Sadržaj prezentacije

2. UVOD U VHDL

- Organizacija VHDL koda:
- Library
- Entity
- Arhitecture
- Stilovi projektovanja:
- Funkcionalni opis
- Strukturni opis
- Projektne jedinice
- Simulacija VHDL koda
- Sinteza VHDL koda
- Konfiguracija entitea i arhitekture
- Uloga VHDL-a u procesu projektovanja

VHDL

- VHDL - standardni jezik za opis digitalnih kola i sistema.
- VHDL standard usvojen 1987. god. (IEEE 1076).
- Revidiran i trenutno aktuelni VHDL standard, IEEE 1164, usvojen 1993. godine.
- Skraćenica od *VHSIC Hardware Description Language.*
- *VHSIC* skraćenica od *Very High Speed Integrated Circuits* (Integrisana kola veoma velike brzine rada)

VHDL

Prvobitna namena:

- Jezik za **dokumentaciju**
- Jezik za **simulaciju**

Savremena primena:

- **Sinteza hardvera**
- Obiman i složen jezik, ali
- za sintezu, bitan je samo jedan manji deo mogućnosti VHDL jezika.
- sredstvo za unos dizajna u **CAD** sisteme za sintezu hardvera.
- Jednom napisan i verifikovan VHDL kod može se iskoristiti za realizaciju opisanog kola u **različitim** implementacionim tehnologijama, kao što su PLD ili **ASIC** kola.

Programski jezik VS jezik za opis hardvera

Programski jezik:

- Opis algoritma
- Sekvencijalni model izračunavanja
- Kompajlira se u mašinski program radi izvršenja na računaru

Jezik za opis hardvera:

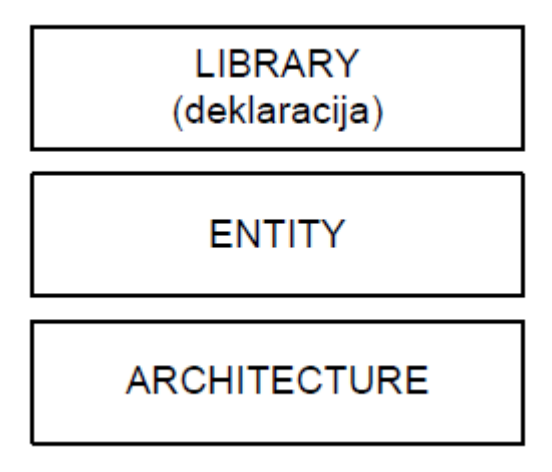
- Opis algoritma
- Opis strukture digitalnog sistema
- **Konkurentni** model izračunavanja
- Kompajlira se radi simulacije u RTL simulatoru
- Sintetiše se radi implementacije u hardveru

Termini:

- **Program** - opis algoritma u programskom jeziku
- **Kod** – opis algoritma ili strukture u jeziku za opis hardvera

Organizacija VHDL koda

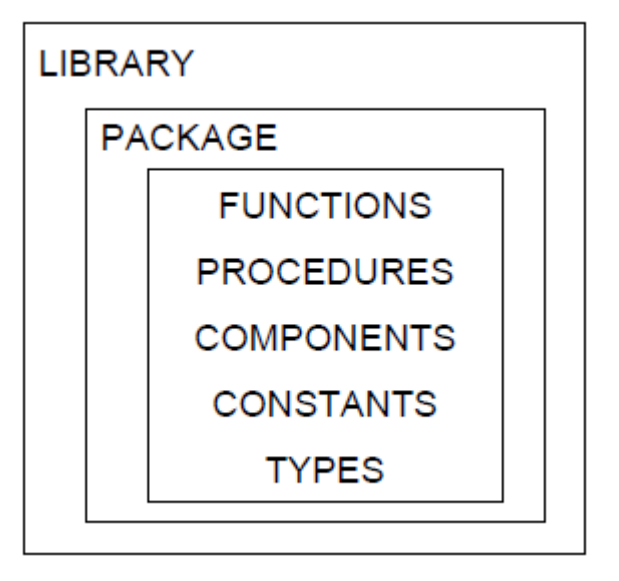
- Svaki celovit VHDL kod, tj. onaj koji se može simulirati ili sintetizovati, sastoji se iz tri sekcije:



LIBRARY

- LIBRARY (biblioteka)
- Kolekcija često korišćenih delova VHDL koda.
- Jednom se piše više puta koristi
- Sadrži pakete (PACKAGE), a paketi sadrže:

- Funkcije (FUNCTION)
- Procedure (PROCEDURE)
- Komponente (COMPONENT)
- Konstante (CONSTANT)
- Tipove podataka (TYPE)



LIBRARY

- Biblioteka se uključuje u projekat pomoću dve naredbe:

LIBRARY ime_biblioteke;

USE ime_biblioteke.ime_paketa.delovi_paketa;

- LIBRARY - definiše ime biblioteke
- USE – definiše delove biblioteke koje želimo da koristimo

Bar tri paketa iz tri različite biblioteke su neophodna u svakom projektu:

- *std_logic_1164* (iz biblioteke *ieee*)
- *standard* (iz biblioteke *std*) i
- *work* (iz biblioteke *work*)

```
LIBRARY IEEE;                    -- tačka-zarez (;) označava
USE IEEE.STD_LOGIC_1164.ALL;     -- kraj naredbe ili deklaracije
LIBRARY STD;                     -- dupla crta (--) označava komentar
USE STD.STANDARD.ALL;
LIBRARY WORK;
USE WORK.ALL;
```

LIBRARY

- Standardne biblioteke:
- **STD** je biblioteka resursa (standardni tipovi podataka i sl.).
- **WORK** je projektna biblioteka (za smeštanje fajlova projekata)
- **IEEE** biblioteka za sintezu, paketi :
- *std_logic_1164*: definiše višenivovske logičke sisteme
- *numeric_std*: podrška za aritmetiku
- Biblioteke *std* i *work* po automatizmu uključene u svaki projekat.
- VHDL kod za sintezu obično počinje linijama:

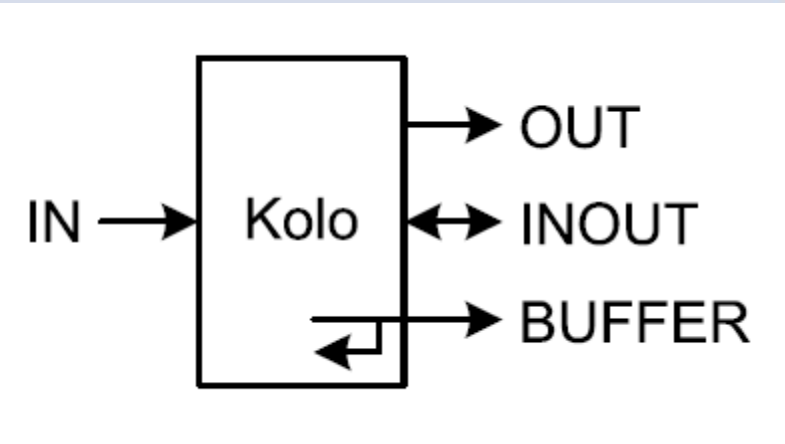
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY

