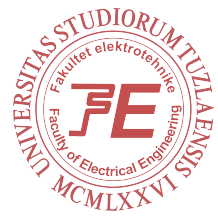
UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE



Jezici za opis hardvera - HDL

Dr. Sc. Asmir Gogić, vanr. prof.

Tuzla, 2020.

Jezici za opis digitalnih sistema

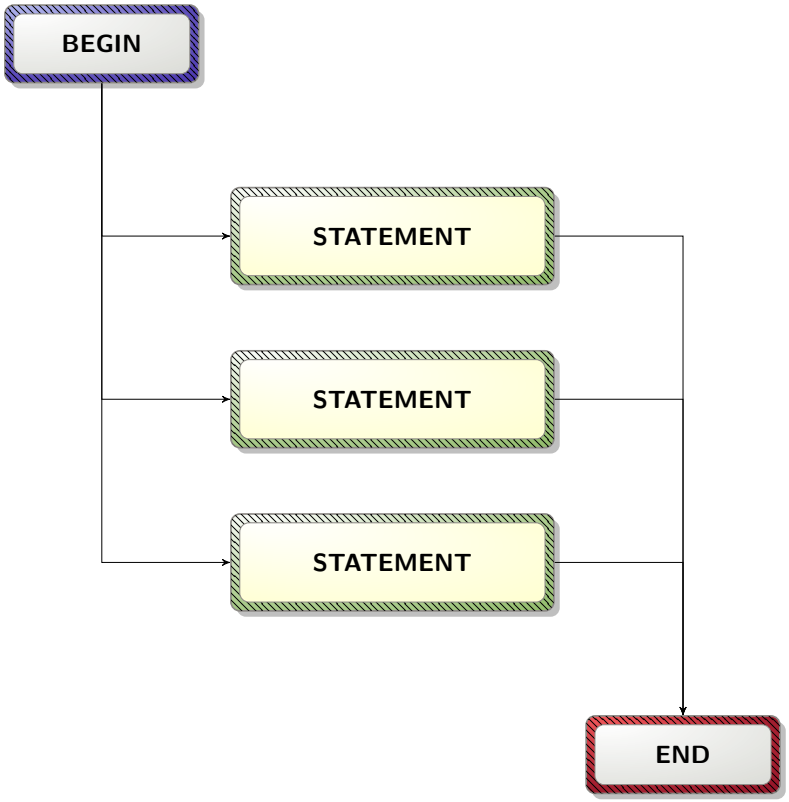
- Generalno, digitalni logički sklopovi se mogu opisati na jedan od sljedećih načina:
  - ***Logička shema*** - sastoji se od međusobno povezanih simbola logičkih kola. Jednostavna za razumijevanje i analizu funkcije sklopa ali nije pogodna za opis sistema sa velikim brojem logičkih kola.
  - ***Bool-ove jednačine*** - predstavljaju formalan način za definiranje veze između logičkih varijabli. Pogodan metod za opis logičkih sklopova ako se radi o dizajnu na "papir".
  - ***Tablice istinitosti*** - univerzalni način opisivanja ponašanja (*behaviour*) logičkog sklopa.
  - ***Vremenski dijagrami*** - praktičan način opisa ponašanja sekvencijalnog sklopa. Nepraktični za opis logičkih sklopova u CAD[1] alatima.
  - ***HDL - Hardware Description Language*** - predstavlja efikasan metod za opis funkcionalnosti i ponašanja logičkog sklopa baziranog na programabilnom hardveru. Najpoznatiji HDL jezici su: Verilog HDL, VHDL, Verilog-A, VHDL-AMS...

