Zaawansowany VHDL

Program wykładu

Opis strukturalny

`map`, `generate`

Pojęcia leksykalne

obiekty: sygnały, zmienne, **stałe, parametry**

Instrukcje sekwencyjne

`process`, `wait`, `if`, `case`, `loop`, `next`, `exit`, `assert`, `function`, `procedure`

Instrukcje współbieżne

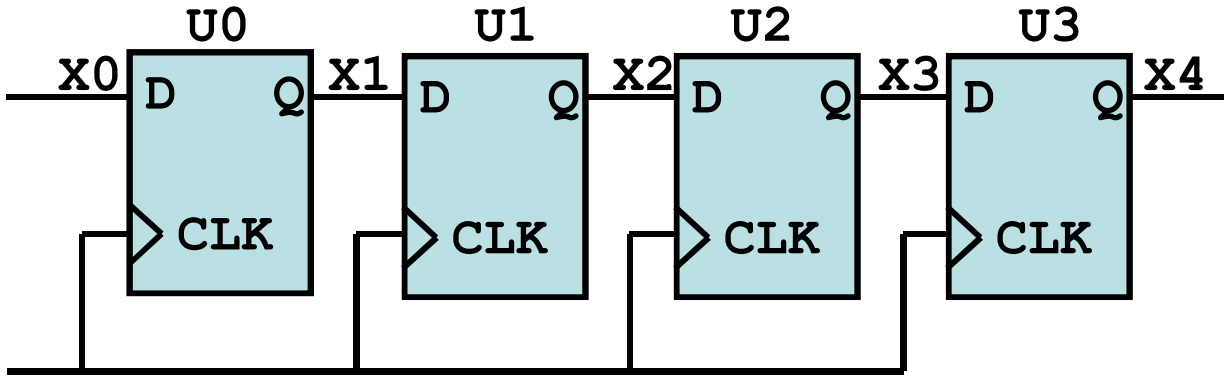
przypisania: bezwarunkowe, warunkowe i decyzyjne, **podprogramy,** `block`

Instrukcja generate

Składnia:

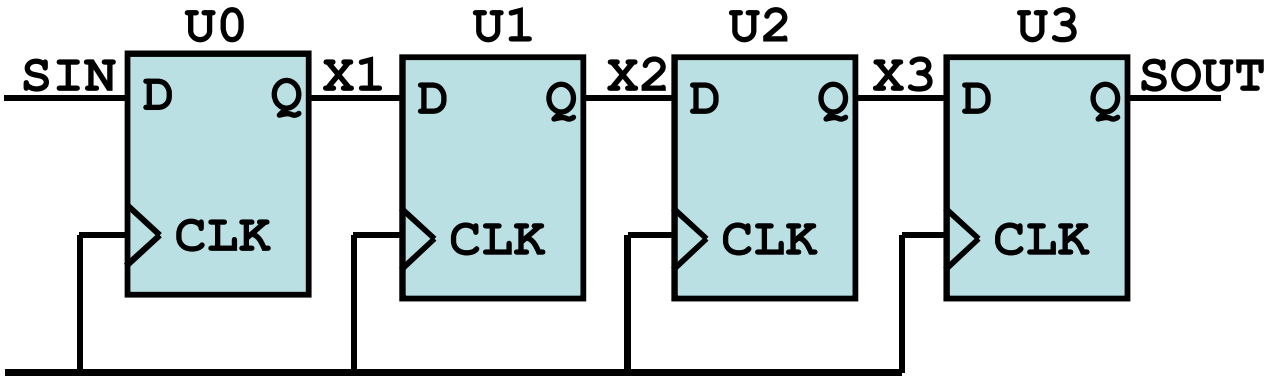
[*etykieta* :]
{[**for** *instrukcja* **if** *warunek*]} **generate**
{instrukcje_współbieżne}
end generate;

Przykład:




```
gen1: for i in 0 to 3 generate
        U: DFF port map (X(i), clk, X(i+1));
end generate;
```

Instrukcja generate




```
for i in 0 to 3 generate
  if (i=0) generate
    UA: DFF port map (SIN, CLK, X(i+1)); end generate;
  if ((i>0) and (i<3)) generate
    UB: DFF port map (X(i), CLK, X(i+1)); end generate;
  if (i=3) generate
    UC: DFF port map (X(i), CLK, SOUT); end generate;
end generate;
```

Deklaracje stałych

Deklaracje stałych

- **skalarnych:**

  constant *name*: *type* **:=** *expression*;

- **tablicowych:**

  constant *name*: *array_type* **[(***index***)]** **:=** *expression*;

np:

```
constant Vcc: real := 5.0;
constant Cycle: time := 50 ns;
constant five: bit_vector := "0101";
constant SIX: std_logic_vector (8 to 11):= "0110";
```