- Sekcija *entity* definiše **ime kola koje se projektuje i imena i osnovne karakteristike njegovih ulaza i izlaza**, tzv. pinova, odnosno *portova* (*ports*).
- Sintaksa deklaracije entiteta je sledećeg oblika:

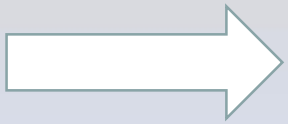
```
ENTITY ime_entiteta IS
PORT (
ime_porta : smer_signala tip_si
ime_porta : smer_signala tip_si
. . . );
END ime_entiteta;
```

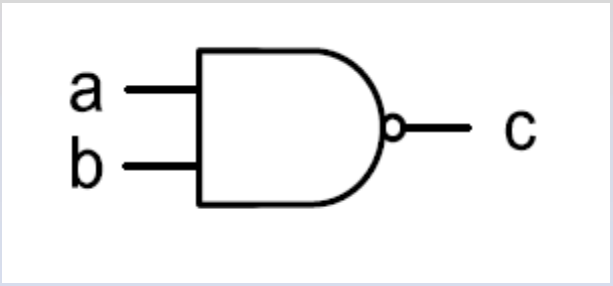


- Posredstvom portova kolo razmenjuje signale sa drugim kolima (entitetima) iz svog okruženja.
- Za svaki port se navodi njegovo **ime, smer i tip.**
- **Smer signala:** IN – ulaz, OUT – izlaz, INOUT – dvosmerni port (ulaz/izlaz), BUFFER - izlaz koji se može koristiti kao interni signal.
- **Tip signala**: BIT, STD_LOGIC, INTEGER . . .

ENTITY

```
1 ENTITY ni_kolo IS
2 PORT (a, b : IN BIT;
3 c : OUT BIT);
4 END ni_kolo;
5
```





- **Opisuje spoljni pogled na kolo, a ne funkciju kola**
- Više portova istog smera i tipa mogu biti deklarisani u istoj liniji
- Linije PORT konstrukcije završavaju se znakom (;), osim poslednje, nakon koje sledi zatvorena zagrada
- Ime entiteta može da sadrži slova, cifre i crtu za podvlačenje.
- Ograničenja: ne može početi cifrom, ne može biti ključna reč
- VHDL ne pravi razliku između velikih i malih slova

ARCHITECTURE

- **Arhitektura (ARCHITECTURE) – sadržati opis funkcionisanja (*ponašanja*) ili opis unutrašnje strukture kola.** Njena sintaksa je sledeća:

ARCHITECTURE ime_arhitekture OF ime_entiteta IS

[deklaracije]

BEGIN

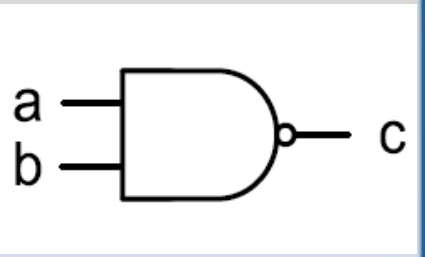
[kod]

END ime_arhitekture;

- **Uvek je pridružena jednom entitetu** (ime_entiteta)
- [deklaracije]: definiše interne signale i konstante
- [kod]: kod arhitekture
- Za pisanje koda arhitekture koriste se tzv. ***konkurentne*** naredbe.

ARCHITECTURE

```
1 ARCHITECTURE ni_funkcija OF ni_kolo IS
2 BEGIN
3 c <= a NAND b;
4 END ni_funkcija;
5
```



- Kolo obavlja NI operaciju (NAND) nad dva ulazna signala, *a* i *b*, i rezultat dodeljuje (**<=**) izlaznom pinu *c*.
- Naredba dodele se izvršava uvek kad se na nekom od signala *a* ili *b* desi **događaj**.
- *Događaj na signalu* - promene vrednosti signala.
- Naredba dodele je *senzitivna* na promenu vrednosti bilo kog signala s desne strane znaka <=.
- Ova naredba se ne izvršava samo jedanput, već uvek kad se na nekom od signala *a* ili *b* desi događaj.

ARCHITECTURE

- Iako je svaka arhitektura pridružena tačno jednom entitetu, entitetu se, s druge strane, može pridružiti jedna od eventualno nekoliko raspoloživih, različitih arhitektura, od kojih svaka na neki specifičan način opisuje isto kolo.
- Na primer, jedan, apstraktniji opis, može se koristiti za simulaciju, a drugi, detaljniji, za sintezu.
- Spoj entiteta i arhitekture se naziva *konfiguracija.*

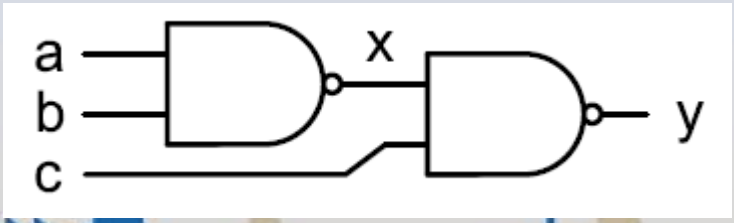
Stilovi projektovanja u VHDL-u

- **Funkcionalni** (ili bihejvioralni)
- Konkurentne naredbe (*dataflow,* tj. model toka podataka)
- Sekvencijalne naredbe (sekvencijalni model, kao kod programskih jezika)
- **Strukturni** (ekvivalentno blok dijagramu)
- **Mešovit** (kombinacija konkurentnih/sekvencijalnih naredbi i strukturnog opisa)

Funkcionalni opis - konkurentno izvršenje naredbi

```
1  ------------------------------------
2  ENTITY nand_mreza IS
3  PORT(a, b, c: IN BIT;
4  y : OUT BIT);
5  END nand_mreza;
6  ------------------------------------
7  ARCHITECTURE dataflow OF nand_mreza IS
8  SIGNAL x : BIT;
9  BEGIN
10 y <= x NAND c;
11 x <= a NAND b;
12 END dataflow;
13 ------------------------------------
```

Deklaracija internog signala



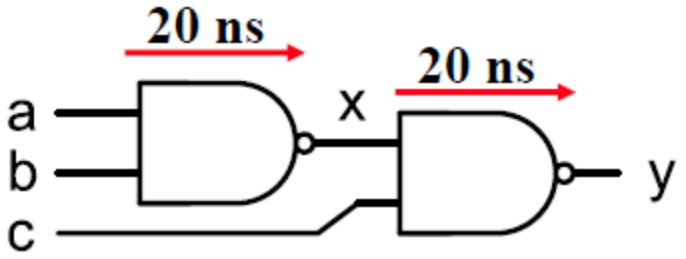
- Arhitektura sadrži konkurentni kod
- Naredbe se iniciraju promenama signala
- Redosled pisanja naredbi nije od značaja (isti efekat ima sledeći kod):

```
10 x <= a NAND b;
11 y <= x NAND c;
```

Modeliranje propagacionog kašnjenja

- Propagaciono kašnjenje predstavlja vreme koje protekne od trenutka promene vrednosti ulaznog signala do trenutka odgovarajuće promene vrednosti izlaznog signala.

```
1  ------------------------------------
2  ENTITY nand_mreza IS
3  PORT(a, b, c: IN BIT;
4  y : OUT BIT);
5  END simple_circuit;
6  ------------------------------------
7  ARCHITECTURE dataflow OF nand_mreza IS
8  SIGNAL x : BIT;
9  BEGIN
10 y <= x NAND c after 20 ns;
11 x <= a NAND b after 20 ns;
12 END dataflow;
13 ------------------------------------
```



Modeliranje propagacionog kašnjenja

Naredba dodele proširena klauzulom **after**:

x <= a NAND b **after** 20 ns;

δ - kašnjenje: beskonačno malo kašnjenje (δ>0)

x <= a NAND b;

je identična naredbi:

x <= a NAND b AFTER 0 ns;

Navođenje propagacionog kašnjenja nije dozvoljeno u kodu za sintezu!

