# 19E042DE1

Univerzitet u Beogradu, Elektrotehnički fakultet

Nikola Petrović
Konsultacije po dogovoru - 102G

2022

# Motivacija

VHDL uvod

Osnovni pojmovi

Signali i tipovi podataka

Nizovi

Operatori

Sekvencijalne naredbe

Konkurentne i sekvencijalne naredbe

# Motivacija

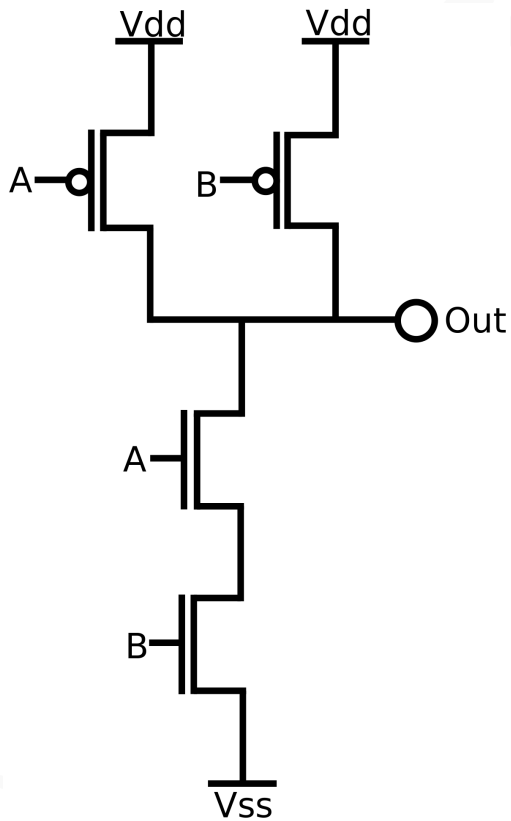
Do sada su rađena jednostavna kombinaciona kola:

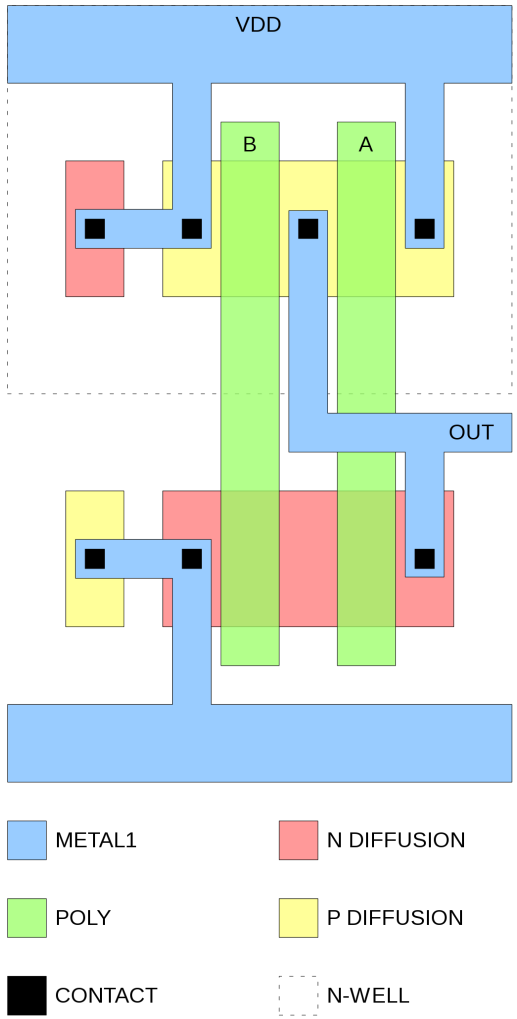
- NAND


Q = A NAND B

Truth Table

| Input A | Input B | | Output Q |
|---|---|---|---|
| 0 | 0 | | 1 |
| 0 | 1 | | 1 |
| 1 | 0 | | 1 |
| 1 | 1 | | 0 |

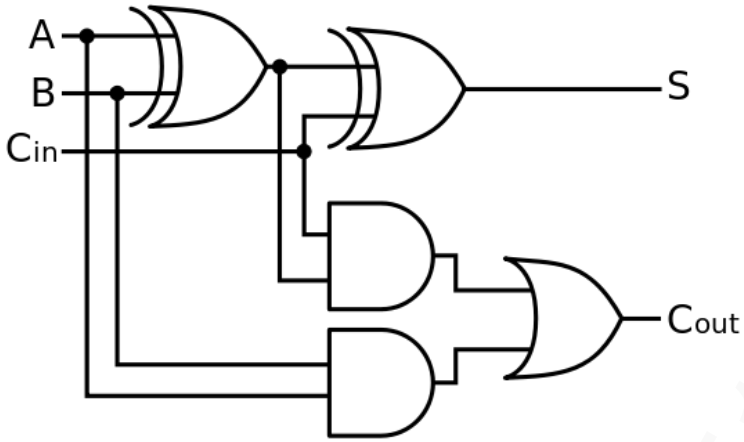


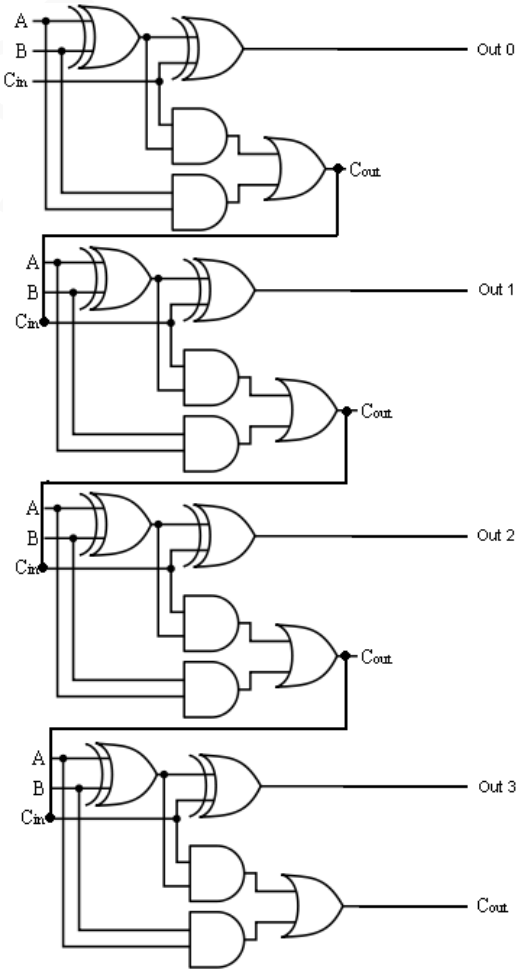



# Motivacija - nastavak

I malo složenija kola:

- Potpuni Sabirač

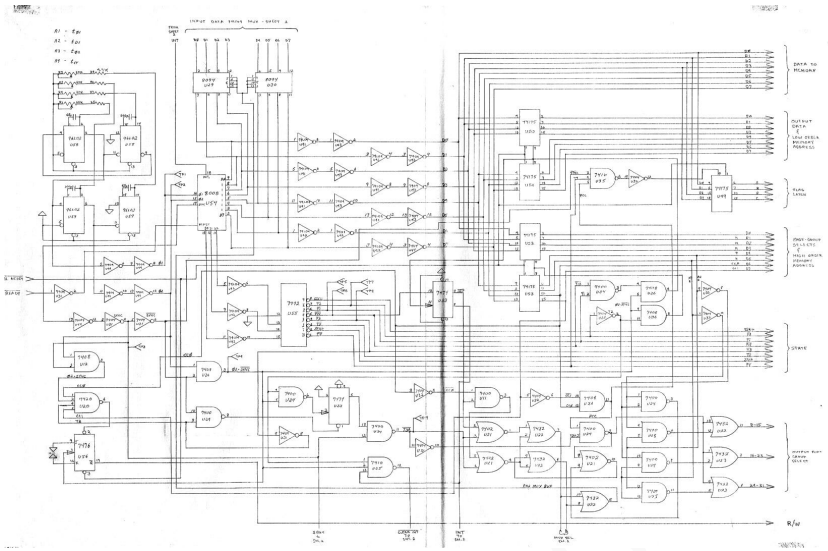


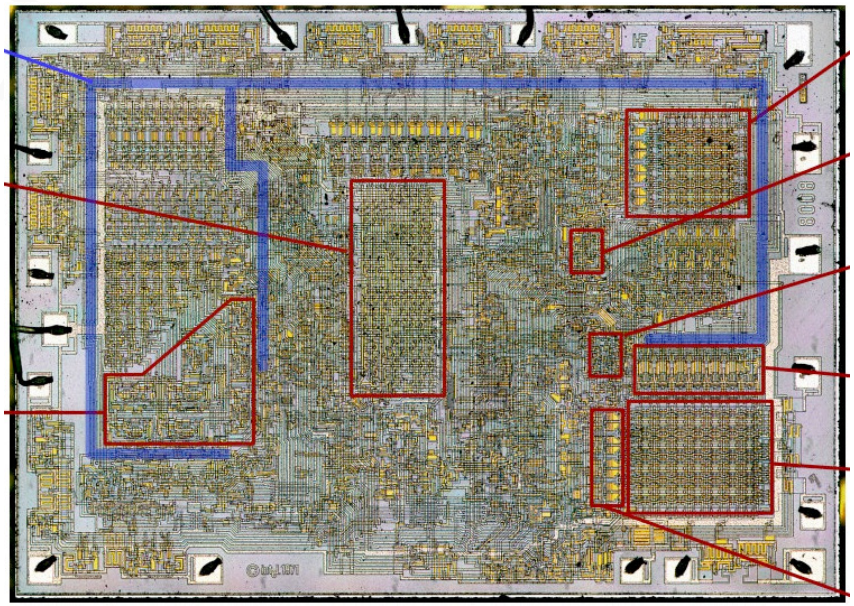



# Motivacija - nastavak

Koja postaju sve složenija:

