Uvod u HDL

1

Opis digitalnog sistema

1. Rečima – najčešće željeni način ponašanja – behavioral model
2. Funkcionalnom tabelom – praktično za sisteme sa malim brojem ulaza i izlaza
3. Jedačinama Bulove algebre – kombinacione mreže
4. Električnom šemom – pogodno za realizaciju digitalnog sistema, omogućuje jedna vid sinteze i simulacije sistema - Schematic Capture
    1. Deo CAD Computer Aided Design
    2. Deo EDA Electronic Design Automation – ECAD
    3. Najčešće za potrebe pravljenja PCB-a (Printed Circuit Board)

Danas - za složene i programabilne sisteme mnogo efikasniji

HDL – Hardware Description Language

Slični tipičnim programskim jezicima – dele neke karakteristike sa njia (objektno orjentisano programiranje, konkuretno programiranje, itd.) ali im je osnovna namena OPIS hardvera u odnosu na posao koji treba neki procesor da izvrši.

2

Digitalne, elektronske, komponente se mogu klasifikovati u dve velike grupe

- fiksne
- programabilne

Pri ovome govorimo i mislimo o hardveru komponenti. Na primer: Računar je programabilan. Možemo da napišemo različite programe i da se računar izvođenjem programa ponaša na različite načine. Ali hardver računara je fiksan. Izvođenje programa ne menja hardverske veze u procesoru ili nekom drugom delu računara (zaboravite na trenutak memorije).

Fiksne komponente – kada su proizvedene određena im je logička funkcija i ne može se menjati

Programabilne komponente – korisnik može da ih prilagodi, programira, za određenu funkciju. Za sada to su neke logičke funkcije. Kombinacione mreže. Razvoj tehnologije je to omogućio. Da se sa kola malog i srednjeg stepena integracije koja su po pravilu fiksne komponente pređe na korišćenje komponenta višeg stepena integracije i koja mogu da se prilagode potrebi korisnika.

3

Koja je ideja da napravimo programabilnu komponentu u kojoj možemo da realizujemo razne kombinacione mreže?

Bilo koja logička funkcija može da se predstavi u obliku sume proizvoda!

I + ILI (ako ima NE)

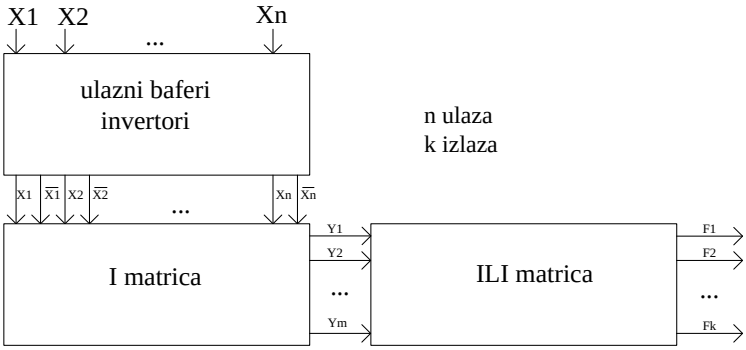
Šta nam treba?

Dovoljan broj ulaza.
Dovoljan broj izlaza.

Mogućnost programiranja I-ILI povezanosti.

4

Primer i ideja jedne programabilne komponente



Šta i koliko može da se programira zavisi od tipa komponente.

Da li su obe matrice programabilne?
Da li je samo I odnosno ILI matrica programabilna?
Koliko su programabilne?
Šta može da učestvuje u Y i koliko ih ima (m?)? Koliko ulaza u I kola.
Šta može da učestvuje u F i koliko? Koliko ulaza u ILI kola.