Port smera OUT se ne može koristiti kao ulazni signal!

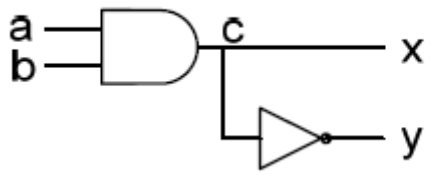
- Ulazni port (smer *in*) se može naći u naredbi dodele samo s **desne**, a izlazni port (smer *out*) samo s **leve** strane znaka (**<=**).

```
1  ---------------------------------
2  LIBRARY IEEE;
3  IEEE.STD_LOGIC_1164.ALL;
4  ---------------------------------
5  ENTITY pr5 IS
6  PORT(a,b : IN STD_LOGIC;
7  x,y : OUT STD_LOGIC);
8  END pr5;
9  ARCHITECTURE pogresno OF pr5 IS
10 BEGIN
11 x <= a AND b;
12 y <= NOT x;
13 END pogresno;
14 ---------------------------------
```

Neispravno, x je izlazni **port**

```
1  ---------------------------------
2  ARCHITECTURE ispravno OF pr5 IS
3  SIGNAL c : STD_LOGIC;
4  BEGIN
5  c <= a AND b;
6  x <= c;
7  y <= NOT c;
8  END ispravno;
9  ---------------------------------
```

Ispravno, interni **signali** se mogu koristiti i kao ulazi i kao izlazi



Port smera OUT se ne može koristiti kao ulazni signal!

- Ova greška se može ispraviti ako se u entitetu port *x* deklariše sa smerom *buffer*. Međutim, korišćenje smera *buffer* se ne preporučuje, jer može dovesti do suptilnih grešaka koje se teško otkrivaju.
- Druga mogućnost je da se port *x* deklariše sa smerom *inout*. Međutim, ni ovo nije dobro rešenje, jer *x* nije bidirekcioni već izlazni signal.
- Rešenje koje se preporučuje je ono koje je dato na prethodnom slajdu. U ovom rešenju, uveden je pomoćni, interni signal *c* koji se u liniji 5 koristi za čuvanje međurezultata "*a and b*", a na osnovu kojeg se formiraju oba izlaza: *x* je isto što i *c* (linija 6), dok je *y* komplement od *c* (linija 7).
- Kod sa prethodnog slajda je ispravan jer se, za razliku od *in* i *out* portova, interni signali u naredbama dodele mogu koristiti bilo kao ulazi, bilo kao izlazi. Takođe, uočimo da su svi signali u ovom primeru tipa *std_logic*. Kao što je već rečeno, ovaj tip podataka se standardno koristi u kodu za sintezu za predstavljanje binarnih signala.

Opis tabele istinitosti

- Omogućava modeliranje (opisivanje) funkcije ili ponašanja kola bez ulaženja u strukturne detalje.

```
 1 ARCHITECTURE funct OF nand_mreza IS
 2 BEGIN
 3 WITH (a & b & c) SELECT
 4 y <= '1' WHEN "000",
 5 '0' WHEN "001",
 6 '1' WHEN "010",
 7 '0' WHEN "011",
 8 '1' WHEN "100",
 9 '0' WHEN "101",
10 '1' WHEN "110",
11 '1' WHEN "111";
12 END funct;
```

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Opis tabele istinitosti

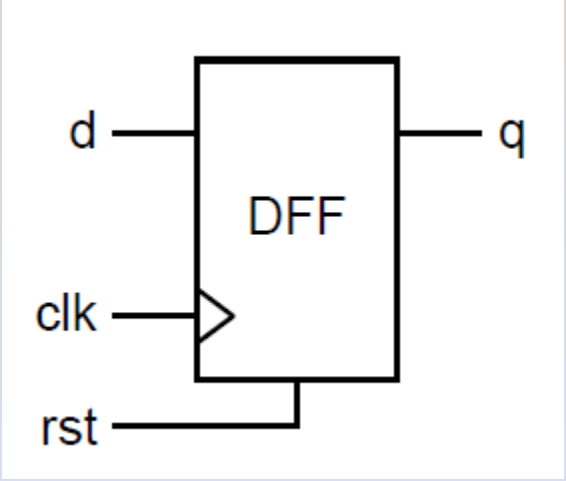
- Kao što se može videti, sada kod nema direktne veze sa strukturom polazne mreže, već samo **opisuje** njenu funkciju.
- Naredba *select* je senzitivna na sva tri ulaza, *a*, *b* i *c*, koji su operatorom & povezana u trobitni vektor.
- Naredba *select* se jedanput izvrši uvek kad se desi događaj na bilo kom od ovih signala, a izlazu *y* se dodeli vrednost iz *when* grane koja odgovara trenutnoj vrednosti trobitnog ulaza.

Sekvencijalne naredbe

- Konkurentne naredbe su pogodne za opis hardvera gde različiti podsistemi jednog složenog sistema rade u paraleli.
- Međutim, ponašanje pojedinih podsistema, ili kola, često je lakše opisati **sekvencijalnim** kodom.
- Konstrukcija ***process*** predstavlja jedan od načina kako se u VHDL-u može nametnuti sekvencijalno izvršenje naredbi.
- Naredbe obuhvaćene procesom izvršavaju se sekvencijalno (jedna za drugom), kao kod bilo kog programskog jezika.
- Kao što arhitektura predstavlja okvir za konkurentni, tako proces predstavlja okvir za sekvencijalni kod.
- Kao što su konkurentne naredbe dodele osetljive na primenu signala s desne strane znaka dodele, tako je proces osetljiv na promenu signala iz njegove liste senzitivnosti.
- Procesi u VHDL-u koriste za modeliranje sekvencijalnih digitalnih kola, tj. kola s memorijom, kao što su to flip-flopovi, registri, brojači i konačni automati.