- Intel 8008
- 3500 tranzistora
- Bio aktuelan 1972

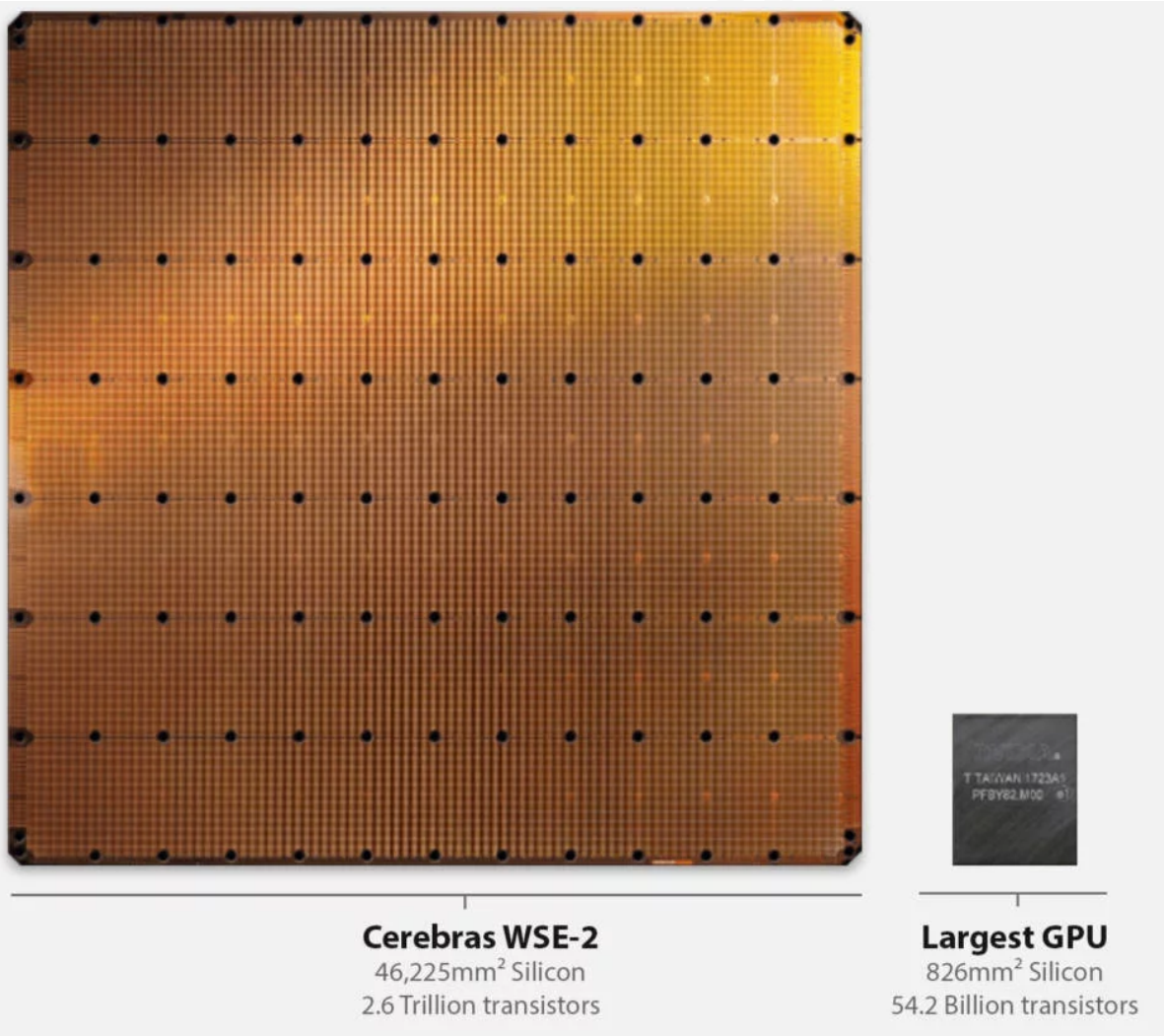




# Motivacija - nastavak

Šta je sada aktuelno:

- Cerebras WSE-2
- Trenutno najveći čip (za sada)



# Šta je VHDL?

**V**HSIC **H**ardware **D**escription **L**anguage

- VHSIC – Very High-Speed Integrated Circuit
- Modelovanje digitalnih elektronskih sistema

Internacionalni standard

- IEEE Std 1076

Verzije VHDL-a:

- VHDL'87, VHDL'93, VHDL2001, VHDL2006, VHDL2008 (major update), VHDL2019

Gde se koristi?

- Za pisanje ASIC/FPGA RTL koda za sintezu
- Sistem arhitekte za "high-level" simulacije sistema
- Kod verifikacije za pisanje testova za sve nivoe simulacije
- Za modelovanje ASIC/FPGA ćelija i makroa

# Hardware Description Language (HDL)

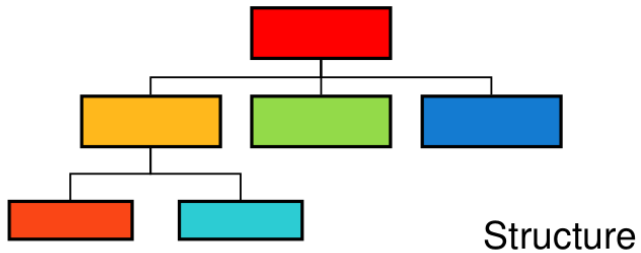
VHDL nije softverski jezik.

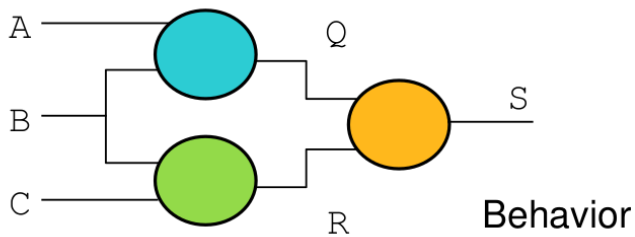
VHDL je jezik za opis hardvera (eng. Hardware Description Language (HDL)).

- Programski jezik sa specijalnim konstrukcijama za modelovanje hardvera.

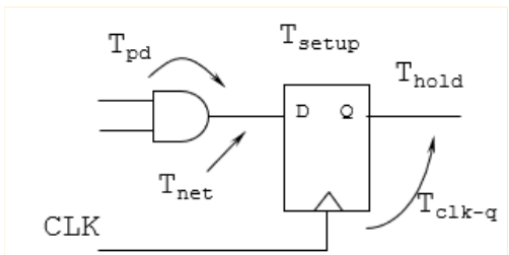
VHDL podržava:

- Strukturni opis (eng. Structure)
  - Fizički opis (netlist-e i hiararhije)
  - Softverski opis (potprogrami)
- Hardverski opis (eng. Hardware behavior)
  - Serijski (sekvencijalni) opis
  - Paralelni (konkurentni) opis
- Vremena (eng. Timing)
- Nivoe apstrakcije


