

VHDL

Jezik za definiciju, opis i simulaciju digitalnih sistema

VHDL ?

Very Hard Difficult Language

u slobodnom prevodu: Veoma težak i problematičan jezik

VHSIC *Very High Speed Integrated Circuit*

H *Hardware*

D *Description*

L *Language*

Istorija VHDL-a

- Specifikacija je predložena 1980(81) od strane američkog ministarstva odbrane radi dokumentovanja ponašanja integrisanih kola koje su poručivali sa opremom od drugih proizvođača
- 1983 Intermetrics, IBM i Texas Instruments dobijaju grant da razviju VHDL
- 1985 finalna verzija jezika je prezentovana

Istorija VHDL-a

- Praktično, VHDL je bio alternativa za veliku i kompleksnu korisničku dokumentaciju
- Veoma brzo se rodila ideja za simulaciju ovakve vrste dokumentacije
- Sledeći korak je bio razvoj alata za sintezu integrisanih kola opisanih u VHDL-u

Razvoj i standardizacija

- Razvijen od strane IBM, Texas Instruments i Intermetrics početkom 80-tih godina 20-tog veka, na osnovu sintakse iz ADA programskog jezika
- Standardizacija
 - VHDL standard 1987 (IEEE 1076-1987) osnovna verzija
 - VHDL standard 1993 (IEEE 1076-1993) proširena i unapređena verzija, danas najčešće korišćena
 - VHDL standard 2000 (IEEE 1076-2000)
 - VHDL standard 2002 (IEEE 1076-2002) male izmene i pokušaj uvođenja zaštićenih tipova (slično konceptu klasa iz C++)
 - Jun 2006, izašla treća preliminarina verzija (engl. *Draft 3.0*); VHDL-2006
 - Februar 2008, prihvaćena četvrta verzija (engl. *Draft 4.0*); VHDL-2008
 - Septembar 2008, konačni predlog standarda predat IEEE na reviziju.
 - VHDL standard 2008 (IEEE 1076-2008)
 - VHDL standard 2019 (IEEE 1076-2019)

Proširenja osnovnih standarda

- Postoje dodatni standardizovani paketi koji proširuju osnovnu funkciju VHDL-a
 - IEEE 1164 tipovi podataka sa višestrukom vrednošću
signal može dobiti 9 različitih vrednosti
ako je definisan tipom `std_ulogic`
 - IEEE 1076.3 proširenje numeričkih vrednosti
 - IEEE 1076.4 opis vremenskog ponašanja

Tradicionalni programski nasuprot HDL jezika

- Proceduralni jezici (npr. C++) omogućuju programski opis
 - Matematičkih izraza
 - Manipulaciju sa podacima
 - Izvršenje na **unapred definisanoj fizičkoj arhitekturi**
- Jezici za opis fizičke arhitekture opisuju celokupan sistem, uključujući i fizičku arhitekturu za izvršenje
 - Model ponašanja (engl. *Behavioral*)
Koja je ulaga sistema? Sta on radi?
 - Strukturni model (engl. *Structural*)
Od čega se sistem sastoji?
 - Funkcionalne osobine (engl. *Functional properties*)
Kako se sprežemo sa sistemom?
 - Fizičke osobine (engl. *Physical properties*)
Koliko je sistem brz?
Koja je njegova veličina izražena u mm² ili broju logičkih kapija?

Namena HDL-a

- Opis sistema može biti na različitim nivoima apstrakcije
 - Prekidački nivo (engl. *Switch level*), opis prekidačkih osobina tranzistora
 - RTL (engl. *Register Transfer Language*) nivo, modelovanje kombinacionih funkcija i sekvencijalnih logičkih elemenata
 - Strukturalni nivo opis hijerarhijskog povezivanja sistema
- HDL opis se može iskoristiti za:
 - Simulaciju verifikacija sistema sa analizom performansi
 - Sintezu prvi korak u proizvodnji fizičke arhitekture
(praktično, namenskog integrisanog kola)

Motivacija

- U ranim fazama, digitalna kola su projektovana ručno na nivou električnih odnosno logičkih šema.
- Osnovni SW alati bili su poput “P spice”-a.
- Sa porastom gustine pakovanja komponenata u integrisna kola, ograničenja tradicionalnih šematskih metoda su postala usko grlo.
- Otvorila se potreba za formalizacijom opisa kompleksnih digitalnih sistema, metodologijom koja “preskače” analizu i sintezu na nivou logičkih šema.