Sekvencijalne naredbe

```
1  ----------------------------------
2  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ----------------------------------
4  ENTITY dff IS
5  PORT (d, clk, rst: IN STD_LOGIC;
6  q: OUT STD_LOGIC);
7  END dff;
8  ----------------------------------
9  ARCHITECTURE behavior OF dff IS
10 BEGIN
11 PROCESS (rst,clk)
12 BEGIN
13 IF(clk'EVENT AND clk='1') THEN
14 IF(rst='1') THEN
15 q <= '0';
16 ELSE
17 q <= d;
18 END IF;
19 END IF;
20 END PROCESS;
21 END behavior;
```



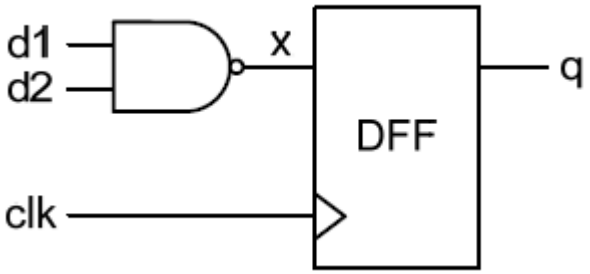
Lista senzitivnosti (promena bilo kog signala iz liste pokreće proces)

Desio se događaj na signalu

Detektuje rastuću ivicu takta

Kombinovani opis - konkurentni/sekvencijalni kod

```
1  ----------------------------------------
2  ENTITY pr8 IS
3  PORT(d1, d2, clk: IN BIT;
4  q : OUT BIT);
5  END pr8;
6  ----------------------------------------
7  ARCHITECTURE primer OF pr8 IS
8  SIGNAL x : BIT;
9  BEGIN
10 x <= d1 NAND d2;
11 PROCESS(clk)
12 BEGIN
13 IF(clk'EVENT AND clk='1') THEN q <= x;
14 END IF;
15 END PROCESS;
16 END primer;
```



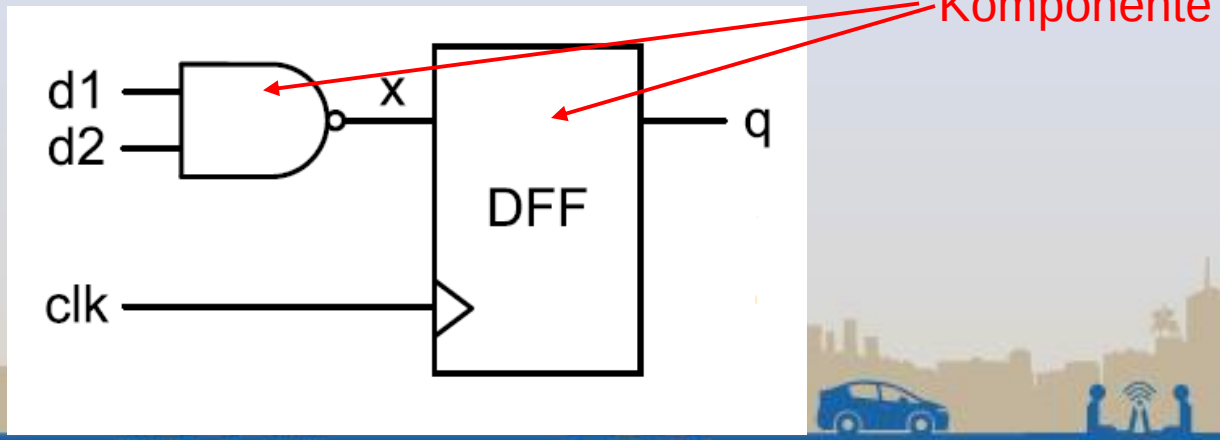
Proces se izvršava konkurentno sa ostalim naredbama iz arhitekture

Kombinovani opis - konkurentni/sekvencijalni kod

- Kod arhitekture (linije 10 – 15) sadrži jedan proces koji opisuje D flip-flop (linije 11 – 15) i jednu konkurentnu naredbu dodele koja obavlja NI operaciju (linija 10). Iako se naredbe unutar procesa izvršavaju sekvencijalno, proces kao celina je konkurentan u odnosu na druge naredbe sadržane u istoj arhitekturi.
- Tako, možemo smatrati da se naredba dodele (linija 10) i proces (linije 11 - 15) izvršavaju konkurentno, slično kao što bi u fizičkom kolu, NI kolo i D flip-flop radili istovremeno i nezavisno.
- Interni signal *x* se koristi za spregu ove dve sekcije koda.
- Konkurentna naredba dodele iz linije 10 nije neophodna i može se izostaviti ako se u procesu naredba *q* <= *x* (linija 13) zameni naredbom *q* <= *d1 nand d2*.

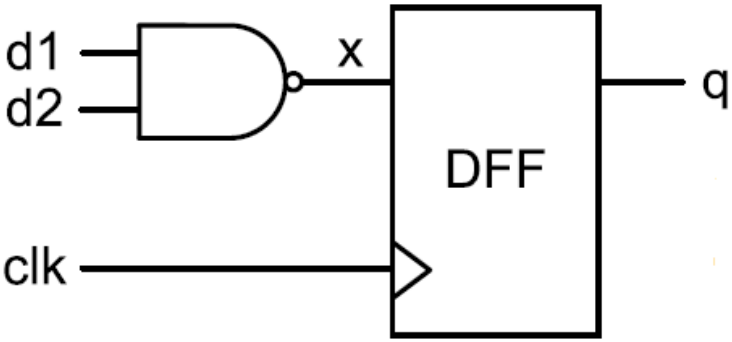
Strukturni opis

- Strukturno modeliranje u VHDL-u zasnovano je na konceptu komponenti.
- Komponenta je celoviti VHDL opisi (entitet plus arhitektura) koji se jednom piše, a onda koristi za konstrukciju složenijih modela.
- Više komponenti se pakuje u jedan paket.
- Paket se smešta u biblioteku.
- Biblioteka se uključuje u one delove projekata gde želimo da koristimo ranije projektovane komponente.
- Sistem se opisuje kao skup povezanih kola manje složenosti.
- U suštini, tekstualni opis blok dijagrama.



Strukturni kod

```
1  ---------------------------------
2  LIBRARY IEEE;
3  LIBRARY nasa_biblioteka;
4  USE IEEE.STD_LOGIC_1164.ALL;
5  USE nasa_bibloteka.nas_paket.all;
6  ---------------------------------
7  ENTITY pr9 IS
8  PORT(d1, d2, clk: IN BIT;
9  q : OUT BIT);
10 END pr9;
11 ---------------------------------
12 ARCHITECTURE struct OF pr9 IS
13 -- komponenta dff -----------
14 COMPONENT dff IS
15 PORT (d, clk: IN STD_LOGIC;
16 q: OUT STD_LOGIC);
17 END COMPONENT;
18 -- komponenta ni_kolo --------
19 COMPONENT ni_kolo IS
20 PORT (a, b : IN STD_LOGIC;
21 c: OUT STD_LOGIC);
22 END COMPONENT;
23 SIGNAL x : STD_LOGIC;
24 BEGIN
25 K1: ni_kolo PORT MAP(d1, d2, x);
26 K2: dff PORT MAP(x, clk, q);
27 END struct;
28 ---------------------------------
```



- U kodu se koriste dve komponente, *ni_kolo* i *dff*.
- Pretpostavka je da se funkcionalni opisi ovih komponenti nalaze u paketu *nas_paket* iz biblioteke *nasa_biblioteka*.
- Naredba iz linije 25 instancira (kreira) jedan primerak komponente *ni_kolo* (K1) i povezuje ga sa portovima i internim signalima kola koje se projektuje.