5

PLA – Programmable Logic Array
PAL – Programmable Array Logic
GAL – Generic Array Logic
} PLD

CPLD – Complex Programmable Logic Device
FPGA– Fild Programmable Gate Array
} CPLD

ROM – Read Only Memory
PROM – Programmable ROM
EEPROM – Electrically EPROM
} Memory

ASIC– Application Specific Integrated Circuit } ASIC

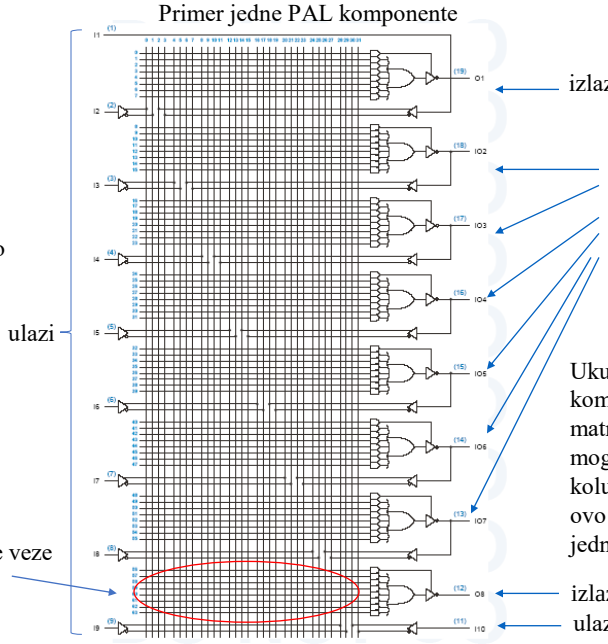
6

Primer jedne PAL komponente

Maksimalno
10+6 ulaza
odnosno
2+6 izlaza

„Bilo koja“ kombinaciona mreža koja ima maksimalno 10+X ulaza,
odnosno
2+6-X izlaza
X = 0,1,..6

20-pinsko pakovanje
10. Pin GND
20. Pin VDD



Ukupan broj programabilnih veza u ovoj komponenti je 2048 u I matrici. ILI matrica je fiksna. Po svakom I kolu je 32 moguća člana u proizvodu. Po svakom ILI kolu je moguće sumirati 7 proizvoda. I ovo je za današnju tehnologiju jednostavna programabilna komponenta.

7

Primer fajla potreban da bi „programator“ isprogramirao potrebne veze

JEDEC

- JEDEC files are used to specify the fuses that will be burned or connected to each AND gate.

```
□
CUPL(WM)        5.0a  Serial# 60000009
Device          g16v8as  Library DLIB-h-40-2
Created         Tue Sep 26 15:12:26 2006
Name            Decoder
Partno          00
Revision        01
Date            2/17/2005
Designer        F.Rios
Company         UMD
Assembly        None
Location
*QPZ0
*QF2194
*G0
*F0
*L00000 11011111111111111111111111111101
*L00032 01111111111111111111111111111101
*L00064 11110111111111111111111111111101
*L00096 11111111101111111111011111111101
*L00128 11111111101111111011111111111101
*L00160 11111111101101111111111111111101
*L00192 10101011011111111111111111111110
*L00224 10101011111110111011011111111110
*L00256 11101111111111111111111111111101
*L00288 01111111111111111111111111111101
*L00320 11110111111111111111111111111101
```

Number of total fuses

Number of fuse

1 = Blown fuse, no connection
0 = Intact fuse, connected

8

Da bi se na „lak“ način generisao potreban fajl za programator, da bi se eventualno pre toga uradila i simulacija a kasnije i verifikacija isprogramirane komponente započet je razvoj HDL jezika, 80ih godina prošlog veka.

PALASM – PAL Assembler
CUPL – Compiler for Universal Programmable Logic
ABEL - Advanced Boolean Expression Language

Ovi jezici su između ostalog dozvolili dobru dokumentovanost dizajna. Zapisan u čitljivom „kodu“.
Iz istog razloga je započet i razvoj složenijih HDL jezika. Međutim mogućnosti prenosivosti, ponovnog korišćenja, mogućnosti simulacije kompleksnih digitalnih sistema, itd… doveli su to toga da se ovi jezici danas masovno koriste u sintezi i analizi digitalnih sistema.
Dva najpoznatija i najčešće korišćena su

VHDL - Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL)
VERILOG – Verification and Logic

