Prostriedky opisu správania sa

opis správania sa na dvoch úrovniach - sekvenčnej a paralelnej.

Sekvenčná úroveň predstavuje programovanie správania sa procesu. Program, obsahujúci sekvenčné príkazy jazyka VHDL.

Paralelná úroveň definuje vzťahy medzi paralelnými procesmi modelu, hlavne ich vzájomnú komunikáciu - paralelné príkazy jazyka VHDL.

Paralelné príkazy opisujúce správanie sa systému:

- paralelný príkaz priradenia signálu,
- paralelný príkaz ASSERTION,
- príkaz blok,
- paralelné volanie procedúry a
- príkaz proces.

Pri opise správania sa používajú 2 štýly:

Opis toku údajov (dataflow style) modeluje obvod z hľadiska pohybu údajov v spojitom čase medzi komponentami kombinačnej logiky (sčítačky dekódery, primitívne hradlá). Opis na úrovni RTL.

Algoritmický opis správania sa (behavioural style) – obsahuje procesy opísané na sekvenčnej úrovni, ktoré umožňujú opísať výstupy obvodu v diskrétnych okamihoch daných konkrétnymi vstupmi, presnejší opis časových závislostí. Opis sekvenčných aj kombinačných obvodov.

Paralelné príkazy jazyka VHDL

Paralelný príkaz priradenia signálu

Všeobecná forma opisu syntaxe *paralelného príkazu priradenia signálu* (zjednodušená):

concurrent_signal_assignment_statement ::=

[label:] conditional_signal_assignment

| [label:] selected_signal_assignment

options ::=[**guarded**] [delay_mechanism]

conditional_signal_assignment ::=

target <= options { waveform **when** condition **else** }

waveform [**when** condition];

target ::= *signal*_name

waveform ::= waveform_element {, waveform_element}

|**unaffected**

waveform_element ::=

*value*_expression [**after** *time*_expression]

selected_signal_assignment ::=

with expression **select**

target <= options selected_waveforms;

selected_waveforms ::= { waveform **when** choices, }

waveform **when** choices

Výber môže mať jednu z nasledujúcich foriem :

value

value1 **to** value2 -- spojitý rozsah hodnôt

value1 |{value2 |} valueN -- množina hodnôt

Príklady:

```
entity Substractor is
    port (In1, In2: in Integer; Out1: out Integer);
end Substractor;
architecture Simplest of Substractor is
begin
    Out1 <= In2 - In1 after 8 ns;
end Simplest;
```

```
Clock_Process:
    Clock <=
        '1'after 1 ms,
        '0'after 2 ms,
        '1'after 3 ms,
        '0'after 4 ms,
        '1'after 5 ms;
```

```
--Príklad opisuje správanie sa úplneho dekódera 3x8.
entity Decoder is
    port (Sel: Bit_Vector(2 downto 0);
          DOut: out Bit_Vector(7 downto 0));
    constant Delay: Time := 5 ns;
end Decoder;
architecture Selected of Decoder is
begin
    with Sel select
        DOut <=
            "00000001" after Delay when "000",
            "00000010" after Delay when "001",
            "00000100" after Delay when "010",
            "00001000" after Delay when "011",
            "00010000" after Delay when "100",
            "00100000" after Delay when "101",
            "01000000" after Delay when "110",
            "10000000" after Delay when "111",
end Selected;
```

Paralelný príkaz ASSERTION

Všeobecný opis syntaxe *príkazu assertion*:

```
assertion_statement ::=  [assertion_label:]
                         assert condition
                         [ report expression ]
                         [ severity expression ];
```

Príklad príkazu assertion:

```
  assert Enable /= 'X'
    report "Unknown value on Enable"
    severity Error;
```

```
-- paralelný príkaz ASSERTION
entity SRFF is
      port (S, R : in Bit;      Q, Qbar : out Bit);
begin
      SRFF_Constraint_Check:
      assert not (S = '1' and R = '1')
            report "Both S and R equal to '1'"
                  severity Error;
end SRFF;
architecture Constrained of SRFF is
begin
      Q <=       Q after 2 ns when S = '0' and R = '0' else
                 '1' after 2 ns when S = '1' and R = '0' else
                 '0' after 2 ns when S = '0' and R = '1';
end SRFF;
```