Kombinovani opis - konkurentni/strukturni kod

```
1  ----------------------------------
2  LIBRARY IEEE;
3  LIBRARY nasa_biblioteka;
4  USE IEEE.STD_LOGIC_1164.ALL;
5  USE nasa_bibloteka.nas_paket.all;
6  ----------------------------------
7  ENTITY pr9 IS
8  PORT(d1, d2, clk: IN BIT;
9  q : OUT BIT);
10 END pr9;
11 ----------------------------------
12 ARCHITECTURE struct OF pr9 IS
13 -- komponenta dff ----------
14 COMPONENT dff IS
15 PORT (d, clk: IN STD_LOGIC;
16 q: OUT STD_LOGIC);
17 END COMPONENT;
18 SIGNAL x : STD_LOGIC;
19 BEGIN
20 x <= d1 NAND d2;
21 K2: dff PORT MAP(x, clk, q);
22 END struct;
```

- Naredba za instanciranje komponenti je takođe konkurentna naredba koja se u istoj arhitekturi može kombinovati s drugim tipovima konkurentnih naredbi i procesima.
- U kodu koji sledi, iskorišćena je samo jedna komponenta, *dff*, dok je *ni* funkcija realizovana u vidu konkurentne naredbe dodele.

Strukturni opis

- Iako predstavlja najniži nivo opisa, strukturno modeliranje ima višestruku ulogu u projektovanju.
- **Prvo**, strukturno modeliranje predstavlja osnovu za **hijerarhijsko projektovanje**. Po pravilu, složen sistem se deli na nekoliko manjih podsistema od kojih se svaki predstavlja jednom komponentom koja se projektuje nezavisno od drugih komponenti. Ako je to potrebno, podsistem se može dalje podeliti na još manje celine.
- **Drugo**, strukturno modeliranje predstavlja način za **ponovno korišćenje ranije projektovanih kola**. Takva kola, dostupna u obliku VHDL komponenti, jednostavno se instanciraju u novom projektu i tretiraju kao “crne kutije”.
- **Treće**, strukturni VHDL opis se može koristiti **za reprezentaciju konačnog rezultata sinteze.** Tipično, ulaz u alati za sintezu je funkcionalni VHDL opis, a izlaz funkcionalno ekvivalentan strukturni VHDL opis u kome se kako komponente koriste elementarna logička kola. Takav strukturni opis se dalje koristi kao ulaz u alate za fizičko projektovanje.

Projektne jedinice (design units)

- **Osnovni gradivni blokovi VHDL opisa**.
- **Nedeljive** sekcije VHDL koda koje u **potpunosti** moraju biti sadržane u **jednoj** projektnoj datoteci.
- Projektna datoteka može sadržati **proizvoljan** broj projektnih jedinica.
- Kad se projektna datoteka analizira od strane VHDL simulatora ili softvera za sintezu, ona se zapravo razlaže na projektne jedinice.
- U VHDL-u postoji pet tipova projektnih jedinica:
- **Entitet** (primarna)
- **Arhitektura** (sekundarana)
- **Deklaracija paketa** (primarna)
- **Telo paketa** (sekundarana)
- **Konfiguracija** (definiše spoj primarne i sekundarne jedinice)

Projektne jedinca se dalje dele na primarne i sekundarne. Primarna projektna jedinica može samostalno da egzistira, dok je sekundarna uvek ridružena jednoj primarnoj jedinici.

Procesiranje VHDL koda

VHDL projekat se procesira u tri koraka ili faze:

- Analiza
- Elaboracija
- Izvršenje

Procesiranje VHDL koda - Analiza

- Tokom faze ***analize***, softver najpre provera sintaksu VHDL koda.
- Ukoliko ne postoje sintaksne greške, softver razlaže kod na **projektne jedinice koje potom nezavisno prevodi** (tj. kompajlira) u tzv. *međukod*, tj. u oblik koji više nije čitljiv (tekstualan), ali je zato lako "razumljiv" **simulatorima i alatima za sintezu.**
- Međukodovi projektnih jedinica se smeštaju u podrazumevanu projektnu biblioteku (*work*) ili u biblioteku koju je projektant naznačio kao odredišnu biblioteku projekta.

Procesiranje VHDL koda - Elaboracija

- Složeni projekti su obično hijerarhijski organizovani. Vršni nivo hijerarhije može sadržati, u vidu instanciranih komponenti, više strukturno povezanih podsistema.
- Tokom faze ***elaboracije***, softver polazi od deklaracije entiteta vršnog nivoa i spaja ga sa odgovarajućom arhitekturom (shodno specifikaciji konfiguracije). Ako u arhitekturi postoje instancirane komponente, softver zamenjuje svaku instancu komponentu odgovarajućom arhitekturom. Ovaj proces se rekurzivno ponavlja sve dok se sve instancirane komponente ne zamene odgovarajućim arhitekturama.
- Tako, elaboracija, kroz proces izbora i kombinovanja arhitektura, dovodi do kreiranja jednog “ravnog” (ne više hijerarhijskog, već razvijenog) opisa celokupnog sistema.

Procesiranje VHDL koda - Izvršenje

- Konačno, u fazi izvršenja, analiziran i elaboriran opis se prosleđuje softveru za **simulaciju ili** softveru za **sintezu**.
- Prvi simulira i “izvršava” VHDL opis na računaru, dok drugi sintetiše (realizuje) fizički opis kola.