9

ABEL primeri

1. Dvoulazno I kolo sa test vektorima
2. Koder prioriteta koji koduje i ulaz sa najvišim prioritetom kao i ulaz sa narednim prioritetom

```
MODULE and1
TITLE 'and1 gate -
            Instantiated by nand1 - Simple hierarchy example'

" The pins must match the Symbol pins (schematic),
" or interface names (ABEL-HDL) in the upper-level module.

IN1, IN2, OUT1 pin;
EQUATIONS

            OUT1 = IN1 & IN2;

TEST_VECTORS

            ([ IN1, IN2] -> [OUT1])
             [   0,   0] -> [ 0];
             [   0,   1] -> [ 0];
             [   1,   0] -> [ 0];
             [   1,   1] -> [ 1];
END
```

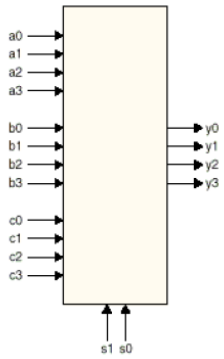
1

```
title 'Dual Priority Encoder'
PRIORTWO device 'P16V8';

" Input and output pins
R7..R0                          pin 1..8;
AVALID, A2..A0, BVALID, B2..B0  pin 19..12 istype 'com';

" Set definitions
A = [A2..A0]; B = [B2..B0];

equations

WHEN R0==1 THEN A=0;
ELSE WHEN R1==1 THEN A=1;
ELSE WHEN R2==1 THEN A=2;
ELSE WHEN R3==1 THEN A=3;
ELSE WHEN R4==1 THEN A=4;
ELSE WHEN R5==1 THEN A=5;
ELSE WHEN R6==1 THEN A=6;
ELSE WHEN R7==1 THEN A=7;

AVALID = ([R7..R0] != 0);

WHEN (R0==1) & (A!=0) THEN B=0;
ELSE WHEN (R1==1) & (A!=1) THEN B=1;
ELSE WHEN (R2==1) & (A!=2) THEN B=2;
ELSE WHEN (R3==1) & (A!=3) THEN B=3;
ELSE WHEN (R4==1) & (A!=4) THEN B=4;
ELSE WHEN (R5==1) & (A!=5) THEN B=5;
ELSE WHEN (R6==1) & (A!=6) THEN B=6;
ELSE WHEN (R7==1) & (A!=7) THEN B=7;

BVALID = (R0==1) & (A!=0) # (R1==1) & (A!=1)
       # (R2==1) & (A!=2) # (R3==1) & (A!=3)
       # (R4==1) & (A!=4) # (R5==1) & (A!=5)
       # (R6==1) & (A!=6) # (R7==1) & (A!=7);

end priortwo
```