Deklaracje parametrów

Deklaracje parametrów

```
generic (generic_interface_list);
generic ( name: type := expression);
```

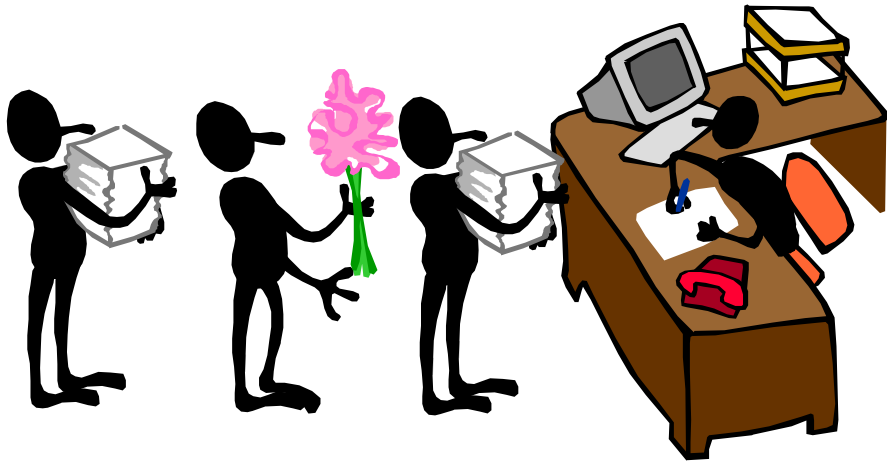
Deklaruje statyczną wartość podobnie jak stała, ale wartość ta może być zmieniana z zewnątrz. Może być deklarowany w: *entities* (dostępny we wszystkich architekturach z nią skojarzonych), blokach i komponentach. Wykorzystywany do parametryzacji modelów (szerokość magistral, czasy opóźnień itp.) Wartość zainicjowana w *entity* może być nadpisana w deklaracji komponentu.

np:

```
entity CPU is
  generic (BusWidth : integer := 16)
  port (
    DataIn  : in  bit_vector(BusWidth-1 downto 0);
    DataOut : out bit_vector(BusWidth-1 downto 0);
```

Instrukcje sekwencyjne

- `process` (*współbieżna!*)
- instrukcje przypisania
- `wait`
- `if`
- `case`
- `null`
- `loop`
- `next`
- `exit`
- `assert`
- **podprogramy**



Instrukcja loop

Przydatna do multiplikacji logiki i w modelowaniu behawioralnym.

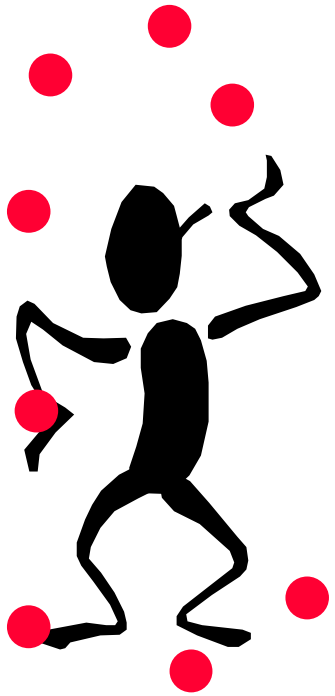
Składnia:

[*etykieta* :]
[while *warunek* | for *index* in *valA* to *valZ*] loop
    instrukcje sekwencyjne ;
end loop;

Przykłady:

```
L: for i in 1 to 10 loop
    instrukcje sekwencyjne;
end loop;

M: while i < 11 loop
    instrukcje sekwencyjne;
    i := i + 1;
end loop;
```