[1] Computer Aided Design

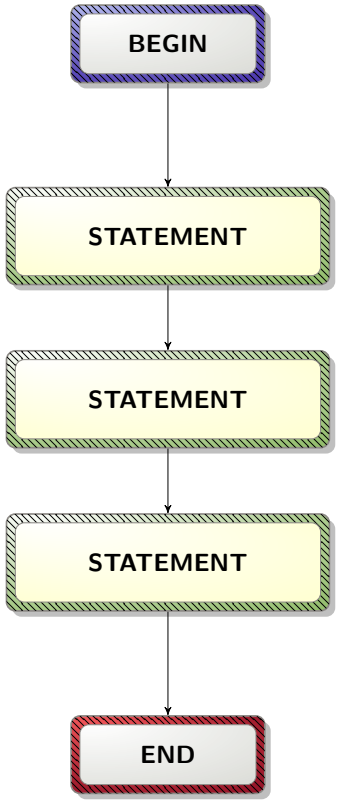
VHDL

- ***VHDL*** - *VHSIC - very high-speed integrated circuit - hardware description language*.
- Rezultat DARPA istraživčkih projekata iz 70 i 80 godina.
- Standardiziran od strane IEEE 1987 godine kao ***IEEE Standard 1076*** (poznat kao VHDL 87)
  - ***IEEE 1076-1987*** - prva standardizirana verzija VHDL-a (v7.2 USAF).
  - ***IEEE 1076-1993*** - unaprjeđenja na bazi povratnih informacija korisnika, najčešće korištena verzija VHDL-a. Revizija svakih 5 godina!
  - ***IEEE 1076-2000*** - *protect* tipovi
  - ***IEEE 1076-2002*** - relaksiranje pravila vezanih za buffer-e i port-ove.
  - ***IEEE 1076-2008*** - integracija PSL-a (*Property Specification Language*) koji omogućuje sintaksu za provjeru očekivanog ponašanja.
- ***HDL/VHDL*** za razliku od ostalih programskih jezika je kao što je C, C++, Java i dr., koji su sekvencijalni ***po pitanju izvršavanja instrukcija***, ***je konkurentan***.
- HDL/VHDL instrukcije se izvršavaju paralelno neovisno o veličini koda.

VHDL - HDL paradigma



CONCURENT FLOW

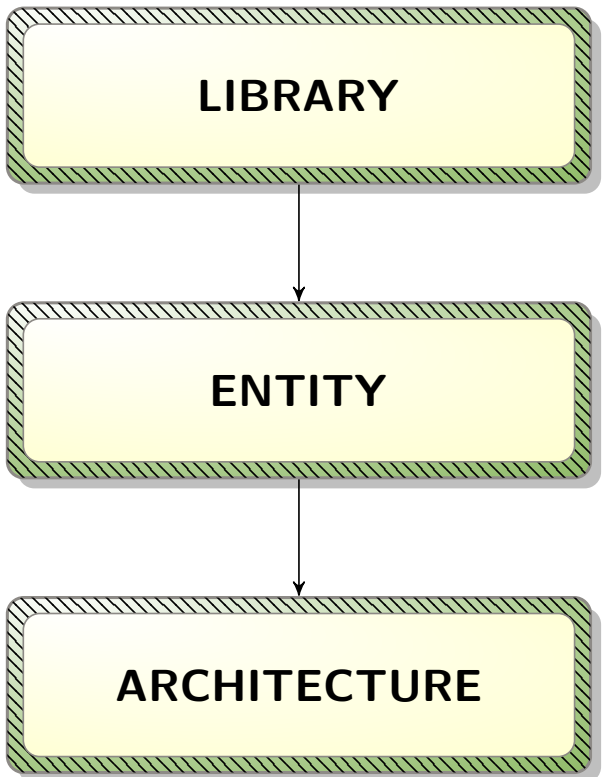


SEQUENTIAL FLOW

VHDL - entity & architecture

- ***Dizajn sistema*** na bazi pristupa tvz. ***"crne kutije"*** implicira ***hijerarhijsku strukturu***.
- Hijerarhijska struktura osigurava grupisanje jedinica/funkcionalnosti u više apstraktne cjeline.
- Ovakav pristup osigurava pojednostavljene arhitekture sistema i re-upotrebu koda.
- Modelovanja sistema na bazi hijerahijskog pristupa definira dvije cjeline: informacije koje ulaze u "kutiju" i funkcionalnost "kutije".
- U VHDL-u crnu kutiju predstavlja ***entity*** a informacije koje ulaze u kutiju se referiraju kao ***architecture***.
- VHDL koncept ***entity*** osigurava mehanizme za modelovanje funkcionalnosti digitalnih sklopova na način da opisuje interface (način i mehanizme interakcije sa vanjskim svijetom).
- Drugačije rečeno ***entity*** definira ulaze i izlaze digitalnog sistema.