Structure

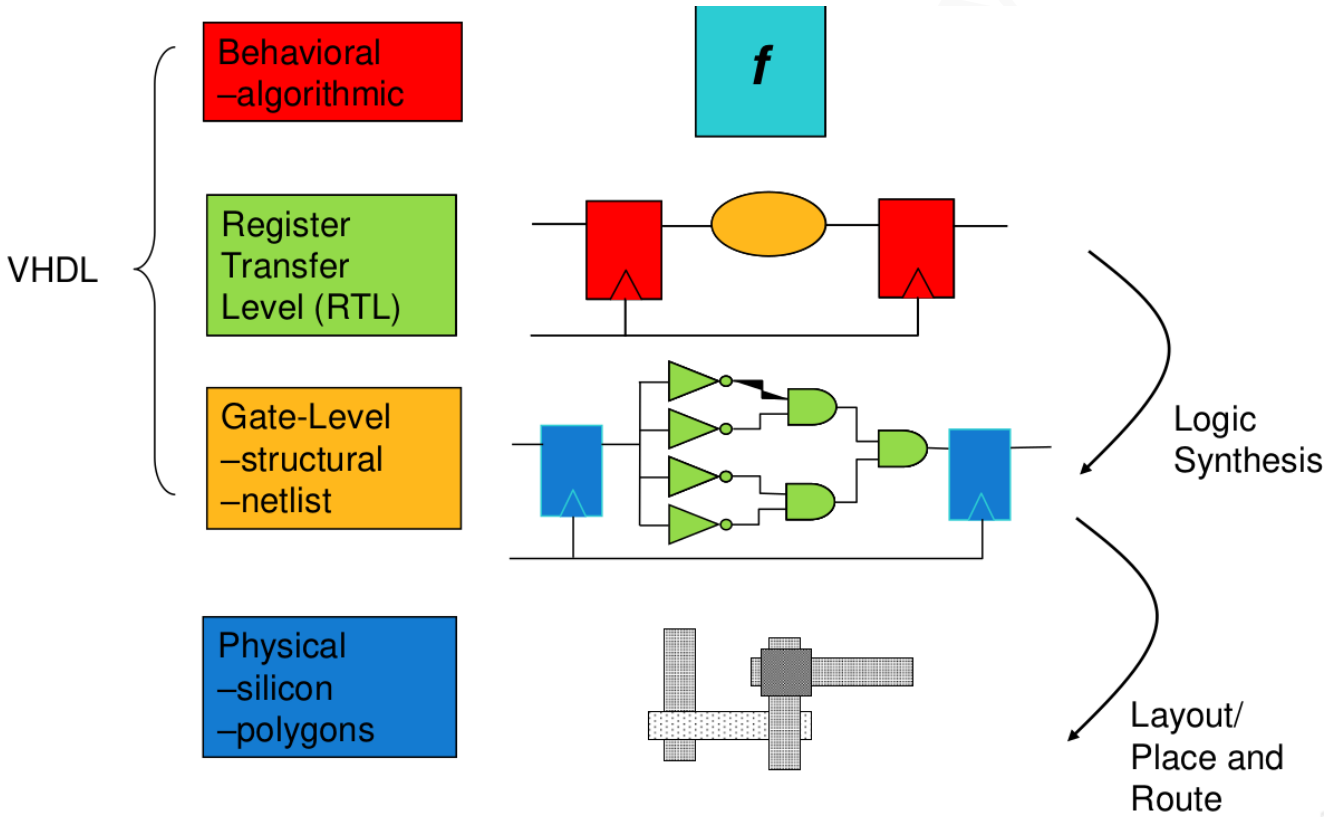


Behavior

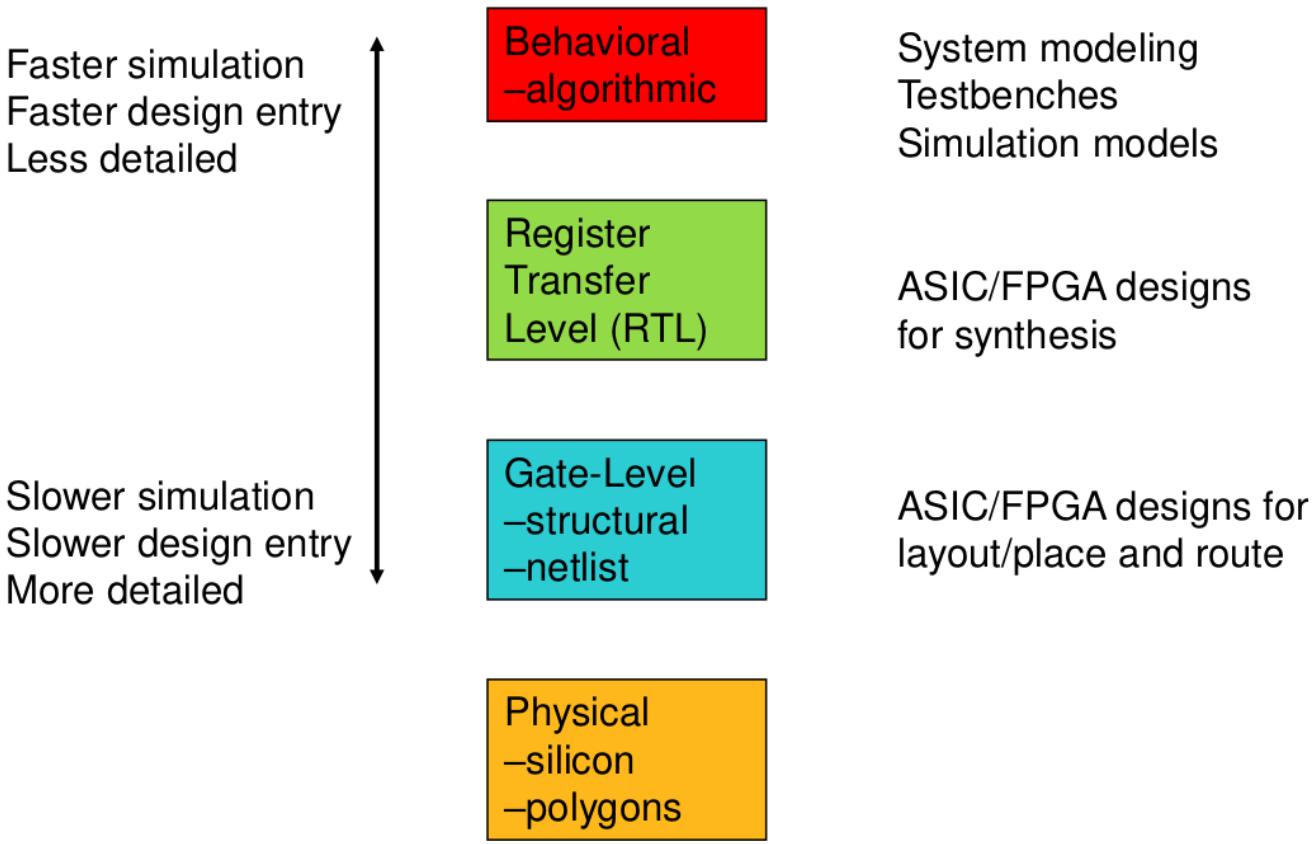


Timing

# Nivoi apstrakcije

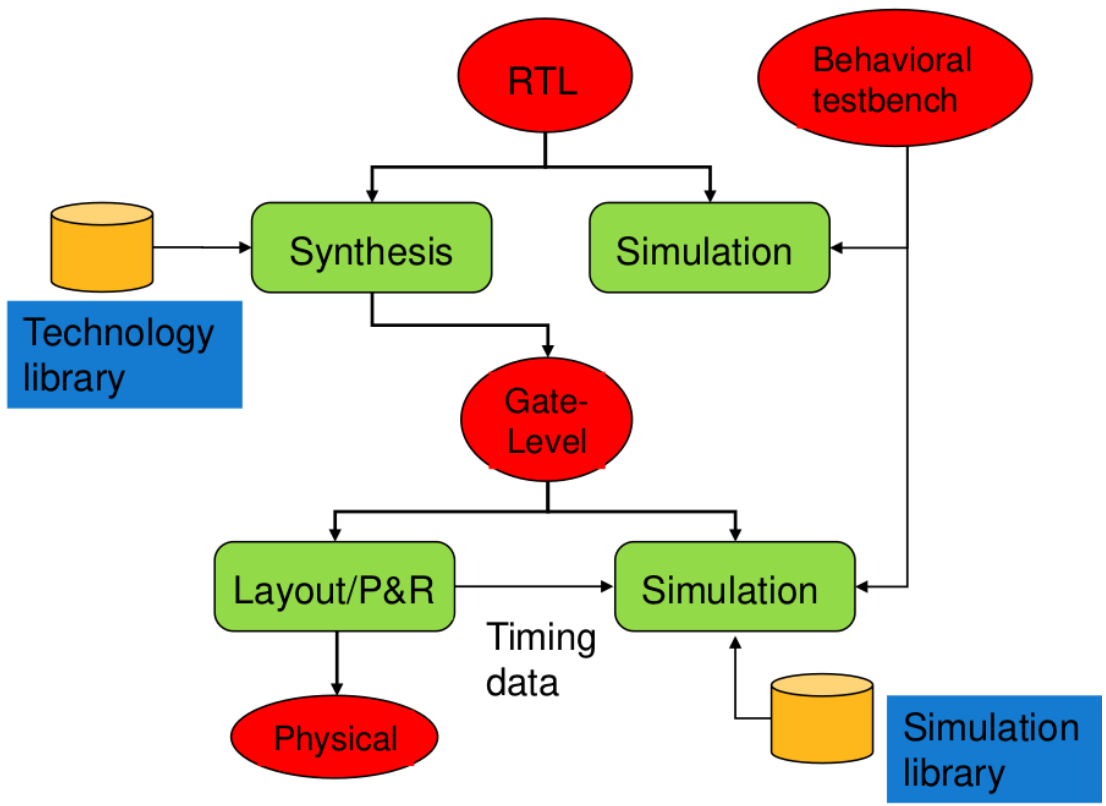


# Kompromisi između nivoa apstrakcije



# Uprošćen tok dizajna (eng. Design flow Sequence)

- Kreirati RTL dizajn
- Kreirati testbench
- Simulirati dizajn radi provere funkcionalnosti
- Izvršiti sintezu na nivo gejtova (eng. gate level synthesis)
- Simulirati dizajn radi provere funkcionalnost i kašnjenja
- Lejaut



# Prednosti VHDL-a

Dizajn na višem nivou apstrakcije

- Možemo naći probleme ranije
- Možemo istražiti alternative

Nezavistan od načina implementacije

- Omogućava promene u "poslednji minut" (ne raditi ovo!)
- Odluku o načinu implementacije možemo ostaviti za kasnije

Fleksibilnost:

- Kod se može upotrebiti više puta
- Isti kod možemo koristiti za različite alate

Jezik je:

- Može se lako pisati i modifikovati

# Problemi korišćenja VHDL-a

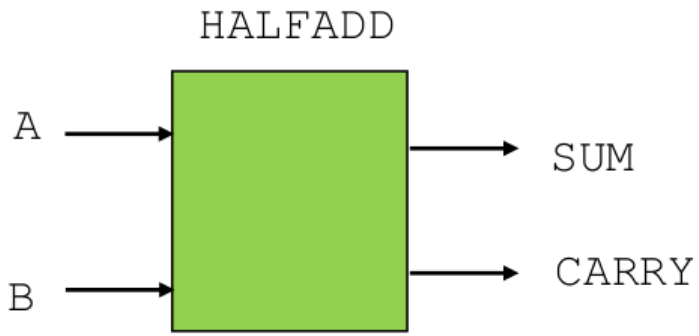
- Novi programski jezik koji je potrebno naučiti
- Novi EDA alati za simulaciju i sintezu koje je potrebno savladati
- Ne postoji standardna "off-the-shelf" metodologija
  - Mora se planirati i implementirati
  - Verovatno se moraju koristiti alati od više različitih kompanija
- VHDL je predviđen za digitalni dizajn
  - Mada postoje analogne ekstenzije (VHDL-AMS)
- Način na koji se piše kod utiče na krajnji ishod projekta
- Potrebno je isplanirati projekat pre nego što se krene sa kodovanjem
- I još dosta drugih stvari...

# Entity

Interfejs sistema.Ne sadrži definiciju ponašanja bloka.

Lista portova mora da sadrži:

- Ime porta (`A, B, ...`)
- Orijentaciju porta (`in / out / inout`)
- Tip porta, npr. `std_logic`

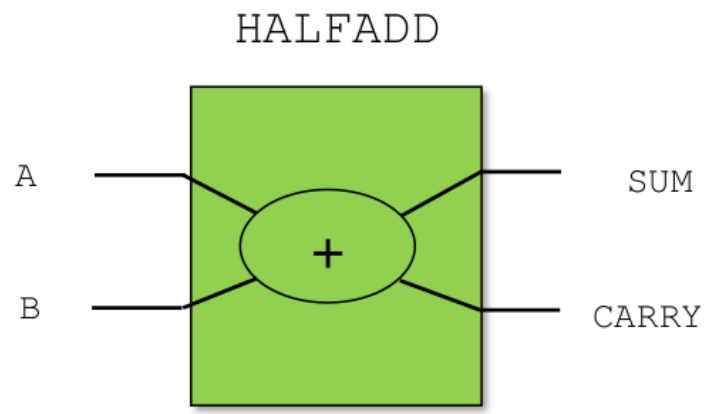



```
library IEEE;
use IEEE.std_logic_1164.all;

entity HALFADD is
    port (
        A, B       : in  std_logic;
        SUM, CARRY : out std_logic
    );
end HALFADD;
```

* `library` i `use` deklaracije omogućavaju korišćenje `std_logic` tipa.

# Architecture

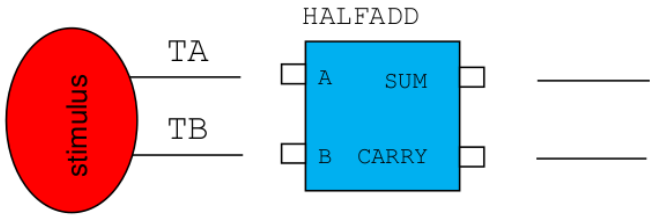
- Opisuje ponasanje bloka
- Mora biti povezana sa odgovarajucim "entity"
- Jedan *entity* može imati više arhitektura.
- Reč begin odvaja deklarativni deo od funkcionalnog dela.




```
architecture RTL of HALFADD is
    -- ovde staviti deklaracije
begin
    SUM   <= A xor B;
    CARRY <= A and B;
end RTL;
```

# Kreiranje hijerarhije: Deklaracija komponenti

- Komponente moraju biti deklarisane kako bi se instancirale unutar arhitekture.
- Signali moraju biti deklarisani da bi se povezali na portove komponente.
- Deklaraciju signala i komponenti odraditi pre ključne reči begin u deklarativnom delu arhitekture.