Motivacija

- VHDL kao jezik za opis HW sintaksno liči na C, ali je striktno orijentisan na opisivanje WH struktura i njihovog ponašanja (behavioral)
- Ključna uloga VHDL-a je da zameni šematski opis HW
- VHDL stupa na tržište u periodu opšte ekspanzije u potreba računara u projektovanju

Glavna prednost VHDL-a

- Pretpostavimo da treba generisati množač
- 10b X 10b daje 20b rezultat
- Implementacija u FPGA troši 107 LUT ~ 400 log. kapija
- 12b X 12b daje 24b rezultat
- Implementacija u FPGA troši 153 LUT ~ 600 log. kapija
- ...
- 16b X 16b daje 32b rezultat
- Implementacija u FPGA troši 271 LUT ~ 1100 log. kapija

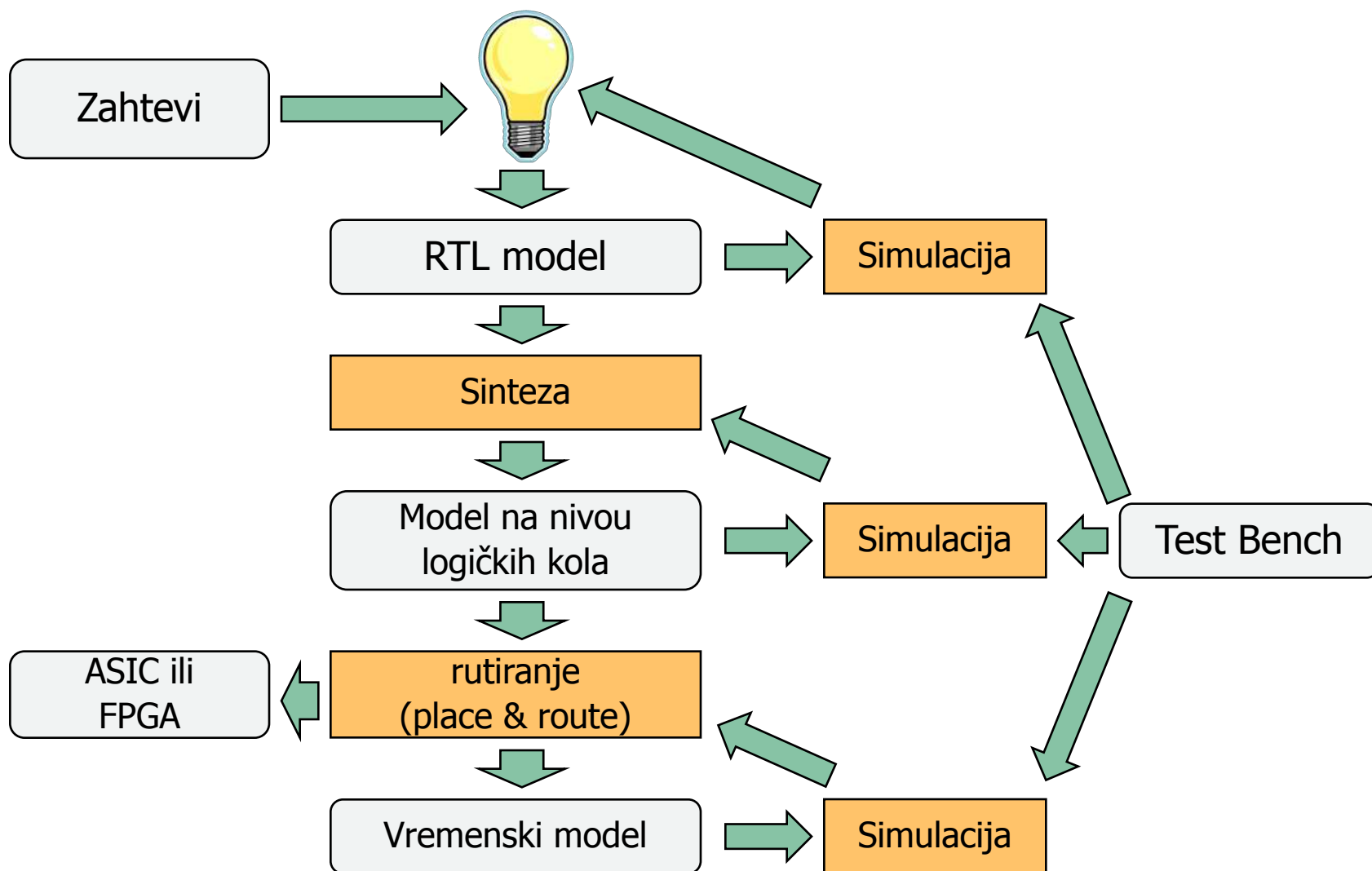
Glavna prednost VHDL-a

- Umeto velikog broja logičkih kapija, množač u VHDL-u se opisuje:

$oY \leq iA * iB;$

- Uporedimo preglednost gornjeg izraza i logičke šeme od više stotina kapija...

Osnovni tok projektovanja digitalnog sistema



Osnovne osobine VHDL-a

- Posедуje konstrukcije za opis paralelizma koji je prisutan u fizičkoj arhitekturi (engl. *Hardware*)
- Implicitno podržava vremenski model ponašanja sistema
- Strogo tipski jezik,
ali ne pravi razliku između malih i velikih slova (nije 'case sensitive')
- Jednostavan opis hijerarhije (strukture) sistema

Osobine VHDL-a

- Dozvoljava dizajneru rad na različitim nivoima apstrakcije
- Opis ponašanja (behavioral)
- RTL
- Bulove funkcije
- Logičke kapije

Prednosti VHDL-a

- Obezbeđuje tehnološku nezavisnost koda u odnosu na tehnologiju implementacije FPGA ili ASIC
- Top-down i bottom-up pristupi su podržani