VHDL - struktura koda



BIBLIOTEKE

INTERFEJS SISTEMA

IMPLEMENTACIJA SISTEMA

VHDL - entity & architecture

```
entity my_entity is
port(
  inputa:  in std_logic;
  outputy: out std_logic;
  bidir:   inout std_logic );
end my_entity;
```

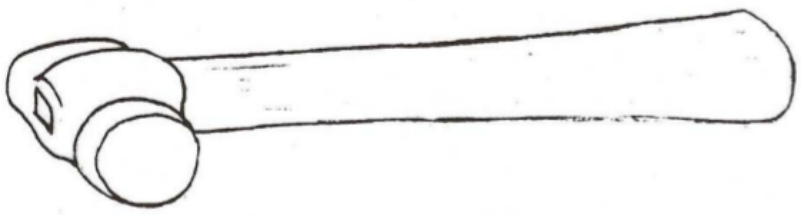
- Svaki entitet ima korisnički definirano ime (identifikator) u datom primjeru *my_entity* te listu interfejsa tj. portova.
- Svaki port sadrži ulazne/izlazne signale kao i njihove tipove.
- Opšta forma definiranja IO signala je

```
sig_name1, sig_name2:  mode datatype;
```

- ***mode*** može biti ***in***, ***out*** ili ***inout***.
- VHDL definira veliki broj tipova podataka ***datatype*** a jedan od njih je i ***std_logic*** koji ima devet stanja.
- VHDL je programerski jezik striktno/jako definiranog tipa, drugačije rečeno svaki objekat mora imati tip podatka i samo predefinirane vrijednosti i operacije mogu biti aplicirane na datim objektom.
- Pored port-ova u okviru entiteta moguće je definirati i ***generics*** koji predstavljaju mehanizme za prosljeđivanje specifičnih informacija u entitet bez definiranja smjera.

VHDL - a strongly typed language

VHDL: a strongly typed language



VHDL - library

- library ***ime_biblioteke*** - definira listu dostupnih paketa unutar kojih su definirani tipovi i podtipovi podataka, vrijednosti koje tipovi mogu poprimiti, operatori i operacije.
- Uvijek je ***prva linija VHDL*** koda a neke od dostupnih biblioteka su: ***ieee***, ***work***, ***std*** i dr.
- Koji paketi će biti analizirani tokom kompajliranja u cilju pronalaženja definicija tipova podataka, operatora ili operacija navodi se sa ***use imebiblioteke.imepaketa.all***
- ... u toku predavanja upoznat ćemo se sa sljedećim paketima IEEE biblioteke:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;
```

VHDL - *library ieee.std_logic_1164*

- Definira sljedeće tipove podataka:
  - ***std_ulogic*** - unresolved tip
  - ***std_ulogic_vector***
  - ***std_logic*** - industry standard logic type koji je resolved. Rezolucija tipa se izvodi na bazi tablice u nastavku
  - ***std_logic_vector***
- ***unresolved*** type - ako su na liniji/vodu (drive) prisutna dva signala, određivanje finalnog stanja nije moguće direktno, potrebna je neka funkcija u suprotnom desit će se greška.
- ***resolved*** type - moguće upisati bilo koju podržanu vijednost i rezultat će uvijek biti poznat.
- ... definirana su stanja ***std_logic*** tipa podataka:
  - *'U'*, – Uninitialized/undefined value - default value.
  - *'X'*, – Strong drive unknown, impossible to determine this value/result.
  - *'0'*, – Strong drive 0 (drive to GND)
  - *'1'*, – Strong drive 1 (drive to VDD)
  - *'Z'*, – High Impedance (floating, undriven, tri-state)
  - *'W'*, – Weak drive Unknown
  - *'L'*, – Weak drive 0 (resistive pull-down)
  - *'H'*, – Weak drive 1 (resistive pull-up)
  - *'-'* – Don't Care (for synthesis minimization)

VHDL - *library ieee.std_logic_1164*

- Unresolved vs. resolved type.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity UnresolvedTb is
end entity;

architecture sim of UnresolvedTb is
    signal Sig1 : std_ulogic := '0';
begin

    -- Driver A
    Sig1 <= '0';

    -- Driver B
    Sig1 <= '1' after 20 ns;

end architecture;
```