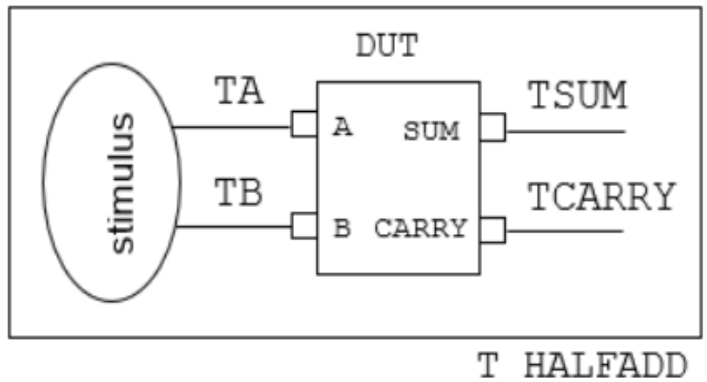

```
library IEEE;
use IEEE.std_logic_1164.all;

entity T_HALFADD is
end T_HALFADD;

architecture BENCH of T_HALFADD is

    component HALFADD
        port(
            A, B       : in std_logic;
            SUM, CARRY : out std_logic
        );
    end component;

    signal TA, TB : std_logic;
    signal TSUM, TCARRY : std_logic;

begin
    -- instanciranje komponenti
    -- generisanje stimulusa
end BENCH;
```

# Kreiranje hijerarhije: Instanciranje komponenti

Svaka instancirana komponenta ima:

- Jedinstveno ime
- Port map-u za povezivanje portova komponenti i lokalnih portova i signala

```
library IEEE;
use IEEE.std_logic_1164.all;

entity T_HALFADD is
end T_HALFADD;

architecture BENCH of T_HALFADD is

    component HALFADD
        port(
            A, B       : in std_logic;
            SUM, CARRY : out std_logic
        );
    end component;

    signal TA, TB : std_logic;
    signal TSUM, TCARRY : std_logic;

begin
    DUT : HALFADD port map (A => TA, B => TB, SUM => TSUM, CARRY => TCARRY);
    -- generisanje stimulusa
end BENCH;
```



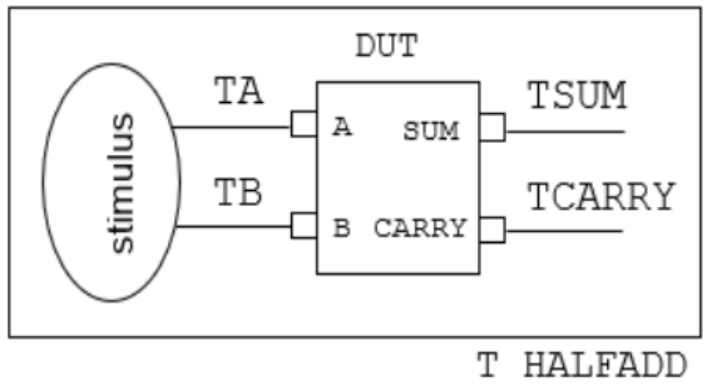

* Ovaj način povezivanja se naziva asocijacija po imenu (eng. named association)

# Poziciona asocijacija portova

Lokalni portovi i signali mapiraju se na portove komponente na osnovu njihovog redosleda deklarisanja.

- Prvi signal TA se mapira na prvi port A komponente DUT.
- Drugi signal TB se mapira na drugi port B komponente DUT itd.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity T_HALFADD is
end T_HALFADD;

architecture BENCH of T_HALFADD is

    component HALFADD
        port(
            A, B       : in std_logic;
            SUM, CARRY : out std_logic
        );
    end component;

    signal TA, TB : std_logic;
    signal TSUM, TCARRY : std_logic;

begin
    DUT : HALFADD port map (TA, TB, TSUM, TCARRY);
    -- generisanje stimulusa
end BENCH;
```




* Ovaj način povezivanja je manje čitljiv i skloniji je greškama nego asocijacija po imenu korišćena u prethodnom primeru.

# Podrazumevana konfiguracija

Deklaracija komponente ne predstavlja direktnu konekciju sa entitijem te komponente.

Ukoliko se prate određena pravila, deklaracija komponente i postojeći entity su automatski linkovani. Ovo je takozvana podrazumevana konfiguracija (eng. default configuration)

Pravila za podrazumevano konfiguraciju:

- Imena za entity i komponentu moraju biti ista
- Imena i tipovi portova moraju biti isti (redosled može biti drugačiji)
  U suprotnom, eksplicitna konfiguracija mora biti napisana kako bi se entity i komponenta linkovali.

```
component HALFADD
    port(
        A, B       : in std_logic;
        SUM, CARRY : out std_logic
    );
end component;
```

```
entity HALFADD is
    port (
        A, B       : in  std_logic;
        SUM, CARRY : out std_logic
    );
end HALFADD;
```

* Koristiti copy/paste da bi sigurni da se portovi i imena poklapaju.

# Paket (*eng. Package*)

Kolekcija deklaracija:

- Tipova
- Konstanti
- Komponenti
- Potprograma

```
library IEEE;
use IEEE.std_logic_1164.all;

package SYSTEM_CONSTS is
    constant HIGH : std_logic := '1';
    constant LOW  : std_logic := '0';
end SYSTEM_CONSTS;
```

# Telo paketa (*eng. Package Body*)

Ne može postojati bez paketa.

Najčešće sadrži:

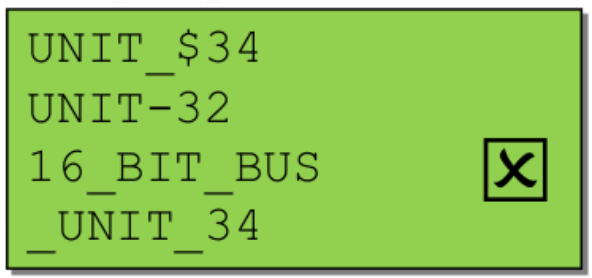
- Definicije potprograma
- Vrednosti konstanti
- Lokalne deklaracije

```
package DEMO_PACK is
    -- konstante
    -- tipovi podataka
    -- deklaracije komponenti
    -- deklaracije sub-programa
end DEMO_PACK;
```

```
package body DEMO_PACK is
    -- vrednosti konstanti
    -- lokalne deklaracije
    -- definicije sub-programa
end DEMO_PACK;
```

# Pravila za imenovanje

- Imena mogu da sadrže samo slova, brojeve i donju crtu ("_").
- VHDL **NE** pravi razliku između velikih i malih slova.
  - Sve je ovo jedan isti objekat: ABC, Abc, abc, aBc
- Imena moraju početi sa slovom.
- Imena mogu biti bilo koje dužine.
  - Alat ili metodologija može ograničiti dužinu imena
  - Od VHDL'93, ilegalna imena se mogu koristiti ako se ime počne sa "\" (*eng. extended identifier*)


# Komentari i razmaci

```
-- ova linija je komentar.
-- svaka linija komentara mora da krene sa: --
-- komentar se zavrsava prelaskom na novu liniju
library IEEE;
use IEEE.std_logic_1164.all;

entity HALFADD is
    port (
        A, B       : in std_logic;
        SUM, CARRY : out std_logic
    );
end HALFADD; -- komentar na kraju linije

architecture BEHAVE of HALFADD is
begin
    -- VHDL je jezik slobodnog formata
    -- dodatni razmaci se mogu koristiti da bi se kod napravio citljivijim
    SUM   <= A xor B;
    CARRY <= A and B;
end BEHAVE;
```

# Koncept tipa signala

Tip signala mora da se definiše kada se signal deklariše, bilo u:

- architecture, u delu za deklaraciju
- Port sekciji entity

Tipovi podataka moraju da se poklapaju!

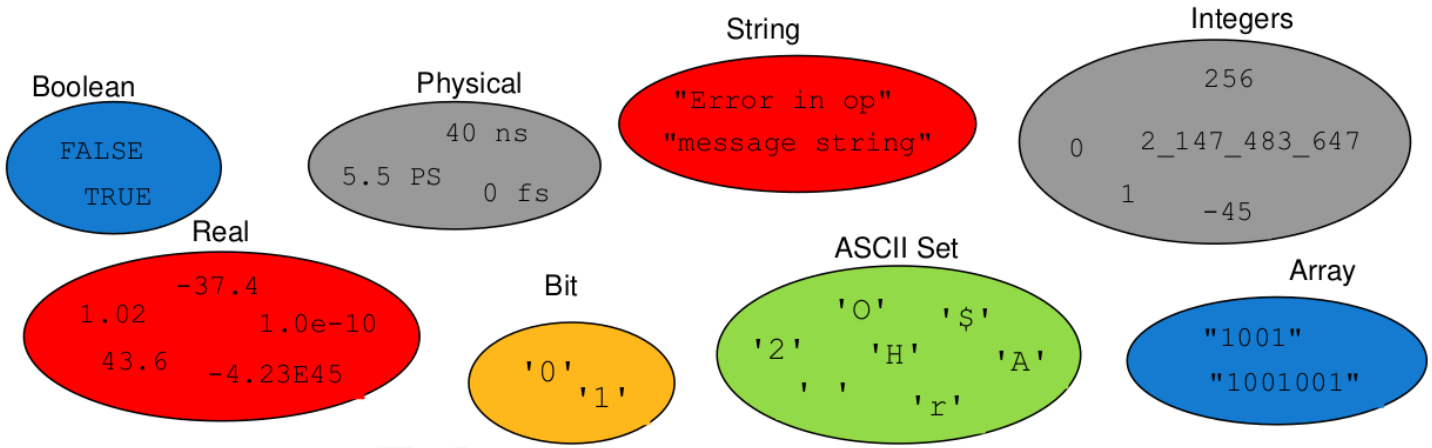
```
library IEEE;
use IEEE.std_logic_1164.all;

entity FULLADD is
    port (
        A, B, CIN  : in  std_logic;
        SUM, CARRY : out std_logic
    );
end FULLADD;

architecture STRUCTURAL of FULLADD is

    signal N_SUM, N_CARRY1, N_CARRY2 : std_logic;

-- component declarations
begin
    -- component instantiations
end STRUCTURAL;
```

# Standardni predefinisani tipovi podataka

Svaki tip ima set potencijalnih vrednosti:

- Standardni tipovi - definisani jezikom
- Ugrađeni paketi dodatnih tipova podataka (std_logic)
- Možemo sami definisati nove tipove

```
package STANDARD is
    type BOOLEAN is (FALSE, TRUE);
    type BIT is ('0', '1');
    type CHARACTER is (-- ASCII set );
    type INTEGER is range <min 32 bit>
    subtype NATURAL is integer range 0 to ...
    type REAL is range <min 64 bit>
    -- BIT_VECTOR
    -- STRING
    -- TIME
end STANDARD;
```



# Upotreba std_logic paketa

- std_logic_1164 paket dolazi prekompajliran sa svakim simulatorom
  - Nalazi se u IEEE biblioteci
- Na početku koda je potrebno dodati library i use klauzulu kako bi paket bio vidljiv.
- library i use navesti pre entity!

```
library IEEE;
use IEEE.std_logic_1164.all;

entity HALFADD is
    port (
        A, B        : in  std_logic;
        SUM, CARRY  : out std_logic
    );
end HALFADD;
```

# Tipovi std_logic

```
package std_logic_1164 is
    type std_ulogic is (
        'U', -- Uninitialized
        'X', -- Forcing 0 or 1
        '0', -- Forcing 0
        '1', -- Forcing 1
        'Z', -- High Impedance
        'W', -- Weak 0 or 1
        'L', -- Weak 0
        'H', -- Weak 1
        '-'  -- don't care
    );
...
```

Postoje dva tipa sa istim vrednostima:

- std_logic
- std_ulogic

# Signali i izvori *eng. Drivers*

Dodeljivanje vrednosti signalu kreira izvor signala (*eng. signal driver*).

Tipovi moraju biti isti sa obe strane dodele ($<=$).

Ako nakačimo više izvora na isti objekat, to može izazvati grešku pri kompajliranju.

- Potrebno je imati specijalni tip podataka

```
architecture RTL of EXAMPLE is
    signal A, B : std_logic;
    signal I1, I2, I3 : integer;
begin
    A <= B; -- Ovo je u redu, isti su tipovi.

