smera brojanja, dir, Up-and-Down brojač postavi na fiksno ´1´**

```
count8up: up_and_down_counter
    GENERIC MAP(N=>8);
    PORT MAP (clk=>clk, rst=>rst, dir=>'1', q=>q);
```

Logika koja se odnosi na Down biće eliminasana tokom faze sinteze

Sinhroni ili asinhroni reset

- **Brojač sa sinhronim ili asinhronim resetom**




```
ENTITY syn_or_asyn_counter IS
   GENERIC(N : NATURAL;
           SYNC : NATURAL);
   PORT (q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
         clk, rst : IN STD_LOGIC);
END syn_or_asyn_counter;
```

SYNC = 1 - sinhroni reset
SYNC ≠ 1 - asinhroni reset

FOR LOOP

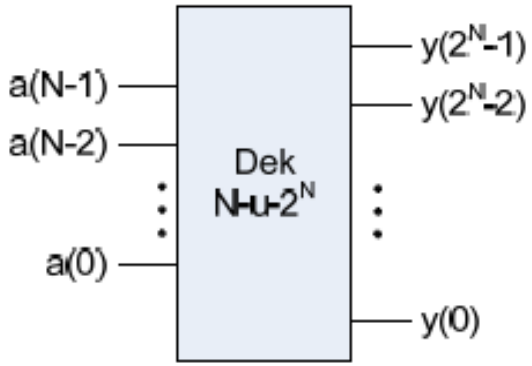
- **Sintaksa:**

```
FOR index IN opseg LOOP
    sekvencijalne naredbe;
END LOOP;
```

- **Koristi se isključivo u sekvencijalnom kodu**

- **U kodu za sintezu opseg mora biti statički**

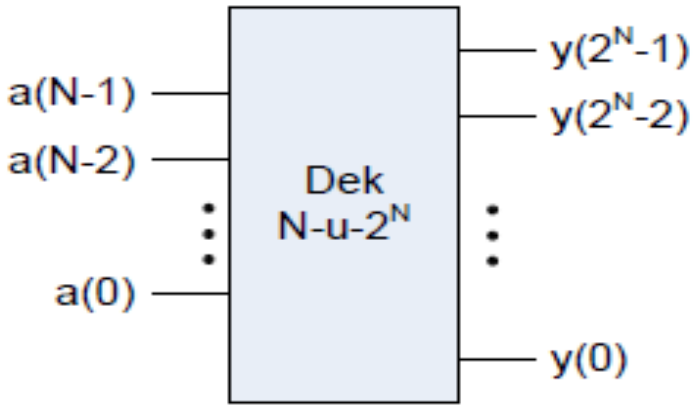
Binarni dekoder - parametrizovan opis



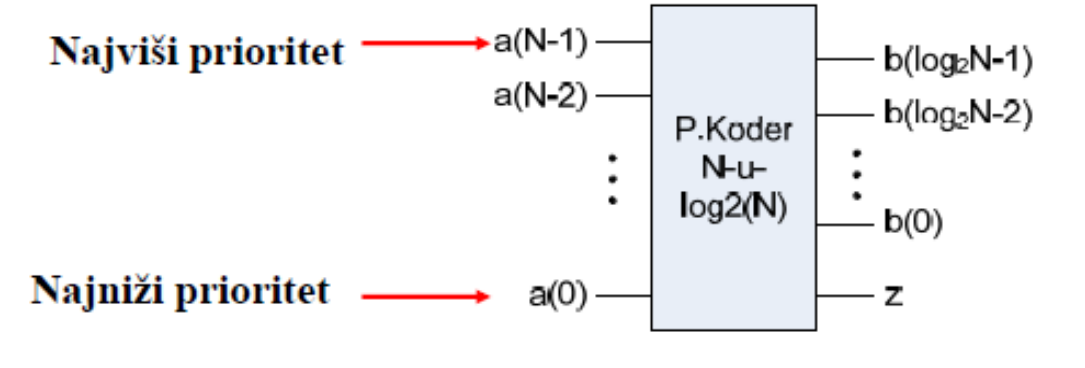

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY bin_decoder IS
  GENERIC(N : NATURAL);
  PORT(a : IN  STD_LOGIC_VECTOR (N-1 DOWNTO 0);
       y : OUT  STD_LOGIC_VECTOR (2**N-1 DOWNTO 0));
END bin_decoder;
```

Binarni dekoder - parametrizovan opis