ERROR

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity ResolvedTb is
end entity;

architecture sim of ResolvedTb is
    signal Sig1 : std_logic := '0';
begin

    -- Driver A
    Sig1 <= '0';

    -- Driver B
    Sig1 <= '1' after 20 ns;

end architecture;
```

PASS

VHDL - *library ieee.std_logic_1164*

- Resolved type - resolution

```
CONSTANT resolution_table : stdlogic_table := (
--      ---------------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -         |    |
--      ---------------------------------------------------------
        ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), --  | U |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), --  | X |
        ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), --  | 0 |
        ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), --  | 1 |
        ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), --  | Z |
        ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), --  | W |
        ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), --  | L |
        ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), --  | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  --  | - |
    );
```

VHDL - *library ieee.std_logic_1164*

- Primjer razrješavanja rezultata za ***resolved type*** na prethodnom primjeru

```
----------------------------------------------------------------
|  U    X    0    1    Z    W    L    H    -          |    |
----------------------------------------------------------------
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), --  | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), --  | X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), --  | 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), --  | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), --  | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), --  | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), --  | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), --  | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  --  | - |
```

VHDL - Pravila

- Nije osjetljiv na ***male/velike karaktere***.

```
Dout <= A and B;
```

je isto što i

```
doUt <= a AnD b;
```

- Nije osjetljiv na ***prazna mjesta (space & tab)***

```
nQ <= In_a or In_b;
```

je isto što i

```
nQ    <= In_a or          In_b;
```

- ***Jednolinijski komentar*** predstavlja svaku liniju koda koja počinje sa dva karaktera *'-'* ili višelinijski komentar sa */* */* parom karaktera.

```
-- Jednolinijski komentar
a_reg <= b_reg
/* ovo je viselinijski kometar
koji se zavrsava u drugom redu */
```

VHDL - Pravila

- Delimiter male zagrade se koriste za grupisanje logičkih izraza:

```
if x =    0    and y =    0    or z =    1    then
  outa <= b and c;
  outb <= b or c;
end if;
```

je isto što i