    B <= I2; -- Tipovi se razlikuju!

    I1 <= I2; -- multiple ...
    I1 <= I3; -- ... drivers
    ...
end RTL;
```

# Rešavanje problema više izvora signala

Svi standardni tipovi su *"unresolved"*

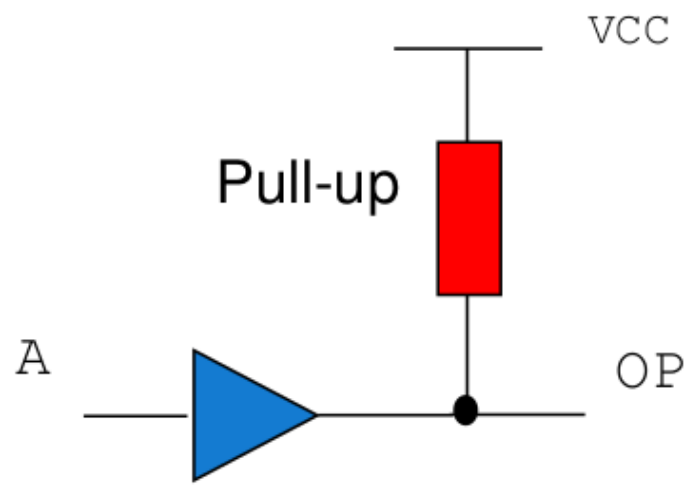
- Više konkurentnih dodela signalu rezultuje greškom pri kompajliranju

Da bi se ovo izbeglo, potreban je *"resolved"* tip

- Više konkurentnih dodela signalu je dozvoljeno
- Posebne funkcije određuju vrednost *"resolved"* signala
- Ovo je bitno za modelovanje hardvera (tri-state, pulldown/up itd.)

std_logic je jedini "*resolved*" tip koji se često koristi.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PULLUP is
    port (
        A  : in  std_logic;
        OP : out std_logic
    );
end PULLUP;

architecture BEHAV of PULLUP is
begin
    OP <= 'H';
    OP <= A;
end BEHAV;
```



# Korišćenje standardnih logičkih tipova

- std_logic je *resolved* tip
  - Dozvoljeno je više izvora
- std_ulogic je *unresolved* tip
  - Nije dozvoljeno više izvora, greška pri kompajliranju

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PULLUP is
    port (
        A  : in  std_logic;
        OP : out std_logic
    );
end PULLUP;

architecture BEHAV of PULLUP is
begin
    OP <= 'H';
    OP <= A;
end BEHAV;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PULLUP is
    port (
        A  : in  std_ulogic;
        OP : out std_ulogic
    );
end PULLUP;

architecture BEHAV of PULLUP is
begin
    OP <= 'H'; -- ulogic is unresolved!
    OP <= A;   -- izazvace gresku
end BEHAV;
```

# Resolved / Unresolved

std_ulogic:

- Daje grešku ukoliko slučajno povezemo više izvora na isti signal

std_logic:

- Koristi se za simulacije na nivou gejtova
- Koristi se za matematicke funkcije
- Potreban je za tri-state logiku
- Nema razlike što se tice brzine izvršavanja simulacije

# Nizovi

std_logic_1164 definiše:

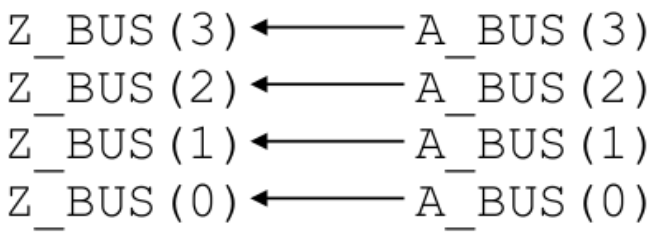
- std_logic_vector (niz std_logic)

VHDL definiše:

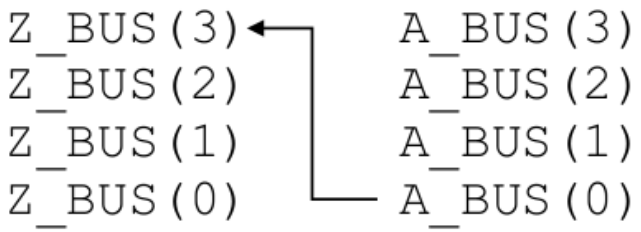
- bit_vector (niz bitova)
- string (niz karaktera)

Pri deklarisanju ovih objekata potrebno je definisati veličinu.

```
Z_BUS <= A_BUS;
```




```
Z_BUS(3) <= A_BUS(0);
```




```
signal A_BUS, Z_BUS : std_logic_vector(3 downto 0);
```

# Indeksi nizova

```
signal Z_BUS : std_logic_vector(3 downto 0);
signal B_BUS : std_logic_vector(3 downto 0);
signal C_BUS : std_logic_vector(0 to 3);
```

- Nizovi mogu biti definisani u oba smera.
- downto je de-facto standard.

```
Z_BUS <= B_BUS;
```

```
Z_BUS(3) <---- B_BUS(3)
Z_BUS(2) <---- B_BUS(2)
Z_BUS(1) <---- B_BUS(1)
Z_BUS(0) <---- B_BUS(0)
```

```
Z_BUS <= C_BUS;
```

```
Z_BUS(3) <---- C_BUS(0)
Z_BUS(2) <---- C_BUS(1)
Z_BUS(1) <---- C_BUS(2)
Z_BUS(0) <---- C_BUS(3)
```

* Elementi niza se dodeljuju po poziciji a ne po broju u nizu!
* Koristiti uvek isti smer za deklarisanje nizova!

# Dodeljivanje vrednosti nizovima

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

- Tipovi podataka moraju da budu isti sa obe strane dodele vrednosti.
- Veličina nizova sa obe strane dodele vrednosti treba biti ista.
- Donja crta "_" može da se koristi pri dodeljivanju vrednosti
  - Povećava čitljivost
  - Dupla donja crta nije dozvoljena
- Numeričke vrednosti se mogu predstaviti u heksadekadnom ili oktalnom sistemu
  - Veličina niza i veličina signala se moraju slagati

```
Z_BUS <= 11;          ☒
Z_BUS <= "1011";      ☑
Z_BUS <= "10111";     ☒
Z_BUS <= "10_11";     ☑
Z_BUS <= "10__11";    ☒

Z_BUS <= X"B";        ☑
Z_BUS <= O"23";       ☒
```

# Dodeljivanje vrednosti nizovima

VHDL2008 dozvoljava da se definiše širina vrednosti koju dodeljujemo.

```
signal E_ARRAY : std_logic_vector(5 downto 0);
```

Vrednosti su ili proširene (uglavnom nulama) ili skraćene na datu širinu:

- `6X"E" = "001110"`

Mogu se koristiti i X i Z vrednosti:

- `6X"ZZ" = "ZZZZZZ"`

Decimalna notacija D je dozvoljena:

- `6D"19" = "010011"`

Ekstenzija za znak S je dozvoljena:

- `6SX"E" = "111110"`

VHDL2002

```
-- razlika u sirinama (6 <= 8)
E_ARRAY <= X"23"
```

X

VHDL2008

```
-- length match 6 <= 6
Z_BUS <= 6X"27"
```

✓

VHDL2008

```
X"6"    = "0110"
X"ZZ"   = "ZZZZZZZZ"
3X"6"   = "110"
10X"E"  = "0000001110"
10SX"E" = "1111111110"
8D"15"  = "00001111"
```

# Dodeljivanje vrednosti nizovima - nastavak

```
signal BIT_1, BIT_2, BIT_3, BIT_4 : std_logic;
signal EX_BUS : std_logic_vector(3 downto 0);
signal BAJT   : std_logic_vector(7 downto 0);
```

- VHDL nam omogućava da dodelimo vrednosti pojedinačnim elementima niza.
- Pri tome treba voditi računa da se pokriju svi elementi niza.

```
-- asocijacija po poziciji
EX_BUS <= (BIT_1, BIT_2, BIT_3, BIT_4);
```

```
-- asocijacija po imenu
BAJT <= (7 => '1', 5 downto 1 => '1', 6 => BIT_2, others => '0');
```

- Pomoću "`others`" niz možemo popuniti vrednošću (0/1) bez obzira na njegovu širinu.

```
BAJT <= (others => '0');
```

# Nabrajanja (*eng. Enumerated type*)

- VHDL dozvoljava da devinišemo svoje tipove podataka.
  - Prvo se definiše tip, a zatim se definišu objekti tog tipa.
- Jedan takav tip je *enumerated* tip koji se koristi pri dizajnu FSM-a.
- Voditi računa o tome da se različiti tipovi podataka ne mogu mešati.

```
type MY_STATE is (RESET, IDLE, DOING_SOMETHING, DONE_EVERYTHING);
...
signal STATE : MY_STATE;
signal TWO_BIT : std_logic_vector(1 downto 0);
...
STATE <= RESET;    ✓
STATE <= "00";     X
STATE <= TWO_BIT;  X
```

* Alati za sintezu dozvoljavaju da se ovaj tip mapira na odredjeni način.

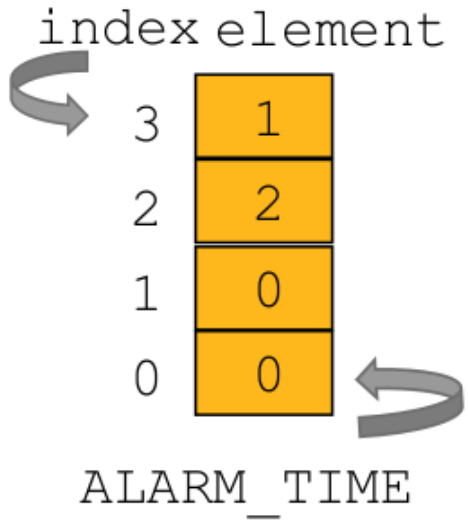
# Nizovi - nastavak

VHDL dozvoljava korisniku da definiše svoje nizove.

