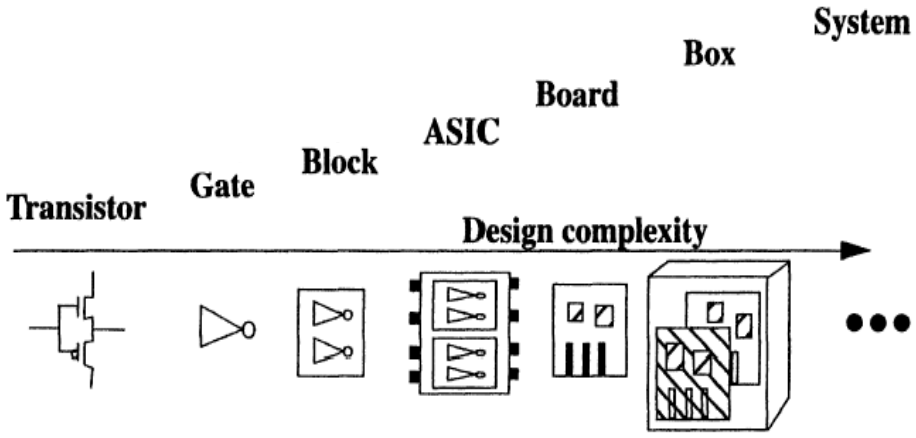
Prednosti VHDL-a u projektovanju digitalnih kola:

- Standard
- Podrška vlade
- Podrška industrije
- Prenosivost
- Mogućnost modelovanja
- Ponovna upotreba
- Tehnološka i proizvodna nezavisnost
- Dokumentacija
- Nova metodologija projektovanja

3

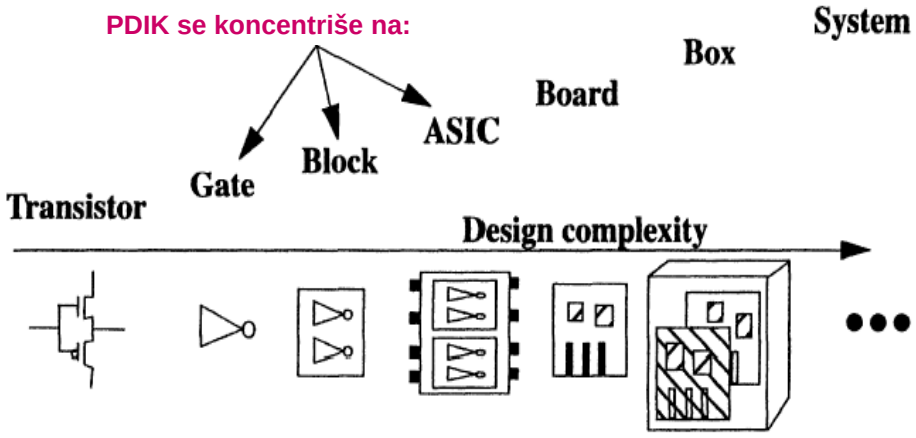
Mogućnosti modelovanja u VHDL-u:



4

Mogućnosti modelovanja u VHDL-u:

PDIK se koncentriše na:



5

Logička sinteza:

Sinteza, u domenu digitalnog projektovanja, predstavlja proces prevođenja i optimizacije.

Logička sinteza je proces

uzimanja neke forme ulaza (VHDL),

prevođenja u formu (Bulove jednačine i specifičnosti alata), i

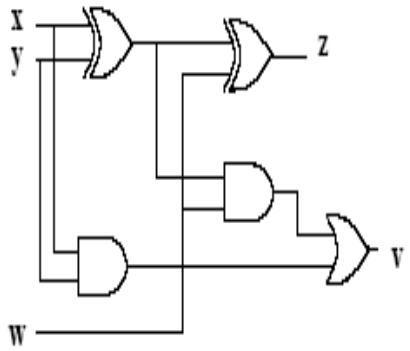
optimizacije u pogledu kašnjenja prostiranja i/ili površine.