```
if (((x =    0   ) and (y =    0   )) or (z =    1   ))
  outa <= b and c;
  outb <= b or c;
end if;
```

VHDL - Pravila

- Identifikatori predstavljaju imena koja dodijeljujemo signalima, vektorima signala, varijablama ili funkcijama u VHDL-u.
- Generalno, ***identifikatori trebaju biti intuitivni*** tj. treba ih definirati tako da instant odražavaju njihovu namjenu.
- Neophodno je ***pridržavati se*** određenih ***pravila*** prilikom ***kreiranja identifikatora***:
  - mogu biti proizvoljne dužine,
  - sadržavaju samo karaktere A-Z, a-z, 0-9 i '_',
  - moraju počinjati karakterom,
  - ne smiju se završavati sa karakterom '_' i ne smiju imati dva sukcesivna karaktera '_'.

VHDL - Pravila - identifikatori, ključne riječi

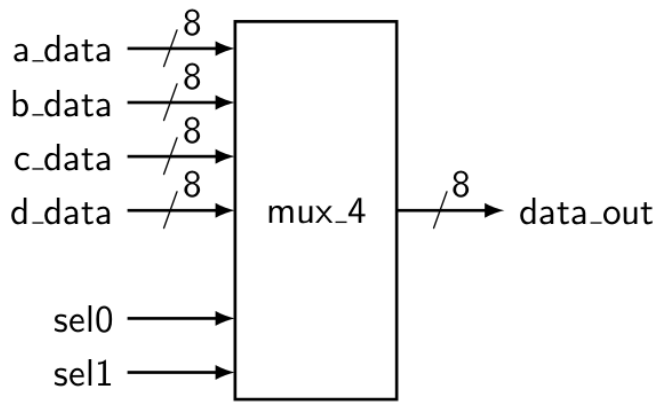
Valid identifies.	Invalid identifies.
data_bus --descriptive name	3Bus_val -- begins with a number chr
WE --classic "write enable"	DDD -- not self commenting
div_flag --real winner	mid_$num -- illegal character
port_A --provides some info	last__val-- consec. underscores
in_bus --input bus	str_val_ -- ends with underscore
clk --classic clock	in -- uses VHDL reserved word
clk_in	@#$%% -- total garbage
clk_out	it_sucks -- try to avoid
mem_read_data	Big_vAlUe-- valid but ugly
--	pa -- possibly lacks meaning
--	sim-val -- illegal character (dash)
--	DDE_SUX -- no comment

- Svaki jezik pa i VHDL ima ključne riječi koje se ne mogu koristiti kao identifikatori.

access	after	alias	all	attribute	block
body	buffer	bus	constant	exit	file
for	function	generic	group	in	is
label	loop	mod	new	next	null
of	on	open	out	range	rem
return	signal	shared	then	to	type
until	use	variable	wait	while	with

Entity - bus/bundle

- Digitalni sistemi veoma često operiraju sa grupama ulaznih signala koje su udružene u jednu cjelinu koju nazivamo sabirnica/bus/bundle.
- Vrijednosti pojedinih signala sabrinice kao zasebna cjelina ne predstavljaju korisnu informaciju ali kao cjelina u sabirnici mogu predstavljati zapis jednog byte (8 bitna sabirnica), ASCII karakter, word-a (32-bitna sabirnica), itd.
- VHDL osigurava vektor tip podatka ***std_logic_vector*** za opis signala sabirnice. Signali u sabirnici se definiraju sa ***to*** i ***downto*** ključnim riječima.
- Ako želimo da se na lijevoj strani definicije sabirnice nalazi najznačajniji bit onda koristimo notaciju ***downto*** u suprotnom ***to***
- Za hipotetički digitalni sklop na slici definirati *interface* koristeći VHDL jezik.



Dodjeljivanje vrijednosti i logički operatori

- Generalno u okviru VHDL koda dodjeljivati ćemo vrijednosti varijablama i signalima.
- Neposredno prije upotrebe signala ili varijabli iste je neophodno deklarisati.
- Deklaracija IO signala interface-a izvodi se u okviru ***entity*** segmenta VHDL koda.
- Varijable i signale deklarišemo u okviru *architecture* segmenta.
- ***Dodjeljivanje vrijednosti signalu*** se izvodi sa operatorom ***<=*** a ***varijabli*** sa ***:=***
- ***std-logic-vector*** i ***std-logic*** tipovi podataka podržavaju sljedeće logičke operatore ***and***, ***or***, ***not***, ***xor***, ***xnor***, ***nand*** i ***nor***.
- Logički operatori imaju isti prioritet prilikom kompajliranja te je stoga potrebno koristiti delimitere (zagrade) kako bi se naglasio redoslijed izvršavanja.

primjer kreiranja vektora signala tipa ***std_logic*** i inicijalizacije vektora vrijednostima 0.