```
type T_CLOCK_TIME is array (3 downto 0) of integer;
signal CURRENT_TIME, ALARM_TIME : T_CLOCK_TIME;
```

```
ALARM_TIME(3) <= 1;
ALARM_TIME(2) <= 2;
ALARM_TIME(1) <= 0;
ALARM_TIME(0) <= 0;

-- assigned with aggregate
CURRENT_TIME <= (1,2,0,0);
```

```
CURRENT_TIME <= ALARM_TIME;
```



# Niz nizova

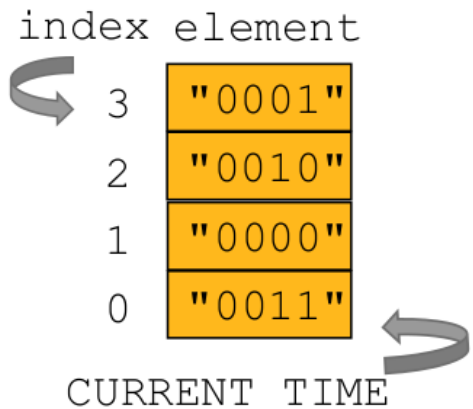
Element niza može biti drugi niz.

- *eng. "array of arrays"*
- Višedimenzionalni nizovi su takođe dozvoljeni

```
type T_4X4 is array (3 downto 0) of std_logic_vector(3 downto 0);
signal CURRENT_TIME, ALARM_TIME : T_4X4;
```

```
-- element assignments
ALARM_TIME(3) <= "0001";
ALARM_TIME(0) <= (others => '0');

-- array assigned with aggregate
CURRENT_TIME <= ("0001","0010",
                 "0000","0011");
ALARM_TIME <= (others => "0000");
```

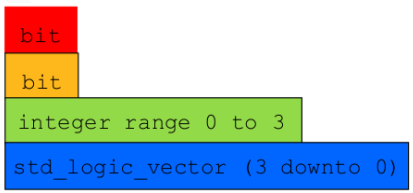
```
CURRENT_TIME <= ALARM_TIME;
```



# Records

- Kolekcija signala različitih tipova: record
  - Eksplicitno imenovati elemente
- Vrednosti se mogu dodeliti asocijacijom po poziciji ili po imenu.
- Pojedinačnim elementima se pristupa pomoću tačke ".".

```
type T_PACKET is record
    BYTE_ID : std_logic;
    PARITY  : std_logic;
    ADDRESS : integer range 0 to 3;
    DATA    : std_logic_vector(3 downto 0);
end record;
signal TX_DATA, RX_DATA : T_PACKET;
...
RX_DATA <= TX_DATA;

RX_DATA.ADDRESS <= 3;

-- asocijacija po poziciji
TX_DATA <= ('1', '0', 2, "0101");
-- asocijacija po imenu
RX_DATA <= ( BYTE_ID => '1',
             PARITY  => '0',
             ADDRESS => 1,
             DATA    => "0100");
```



# Logički operatori

- `and, or, nand, nor, xor, xnor` (operatori jednakog prioriteta)
- `not` (operator višeg prioriteta)
- Opracije su predefinisane za:
  - `bit`
  - `bit_vector`
  - `boolean`
- U paketu `std_logic_1164` operacije su definisane za:
  - `std_ulogic, std_logic`
  - `std_ulogic_vector, std_logic_vector`
- Operacija šiftovanja je dodata u VHDL'93
  - `sll, srl, sla, sra, rol, ror`
  - Operacije su predefinisane samo za `bit_vector`

# Primer

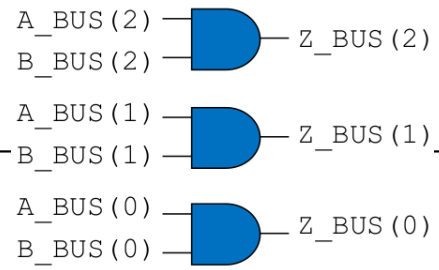
Koristiti zagrade za kontrolisanje evaluacije izraza (kao i za povećanje čitljivosti).

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MVLS is
    port (
        A, B, C : in std_logic;
        OP      : out std_logic
    );
end MVLS ;

architecture EX of MVLS is
begin
    OP <= A and not(B or C);
end EX;
```

# Logičke operacije na nizovima

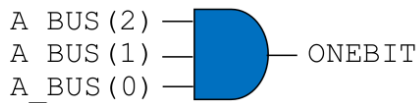
- Operandi moraju biti istog tipa i dimenzija
- Svi elementi operacije moraju biti istog tipa

```
signal A_BUS, B_BUS, Z_BUS : std_logic_vector(2 downto 0);
...
Z_BUS <= A_BUS and B_BUS;
...
```




- VHDL08: Logičke operacije se mogu koristiti kao operacije redukcije:

```
signal A_BUS : std_logic_vector(2 downto 0);
signal ONEBIT : std_logic;
...
ONEBIT <= and A_BUS
```



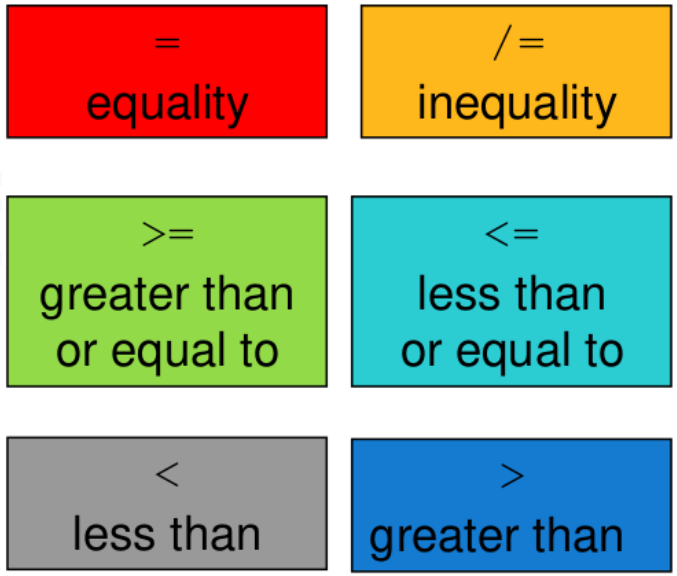
# Relacioni operatori

- Rezultat je `TRUE` ili `FALSE`
- Mogu se koristiti unutar `if` naredbe

```
signal BOOLA, BOOLB : boolean;
signal INTA, INTB : integer;
...
BOOLA <= INTA = INTB;
BOOLB <= INTA <= INTB; -- OK!
```

```
if A = B then
    OP <= '1';
else
    OP <= '0';
end if;
```

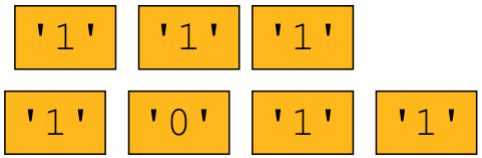
```
type T_COLOR is (RED, BLUE, GREEN);
-- RED < BLUE < GREEN
-- for std_logic '0' < '1' < 'Z'
```



# Relacione operacije na nizovima

- Mogu biti različitih širina
- Poravnjaju su nalevo pa se onda upoređuju
- Rezultati mogu biti neočekivani
- Nizovi nemaju numeričko značenje
  - Da li je vrednost binarna ili komplement dvojke?

```
signal A : std_logic_vector(2 downto 0);
signal B : std_logic_vector(3 downto 0);
...
if A > B then
...
```

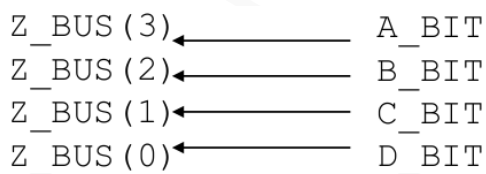


# Konkatenacija

```
signal A_BUS, B_BUS, Z_BUS : std_logic_vector(3 downto 0);
signal A_BIT, B_BIT, C_BIT, D_BIT : std_logic;
signal BYTE : std_logic_vector(7 downto 0);
```

- Možemo kreirati niz od pojedinačnih elemenata
- Možemo kreirati širi niz od pojedinačnih nizova
- Standardna pravila važe sa širine nizova

```
Z_BUS <= A_BIT & B_BIT & C_BIT & D_BIT;
```




```
BYTE <= A_BUS & B_BUS;
```

# Dodela vrednosti delu niza

```
signal Z_BUS, A_BUS : std_logic_vector(3 downto 0);
signal B_BIT : std_logic;
signal BYTE : std_logic_vector(7 downto 0);
```

- Podniz mora imati isti smer kao i niz
- Podniz se može koristiti sa bilo koje strane dodele

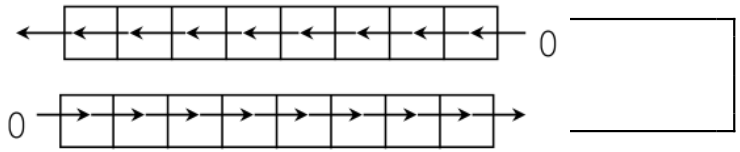
```
A_BUS <= BYTE(5 downto 2);
Z_BUS(1 downto 0) <= '0' & B_BIT;
```

```
BYTE(5 downto 2) <= A_BUS;     -- u redu
Z_BUS(0 to 1) <= '0' & B_BIT; -- nije u redu
```

# Rotiranje i pomeranje

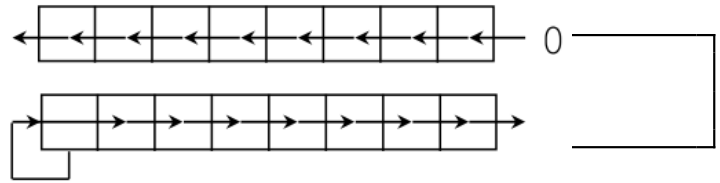
Logičko pomeranje:

```
BAJT <= BAJT(6 downto 0) & '0';      -- LEVO
BAJT <= '0' & BAJT(7 downto 1);      -- DESNO
```



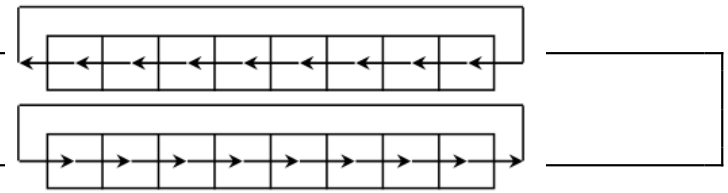

Aritmetičko pomeranje:

```
BAJT <= BAJT(6 downto 0) & '0';      -- LEVO
BAJT <= BAJT(7) & BAJT(7 downto 1); -- DESNO
```




Rotacije:

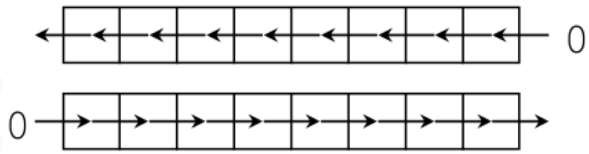
```
BAJT <= BAJT(6 downto 0) & BAJT(7); -- LEVO
BAJT <= BAJT(0) & BAJT(7 downto 1); -- DESNO
```