Príkaz block

Syntax príkazu block:

block_statement ::=
   *block*_label:
   block [(*guard*_expression)] [**is**]
    block_header
    { block_declarative_item}
   begin
    { concurrent_statement }
   end block [*block*_label] ;

block_header ::=
   [**generic** (*generic*_interface_list);
   [**generic map** (*generic*_association_list);]]
   [**port** (*port*_interface_list);
   [**port map** (*port*_association_list);]]
block_declarative_item ::=
   subprogram_declaration
   | subprogram_body
   | type_declaration
   | subtype_declaration
   | constant_declaration
   | signal_declaration
   | component_declaration
   | configuration_specification
   | use-clause

Použitie príkazu block:

```
S <= '0' after 4ns;
L1: block
   begin
       T <= '1' after 5ns;
end block;
```

ekvivalentný zápis:

```
S <= '0' after 4ns;
T <= '1' after 5ns;
```

```
--Použitie stráženého príkazu block:
entity d_ff is
   port (d: in Bit; clk: in Bit; q: out Bit);
end d_ff;
architecture beh of d_ff is
signal dff_s: Bit;
begin
   label0: block (clk='0' and not clk'stable) is
   begin
      dff_s <= guarded not d;
   end block;
   q<= not dff_s;
end beh;

entity Guard_gate is
  port( A, EN: in Bit;
        Y: out Bit bus);
end Guard_gate;
architecture beh of Guard_gate is
signal temp_Y: bit bus;
begin
     B: block (EN = '1') is
     begin
          temp_Y <= guarded A ;
     end block;
     Y<= temp_Y;
end beh;
```

Príkaz proces

Syntax *príkazu proces*:

process_statement ::=

[*process*_label:]

[**postponed**] **process** [(sensitivity_list)] [**is**]

process_declarative_part

begin

{sequential_statement}

end [postsponed] process [*process*_label];

Deklaračná časť príkazu process môže obsahovať nasledujúce deklarácie:

- základné deklarácie (typu, podtypu, konštanty, aliasu, podprogramu, súboru, use klauzula)
- telo podprogramu
- deklarácie premenných
- deklarácie atribútov
- špecifikácie atribútov

Deklarácie premenných sú typické pre príkazy process a podprogramy.

Syntax deklarácie premennej:

variable_declaration ::=

[**shared**] **variable** identifier_list : subtype_indication [:= initial_value] ;

Príklad procesu, ktorý deklaruje jednu premennú:

```
P1:   process
   variable count : Integer := 0;
   begin
      count := count + 1;
      wait;
   end process;
```

Sekvenčné príkazy jazyka VHDL

Príkaz wait

Syntax *príkazu wait*:

```
wait_statement ::= [label:] wait [sensitivity_clause]
                          [condition_clause] [timeout_clause] ;
sensitivity_clause::= on signal_list;
condition_clause ::= until condition;
timeout_clause ::= for time_expression;
```

Príklad použitia príkazu **wait**:

```
And_Process:
process
begin
   wait on A, B until Enable = '1';
   T <= transport A and B;
end process;
```

Príklad použitia zoznamu citlivosti:

```
Or_Process:
  process (In1, In2) -- sensitivity list
  begin
    Output <= In1 or In2;
  end process;
```

Tento proces je ekvivalentný procesu:

```
Or_Process:
  process
  begin
    Output <= In1 or In2;
    wait on In1, In2;
  end process;
```

Priradenie premennej (Variable Assignment)

Všeobecný opis syntaxe príkazu priradenia premennej:

variable_assignment_statement ::=
[label:] target := expression;
target ::= *variable*_name | aggregate

```
-- THIS VHDL IS NOT LEGAL
variable V: Integer := 0;
Process_One:
  process
  begin
    V := V + 1 ;
    wait;
  end process;
```

```
Process_Two:
  process
    variable X: Integer;
  begin
    X := V;
    wait;
  end process;
```