```
yout:  out  std_logic_vector(7 to 0) := (others => '0');
```

primjer kreiranja signala tipa ***unsigned*** i incijalizacije sa vrijednošću 0x00

```
signal tdata: unsigned(7 downto 0):=x"00";
```

Dodjeljivanje vrijednosti

- ***architecture*** - arhitektura VHDL koda definira ***funkcionalnost sistema/modula***. Drugačije rečeno šta sistem radi sa ulaznim signalima.
- Realizacija sistema u bloku ***architecture*** direktno će se odražavati na činjenicu kako će se stvarni sistem sintetizirati.

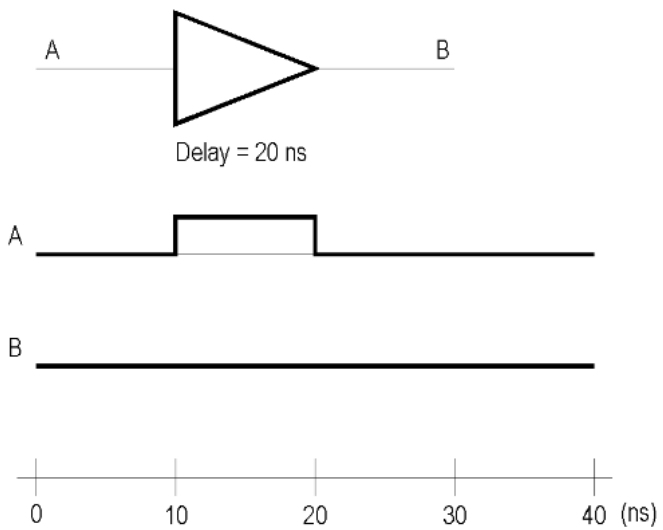
```
library ieee;
use ieee.std_logic_1164.all;
-- ENTITY
entity circuit1 is
  port (
    A,B,C : in std_logic;
    F : out std_logic);
end circuit1;
-- ARCHITECTURE
architecture circuit1_arc of circuit1 is
  signal sig_1: std_logic; -- definicija signala
  variable var_1: integer; -- definicija varijable

begin
  F <= not (A and B and C); -- dodjeljivanje vrijednosti signalu
  sig_1 <= A;
  var_1 := 34; -- dodjeljivanje vrijednosti varijabli
end circuit1_arc;
```

Dodjeljivanje vrijednosti

- ***Dodjeljivanje vrijednosti*** signala može biti ***odloženo predefiniran broj vremenskih jedinica*** unutar procesa.

```
signal <= expression [after delay];
b <= a after 20 ns;
```

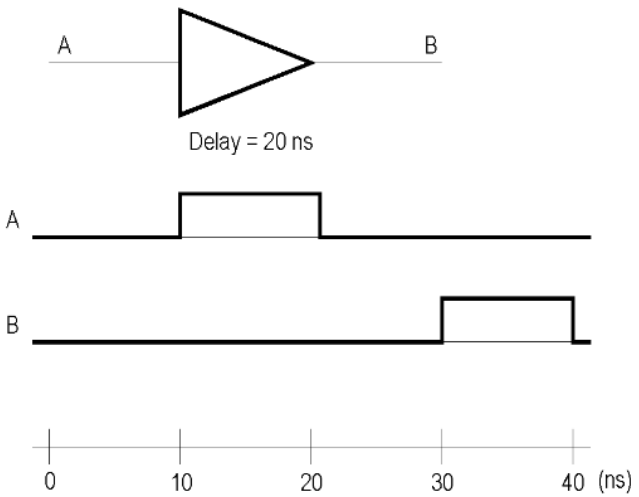
- Korištenjem prethodnog principa možemo jednostavno modelovati "inerciju" komponenti. U sljedećem primjeru, činjenica da ulazni impuls buffer-a je kraći od modelovane "inercije" buffer-a ima za posljedicu da dati impuls neće biti detektovan.



odjeljivanje vrijednosti

- ***Transportno kašnjenje*** (vrijeme propagacije signala kroz komponentu) možemo modelovati na sljedeći način

```
b <= transport a after 20 ns;
```



If statement

- ***IF*** iskaz selektuje jedan ili više sekvenci iskaza za izvršenje na bazi evaluacije uslova u datom iskazu. Opšti oblik je