- Kao standardizovan jezik obezbeđuje prenosivost koda između alata za sintezu i alata za simulaciju
- Obezbeđena je i prenosivost između alata različitih proizvođača
- VHDL je jezik otvorenog standarda (Open standard language)

Osnovne sekcije VHDL koda

- Deklaracija biblioteka (LIBRARY DECLARATIONS)
 - Sadrži listu svih biblioteka korišćenih u dizajnu
- Entitet sa opisom portova (ENTITY)
 - Specificira ulazno/izlazne portove bloka
- Opis arhitekture (ARCHITECTURE)
 - Sadrži VHDL kod koji opisuje funkcionalnost bloka

Primer kompletnog VHDL modula

```
-- ovo je VHDL komentar
-- postoje samo linijski komentari
-- blokovski komentari nisu podržani (npr. /* ... */)

-- uključujemo podršku za std_logic tip iz IEEE biblioteke
library IEEE;
use IEEE.std_logic_1164.all;

-- ovo je opis ENTITETA
entity ANDGATE is
port ( IN1  : in  std_logic;  -- prvi ulazni port
        IN2  : in  std_logic;  -- drugi ulazni port
        OUT1 : out std_logic    -- izlazna vrednost, logičko I između
        );                      -- vrednosti signala na ulaznim portovima
end ANDGATE;

-- ovo je opis ARHITEKTURE
architecture RTL of ANDGATE is
begin
    OUT1 <= IN1 and IN2;        -- definicaja izlazne vrednosti na osnovu
    ulaza
end RTL;
```

Struktura VHDL modula

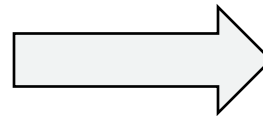
modul

entitet

Deklaracija
sprežnog sistema

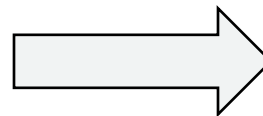
arhitektura

Funkcionalna
definicija
sistema



Opisuje ulaze i izlaze sistema.

Može da sadrži i parametrizovane vrednosti.