2

10

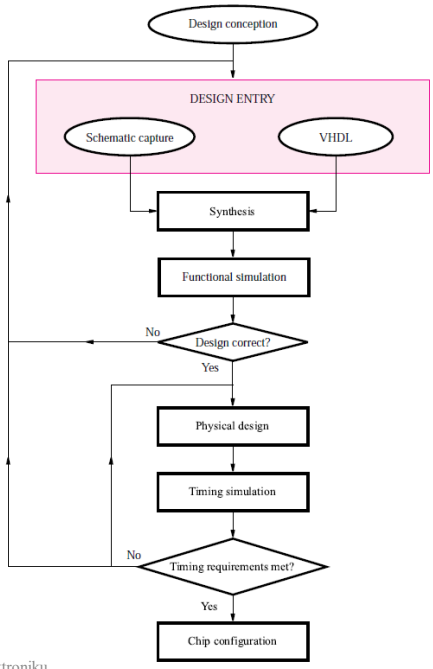
Primer

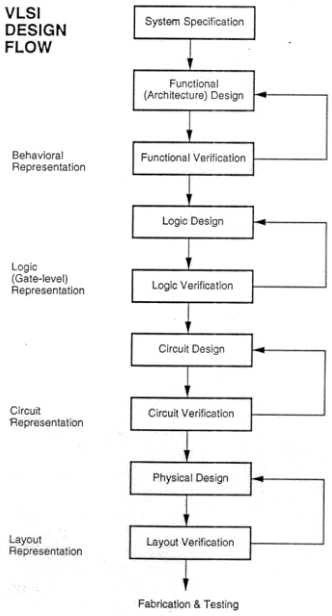



```
module Mux12T4
title '12 to 4 multiplexer'
        a0..a3      pin      1..4;
        b0..b3      pin      5..8;
        c0..c3      pin      9, 11, 12, 13;
        s1,s0       pin      18,19;
        y0..y3       pin      14..17;
        H       =       [1,1,1,1];
        L       =       [0,0,0,0];
        X       =       .x.;
        select  =       [s1, s0];
        y       =       [y3..y0];
        a       =       [a3..a0];
        b       =       [b3..b0];
        c       =       [c3..c0];
equations
        when (select == 0) then y = a;
        when (select == 1) then y = b;
        when (select == 2) then y = c;
        when (select == 3) then y = c;

test_vectors ([select, a, b, c] -> y)
                [0      , 1, X, X] -> 1;"select = 0, gates lines a to output
                [0      ,10, H, L] -> 10;
                [0      , 5, H, L] -> 5;
                [1      , H, 3, H] -> 3;"select = 1, gates lines b to output
                [1      ,10, 7, H] -> 7;
                [1      , L,15, L] -> 15;
                [2      , L, L, 8] -> 8;"select = 2, gates lines c to output
                [2      , H, H, 9] -> 9;
                [2      , L, L, 1] -> 1;
                [3      , H, H, 0] -> 0;"select = 3, gates lines c to output
                [3      , L, L, 9] -> 9;
                [3      , H, L, 0] -> 0;
end
```

11

Tipičan CAD proces





12

Šta je VHDL?

VHDL je programski jezik koji je dizajniran i optimizovan za opisivanje ponašanja digitalnih kola i sistema. Kao takav, VHDL kombinuje sledeće karakteristike:

1. Jezik za simulaciono modelovanje. VHDL ima mnogo funkcija pogodnih za opisivanje (do nivoa detalja) ponašanja elektronskih komponenti u rasponu od jednostavnih logičkih gejtova do kompletnih mikroprocesora i namenskih čipova. Karakteristike VHDL-a omogućavaju da se električni aspekti ponašanja kola (kao što su vremena porasta i pada signala, kašnjenja kroz gejtove i funkcionalni rad) precizno opisuju. Dobijeni VHDL simulacioni modeli se zatim mogu koristiti kao gradivni blokovi u većim kolima (koristeći šeme, blok dijagrame ili VHDL opise na nivou sistema) u svrhu simulacije.

13

2. Jezik za unos dizajna. Baš kao što programski jezici visokog nivoa dozvoljavaju da se složeni koncepti dizajna izraze kao kompjuterski program VHDL omogućava da se ponašanje složenih elektronskih kola prikaže u sistemu dizajna za automatsku sintezu kola ili za simulaciju sistema. Kao Pascal, C i C++, VHDL uključuje funkcije korisne za tehnike strukturiranog dizajna i nudi bogat skup funkcija kontrole i predstavljanja podataka. Za razliku od ovih drugih programskih jezika, VHDL pruža funkcije koje omogućavaju opisivanje istovremenih događaja. Ovo je važno jer je hardver koji se opisuje korišćenjem VHDL-a inherentno istovremen u svom radu. Korisnicima PLD programskih jezika kao što su PALASM, ABEL, CUPL i drugi će istovremene karakteristike VHDL-a biti prilično poznate. Oni koji su programirali samo koristeći softverske programske jezike će, međutim, imati neke nove koncepte za razumevanje. Konkuretno programiranje. Konkuretno izvršavanje programa.

3. Jezik za testiranje. Jedan od najvažnijih (i nedovoljno iskorišćenih) aspekata VHDL-a je njegova sposobnost da prihvati specifikacije performansi za kolo, u obliku koji se obično naziva testbench. Testbench su VHDL opisi stimulusa kola i odgovarajućih očekivanih izlaza koji potvrđuju ponašanje kola tokom vremena. Testbench treba da budu sastavni deo svakog VHDL projekta i treba da se kreiraju paralelno sa drugim opisima kola.

14

4. Netlist Language. VHDL je moćan jezik sa kojim se unose novi dizajni na visokom nivou, ali je takođe koristan kao oblik komunikacije niskog nivoa između različitih alata u računarskom okruženju dizajna. Karakteristike strukturalnog jezika VHDL-a omogućavaju da se efikasno koristi kao jezik liste povezanosti mreža, zamenjujući (ili povećavajući) druge jezike liste povezanosti mreža kao što je EDIF.