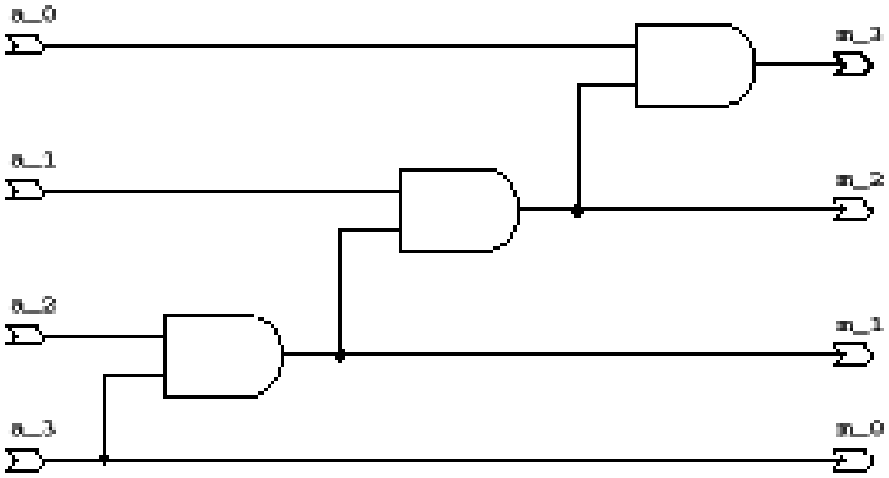


Instrukcja loop

Przykład: powielanie logiki

```
entity multi_and is
   port (a: in bit_vector (0 to 3);
         m: out bit_vector (0 to 3));
end multi_and;

architecture example of multi_and is
begin
  process (a)
    variable b: bit;
    begin
      b := '1';
      for i in 0 to 3 loop
        b := a(3-i) and b;
        m(i) <= b;
      end loop;
  end process;
end example;
```



Instrukcja next

Składnia:

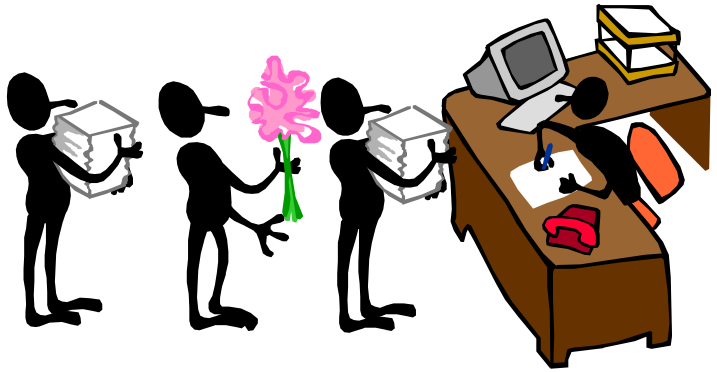
```
next [etykieta] [when warunek];
```

Przykłady:

```
for i in 0 to max_limit loop
   if a(i) = 0 then next;
   end if;
   g(i) := a(i);
end loop;
```

Natychmiastowe przejście do kolejnej iteracji

```
L1: while i < 5 loop
L2: while j < 5 loop
     ...
        next L2 when i=j;
     ...
     end loop L2;
     end loop L1;
```



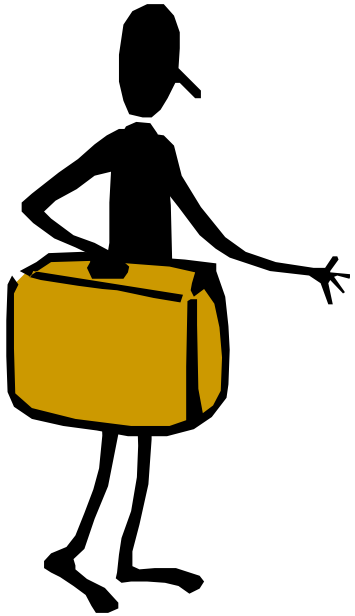
Instrukcja `exit`

Składnia:

```
exit [etykieta] [when warunek];
```

Przykład:

```
for i in 0 to max_limit loop
      if a(i) = 0 then exit;
      end if;
      g(i) := a(i);
end loop;
...
```



Natychmiastowe opuszczenie pętli

Instrukcja assert

Drukuje na konsoli komunikat podczas symulacji.

Składnia:

```
assert warunek [report string] [severity level];
level: FAILURE | ERROR | WARNING | NOTE
```

Przykład:

```
assert (Machine_Code = "0000")
  report "Illegal Opcode"
  severity FAILURE;
```

W VHDL'92 umożliwiono używanie `report` bez `assert`.

Instrukcja `assert` może występować w deklaracji `entity` (ponieważ jest bierna – nie ma przypisania).

Podprogramy – instrukcja function

Funkcje zwracają jedną wartość.

Składnia:

```
function nazwa [(parametr : typ ; ...)] return typ is
deklaracje
begin
    instrukcje sekwencyjne ;
end [nazwa ];
```

Przykład: (z biblioteki std_logic_1164, sprawdza czy wartość jest *unknown*)

```
function Is_X (s : STD_ULOGIC) return BOOLEAN is
begin
  case s is
    when 'U' | 'X' | 'Z' | 'W' | '-' => return true;
    when others                       => null;
  end case;
  return false;
end function Is_X;
```

Podprogramy – instrukcja `procedure`

Procedury mogą zwracać więcej niż jedną wartość.