6

Logička sinteza:

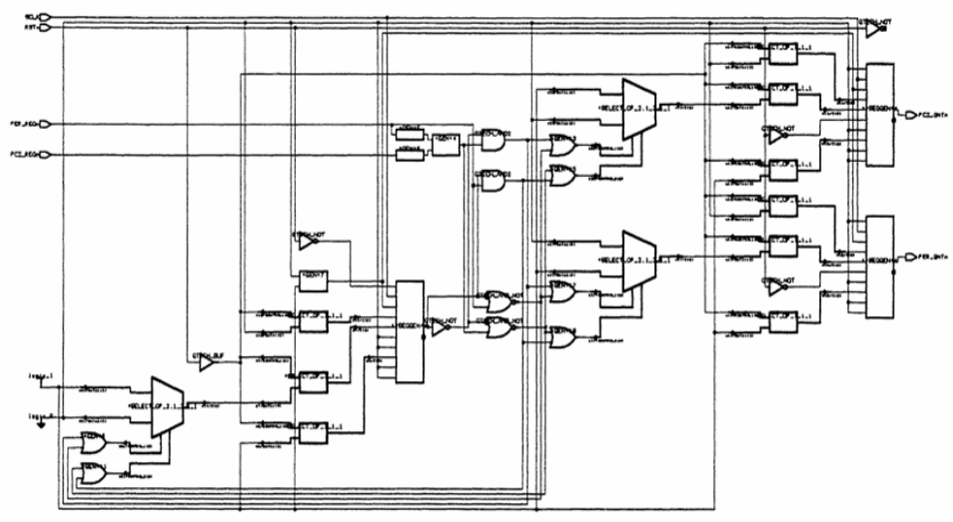
```
entity my_ckt is
port(x, y, w :in bit;
     v, z : out bit)
end entity my_ckt;

architecture behavioral of my_ckt is
begin
--
-- some code here
--
end architecture behavioral;
```

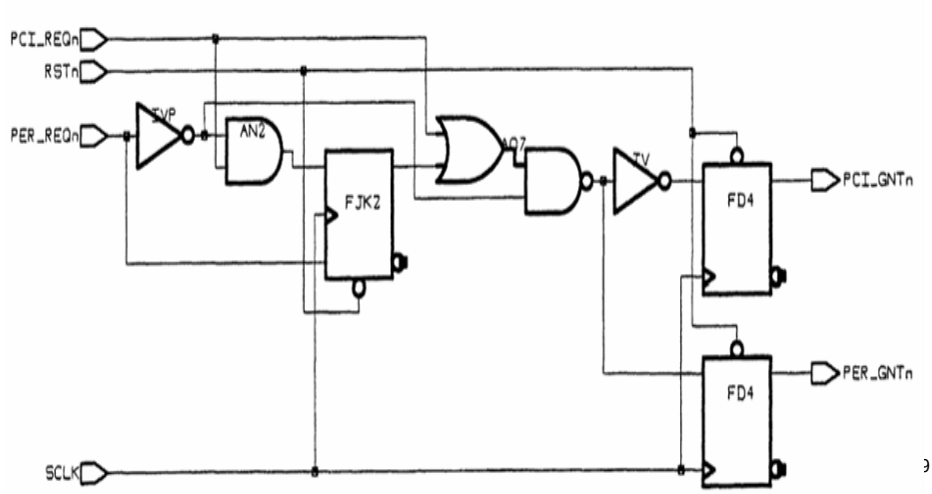


7

Sintetizovana šema:



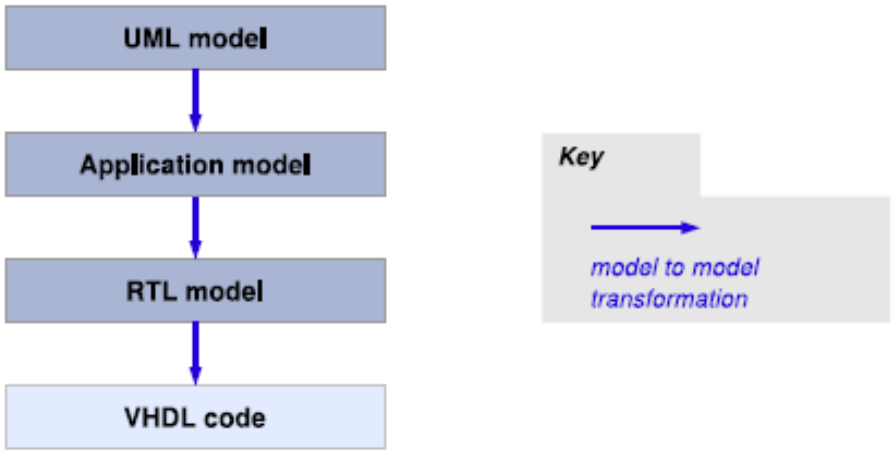
Optimizovana šema:



Generisanje VHDL koda

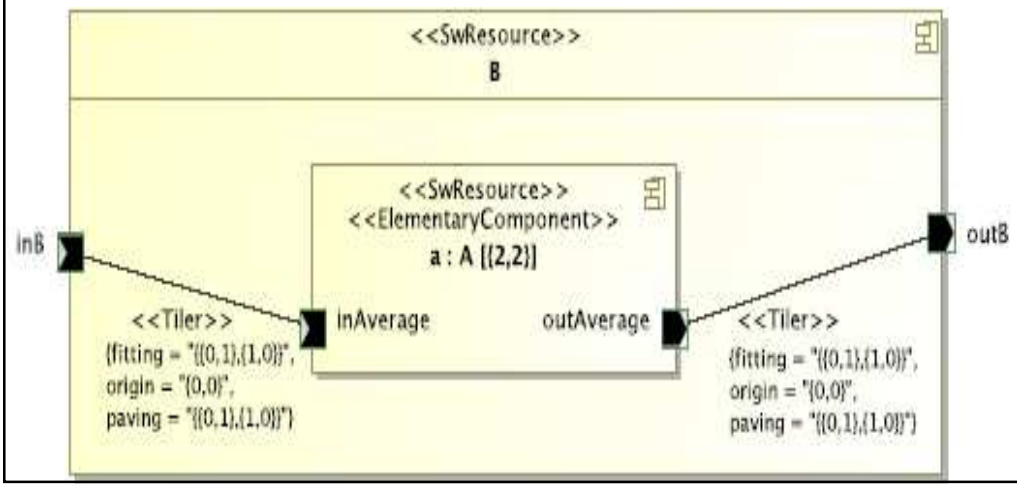
10

Jedan način generisanja VHDL koda:

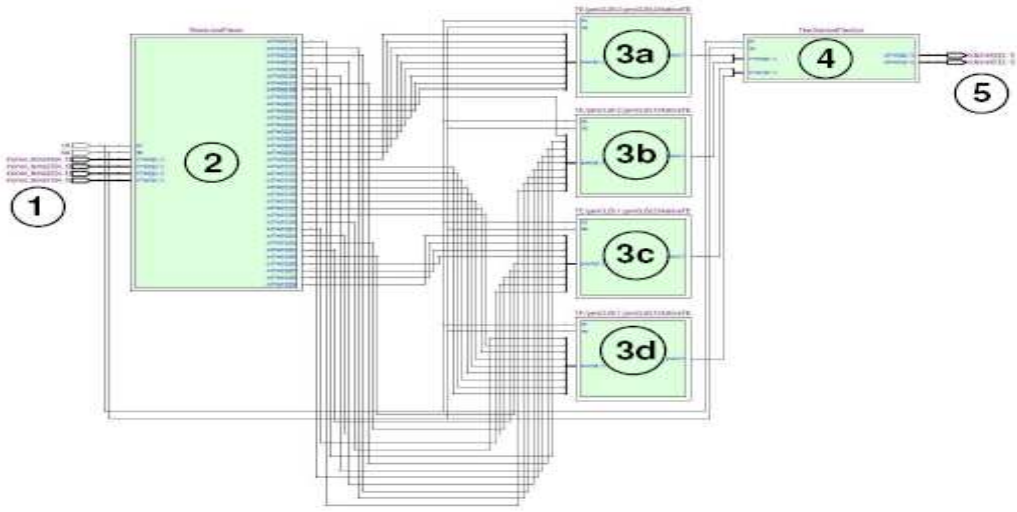


11

Modelovanje u UML-u dela algoritma za detekciju koji se koristi u anti-collision radaru:



Rezultat sinteze automatski generisanog VHDL koda:



14

Generalno, neostvarivi opis:

Skup procesa je zaključan ako svaki proces u skupu čeka događaj koji može da izazove samo drugi proces u skupu.

15

Synthesizable Subset of VHDL

Postoje konstrukcije koje je nemoguće prevesti u hardver
- Operacije nad fajlovima uključujući i tekstualni ulaz/izlaz
- Naredbe obaveštenja

Postoje neki dobri načini pisanja opisa za sintezu

Kod nekih konstrukcija mogu se sintetizovati samo prosti slučajevi
- Petlje (samo petlje sa konstantnim granicama)
- Function Calls (samo sa jednostavnim konstantnim parametrima)

Podsećanje: VHDL je razvijen za simulaciju!

16

Synthesizable Subset of VHDL

Postoji mnogo načina opisivanja ponašanja u VHDL-u
- Sintetizovani rezultat ne mora biti isti
- Može se desiti da je nemoguće sintetizovati VHDL opis

Podskup VHDL-a koji se može sintetizovati nije jedinstven
- Različiti alati mogu isti VHDL opis sintetizovati sasvim drugačije
- Skup neostvarivih naredbi različit je za različite alate

17

Zašto obraćati pažnju?

- Ako kod nije ostvariv, on može biti **neefikasan** (spor, zahtevati veliku potrošnju, neizvodljiv)
- Šta onda?
  - Možemo očekivati da alat za optimizaciju nađe različita rešenja optimizujući projekat po **brzini, površini i potrošnji.**

18

Podskup VHDL-a:

Većina alata za sintezu prihvata (prepoznaje) samo ***podskup*** komandi VHDL-a!

- Podskup VHDL komandi:
  - IEEE Standard je za VHDL Register Transfer Level Synthesis (RTL)

Dakle, provera se uvek vrši na RTL nivou.

19

IEEE Synthesis Standard in 1999 Supports VHDL-87 ONLY

VHDL-87 podržava samo:

- ASCII character set (not full)
- Nije podržano sledeće:

  group, impure, inertial, literal, postponed, pure, reject, rol, ror, shared,

  sla, sll, sra, srl, unaffected, xnor, protected.

20

Synthesizable Data Types:

- Enumeration types, including: **boolean, bit, character**
- integer types, including **integer**
- One-dimensional arrays of scalars, including: **bit_vector** and **strings**
- IEEE std_logic_1164 types:
  std_ulogic, std_ulogic_vector, std_logic, std_logic_vector
- IEEE ***numeric_bit*** package types: unsigned and signed
- IEEE ***numeric_std*** package types: unsigned and signed

Neprihvatljivo: brojevi u fiksnom zarezu i brojevi u pokretnom zarezu!

21

Synthesizable Scalar Types:

- Sledeće će biti implementirano kao ***bit***:
  - boolean, bit, std_ulogic, std_logic

Ostali tipovi enumeration tretiraju se kako odluči alat!

- Dodela stanja može se izvesti kao:

```
type state is (idle, preamble, data, crc, ok, error);
attribute enum_encoding of state: type is
        "000 001 010 011 100 111";
```

Koristiti isti broj bitova!

22

Rad sa Integer-ima:

Podržani do 32-bit integer:
opseg: -2^{31} to $2^{31}-1$.

Primeri:

type *sample* **is range** –64 **to** 63;
formiraće *sample* kao 7-bit integer u dvojičnom komplementu.

type *table_index* **is** natural **range** 0 **to** 1023;
implementiraće *table_index* kao 10-bitni neoznačeni broj.