Priradenie signálu (Signal assignment)

Všeobecný opis syntaxe *príkazu priradenia signálu*:

signal_assignment_statement ::=

[label:] target <= [delay_mechanism] waveform;

delay_mechanism ::= **transport** | [**reject** *time*_expression] **inertial**

target ::= *signal*_name | aggregate

waveform ::= waveform_element {, waveform_element}

| **unaffected** -- only concurrent

waveform_element ::=

*value*_expression [**after** *time*_expression]

| **null** [**after** *time*_expression] -- only guarded signal or aggregate in sequential statement

```
entity Sub is
    port(In1, In2: in Integer; Out1: out Integer);
end Sub;
architecture A_Simplest of Sub is
begin
  process (In1,In2)
  begin
     Out1 <= In2-In1 after 8 ns;
   end process;
end A_Simplest;
```

```
architecture Simplest of Sub is
begin
    Out1 <= In2-In1 after 8 ns;
end Simplest;
```

```
Clock_Process:
process
begin
Clock <=
        '1'after 1 ms,
        '0'after 2 ms,
        '1'after 3 ms,
        '0'after 4 ms,
        '1'after 5 ms;
wait;
end process;
```

```
Clock_Process:
Clock <=
        '1'after 1 ms,
        '0'after 2 ms,
        '1'after 3 ms,
        '0'after 4 ms,
        '1'after 5 ms;
```

```
signal A: Bit := '0';
signal B: Bit := '1';
P1:
process
begin
  B <= A;
  assert (B = A)
    report "B nerovna sa A"
      severity error;
  wait on B;
end process;
```

```
signal A: Bit := '0';
signal B: Bit := '1';
P1:
process
begin
  B <= A;
  wait on B;
  assert (B = A)
    report "B nerovna sa A"
      severity error;
end process;
```

Oneskorenie priradenia signálu

VHDL rozoznáva 2 typy oneskorení: ***zotrvačné*** (inertial) a ***prenosové*** (transport).

-- The following assignments are equivalent:

Output_pin <= Input_pin **after** 10 ns;

Output_pin <= **inertial** Input_pin **after** 10 ns;

Output_pin <= **reject** 10ns **inertial** Input_pin **after** 10 ns;

-- Assignment with a *pulse rejection limit* less than the time expression:

Output_pin <= **reject** 5ns **inertial** Input_pin **after** 10 ns,

not Input_pin **after** 20 ns;

-- Assignments using transport delay:

Output_pin <= **transport** Input_pin **after** 10 ns;

Output_pin <= **transport** Input_pin **after** 10 ns, **not** Input_pin **after** 20 ns;

-- Their equivalent assignments:

Output_pin <= **reject** 0ns **inertial** Input_pin **after** 10 ns;

Output_pin <= **reject** 0ns **inertial** Input_pin **after** 10 ns, **not** Input_pin **after** 20 ns;

Aktualizácia ovládača – prenosové oneskorenie:

- Všetky staré transakcie plánované neskôr, alebo súčasne ako prvá nová transakcia sa zrušia.
- Nové transakcie sa pripoja na koniec ovládača.

Ďalšie pravidlá pre zotrvačné oneskorenie:

- Všetky nové transakcie sa označia.
- Staré transakcie sa označia, ak sú naplánované v čase, ktorý je menší než čas prvej novej transakcie mínus minimálna šírka impulzu.

- Ostatné staré transakcie sa označia, ak bezprostredne predchádzajú označenú transakciu a ich hodnota je zhodná s hodnotou tejto transakcie.
- Transakcia, ktorá určuje okamžitú hodnotu ovládača signálu sa označí.
- Všetky neoznačené transakcie v ovládači signálu sa zrušia.