```
if condition then statement_sequence
  [elsif condition then statement_sequence...]
  [else statement_sequence]
end if;
```

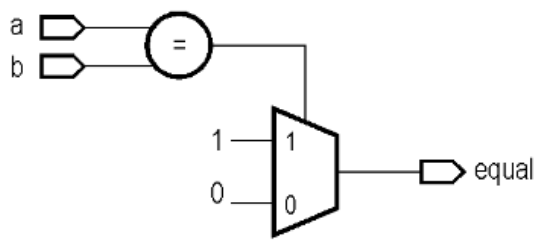
- Uslov ***IF*** iskaza je bool tip koji može biti TRUE ili FALSE.

```
process (a, b)
begin
  if a = b then
    result <= 0;
  elsif a < b then
    result <= -1;
  else
    result <= 1;
  end if;
end process;
```

If statement

```
library ieee;
use ieee.numeric_bit.all;
entity comp is
  port (a, b: in unsigned (7 downto 0);
  equal: out bit);
end comp;
architecture functional of comp is
begin
  process (a, b)
  begin
    if a = b then
      equal <= '1';
    else
      equal <= '0';
    end if;
   end process;
end functional;
```

- Implementacija komparatora dva 8-bitna signala



If statement

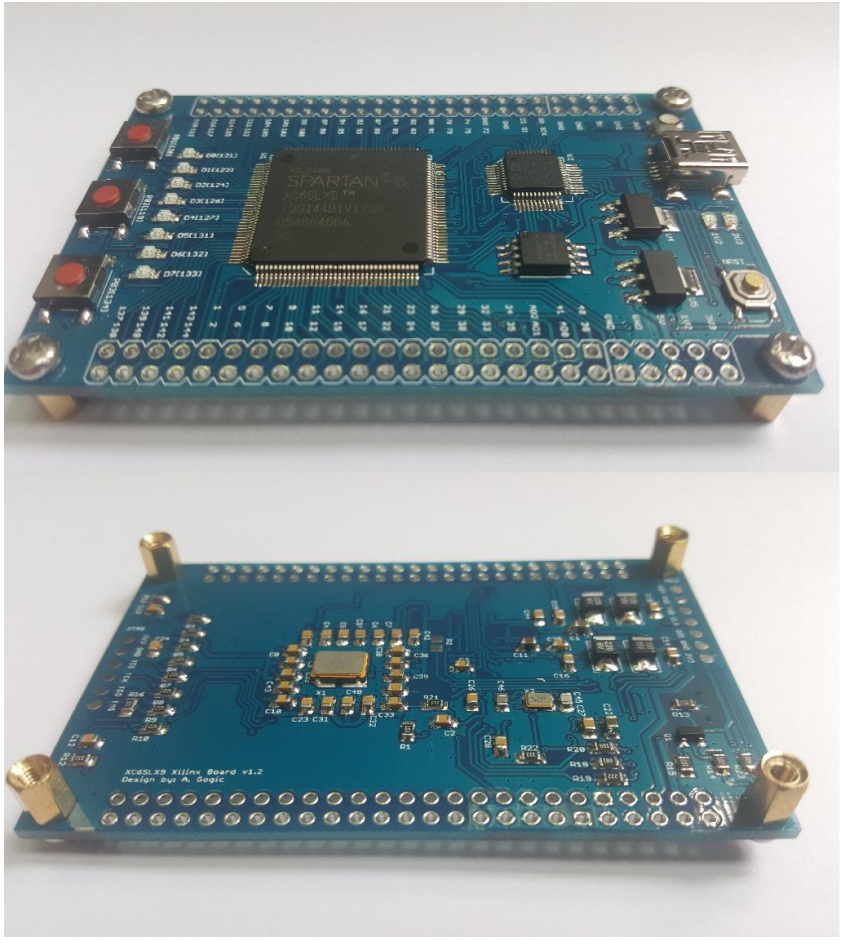
- Naredna dva primjera implementiraju istu funkcionalnost

```
process (a, b, enable)
  begin
    z <= a;
    if enable = '1' then
      z <= b;
    end if;
end process;
```