23

Definisanje polja koja se mogu hardverski sintetizovati:

Ispravne definicije polja:

type *coeffs* **is array** (3 **downto** 0) **of** integer; -- 4 n-bitna elementa

type *channel_states* **is array** (0 **to** 7) **of** state; -- 8 m-bitna elementa

-- usvaja se da je state tipa enumeration

subtype *word* **is** bit_vector (31 **downto** 0); -- 32 single bita

type reg_file **is array** (0 **to** 15) **of** word; -- 16 32-bit elementa

24

Sledeće se ne može sintetizovati!

type *color* **is array** (red, green, blue);
type *plane_status* **is array** (color) **of** boolean;
-- color NIJE integer! (može ipak da radi)

type *matrix* **is array** (1 **to** 3, 1 **to 3**) **of** real;
-- NE MOGU se sintetizovati 2-D polja (bilo bi lepo!)
-- NE MOGU se sintetizovati floating-point brojevi (još uvek!)

type *reg_file_set* **is array** (0 **to** 3) **of** reg_file;
-- elementi su vektori ne-bitova! (videti prethodne slajdove)

25

Koristiti:

Označene integere:

- Koristiti ***signed*** definiciju iz IEEE biblioteka ***numeric_bit*** i ***numeric_std***.

Neoznačene integere:

- Koristiti ***unsigned*** definiciju iz IEEE biblioteka ***numeric_bit*** and ***numeric_std***.

26

Kako dodeljivanja utiču na sintezu?

- Alat za sintezu **će najčešće ignorisati** 'U', 'Z', 'X' i sve ostale vrednosti.

Posle sinteze, svaki bit se preslikava u 0 ili 1.

- Ako promenljivoj dodelimo 'Z', biće implementirana kao tri-state buffer.

Na primer:

```
request <= 'Z';
-- implementiraće tri-state buffer da ostvari request.
```

- Koristiti ***std_match*** funkciju iz IEEE biblioteke ***numeric_std***

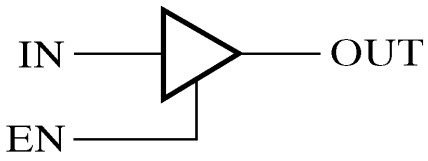
za poređenje umesto upotrebe '='!

Ovo garantuje da ćemo dobiti iste rezultate u simulaciji i sintezi. Na primer:

```
std_match ('0', '0')  -- vraća true
std_match ('0', '1')  -- vraća false
```

27

Setimo se Buffera …



(a) Logic symbol

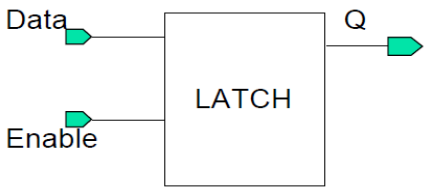
EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

(b) Truth table

28

If Statement => LATCH

process (ENABLE, DATA)
begin
if ENABLE = '1' **then**
Q <= DATA;
end if;
end process;



29

Kombinaciona kola: Multiplekser

Za logiku multipleksera koristiti “select”:

```
with addr(1 downto 0) select
    request <= request_a when “00”,
               request_b when “01”,
               request_c when “10”,
               request_d when “11”;
```

Ne koristiti “else” naredbu, osim ako treba implementirati *priority encoder*!

30

Kombinaciona kola: Pravila za procese

Možemo koristiti naredbu **process**:

- **Proces mora biti osetljiv na sve ulaze**
- **Svi izlazi moraju biti dodeljeni u mogućim izvršenjima**
- **Promenljive moraju biti dodeljene pre čitanja**
  - *U suprotnom, promenljive će biti implementirane koristeći memoriju a ne kombinaciona kola*

31

Kombinaciona kola - primer procesa (1):

```
read_sample : process (read_enable, sample, limit_exceeded, ready)
begin
    if  std_match(read_enable, '1') then
          data   <= sample;
          parity <= calc_parity(sample);
          status <= ready and not limit_exceeded;
    else
          data   <= sample;
          parity <= calc_parity(sample);
          status <= ready and not limit_exceeded;
    end if;
end process read_sample;
```