Simulacija VHDL koda - Testbenč

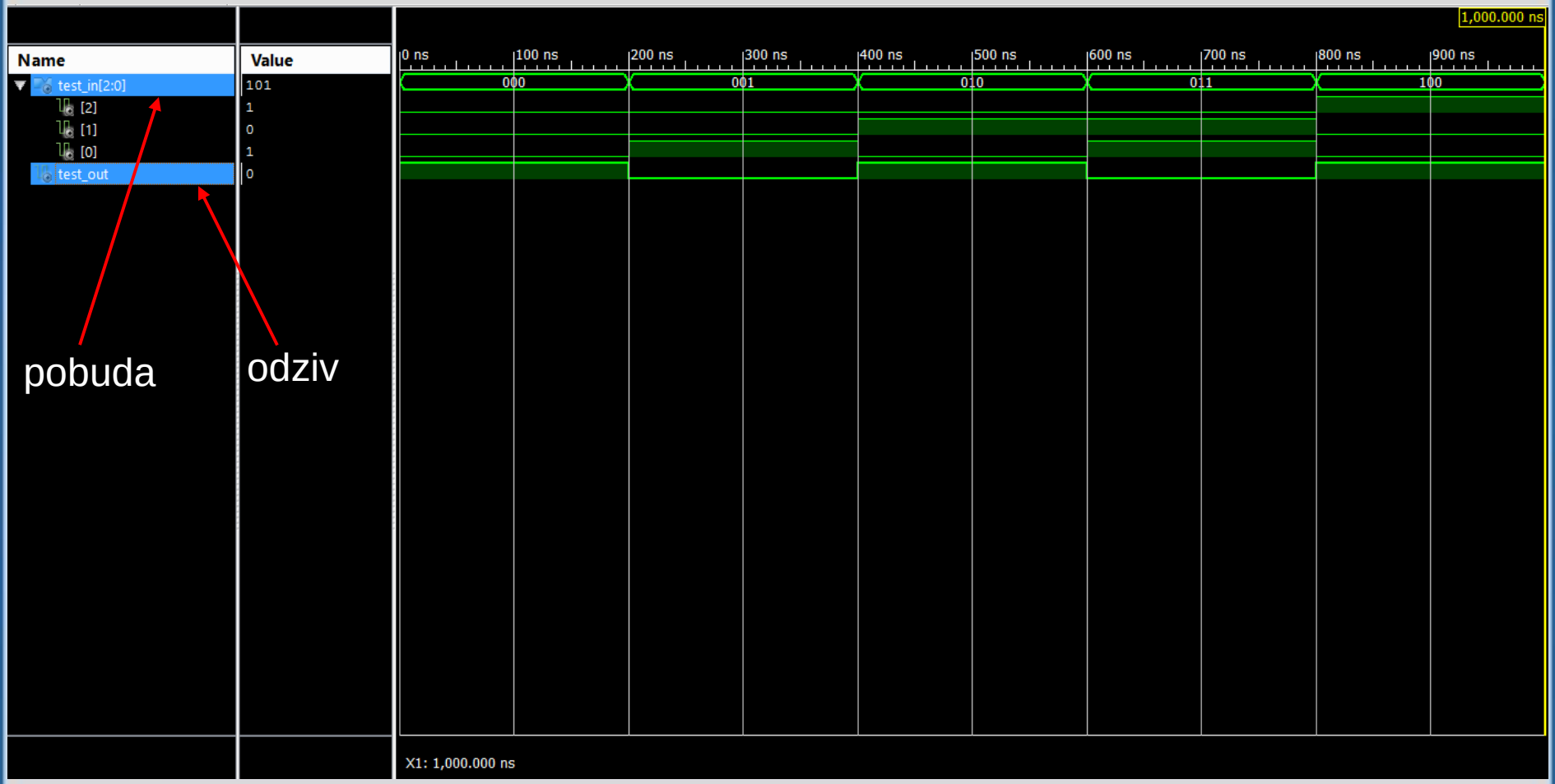
```vhdl
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

 ----------------------------------------
 ENTITY nand_mreza IS
 PORT(a, b, c: IN BIT;
 y : OUT BIT);
 END nand_mreza;
 ----------------------------------------
 ARCHITECTURE dataflow OF nand_mreza IS
 SIGNAL x : BIT;
 BEGIN
 y <= x NAND c after 0 ns;
 x <= a NAND b after 20 ns;
 END dataflow;
 ----------------------------------------
```

Analiza ?