Opisuje sadržaj entiteta (modula).

Definiše ponašanje tj. Funkciju sistema.

Režimi (modovi) VHDL portova

In:

Podaci teku samo u entitet.

Izvor režima porta je spolja u odnosu na entitet, koristi se prvenstveno za ulaze takta, kontrolne ulaze (kao što su `clock`, `load` i `enable`).

Out:

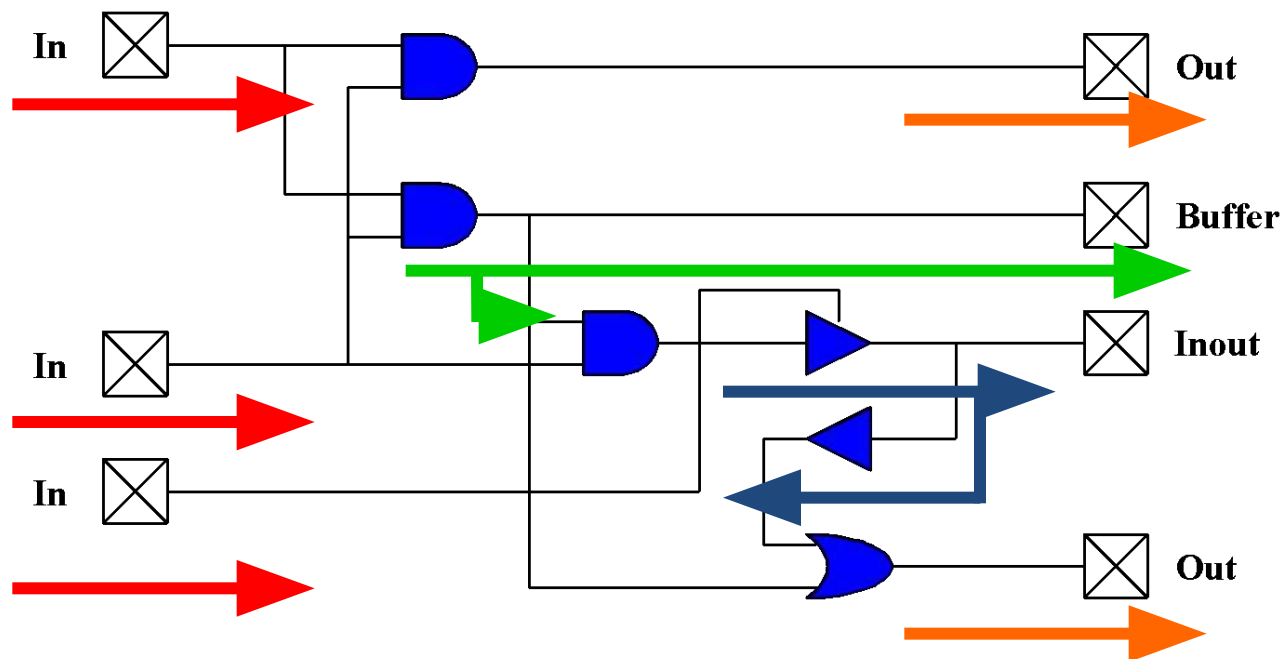
Podaci teku samo iz svog izvora na izlazni port entiteta.

Buffer:

Za unutrašnju povratnu spregu nekog izlaznog podatka (tj. da se izlazni port takođe koristi kao izvor unutar arhitekture)

Inout:

Za dvosmerne signale, mora se deklarirati port u `inout` režimu, što dozvoljava protok podataka unutar ili izvan entiteta.