32

Kombinaciona kola - komentar primera procesa:

- Svaka promenljiva/signal koja se pojavljuje na desnoj strani dodeljivanja ili if-naredbe, pojavljuje se, takođe, u listi osetljivosti (sensitivity list) **OSIM lokalnih promenljivih/signala. Treba primetiti da lokalne promenljive ne mogu da se pojave u listi osetljivosti (sensitivity list).**
  - zadovoljen prvi zahtev
- U svakom izvršavanju dodeljene su sve promenljive
  - zadovoljen drugi zahtev
- U ovom procesu nema promenljivih!
  - zadovoljen treći zahtev

33

Kombinaciona kola - primer procesa (2):

```
adder : process (sel, a, b, c)  -- svi znacajni ulazi
    variable operand : integer; -- LOCAL variable. Nije u sensitivity list.
begin  -- sve izvršni threads moraju da dodele operande.
    if sel='1' then
        operand := a;     -- operand dodeljen i NOT READ
    else
        operand := b;     -- operand dodeljen i NOT READ
    end if;
    sum <= operand + c;   -- operand je već dodeljen. READ IS OK
end process adder;          -- nema potrebe memorisati operand!
```

34

Sekvencijalna kola – generalno:

- *Nije dobro podržana sinteza asinhronih kola. Asinhroni znači da nema Clock ili Enable signala.*
- Dobra podrška samo za sintezu **synhronih** kola (clock ili enable mora da postoji)
- Clock signal mora biti definisan kao tip:
  bit, std_ulogic ili njegovi podtipovi (npr: std_logic)

35

Ivično okidan proces:

process_label : **process** (clock_signal_name)
begin
-- videti sledeći slajd za odgovarajuću definiciju
-- ispravnih naredbi za valid clock_edge.
if clock_edge **then**
-- NIJE dopuštena naredba wait!
-- NIJE dopušteno referenciranje neke ivice!
sequential_statements;
end if;
end process_label;

36

Definisanje rastuće ivice:

rising_edge (*clock_signal_name*) -- best one! from IEEE library …

clock_signal_name'event **and** clock_signal_name='1'
clock_signal_name='1' **and** clock_signal_name'event

-- nadalje, **not** prethodi za **and**
-- takođe, 'stable je atribut za signale
not clock_signal_name'stable **and** clock_signal_name='1'
clock_signal_name='1' **and** **not** clock_signal_name'stable

37

Definisanje opadajuće ivice:

falling_edge (*clock_signal_name*) -- best one! from IEEE library …

clock_signal_name'event **and** clock_signal_name='0'
clock_signal_name='0' **and** clock_signal_name'event

-- nadalje, **not** prethodi za **and**
-- takođe, 'stable je atribut za signale
not clock_signal_name'stable **and** clock_signal_name='0'
clock_signal_name='0' **and** **not** clock_signal_name'stable

38

Sinhroni brojač sa asinhronom kontrolom:

```
count_byte: process (clk, rst_n, load, load_data) – sens. list: clk + async controls
    variable count : unsigned(7 downto 0);
begin
    if  std_match(rst_n, '0') then     -- async negated reset
          count := "00000000";
          q <= count;                  -- q is a signal defined elsewhere.
    elsif  std_match(load, '1') then  -- async load
          count := load_data;
          q <= count;
    elsif  rising_edge(clk) then       -- only one check for the edge!
          count := count + 1; -- count IS a STATIC variable implying storage.
          q <= count;          -- NOTE that we need storage for this to work.
    end if;
end process count_byte;
```

39

Drugi oblik sinhrone kontrole:

```
shift_reg : process
    variable stored_value : bit_vector(7 downto 0);
begin
    -- wait statement must be first statement (no more allowed)
    wait until clk='1' -- rising edge. It will be '0' here for falling edge
    if  load='1' then   -- synchronous load
      stored_value := load_data_in;  -- load_data_in externally defined
      q <= stored_value;
    else
      -- NOTE that stored_value MUST be STORED for this to work.
      stored_value := stored_value(6 downto 0) & serial_data_in;
      q <= stored_value;
    end if;
end process shift_reg;
```