5. Standardni jezik. Jedan od najubedljivijih razloga da steknete iskustvo i znanje o VHDL-u je njegovo usvajanje kao standarda u zajednici elektronskog dizajna. Korišćenje standardnog jezika kao što je VHDL će praktično garantovati da nećete morati da bacate i ponovo preuzimate koncepte dizajna samo zato što metod unosa dizajna koji ste izabrali nije podržan u novijoj generaciji alata za dizajn. Korišćenje standardnog jezika takođe znači da je veća verovatnoća da ćete moći da iskoristite prednosti najsavremenijih alata za dizajn i da ćete imati pristup bazi znanja hiljada drugih inženjera, od kojih mnogi rešavaju slične probleme.

VHDL: Jednostavno ili složeno koliko želite. Iako je istina da je VHDL veliki i složen jezik, zapravo nije teško početi sa njim. Koristite ga kako vam je potrebno i istražujte napredne funkcije dok postajete sigurniji.

15

Kako se koristi VHDL?
VHDL je programski jezik opšte namene optimizovan za dizajn elektronskih kola. Kao takav, postoji mnogo tačaka u celokupnom procesu dizajna u kojima VHDL može pomoći.

Za specifikaciju dizajna: VHDL se može koristiti na početku, dok još dizajnirate na visokom nivou, da biste prikazali performanse i zahteve interfejsa svake komponente u velikom sistemu. Ovo je posebno korisno za velike projekte koji uključuju mnogo članova tima. Koristeći pristup dizajnu odozgo nadole, dizajner sistema može definisati interfejs za svaku komponentu u sistemu i opisati zahteve prihvatanja tih komponenti u obliku testbench-a visokog nivoa. Definicija interfejsa (obično izražena kao deklaracija VHDL entiteta) i specifikacija performansi visokog nivoa mogu se zatim preneti na druge članove tima radi dovršavanja ili usavršavanja.

Za unos dizajna: Unos dizajna je ona faza u kojoj se detalji sistema unose u sistem projektovanja zasnovan na računaru. U ovoj fazi možete izraziti svoj dizajn (ili delove vašeg dizajna) kao šeme (bilo na nivou ploče ili čisto funkcionalne) ili koristeći VHDL opise. Ako ćete koristiti tehnologiju sinteze, onda ćete želeti da napišete VHDL delove dizajna koristeći stil VHDL koji je prikladan za sintezu. Faza unosa dizajna može uključivati alate i metode unosa dizajna koje nisu VHDL. U mnogim slučajevima, opisi dizajna napisani u VHDL-u se kombinuju sa drugim prikazima, kao što su šeme, da bi se formirao kompletan sistem.

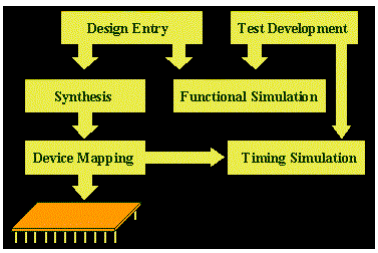
16

Za simulaciju dizajna: Jednom kada uđete u sistem projektovanja zasnovan na računaru, verovatno ćete želeti da simulirate rad vašeg kola da biste saznali da li će ono ispuniti funkcionalne i vremenske zahteve razvijene tokom procesa specifikacije. Ako ste kreirali jedan ili više test bench-a kao deo vaše specifikacije dizajna, onda ćete koristiti simulator da primenite test bench-a na svoj dizajn onako kako je napisan za sintezu (funkcionalna simulacija) i eventualno korišćenje verzije nakon sinteze dizajna takođe.

Za projektnu dokumentaciju: Karakteristike strukturiranog programiranja VHDL-a, zajedno sa njegovim karakteristikama upravljanja konfiguracijom, čine VHDL prirodnim oblikom u kojem se dokumentuje veliko i složeno kolo. Na vrednost korišćenja jezika visokog nivoa kao što je VHDL za projektnu dokumentaciju ukazuje činjenica da Ministarstvo odbrane SAD sada zahteva VHDL kao standardni format za komunikaciju zahteva dizajna između vladinih podizvođača.