Składnia:

```
procedure nazwa [(parametr;...)] is
deklaracje
begin
    instrukcje sekwencyjne;
end [nazwa]
```

parametry:

```
{[variable] nazwy: [in | out | inout] typ [:=wyrażenie]; |
 [signal] nazwy: [in | out | inout] typ;}
```

Przekazanie parametrów do procedury : `in inout`
Przekazanie parametrów z procedury : `out inout`

Podprogramy – instrukcja procedure

Przykład:

```
procedure vect_to_int (z: in bit_vector (1 to 8);
                       zero_flag: out boolean;
                       q: inout integer) is
begin
    q := 0;
    zero_flag := true;
    for i in 1 to 8 loop
        q := q * 2;
        if z(i) = '1' then
            q := q + 1;
            zero_flag := false;
        end if;
    end loop;
    return;
end vect_to_int;
```

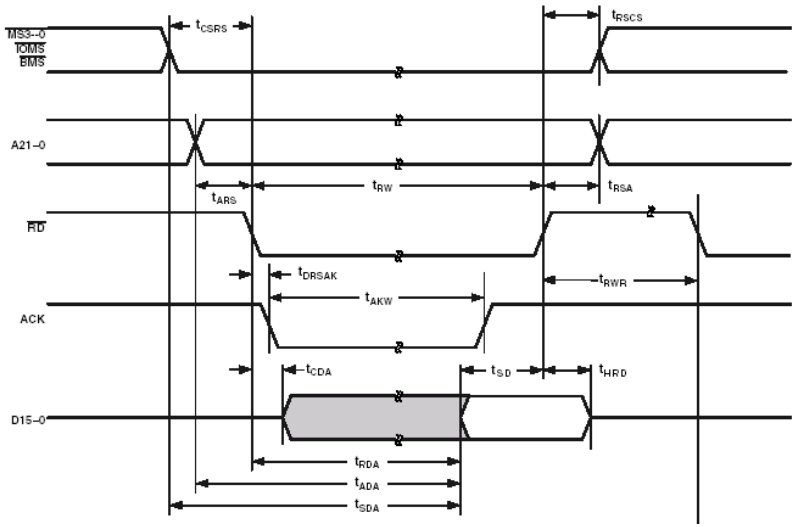
Wywołanie: `vect_to_int (s,t,u);`

Podprogramy – instrukcja procedure

```
procedure read_cycle (
 address:      in  STD_LOGIC_VECTOR(15 downto 0); -- input parameter
 signal addr: out STD_LOGIC_VECTOR(15 downto 0); -- DSP addres bus
 signal data: in  STD_LOGIC_VECTOR(15 downto 0); -- DSP data
 data_read:    out STD_LOGIC_VECTOR(15 downto 0); -- reading result
 signal rd:    out STD_LOGIC;                     -- DSP signal
 signal ms3:   out STD_LOGIC                      -- DSP signal)
is

constant T_CSRS: TIME := H_clk_per/2 - 3ns;
constant T_ARS:  TIME := H_clk_per/2 - 3ns;
constant T_RW:   TIME := H_clk_per - 2ns + (wait_states*H_clk_per);
constant T_RSA:  TIME := H_clk_per/2 - 2ns;

begin
   ms3 <= '0';
   addr <= address;
   wait for T_ARS;
   rd <= '0';
   wait for T_RW;
   rd <= '1';
   data_read := data;
   wait for T_RSA;
   addr <= (others => 'Z');
   ms3 <= '1';
end procedure read_cycle;
```



Instrukcje współbieżne

- przypisania wartości sygnałom
  - bezwarunkowe (*unconditional*)
  - warunkowe (*conditonal*)
  - decyzyjne (*selected*)
- **podprogramy**
- `block`

Współbieżne wywołanie podprogramu

W obydwu poniższych przykładach skutek jest ten sam:

```
architecture concurrent of SUB_CALL is
begin
   vect_to_int (bitstuff, flag, number);
end concurrent ;
```

```
architecture sequential of SUB_CALL is
begin
   process (bitstuff, number)
   begin
      vect_to_int (bitstuff, flag, number);
   end process;
end sequential ;
```

Instrukcja `block`

Grupuje instrukcje współbieżne.
Bloki można zagnieżdżać, tworząc hierarchię.

Składnia:

[*etykieta* :] `block` [(*wyrażenie_typu_boolean*)] [`is`] ;
[*deklaracje*]
`begin`
    instrukcje współbieżne ;
`end block` [*etykieta*] ;

deklaracje:

- stałych, typów, sygnałów
- podprogramów
- klauzula `use`
- komponentów

Instrukcja `block`

Wyrażenie_typu_boolean automatycznie generuje sygnał `guard`, który może warunkować przypisania wartości sygnałom przez klauzulę `guarded`.

Przykład:

```
B1: block (control)
begin
    s <= guarded '1';
end block B1;
```

Przypisanie `s <= '1'` zajdzie, jeżeli `control` jest równe `true`.

Uwaga!!! Nie wszystkie narzędzia dopuszczają korzystanie z tej opcji.

Ciąg dalszy nastąpi…