40

Synchronous and Asynchronous Resets

Synchronous Reset	Asynchronous Reset
Large number of gates	Less number of gates
Reset pulse should be large enough for clock to become active	Only to cover setup and hold times
No metastability problems	Mestability problems
Slow	Fast
Consumes more power	Consumes less power
Small reset glitches are automatically filtered	May reset the circuit if glitch covers setup and hold time

- No choice is in general better, depends on design
- Can convert asynchronous reset to synchronous and then use as asynchronous input to all ip-ops

41

Logika osetljiva na nivo:

- Nema Clock signala
- Obično se koristi *enable signal*

Memorija potrebna:

- ako postoji put koji ne dodeljuje sve signale/promenljive
- kada se signal/promenljiva čita pre nego je dodeljena (kada nema Clock signala)

42

Primer latch-a (1):

```
latch : process (enable, d) – Need: enable and read (see below)
-- sensitivity list includes all variables/signals read.
begin
-- Place everything within an if-statement of
-- the enabling signal. Place sequential statements afterwards
if  enable = '1' then  -- enable is read here
      q <= d;          -- d is read here
end if;
end process  latch;
-- Note that if enable is zero, there is no value stored for q. Thus, we
-- need a storage element for implementing q (NOT combinational)
```

43

Kada se signal/promenljiva “čita”?

Signal se čita:

- ako se javlja na desnoj strani naredbe dodeljivanja
- ako se javlja u bilo kome uslovnom izrazu

U osnovi, ako se signal/variable javlja u telu procesa, (ili procedure, …), i nije dodeljena vrednost, onda se “čita”.

44

Primer latch-a (2):

```
latch_with_reset : process (enable, reset, d)
   variable stored_value : bit;
begin
if reset='1' then
   stored_value := '0';
elsif enable='1' then     -- stored_value is not assigned
   stored_value := d;     -- when reset=enable=0.
end if;                   -- Thus, we create storage for it!
   q <= stored_value;
end process latch_with_reset;
```

45

Modeling Finite State Machines (I of III)

```
architecture rtl of  state_machine  is
    type state is (ready, ack, err);
    signal current_code, next_state : state;
begin
-- Define two processes inside the architecture:
--  1. A combinational process (call next_state_and_output), and
--   2. A register process for storing the current state (state_reg).

next_state_and_output : process (current_state, in1, in2)
  -- everything depends on:  current state  and  all the inputs.
  begin
    case current_state is  -- case statement for parallel implementation.
      when ready =>
      -- Here, current_state=ready, provide the logic for computing
      -- the next_state
```

46

Modeling Finite State Machines (II of III)

```
        out1 <= '0';              -- Note that ALL outputs and current_state
        if  in1 = '0'  then       -- are assigned on EVERY execution path.
          out2 <= '1';            -- Else, synthesis may not be combinational.
          next_state <= ack;
        else
          out2 <= '0';
          next_state <= ready;
        end if;
    when ack =>                   -- Similarly. Make sure to do this for ALL states.
        …
    when err  =>
        …
    end case;
end process next_state_and_output;
```

47

Modeling Finite State Machines (III of III)

```
-- the second process is here to define the logic for the register that stores
-- the current state. It has an asynchronous reset input.
    state_reg : process (clk, reset)
    begin
      if  reset='1' then
          current_state <= ready;
      elsif clk'event and clk='1' then
          current_state <= next_state;  -- note the assignment for next_state
      end if;                       -- causing next_state to be read.
    end
end rtl;
```

48

Metacomment:

Alat za sintezu ignorisaće ceo kod između:

-- rtl_synthesis off

…

-- rtl_synthesis on

Međutim, simulator će to tretirati kao regularan kod.

49


Dobro proučiti
raspoloživi alat za sintezu i
VHDL kod prilagoditi alatu!