Kao alternativa šemi: Šeme su dugo bile deo dizajna elektronskih sistema i malo je verovatno da će uskoro izumreti. Šeme imaju svoje prednosti, posebno kada se koriste za prikazivanje kola u obliku blok dijagrama. Iz tog razloga mnogi VHDL alati za dizajn sada nude mogućnost kombinovanja šematskih i VHDL reprezentacija u dizajnu.

17



Gde se VHDL uklapa u ovaj dijagram? VHDL (zajedno sa drugim oblicima unosa, kao što su šeme i blok dijagrami) će se koristiti za unos dizajna. VHDL izvorni kod se može uneti u simulaciju, omogućavajući mu da bude funkcionalno verifikovan, ili može biti prosleđen direktno alatima za sintezu za implementaciju u određenom tipu uređaja. Na strani razvoja testa, mogu se kreirati VHDL testbench koje testiraju kolo kako bi se proverilo da li ispunjava funkcionalna i vremenska ograničenja specifikacije. Testbench se mogu uneti pomoću editora teksta ili mogu biti generisani iz drugih oblika informacija o test stimulansima, kao što su grafički talasni oblici. Za preciznu vremensku simulaciju kola posle rutiranja, verovatno ćete koristiti program za generisanje vremenskog modela dobijen od dobavljača uređaja ili treće strane dobavljača simulacionog modela. Alati za generisanje modela, kao što je ovaj, obično generišu VHDL izvorne datoteke sa vremenskim napomenama koje podržavaju veoma precizne simulacije na nivou sistema.

18

Kada je reč o jezicima za opis hardvera, VHDL nije jedini. Verilog HDL postoji već dugi niz godina i ima veliki broj fanova koji ga koriste i za simulaciju i za sintezu. Kako se Verilog razlikuje od VHDL-a? Prvo, Verilog ima određenu prednost u dostupnosti simulacionih modela. Ubrzo nakon svog uvođenja (od strane Gatevai Design Automation sredinom 1980-ih), Verilog se uspostavio kao de fakto standardni jezik za ASIC simulacione biblioteke. Široka popularnost Verilog HDL-a za ovu svrhu ostavila je VHDL donekle iza u podršci za širok spektar ASIC uređaja. Inicijativa VITAL poboljšava VHDL standardnim metodom zapisa o kašnjenju koji je sličan onom koji se koristi u Verilog-u. VITAL će olakšati dobavljačima ASIC-a i drugima da brzo kreiraju VHDL modele iz postojećih Verilog biblioteka. Još jedna karakteristika koja je definisana u Verilogu (u Open Verilog International objavljenom standardu) je interfejs programskog jezika, ili PLI. PLI omogućava piscima simulacionih modela da izađu van Verilog-a kada je to potrebno za kreiranje bržih simulacionih modela, ili da kreiraju funkcije (koristeći jezik C) koje bi bilo teško ili neefikasno implementirati direktno u Verilogu. U korist VHDL-a su njegovo usvajanje kao IEEE standard (standard 1076) i njegove karakteristike upravljanja dizajnom višeg nivoa. Ove karakteristike upravljanja dizajnom, koje uključuju deklaraciju konfiguracije i karakteristike biblioteke VHDL-a, čine VHDL idealnim jezikom za upotrebu u velikim projektima. U većini aspekata, međutim, VHDL i Verilog su identični u funkciji. Sintaksa je drugačija (sa Verilogom koji liči na C, a VHDL više kao Pascal ili Ada), ali osnovni koncepti su isti.

19

Oba jezika se lako uče, ali ih je teško savladati.

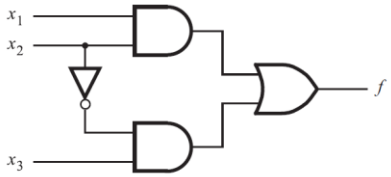
Kada naučite jedan od ovih jezika, nećete imati problema da pređete na drugi.

Suština je sledeća: raspravljanje o jezičkim karakteristikama je besmisleno i najbolje je ostaviti za slobodno vreme.