```
process (a, b, enable)
  begin
    if enable = '1' then
      z <= b;
    else
      z <= a;
    end if;
end process;
```

FPGA razvojna ploča

- ***FPGA:***
  Xilinx Spartan VI XC6SLX9
- ***Loader:***
  ARM Cortex M3
  STM32F103 MCU
- ***Memory:***
  SST25VF032B
- 50MHz clock
- 3x Tactile switches
- 8x LED's
- 5V, 3V3 & 1V8 power rail's



- Xilinx ISE + Custom USB bitstream loader

FPGA Spartan 6 familija

Device	Logic Cells	Total Slices	SLICEMs	SLICELs	SLICEXs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Registers (Kb)	Number of Flip-Flops
XC6SLX4	3,840	600	300	0	300	2,400	75	38	4,800
XC6SLX9	9,152	1,430	360	355	715	5,720	90	45	11,440
XC6SLX16	14,579	2,278	544	595	1,139	9,112	136	68	18,224
XC6SLX25	24,051	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45	43,661	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75	74,637	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100	101,261	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150	147,443	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304
XC6SLX25T	24,051	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45T	43,661	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75T	74,637	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100T	101,261	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150T	147,443	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304

FPGA Spartan 6 - DSP SLICES

Device	Total DSP48A1 Slices per Device	Number of DSP48A1 Columns per Device	Number of DSP48A1 Slices per Column
XC6SLX4	8	1	8
XC6SLX9	16	1	16
XC6SLX16	32	2	16
XC6SLX25	38	2	18/20
XC6SLX45	58	2	30/28
XC6SLX75	132	3	44/40/48
XC6SLX100	180	4	48/44/40/48
XC6SLX150	180	4	48/44/40/48
XC6SLX25T	38	2	18/20
XC6SLX45T	58	2	30/28
XC6SLX75T	132	3	44/40/48
XC6SLX100T	180	4	48/44/40/48
XC6SLX150T	180	4	48/44/40/48

Primjer 1

PRIMJER 1

Napisati VHDL kod koji će implementirati sljedeću logičku funkciju sa tri logičke promjenljive $F = AB + \overline{B}C$. Kao ulaze koristi tastere na IO pinovima P119, P120, P134 te pinove P121, P123, P124 kao indikatore ulaza i pin P126 kao izlaz logičke funkcije.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
        Port (
                BTN0: in std_logic;
                BTN1: in std_logic;
                BTN2: in std_logic;
             LED0: out std_logic := '0';
             LED1: out std_logic := '0';
             LED2: out std_logic := '0';
             YOUT: out std_logic := '0');
end top;

architecture arc_top of top is

begin
        LED0 <= BTN0;
        LED1 <= BTN1;
        LED2 <= BTN2;
        YOUT <= (BTN0 and BTN1) or ((not BTN1) and BTN2);

end arc_top;
```

Primjer Dz

PRIMJER 2

Napisati VHDL kod koji će implementirati funkcionalnost digitalnog sklopa koji poredi dva dvo-bitna binarna broja. Izlaz digitalnog sklopa je 1 ako su binarni brojevi jednaki u suprotnom je u stanju 0.

PRIMJER 3

Napisati VHDL kod koji će implementirati funkcionalnost digitalnog sklopa koji vrši sabiranje dva dvo-bitna broja. Izlaz digitalnog sklopa su 3 bita Y0, Y1 i Y2.

PRIMJER 4

Napisati VHDL kod koji će implementirati funkcionalnost digitalnog sklopa koji vrši oduzimanje dva dvo-bitna broja. Izlaz digitalnog sklopa je Y i prijenos C.

Literatura

- RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability, 1st Editon by Pong P. Chu, 2006.
- Digital Systems Design Using VHDL 2 nd Edition, by Charles H. Roth, Jr. and Lizy Hurian John, Thomson, 2007.
- The Designer's Guide to VHDL, Third Edition, Peter J. Ashenden, 2008.