# Operatori za rotaciju i pomeranje

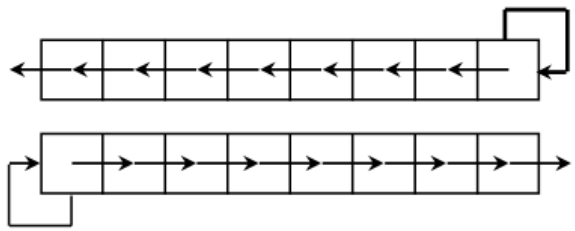
Logičko pomeranje:

```
BAJT <= BAJT sll 1; -- LEVO
BAJT <= BAJT srl 1; -- DESNO
```



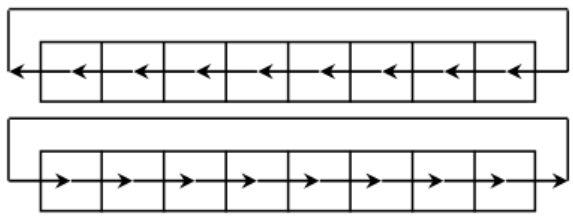

Aritmetičko pomeranje:

```
BAJT <= BAJT sla 1; -- LEVO
BAJT <= BAJT sra 1; -- DESNO
```



Rotacije:

```
BAJT <= BAJT rol 1; -- LEVO
BAJT <= BAJT ror 1; -- DESNO
```



# Proces

- Sekvencijalna logika se smešta unutar procesa (`process`).
- Procesi su definisati unutar arhitekture, gde:
  - Čitaju ulazne portove i lokalne signale
  - Pišu izlazne portove i lokalne signale
- Labela za proces je opcionalna.
- Procesi su međusobno konkurentni.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MUX is
    port (
        A, B, SEL : in std_logic;
        OP : out std_logic
    );
end MUX;

architecture BEHAVE of MUX is
begin

    MUXPROCESS: process (A, B, SEL)
    begin
        if (SEL = '1') then
            OP <= A;
        else
            OP <= B;
        end if;
    end process MUXPROCESS;

end BEHAVE;
```

# Lista osetljivosti

Proces se izvršava kada bilo koji signal iz liste osetljivosti promeni vrednost.

- eng. *Signal event*

Izlazni signali su ažirirani tek kada proces završi izvršavanje.

Bilo koji signal može biti u listi osetljivosti:

- Simulatori ne proveravaju signale u listi
- Većina alata za sintezu proverava listu

```
MUXPROCESS: process (A, B, SEL)
begin
    if (SEL = '1') then
        OP <= A;
    else
        OP <= B;
    end if;
end process MUXPROCESS;
```

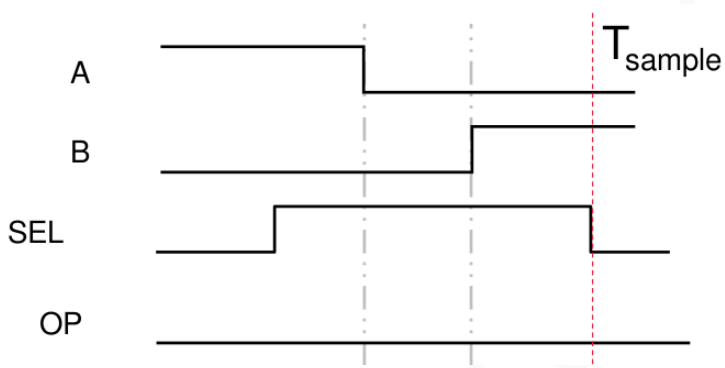
* Šta ako SEL nije u listi osetljivosti?

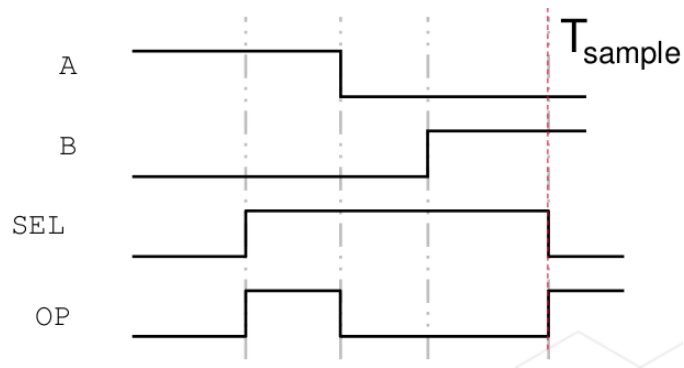
# Nepotpuna lista osetljivosti

Lista osetljivosti, za kombinacionu logiku, treba sadržati sve signale koji su pročitani u procesu.

```
MUXPROCESS: process (A, B)
...
```

```
MUXPROCESS: process (A, B, SEL)
...
```



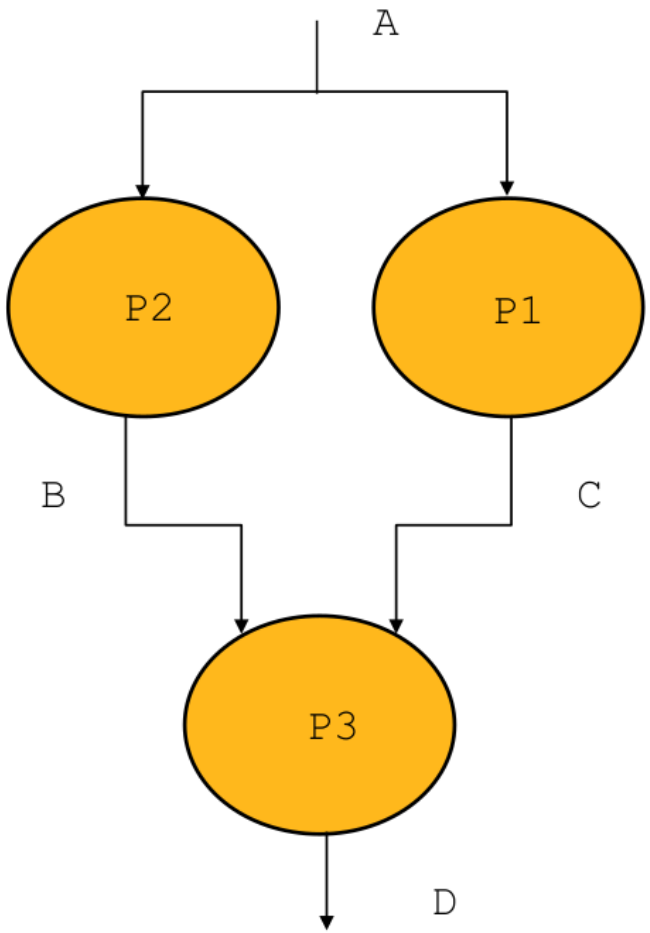
# process (all)

- VDHL 2008
- Koristi se u kombinacionim procesima
- Dodeljuje listi osetljivosti sve signale koji su pročitani u procesu
- Sme se koristiti i kod procedura
- Zanemarljiv uticaj na vreme kompajliranja
- Smanjuje grešku da se signal izostavi u listi osetljivosti

```
-- kompajler ekspanduje process (all) na
-- process(A, B, SEL)
-- gde su A, B, SEL ulazi
MUXPROCESS: process (all)
    if SEL = '1' then
        OP <= A;
    else
        OP <= B;
    end if;
end process MUXPROCESS;
```

# Signali i procesi

- Arhitektura može sadržati više procesa:
  - Spojeni signalima
  - Tip signala mora biti definisan
- Naredbe u procesima se izvršavaju sekvencijalno
  - eng. *serial hardware*
- Procesi sa istom listom osetljivosti se izvršavaju paralelno
  - eng. *concurrent hardware*
- Izlaz jednog procesa može biti ulaz drugog



# if

- Naredbe u samostalnom if se izvršavaju samo ako je uslov tačan.
- Možemo dodati bezuslovno else da izvršimo neke naredbe ukoliko uslov za if nije tačan.
- Možemo dodati jedan ili više elsif kako bi proverili dodatne uslove.
  - Prvo se proveravaju uslovi koji su prvi deklarisani
  - Poslednje else je opciono

```
if USLOV then
-- sekvencijalne naredbe
end if;
```

```
if USLOV then
-- sekvencijalne naredbe
else
-- sekvencijalne naredbe
end if;
```

```
if USLOV1 then
-- sekvencijalne naredbe
elsif USLOV2 then
-- sekvencijalne naredbe
elsif USLOV3 then
-- sekvencijalne naredbe
else
-- sekvencijalne naredbe
end if;
-- uslovi 1,2,3 se mogu preklapati
```

# if - primer

- Uslovi se proveravaju po redosledu pojavljivanja
- Uslovi mogu da se preklapaju

```
library IEEE;
use IEEE.std_logic_1164.all;

entity IF_PRIMER is
    port (
        A,B,C,S : in  std_logic_vector(3 downto 0);
        OP      : out std_logic_vector(3 downto 0)
    );
end IF_PRIMER;

architecture RTL of IF_PRIMER is
begin
    IF_PR: process (all)
    begin
        if (S = "0000") then      -- 1
            OP <= A;
        elsif (S <= "0101") then -- 2
            OP <= B;
        else
            OP <= C;
        end if;
    end process IF_PR;
end RTL;
```

```
if (S <= "0101") then    -- 2
    OP <= A;
elsif (S = "0000") then -- 1
    OP <= B;
...
```

* Šta se dobija zamenom uslova 1 i 2?

# if - organizacija koda

- `if` može sadržati više provera
- `if` može sadržati drugu `if` naredbu
- Voditi računa o uslovima koji se proveravaju sa `if`

```
if (USL1 = '1') and (USL2 = '1') then
    -- USL1 = '1', USL2 = '1'
else
    -- USL1 = '1', USL2 = '0' or
    -- USL1 = '0', USL2 = '1' or
    -- USL1 = '0', USL2 = '0'
end if;
```

```
if USL1 = '1' then
    -- USL1 = '1'
elsif USL2 = '1' then
    -- USL1 = '0', USL2 = '1'
else
    -- USL1 = '0', USL2 = '0'
end if;
```

```
-- isti uslov
if (USL1 and USL2) = '1' then
...
```

- VHDL2008: `std_logic.'1' = true`

```
if USL1 = '1' then
    if USL2 = '1' then
        -- USL1 = '1', USL2 = '1'
    else
        -- USL1 = '1', USL2 = '0'
    end if;
else
    -- USL1 = '0'
end if;
```

```
-- Los kod
if USL1 = '1' and USL2 = '1' then
...
```

```
-- vhdl2008
if USL1 and USL2 then
    ...
```

# case

```
case izraz is
    when VREDNOST_1 =>
        -- jedna vrednost
        <sekvencijalne naredbe>
    when VREDNOST_2 | VREDNOST_4 =>
        -- lista vrednosti
        <sekvencijalne naredbe>
    when VREDNOST_M to VREDNOST_N =>
        -- opseg vrednosti
        <sekvencijalne naredbe>
    when others =>
        -- ostatak vrednosti
        <sekvencijalne naredbe>
end case;
```

# case - primer

- Vrednosti ne smeju da se preklapaju
- Sve moguće vrednosti treba pokriti
- Ukoliko se ne pokriju sve opcije, dobija se greška pri kompajliranju

```
entity CASE_PRIMER is
    port (
        A, B, C, S : in  std_logic_vector(2 downto 0);
        OP         : out std_logic_vector(2 downto 0)
    );
end CASE_PRIMER;
architecture RTL of CASE_PRIMER is
begin
    CASEPROC: process (all)
    begin
        case S is
            when "001" =>
                OP <= A;
            when "100" =>
                OP <= C;
            when "110" | "011" =>
                OP <= A;
            when others =>
                OP <= "0000";
        end case;
    end process CASEPROC;
end RTL;
```

# case - upotreba opsega

- `case` opseg može da se koristi sa diskretnim numeričkim tipovima podataka
  - Na primer, celi brojevi (eng. *integer*)

```
entity SLUCAJ_CEOBROJ is
    port ( A, B, C, SINT : in integer range 0 to 15;
           OP : out integer range 0 to 15);
end SLUCAJ_CEOBROJ;
architecture RTL of SLUCAJ_CEOBROJ is
begin
    SIG_INT: process (all)
    begin
        case SINT is
            when 0 to 4 =>
                OP <= B;
            when 5 =>
                OP <= C;
            when others =>
                OP <= A;
        end case;
    end process SIG_INT;
end RTL;
```

- Nizovi nemaju numeričke vrednosti asocirane sa njima.