Arhitektura

- Sastoji se iz niza konkurentnih iskaza koji se izvršavaju potpuno paralelno
- Moguće je
 - Koristiti bulove jednačine
 - Opis toka podatka
 - Procesi
 - Instance drugih entiteta

```
-- primer 1: bulove jednačine
X <= (A and S) or (B and not(S));
```

```
-- primer 2: opis ponašanja
X <= A when S = '1' else B;
```

```
-- primer 3: opis ponašanja
with S select
    X <= A when '1'
    B when others;
```

```
-- primer 4:
-- proces sa uslovnim iskazom case
process (A,B,S)
begin
    case S is
        when '1'      => X <= A;
        when others => X <= B;
    end case;
end process;
```

```
-- primer 5:
-- proces sa uslovnim iskazom if
process (A,B,S)
begin
    if S = '1' then
        X <= A;
    else
        X <= B;
    end if;
end process;
```

VHDL identifikatori

- Osnovni identifikatori su sačinjeni od alfabetskih, numeričkih i/ili podvučenih znakova. Primenjuju se sledeća pravila:
 - Prvi znak mora biti slovo
 - Poslednji znak ne može biti donja crtica
 - Crtice u nizu nisu dozvoljene
- VHDL rezervisane reči se ne mogu koristiti kao identifikatori. Velika i mala slova su jednaka kada se koriste u identifikatorima. Sledeće je jednako:
`txclk, Txclk, TXCLK, TxClk`
- Slede legalni identifikatori:
`tx_clk, Three_State_Enable, sel7D, HIT_1124`
- Dole navedeni *nisu* legalni identifikatori:
 - `tx_clk` -- identifikator mora početi slovom
 - `8B10B` -- identifikator mora početi slovom
 - `large#number` -- samo slova, cifre i crtice
 - `link_bar` -- dve crtice u nizu nisu dozvoljene
 - `select` -- ključne reči (rezervisane reči) se ne mogu koristiti kao identifikatori
 - `rx_clk_` -- poslednji karakter ne može biti crtica (podvlačenje)

Objekti podataka

- **Konstante**: sadrže vrednost koja se ne može menjati unutar opisa podataka

```
constant cWIDTH : integer := 8;
```

- **Signali**: povezuju komponente unutar arhitekture

```
signal sCNT : std_logic_vector (3 downto 0);
```

<= VHDL simbol za dodelu vrednosti signalu
Simulator dodeljuje vrednosti svim signalima
u istom vremenskom trenutku
(odložena dodela pre ponovnog aktiviranja procesa)

- **Promenljive**: Lokalne vrednosti unutar procesa

```
variable vRESULT : std_logic := '0';
```

:= VHDL simbol za dodelu vrednosti promenljivoj
Simulator dodeljuje vrednost promenljivoj u trenutku kada
nađe na traženu dodelu (nema odlaganja dodele)

Standardni tipovi signala

std_ulogic & std_logic

Vrednost	Interpretacija
U	Neinicijalizovano <i>Uninitialized</i>
X	Nepoznato <i>Forcing Unknown</i>
0	Nula <i>Forcing 0</i>
1	Jedinica <i>Forcing 1</i>
Z	Visoka impedansa <i>High Impedance</i>
W	Slabo nepoznato <i>Weak Unknown</i>
L	Slaba nula <i>Weak 0</i>
H	Slaba jedinica <i>Weak 1</i>
-	Nedefinisano <i>Don't Care</i>

Jednobitni i vektorski signali

- Dodela vrednosti jednobitnom signalu se realizuje putem jednostrukih znakova navoda:

```
ARCHITECTURE ...  
    signal sEN : std_logic;  
    ...  
BEGIN  
    sEN <= '1' when sSEL = '1' else '0';  
END primer;
```

- Dodela vrednosti vektorskom signalu se realizuje putem dvostrukih znakova navoda:

```
ARCHITECTURE ...  
    signal sITEM_NO : std_logic_vector(3 downto 0);  
    ...  
BEGIN  
    sITEM_NO <= "0101";  
END primer;
```

Unutar dvostrukih navodnika se moraju navesti vrednosti za sve bite koje sadrži vektor.