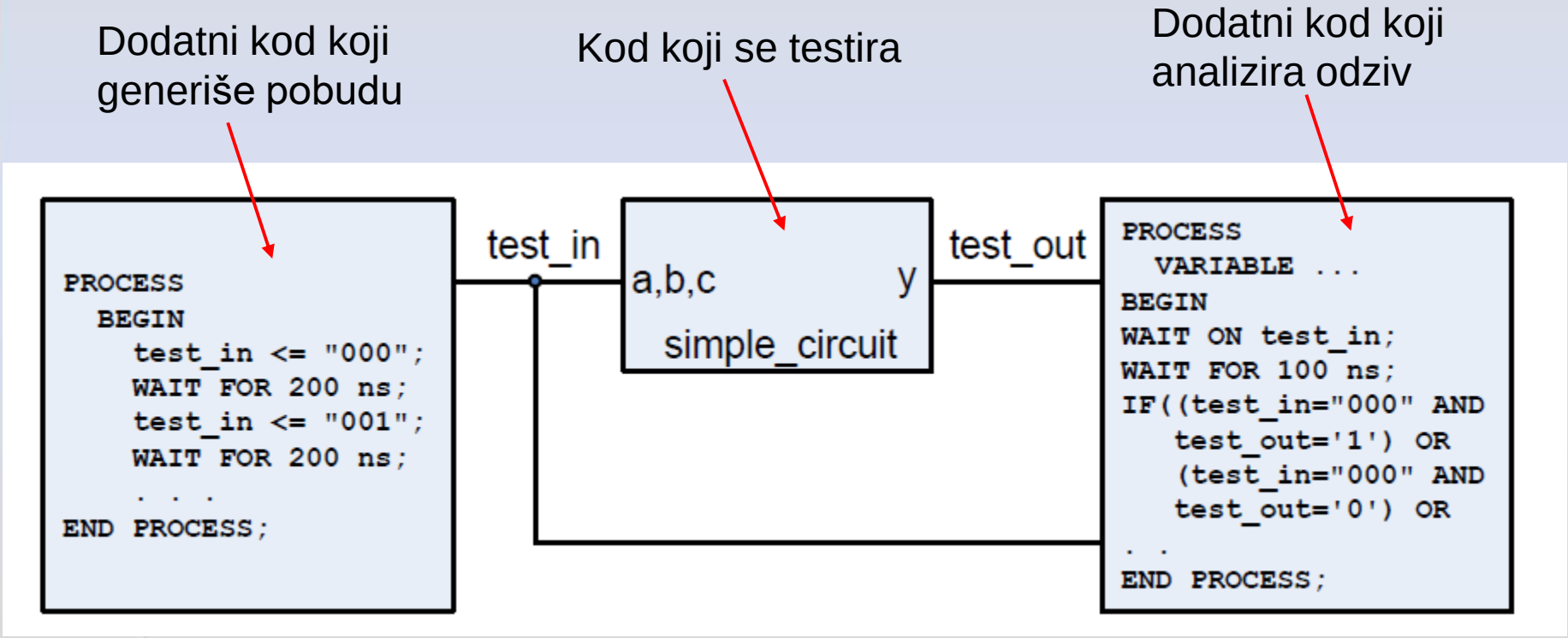
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
 ------------------------------------------------------------------
 ENTITY nand_mreza_testbench IS
 END nand_mreza_testbench;
 ------------------------------------------------------------------
 ARCHITECTURE tb_arch OF nand_mreza_testbench IS
 COMPONENT nand_mreza
 PORT(a, b, c: IN BIT;
 y : OUT BIT);
 END COMPONENT;
 SIGNAL test_in : BIT_VECTOR(2 DOWNTO 0);
 SIGNAL test_out : BIT;
 BEGIN
 -- Instanciranje kola koje se testira ---------------------------
uut: nand_mreza PORT MAP(test_in(2),test_in(1),test_in(0),test_out);
 -- Generator stimulansa (test vektora) --------------------------
PROCESS
 BEGIN
 test_in <= "000";
 WAIT FOR 200 ns;
 test_in <= "001";
 WAIT FOR 200 ns;
 test_in <= "010";
 WAIT FOR 200 ns;
 test_in <= "011";
 WAIT FOR 200 ns;
 test_in <= "100";
 WAIT FOR 200 ns;
 test_in <= "101";
 WAIT FOR 200 ns;
 test_in <= "110";
 WAIT FOR 200 ns;
 test_in <= "111";
 WAIT FOR 200 ns;
 END PROCESS;
 END tb_arch;
 ------------------------------------------------------------------
```

Simulacija VHDL koda - Testbenč



Simulacija VHDL koda - Testbenč

- Testbenč sadrži: generator stimulansa, kolo koje se testira i verifikator koji proverava ispravnost odziva kola.
- Testbenč je kompletan VHDL modul sa entitetom i arhitekturom.

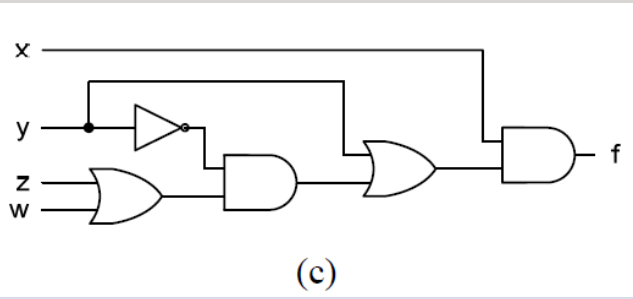


Sinteza VHDL koda

- **Preslikavanje jezičkih konstrukcija iz VHDL koda na hardverske elemente identičnog ponašanja (funkcije).**
- **Nije moguće sintetizovati proizvoljan VHDL opis !**
- Pojedine jezičke konstrukcije se ne mogu sintetizovati.
- Pojedine se mogu sintetizovati uz određena ograničenja.
- Pojedine se mogu sintetizovati.