```
ARCHITECTURE loop_arch OF bin_decoder IS
BEGIN
  PROCESS(a)
  BEGIN
    FOR I IN 0 TO 2**N-1 LOOP
      IF(I = TO_INTEGER(UNSIGNED(a))) THEN
        y(I) <= '1';
      ELSE
        y(I) <= '0';
      END IF;
    END LOOP;
  END PROCESS;
END loop_arch;
```



Prioritetni koder - parametrizovan opis



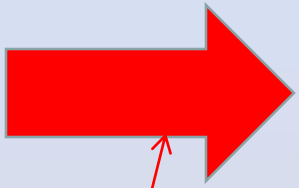

```
ENTITY pencoder IS
    GENERIC(N : NATURAL);
    PORT( a : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
          b : OUT STD_LOGIC_VECTOR (log2c(N)-1 downto 0);
          z : OUT STD_LOGIC);
END pencoder;
```

```
ARCHITECTURE loop_arch OF pencoder IS
  CONSTANT M : NATURAL := log2c(N);
  SIGNAL p : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
BEGIN
-- prioritetno kodiranje -------------
  PROCESS(a)
  BEGIN
    b <= (OTHERS => '0');
    FOR I IN 0 TO N-1 LOOP
      IF(a(i) = '1') THEN
        b <= STD_LOGIC_VECTOR(TO_UNSIGNED(I,M));
      END IF;
    END LOOP;
  END PROCESS;
-- N-to ulazno ILI kolo --------------
  PROCESS(a, p)
  BEGIN
    p(0) <= a(0);
    FOR I IN 1 TO (N-1) LOOP
      p(i) <= a(i) OR p(I-1);
    END LOOP;
  END PROCESS;
  z <= p(N-1);
END loop_arch;
```

Prioritetni koder - parametrizovan opis- sinteza

Nakon razmotavanja petlje (n=4)

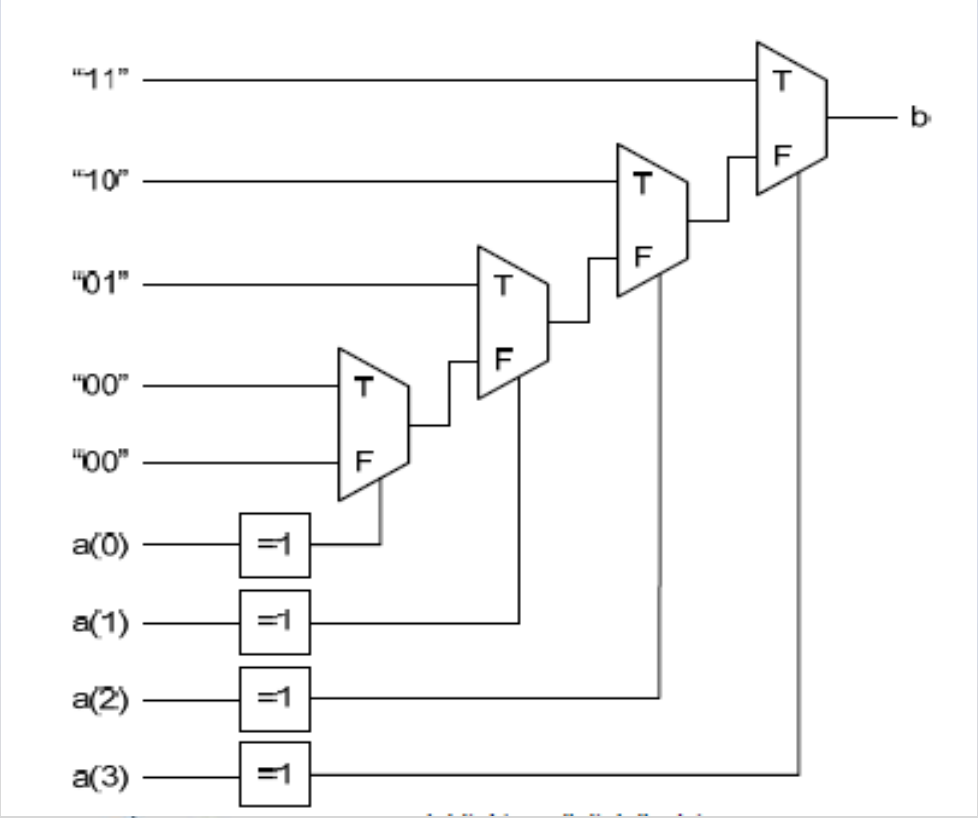
```
b <= "00";
IF (a(0) = '1') THEN
  b <= "00";
END IF;
IF (a(1) = '1') THEN
  b <= "01";
END IF;
IF (a(2) = '1') THEN
  b <= "10";
END IF;
IF (a(3) = '1') THEN
  b <= "11";
END IF;
```