Moguće je iskoristiti i heksadecimalnu definiciju vrednosti sa X"..."

```
sITEM_NO <= X"5";
```

Tipovi podataka

- **Celobrojni tipovi:** `integer`
od -2,147,483,647, $-(2^{31}-1)$, do 2,147,483,647, $(2^{31}-1)$
Signali ili promenljive celobrojnog tipa moraju imati ograničen opseg radi definisanja potrebnog broja bita za fizičku implementaciju
`variable a: integer range -255 to 255;`
- **Tipovi sa pokretnim zarezom:** `real`
od -1.0E38 do +1.0E38
Signali tipa real se ne mogu koistiti za fizičku implementaciju (sintezu) već samo u simulaciji
- **Fizički tipovi:** `time`
`type time is range -2147483647 to 2147483647`
`units`
 `fs;`
 `ps = 1000 fs;`
 `ns = 1000 ps;`
 `us = 1000 ns;`
 `ms = 1000 us;`
 `sec = 1000 ms;`
 `min = 60 sec;`
 `hr = 60 min;`
`end units;`

Tip nabrajanja

- **Tip nabrajanja** je lista vrednosti koje jedan objekat tog tipa može da sadrži. Tipovi nabrajanja su često definisani za automate sa konačnim brojem stanja.

```
type states is ( idle, preamble, data,  
                jam,   nosfd,   error );
```

Sa ovakvom definicijom tipa signala može se deklarirati signal koji je upravo tog tipa:

```
signal current_state : states;
```

Postoje dva tipa nabrajanja definisana IEEE 1076 standardom veoma pogodna za sintezu: `bit` i `boolean`.

Oni su definisani kako sledi:

```
type boolean is ( FALSE, TRUE );  
type bit      is ( '0', '1' );
```

Sekvencijalni iskazi

Koriste se isključivo u VHDL procesima i čine najčešće korišćene iskaze u VHDL izvornim kodovima.

U zavisnosti od opisanog konteksta mogu da realizuju i kombinacione i sekvencijalne mreže.

```
if (condition) then
    do something;
else
    do something different;
end if;

if (condition1) then
    do something;
elsif (condition2) then
    do something different;
else
    do something completely
    different;
end if;

case selection_signal is
    when value_1_of_selection_signal =>
        -- grupa iskaza 1
        (do something)
    when value_2_of_selection_signal =>
        -- grupa iskaza 2
        (do something)
    when value_3_of_selection_signal =>
        -- grupa iskaza 3
        (do something)
    ...
    when last_value_of_selection_signal =>
        -- grupa iskaza x
        (do something)
end case;
```

PRIMERI VHDL KODOVA

Multiplekser

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY MUX8x1 IS PORT (
    iX:    IN    std_logic_vector(7 DOWNTO 0); -- vektor ulaznih signala
    iSEL:  IN    std_logic_vector(2  DOWNTO 0); -- vektor adresnih signala
    iE:    IN    std_logic;                    -- dozvola rada multipleksera
    oY:    OUT   std_logic );                  -- izlaz multipleksera
END MUX8x1;

ARCHITECTURE ARH_MUX8x1 OF MUX8x1 IS BEGIN
oY <= (iX(0) AND NOT(iSEL(2)) AND NOT(iSEL (1)) AND NOT(iSEL(0))) OR
      (iX(1) AND NOT(iSEL(2)) AND NOT(iSEL (1)) AND      (iSEL(0))) OR
      (iX(2) AND NOT(iSEL(2)) AND      (iSEL (1)) AND NOT(iSEL(0))) OR
      (iX(3) AND NOT(iSEL(2)) AND      (iSEL (1)) AND      (iSEL(0))) OR
      (iX(4) AND      (iSEL(2)) AND NOT(iSEL (1)) AND NOT(iSEL(0))) OR
      (iX(5) AND      (iSEL(2)) AND NOT(iSEL (1)) AND      (iSEL(0))) OR
      (iX(6) AND      (iSEL(2)) AND      (iSEL (1)) AND NOT(iSEL(0))) OR
      (iX(7) AND      (iSEL(2)) AND      (iSEL (1)) AND      (iSEL(0))) OR
END ARH_MUX8x1;
```

Multiplexer, arhitektura na drugi način