```
case SIG_VEC is
    when "0101" =>              -- dozvoljeno
        ...
    when "0000" to "0100" =>    -- nije dozvoljeno
        ...
    when others =>              -- dozvoljeno
        ...
end case;
```

# Konkurentne i sekvencijalne naredbe

- Arhitektura sadrži procese
- Može postojati bilo koji broj procesa
- Procesi su međusobno konkurentni
- Komande unutar procesa se izvršavaju sekvencijalno
- Van procesa su dozvoljene konkurentne naredbe

```
architecture DIZAJN of PRIMER is
-- deklaracija signala
begin
    -------------------------------
    P1 : process                  --|
    begin                         --|
    -- sekvencijalne naredbe      --|
    end process P1;               --|
    -------------------------------
    -------------------------------
    P2 : process (all)            --|
    begin                         --|
        if SEL = '1' then         --|
            OP <= A;              --|
        else                      --|
            OP <= B;              --|
        end if;                   --|
    end process P2;               --|
    -------------------------------
    -------------------------------
    Y <= A and B;                 --|
    -------------------------------
end DIZAJN;
```

# Ekvivalentni procesi

Sekvencijalne naredbe:

- Unutar procesa
- Lista osetljivosti mora biti definisana
- Unutar procesa može postojati više dodela
- Kombinaciona logika i registri

Konkurentne naredbe:

- Van procesa - ekvivalent procesu
- Kompilator dodeljuje listu osetljivosti
- Više dodela signalima zahteva više konkurentnih naredbi
- Samo kombinaciona logika

Konkurentne dodele signalima imaju ekvivalentnu formu napisanu pomoću procesa.

```
architecture SEQ of PRIMER is
begin
    P1 : process (all)
    begin
        Y <= A and B;
    end process P1;
    ...
end RTL;
```

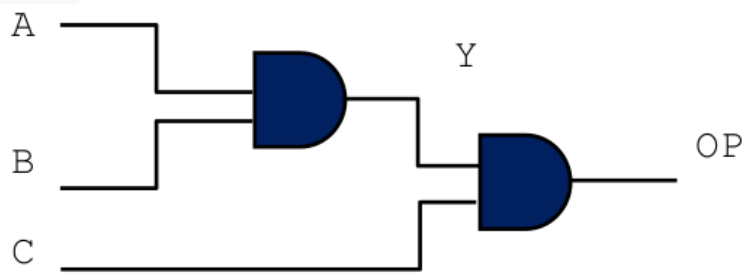
```
architecture CON of PRIMER is
begin
    Y <= A and B;
    ...
end RTL;
```

# Konkurentna dodela vrednosti

- Redosled nije bitan
- Dobijamo ono što smo napisali

```
architecture P1 of ANDS is
begin
    OP <= C and Y;
    Y  <= A and B;
end P1;
```

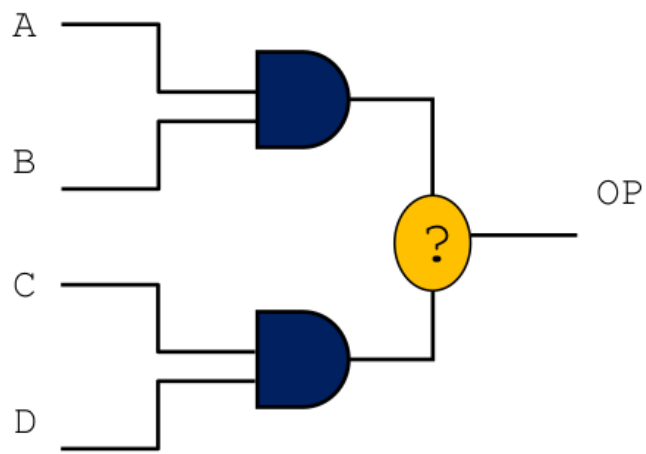
```
architecture P2 of ANDS is
begin
    Y  <= A and B;
    OP <= C and Y;
end P2;
```



# Više konkurentnih dodela

- Konkurentne dodele istom signalu su spojene (eng. *"wired together"*)
- Signali moraju biti *resolved* tipa
- `std_logic` i `std_logic_vector` su često korišćeni *resolved* tipovi
- Da bi se odredila krajnja vrednost potrebna je posebna funkcija
  - eng. *resolution function*
- Više konkurentnih dodela istoj vrednosti uglavnom rezultuje greškom
- Jedan od izuzetaka su trostatička kola

```
architecture CONC of MULT is
    signal OP : std_logic;
begin
    OP <= C and D;
    OP <= A and B;
    ...
end CONC;
```



# Proces - podsetnik

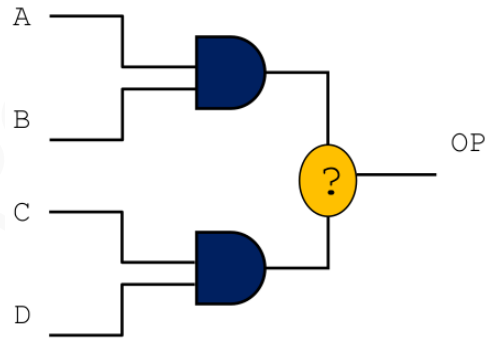
```
architecture DIZAJN of PRIMER is
-- deklaracija signala
begin
    -------------------------------
    P1 : process                  --|
    begin                         --|
    -- sekvencijalne naredbe      --|
    end process P1;               --|
    -------------------------------
    -------------------------------
    P2 : process (all)            --|
    begin                         --|
        if SEL = '1' then         --|
            OP <= A;              --|
        else                      --|
            OP <= B;              --|
        end if;                   --|
    end process P2;               --|
    -------------------------------
    -------------------------------
    Y <= A and B;                 --|
    -------------------------------
end DIZAJN;
```

- Procesi se međusobno izvršavaju konkurentno
- Unutar procesa se naredbe izvršavaju sekvencijalno
- Proces se izvršava kada se desi događaj na signalu iz liste osetljivosti
- Signali se ažuriraju pri izlasku iz procesa

# Više sekvencijalnih dodela

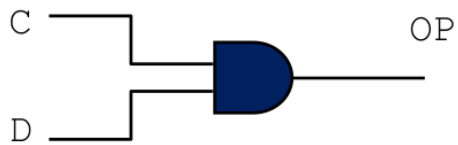
Za više konkurentnih dodela istom signalu je potrebna funkcija rezolucije.

```
architecture CONC of MULT is
    signal OP : std_logic;
begin
    OP <= C and D;
    OP <= A and B;
    ...
end CONC;
```




Signal ima vrednost poslednje dodele u slučaju više sekvencijalnih dodela istom signalu.

```
architecture SEQ of MULT is
    signal A, B, C, D, OP : std_logic;
begin
    SEQPROC : process (all)
    begin
        OP <= A and B;
        OP <= C and D;
    end process SEQPROC;
end SEQ;
```



# Više sekvencijalnih dodela - primer

- Signal uzima vrednost poslednje dodele
- Signalu se dodeljuje vrednost po izlasku iz procesa
- Primeri su ekvivalentni
- Drugi primer ima specifičan način kodovanja:
  - Ukoliko je podrazumevana vrednost istovremeno i najčešća, simulacija je efikasnija
  - Podrazumevana vrednost je definisana i ovaj način kodovanja pomaže pri izbegavanju generisanja leč kola pri sintezi.

```
PROC1 : process (all)
begin
    if SEL = '1' then
        OP <= A;
    else
        OP <= B;
    end if;
end process PROC1;
```

```
PROC2 : process (all)
begin
    OP <= B; -- podrazumevana vrednost
    if SEL = '1' then
        OP <= A;
    end if;
end process PROC2;
```

# Uslovne dodele signalima

- `when` konkurenta verzija `if` naredbe
  - Samo jedna dodela
  - Bezuslovna `else` grana
- Ekvivalent procesu koji sadrži samo jednu `if` naredbu

```
architecture USE_IF of BRANCH is
begin
    IFPROC : process (all)
    begin
        if (T > "0101") then
            OP <= A;
        elsif (T < "0101") then
            OP <= B;
        else
            OP <= C;
        end if;
    end process IFPROC;
end USE_IF;
```

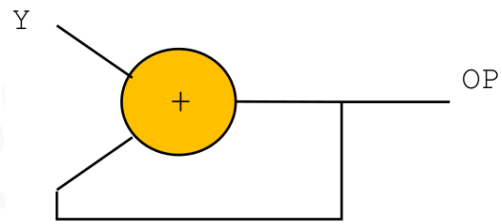
```
architecture USE_WHEN of BRANCH is
begin
    OP <= A when T > "0101" else
          B when T < "0101" else
          C;
end USE_WHEN;
```

- Ne može se koristiti unutar procesa!

# Zero-Delay Feedback Loops

## Ne raditi!

```
OP <= OP + Y;

-- ekvivalentna logika

process (all)
begin
    OP <= OP + Y;
end process;
```




- Konkurentna dodela vrednosti koja sadrži povratnu spregu bez kašnjenja je dozvoljena ali obično ne predstavlja smislenu logiku
- Za proces:
  - Događaj na signalu `Y` okida proces i `OP` se ažurira
  - `OP` je u listi osetljivosti i ponovo okida proces ...
  - ... što ponovo dodeljuje novu vrednost u `OP`
  - proces se ponavlja u nedogled
- Simulator radi ali vreme simulacije se ne pomera

# LAB

1. Lab1 - ALU
2. Lab2 - BCD sabirač