```
IF (a(3) = '1') THEN
  b <= "11";
ELSIF (a(2) = '1') THEN
  b <= "10";
ELSIF (a(1) = '1') THEN
  b <= "01";
ELSIF (a(0) = '1') THEN
  b <= "00";
ELSE
  b <= "00";
END IF;
```

U procesu, konačna vrednost signala, određena je poslednjom dodelom

Prioritetni koder - parametrizovan opis- sinteza



Višeulazno I kolo – parametrizovan opis I konceptualna implementacija

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY andN IS
GENERIC(N : NATURAL);
PORT( a : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
y : OUT STD_LOGIC);
END andN;
ARCHITECTURE loop_arch OF andN IS
BEGIN
PROCESS(a)
VARIABLE v : STD_LOGIC;
BEGIN
v := a(0);
FOR I IN 1 TO (N-1) LOOP
v := v AND a(i);
END LOOP;
y <= v;
END PROCESS;
END loop_arch;
```

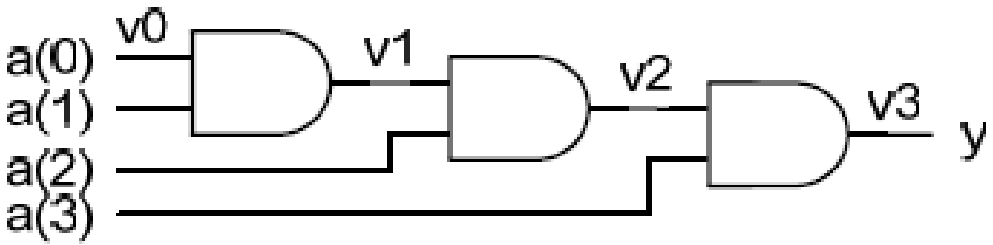
Višeulazno I kolo -parametrizovan opis i konceptualna implementacija

Razmotavanje petlje (n=4)

```
v := a(0);
v := a(1) AND v;
v := a(2) AND v;
v := a(3) AND v;
y <= v;
```

Preimenovanje promenljivih

```
v0 := a(0);
v1 := a(1) AND v0;
v2 := a(2) AND v1;
v3 := a(3) AND v2;
y <= v3;
```