Ilustračné príklady – *prenosové oneskorenie*:

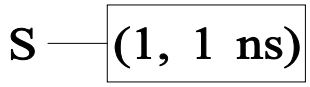
Príklad 1:

signal S:Integer:=0;

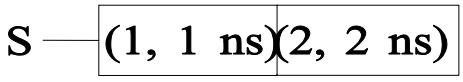
P1:

```
process
begin
  S <= transport 1 after 1 ns;
  S <= transport 2 after 2 ns;
  wait;
end process;
```

Driver of signal S:



S <= **transport** 1 **after** 1 ns;



S <= **transport** 2 **after** 2 ns;

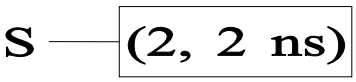
Príklad 2:

signal S:Integer:=0;

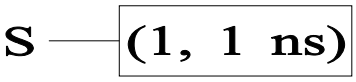
P1:

```
process
begin
  S <= transport 2 after 2 ns;
  S <= transport 1 after 1 ns;
  wait;
end process;
```

Driver of signal S:



S <= **transport** 2 **after** 2 ns;

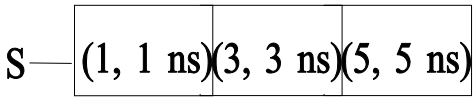


S <= **transport** 1 **after** 1 ns;

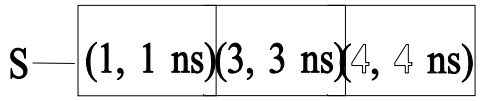
Príklad 3:

```
signal S:Integer:=0;
P1:  process
  begin
    S <= transport 1 after 1 ns, 3 after 3 ns, 5 after 5 ns;
    S <= transport 4 after 4 ns;
    wait;
  end process;
```

Driver of signal S:



S <= **transport** 1 **after** 1 ns, 3 **after** 3 ns, 5 **after** 5 ns;



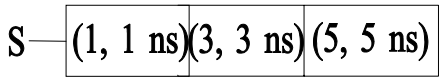
S <= **transport** 4 **after** 4 ns;

Ilustračné príklady – *zotrvačné oneskorenie*:

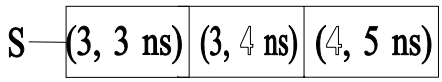
Príklad 1.

```
signal S:Integer:=0;
P1:
  process
  begin
    S <= 1 after 1 ns, 3 after 3 ns, 5 after 5 ns;
    S <= 3 after 4 ns, 4 after 5 ns;
    wait;
  end process;
```

Driver of signal S:



S <= 1 **after** 1 ns, 3 **after** 3 ns, 5 **after** 5 ns;



S <= 3 **after** 4 ns, 4 **after** 5 ns;

Príklad 2.

signal S:Integer:=0;

P1:

process

begin

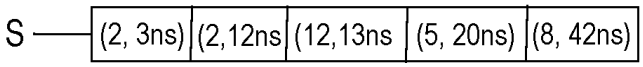
S <= 2 **after** 3ns, 2 **after** 12ns, 12 **after** 13ns, 5 **after** 20ns, 8 **after** 42ns;

S <= **reject** 15ns **inertial** 12 **after** 20ns, 18 **after** 41ns;

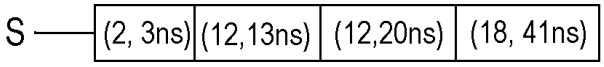
wait;

end process;

Driver of signal S:



S <= 2 **after** 3ns, 2 **after** 12ns, 12 **after** 13ns, 5 **after** 20ns, 8 **after** 42ns;



S <= **reject** 15ns **inertial** 12 **after** 20ns, 18 **after** 41ns;

S <= **reject** t_r **inertial** v_1 **after** t_1 {, v_i **after** t_i};

Pre tento tvar platí vzťah:

$0ns \leq t_r \leq t_1 \wedge 0ns < t_i < t_{i+1}$

Príklad

Nakreslite časový priebeh signálov X a Z, ak model obvodu obsahuje iba nasledujúce 2 procesy:

Z<=X **after**10ns;

X<= '1', '0' **after** 5ns;

a inicializačné honoty oboch signálov sú '0'.