- Otuda je dobra praksa da pisanju VHDL koda za sintezu prethodi crtanje skice konceptualnog dijagrama koji će poslužiti kao model za pisanje koda (slično kao što dijagram toka služi kao model za pisanje programa).

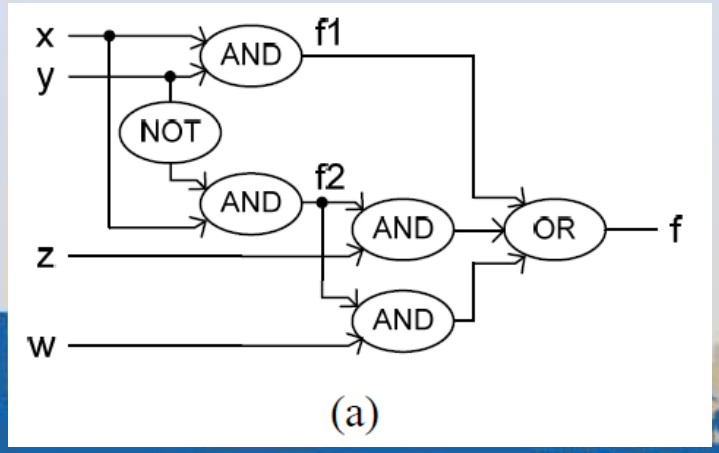
Sinteza VHDL koda

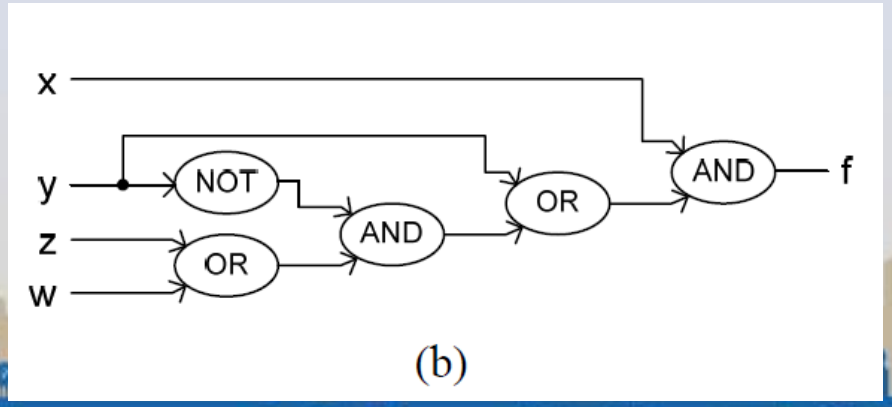
```
1  . . .
2  f1 <= x AND y;
3  f2 <= x AND NOT y;
4  f <= f1 OR (f2 AND z) OR (f2 AND w);
5  . . .
6
7
```



Sinteza VHDL koda se obavlja u nekoliko uzastopnih koraka.

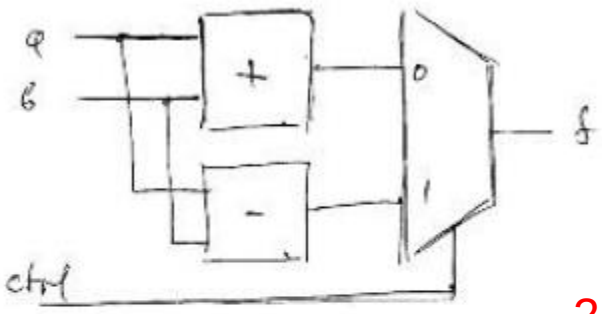
- U prvom koraku, softver za sintezu transformiše polazni VHDL opis u funkcionalnu mrežu, koja za razmatrani VHDL kod izgleda kao na Sl.(a).
- U drugom koraku, funkcionalna mreža se pojednostavljuje primenom različitih metoda za automatsku optimizaciju logičkih funkcija (Sl.(b)).
- U trećem koraku, operatori iz optimizovane funkcionalne mreže se preslikavaju na hardverske elemente (Sl.(c)).





Sinteza VHDL koda

Projektant:

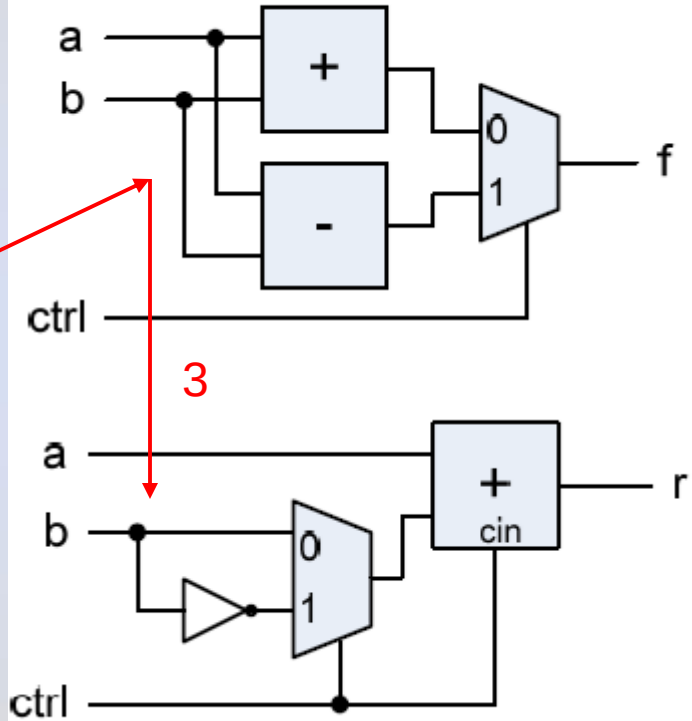


1

2

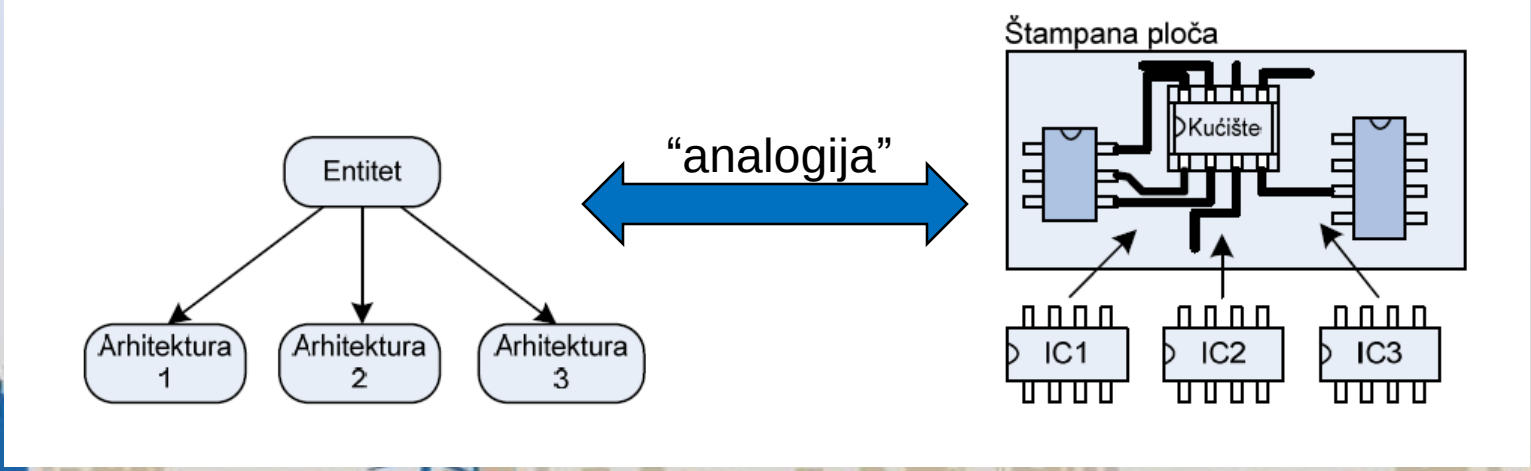
```
. . .
f <= a + b WHEN ctrl='0' ELSE
     a - b;
. . .
```

Softver za sintezu:

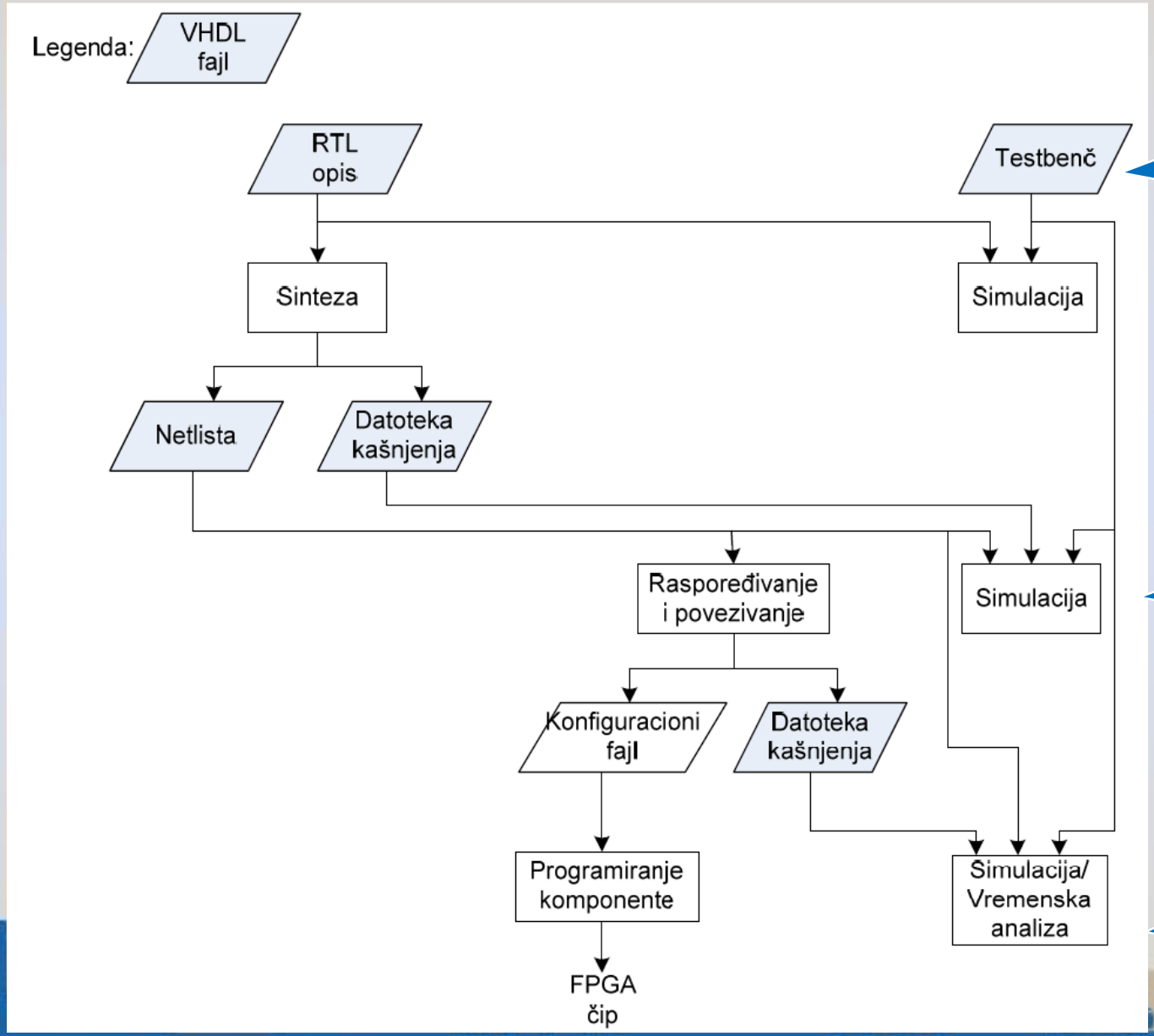


CONFIGURATION

- Entitet i arhitektura su ključni koncepti VHDL-a.
- Razdvajanje specifikacije interfejsa (entitet) od specifikacije funkcije/strukture kola (arhitektura), omogućava projektantu da zadrži isti entitet, a zameni arhitekturu novom, eventualno detaljnijom ili unapređenom verzijom.
- Jedan entitet više arhitektura.
- Konfiguracija - spoj entiteta i arhitekture
- Kako arhitekturu povezati sa entitetom – naredba configuration.



Uloga VHDL-a u procesu projektovanja



Funkcionalana simulacija

Vremenska simulacija nakon sinteze

Vremenska simulacija nakon fizičkog projektovanja