Brojač jedinica - parametrizovan opis i konceptualna implementacija

- Kombinaciono kolo koje prebrojava 1-ce u ulaznom višebitnom signalu

```
ENTITY once_counter IS
    GENERIC(N : NATURAL);
    PORT( a : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
          b : OUT STD_LOGIC_VECTOR (log2c(N)-1 DOWNTO 0));
END once_counter
```

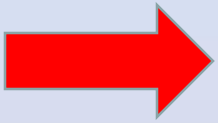
Brojač jedinica - parametrizovan opis i konceptualna implementacija

```
ARCHITECTURE loop_arch OF once_counter IS
BEGIN
    PROCESS(a)
        VARIABLE sum : UNSIGNED(log2c(N)-1 DOWNTO 0);
BEGIN
    sum := (OTHERS => '0');
    FOR I IN 0 TO N-1 LOOP
        IF (a(i) = '1') THEN
          sum := sum + 1;END IF;
        END LOOP;
        b <= STD_LOGIC_VECTOR(sum);
    END PROCESS;
END loop_arch;
```

Brojač jedinica - parametrizovan opis i konceptualna implementacija

Razmotavanje petlje (n=3)

```
sum := 0;
IF (a(0) = '1') THEN
  sum := sum + 1;
END IF;
IF (a(1) = '1') THEN
  sum := sum + 1;
END IF;
IF (a(2) = '1') THEN
  sum := sum + 1;
END IF;
b <= sum;
```



Preimenovanje promenljivih

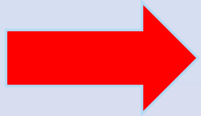
```
sum0 := 0;
IF (a(0) = '1') THEN
  sum1 := sum0 + 1;
END IF;
IF (a(1) = '1') THEN
  sum2 := sum1 + 1;
END IF;
IF (a(2) = '1') THEN
  sum3 := sum2 + 1;
END IF;
b <= sum3;
```

Tekuća vrednost sume se ne prenosi korektno iz iteracije u iteraciju.

Brojač jedinica - parametrizovan opis i konceptualna implementacija

Razmotavanje petlje (n=3)

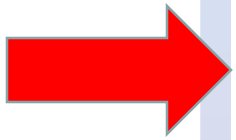
```
sum := 0;
IF (a(0) = '1') THEN
  sum := sum + 1;
END IF;
IF (a(1) = '1') THEN
  sum := sum + 1;
END IF;
IF (a(2) = '1') THEN
  sum := sum + 1;
END IF;
b <= sum;
```

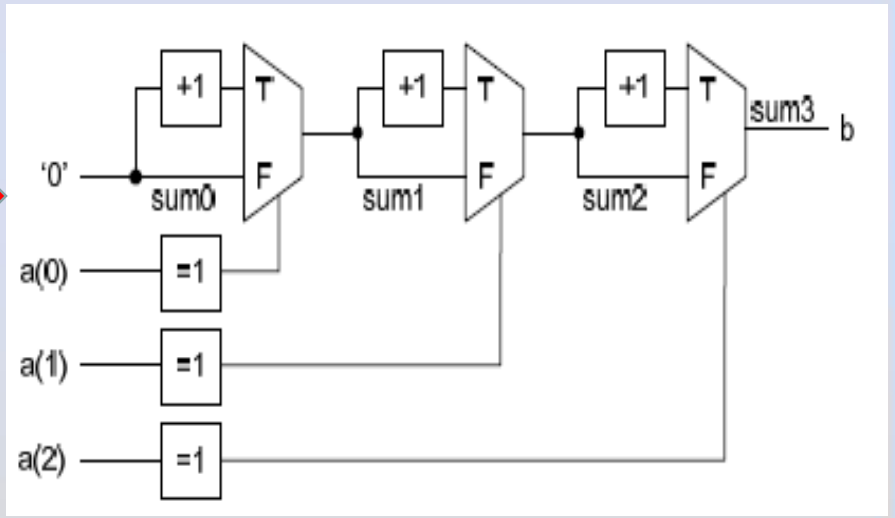


```
sum0 := 0;
IF (a(0) = '1') THEN
  sum1 := sum0 + 1;
ELSE
  sum1 := sum0;
END IF;
IF (a(1) = '1') THEN
  sum2 := sum1 + 1;
ELSE
  sum2 := sum1;
END IF;
IF (a(2) = '1') THEN
  sum3 := sum2 + 1;
ELSE
  sum3 := sum2;
END IF;
b <= sum3;
```