```
ARCHITECTURE ARH_MUX8x1 OF MUX8x1 IS BEGIN
  PROCESS (iX, iSEL, iE) BEGIN
    IF (iE = '1') THEN -- provera vrednosti signala dozvole rada
      -- multipleksiranje dozvoljeno -> odredjivanje izlaznog signala
      -- u zavisnosti od vrednosti adresnog vektora
      CASE iSEL IS
        WHEN "000" => oY <= iX(0);
        WHEN "001" => oY <= iX(1);
        WHEN "010" => oY <= iX(2);
        WHEN "011" => oY <= iX(3);
        WHEN "100" => oY <= iX(4);
        WHEN "101" => oY <= iX(5);
        WHEN "110" => oY <= iX(6);
        WHEN OTHERS => oY <= iX(7);
      END CASE;
    ELSE
      -- multipleksiranje nije dozvoljeno -> postavljanje definisane vrednosti na izlaz
      oY <= '0';
    END IF;
  END PROCESS;
END ARH_MUX8x1;
```

Prioritetni koder 74LS147

(izlazi su u negativnoj logici)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY PRIORITY_CODER IS PORT (
    iU: IN  std_logic_vector(9 DOWNT0 1);
    onA, onB, onC, onD: OUT std_logic );
END PRIORITY_CODER;

ARCHITECTURE ARH_PRIORITY_CODER
OF PRIORITY_CODER IS

    -- vektor koji prima odgovarajucu
    -- binarnu vrednost u zavisnosti od
    -- stanja ulaza koderu
    SIGNAL sBCD_VECTOR :
        std_logic_vector(3 DOWNT0 0);

BEGIN
```

```
PROCESS (iU) BEGIN
    IF      (iU(9)='0') THEN sBCD_VECTOR <= "1001";
    ELSIF   (iU(8)='0') THEN sBCD_VECTOR <= "1000";
    ELSIF   (iU(7)='0') THEN sBCD_VECTOR <= "0111";
    ELSIF   (iU(6)='0') THEN sBCD_VECTOR <= "0110";
    ELSIF   (iU(5)='0') THEN sBCD_VECTOR <= "0101";
    ELSIF   (iU(4)='0') THEN sBCD_VECTOR <= "0100";
    ELSIF   (iU(3)='0') THEN sBCD_VECTOR <= "0011";
    ELSIF   (iU(2)='0') THEN sBCD_VECTOR <= "0010";
    ELSIF   (iU(1)='0') THEN sBCD_VECTOR <= "0001";
    ELSE
        sBCD_VECTOR <= "0000";
    END IF;
END PROCESS;

-- dodela vrednosti izlaznim signalima na osnovu
-- formiranog binarnog vektora
onD <= NOT sBCD_VECTOR(3); -- bit sa težinom 8
onC <= NOT sBCD_VECTOR(2); -- bit sa težinom 4
onB <= NOT sBCD_VECTOR(1); -- bit sa težinom 2
onA <= NOT sBCD_VECTOR(0); -- bit sa težinom 1

END ARH_PRIORITY_CODER;
```


Barelov pomerač

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY BARREL IS PORT (
    -- vektor ulaznih signala
    iX: IN std_logic_vector(7 DOWNTO 0);
    -- vektor adresnih signala
    iSEL: IN std_logic_vector(2 DOWNTO 0);
    -- vektor izlaznih signala
    oY: OUT std_logic_vector(7 DOWNTO 0)
);
END BARREL;

ARCHITECTURE ARH_BARREL OF BARREL IS
    -- vektori koji predstavljaju izlaze
    -- odgovarajucih stepeni Barelovog
    -- pomeraca
    SIGNAL sSTEPEN0,
           sSTEPEN1,
           sSTEPEN2:
        std_logic_vector(7 DOWNTO 0);
BEGIN

    -- stepen 0:
    sSTEPEN0 <= (iX(6 DOWNTO 0) & iX(7))
        WHEN (iSEL(0) = '1')
        ELSE iX;

    -- stepen 1:
    sSTEPEN1 <= (sSTEPEN0(5 DOWNTO 0) &
        sSTEPEN0(7 DOWNTO 6))
        WHEN (iSEL(1) = '1')
        ELSE sSTEPEN0;

    -- stepen 2:
    sSTEPEN2 <= (sSTEPEN1(3 DOWNTO 0) &
        sSTEPEN1(7 DOWNTO 4))
        WHEN (iSEL(2) = '1')
        ELSE sSTEPEN1;

    -- izlaz trostepenog Barelovog pomeraca
    oY <= sSTEPEN2;
END ARH_BARREL;
```