Trebalo bi da izaberete skup alata koji će vam pomoći da obavite svoj posao. Izaberite ove alate na osnovu njihovih karakteristika i njihove podrške za vašu specifičnu aplikaciju i tehnologiju. Ne birajte jezik opisa hardvera samo na osnovu toga kako izgleda

20

Jednostavan VHDL kod



Dva osnovna elementa opisa - koda

1. ENTITY

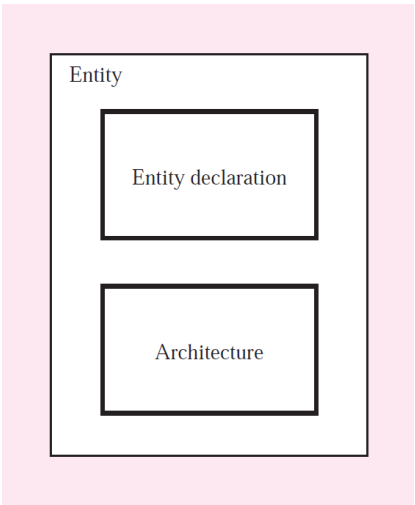
```
ENTITY example1 IS
      PORT ( x1, x2, x3   : IN     BIT ;
                  f       : OUT  BIT ) ;
END example1 ;
```

2. ARCHITECTURE

```
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
      f < = (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```

Za jedan entity može biti definisano više arhitektura. Obrnuto NE.

Opis kola – design entity - entity



```
ENTITY entity_name IS
    PORT ( [SIGNAL] signal_name {, signal_name} : [mode] type_name {;
           SIGNAL] signal_name {, signal_name}  : [mode] type_name } ) ;
END entity_name ;

ARCHITECTURE architecture_name OF entity_name IS
    [SIGNAL declarations]
    [CONSTANT declarations]
    [TYPE declarations]
    [COMPONENT declarations]
    [ATTRIBUTE specifications]
BEGIN
    {COMPONENT instantiation statement ;}
    {CONCURRENT ASSIGNMENT statement ;}
    {PROCESS statement ;}
    {GENERATE statement ;}
END [architecture_name] ;
```

23

Jednobitni potpuni sabirač

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

ENTITY fulladd IS
    PORT ( Cin, x, y  : IN   STD_LOGIC ;
           s, Cout    : OUT  STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= x XOR y XOR Cin ;
    Cout <= (x AND y) OR (x AND Cin) OR (y AND Cin) ;
END LogicFunc ;
```

24

"Dodeljivanje vrednosti"

```
signal_name  <= expression ;
```

```
SIGNAL x1, x2, x3, f  : STD_LOGIC ;
        .
        .
        .
f  <=  (x1 AND x2) OR x3 ;
```

25

Primer logička operacija

```
SIGNAL A, B, C  : STD_LOGIC_VECTOR (1 TO 3) ;
        .
        .
        .
C  <=  A AND B ;
```

This results in $C(1) = A(1) \cdot B(1)$, $C(2) = A(2) \cdot B(2)$, and $C(3) = A(3) \cdot B(3)$.

Primer aritmetička operacija

```
SIGNAL X, Y, S   : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
        .
        .
        .
S  <=  X + Y ;
```

26

```
title 'Dual Priority Encoder'
PRIORTWO device 'P16V8';

" Input and output pins
R7..R0                          pin 1..8;
AVALID, A2..A0, BVALID, B2..B0  pin 19..12 istype 'com';

" Set definitions
A = [A2..A0]; B = [B2..B0];

equations

WHEN R0==1 THEN A=0;
ELSE WHEN R1==1 THEN A=1;
ELSE WHEN R2==1 THEN A=2;
ELSE WHEN R3==1 THEN A=3;
ELSE WHEN R4==1 THEN A=4;
ELSE WHEN R5==1 THEN A=5;
ELSE WHEN R6==1 THEN A=6;
ELSE WHEN R7==1 THEN A=7;

AVALID = ([R7..R0] != 0);

WHEN (R0==1) & (A!=0) THEN B=0;
ELSE WHEN (R1==1) & (A!=1) THEN B=1;
ELSE WHEN (R2==1) & (A!=2) THEN B=2;
ELSE WHEN (R3==1) & (A!=3) THEN B=3;
ELSE WHEN (R4==1) & (A!=4) THEN B=4;
ELSE WHEN (R5==1) & (A!=5) THEN B=5;
ELSE WHEN (R6==1) & (A!=6) THEN B=6;
ELSE WHEN (R7==1) & (A!=7) THEN B=7;

BVALID = (R0==1) & (A!=0) # (R1==1) & (A!=1)
       # (R2==1) & (A!=2) # (R3==1) & (A!=3)
       # (R4==1) & (A!=4) # (R5==1) & (A!=5)
       # (R6==1) & (A!=6) # (R7==1) & (A!=7);

end priortwo
```