Brojač jedinica – parametrizovan opis i konceptualna implementacija

```
sum0 := 0;
IF (a(0) = '1') THEN
    sum1 := sum0 + 1;
ELSE
    sum1 := sum0;
END IF;
IF (a(1) = '1') THEN
   sum2 := sum1 + 1;
ELSE
    sum2 := sum1;
END IF;
IF (a(2) = '1') THEN
   sum3 := sum2 + 1;
ELSE
    sum3 := sum2;
END IF;
b <= sum3;
```





EXIT

EXIT WHEN uslov;

- Piše se unutar LOOP naredbe
- Momentalno napuštanje petlje ako je uslov tačan
- Može i bez uslova:

```
LOOP
    IF (uslov) THEN
    . . .
      EXIT;
  ELSE
  . . .
END LOOP;
```

Višeulazno I kolo – realizacija pomoću naredbe *˝for loop -exite - and loop˝*

```
ARCHITECTURE loop_exit_arch OF andN IS
BEGIN
     PROCESS(a)
         VARIABLE v : STD_LOGIC;
      BEGIN
        v := ´1´; -- podrazumevana vrednost
        FOR I IN 0 TO (N-1) LOOP
         IF a(I) = ´1´ THEN
             v :=´0´ ;
             EXIT;
        END IF;
     END LOOP;
     y <= v;
  END PROCESS;
END loop_exit_arch;
```

Dovoljno je da na ulazu postoji samo jedna 0, pa da rezultat AND operacije bude 0

Brojač vodećih nula – realizacija pomoću *for loop petlje sa exit naredbom*

```
ARCHITECTURE loop_exit_arch OF zero_counter IS
BEGIN
    PROCESS(a)
        VARIABLE sum: UNSIGNED(log2(N)-1 DOWNTO 0);
    BEGIN
      sum := (OTHERS => '0');
      FOR I IN N-1 DOWNTO 0 LOOP
           IF a(I) = '1' THEN
              EXIT;
           ELSE;
                sum := sum + 1;
           END IF;
        END LOOP;
        z <= STD_LOGIC_VECTOR(sum);
      END PROCESS;
END loop_exit_arch;
```

Sa koliko 0 počinje ulazni vektor a?

Brojanje se završava ako se naiđe na 1

NEXT

NEXT WHEN uslov;

- Piše se unutar LOOP naredbe
- Ako je uslov tačan, momentalno se prelazi na sledeću iteraciju petlje

```
FOR ... LOOP FOR ... LOOP
    sekvencijalna naredba 1;
    NEXT WHEN uslov;
    sekvencijalna naredba 2;
END LOOP;
```

→

```
FOR ... LOOP FOR ... LOOP
    sekvencijalna naredba 1;
    IF(NOT uslov) THEN
      sekvencijalna naredba 2;
    END IF;
END LOOP;
```

Brojač jedinica – realizacije pomoću naredbe *next*

```
ARCHITECTURE loop_next_arch OF zero_counter IS
BEGIN
    PROCESS(a)
        VARIABLE sum: UNSIGNED(log2(N)-1 DOWNTO 0);
    BEGIN
        sum := (OTHERS => '0');
        FOR I IN 0 TO N-1 LOOP
            NEXT WHEN a(I)='0';
            sum := sum + 1;
        END LOOP;
        b <= STD_LOGIC_VECTOR(sum);
    END PROCESS;
END loop_next_arch;
```