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

entity Vprior2 is
    port (
        R: in STD_LOGIC_VECTOR (0 to 7);
        A, B: out STD_LOGIC_VECTOR (2 downto 0);
        AVALID, BVALID: buffer STD_LOGIC
    );
end Vprior2;

architecture Vprior2_arch of Vprior2 is
begin
  process(R, AVALID, BVALID)
  begin
    AVALID <= '0'; BVALID <= '0'; A <= "000"; B <= "000";
    for i in 0 to 7 loop
      if R(i) = '1' and AVALID = '0' then
        A <= CONV_STD_LOGIC_VECTOR(i,3); AVALID <= '1';
      elsif R(i) = '1' and BVALID = '0' then
        B <= CONV_STD_LOGIC_VECTOR(i,3); BVALID <= '1';
      end if;
    end loop;
  end process;
end Vprior2_arch;
```

27

Dataflow model - Konkuretni izrazi – redosled kojim se pojavljuju ne menja značenje koda.
Svi se izvršavaju u paraleli. Kombinaciona logika.

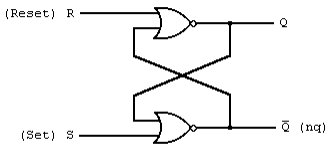
```
entity ckt is
port (A: in BIT:=1;  B: in BIT; Y,Z: out BIT);
end ckt;

architecture ckt of ckt is
begin
B <= A and A;
Y<= A and B;
Z<= B after 10 ns;
end ckt;
```

```
entity ckt is
port (A: in BIT:=1;  B: in BIT; Y,Z: out BIT);
end ckt;

architecture ckt of ckt is
begin
Y<= A and B;
Z<= B after 10 ns;
B <= A and A;
end ckt;
```

A = 1, B = 0, Y = 0, Z = 0

Y = 1, Z = 1

28



SR Latch

```
entity latch is
        port    (s,r : in bit;
                 q, nq : out bit);
end latch;

architecture dataflow of latch is
begin
        q<=r nor nq;
        nq<=s nor q;
end dataflow;
```

s='0', r='0', q='1', nq='0'

↓

r= '1' pa r= '0'

↓

start : r='0',s='0',q='1',nq='0'
round 1: r='1',s='0',q='1',nq='0', The value '0' is scheduled on q.
round 2: r='1',s='0',q='0',nq='0', The value '1' is scheduled on nq.
round 3: r='1',s='0',q='0',nq='1', No new events are scheduled.
round 4: r='0',s='0',q='0',nq='1', No new events are scheduled.

Behavioral model – sekvencijalno izvršavanje – dodela vrednosti na kraju - PROCESS

```
[process_label:]
PROCESS [( signal name {, signal name} )]
        [VARIABLE declarations]
BEGIN
        [WAIT statement]
        [Simple Signal Assignment Statements]
        [Variable Assignment Statements]
        [IF Statements]
        [CASE Statements]
        [LOOP Statements]
END PROCESS [process_label] ;
```

MUX 2/1

```
PROCESS ( Sel, x1, x2 )
BEGIN
    IF Sel = '0' THEN
        f <= x1 ;
    ELSE
        f <= x2 ;
    END IF ;
END PROCESS ;
```

```
PROCESS ( Sel, x1, x2 )
BEGIN
    f <= x1 ;
    IF Sel = 1 THEN
        f <= x2 ;
    END IF ;
END PROCESS ;
```

```
PROCESS ( Sel, x2 )
BEGIN
    IF Sel = 1 THEN
        f <= x2 ;
    END IF ;
END PROCESS ;
```