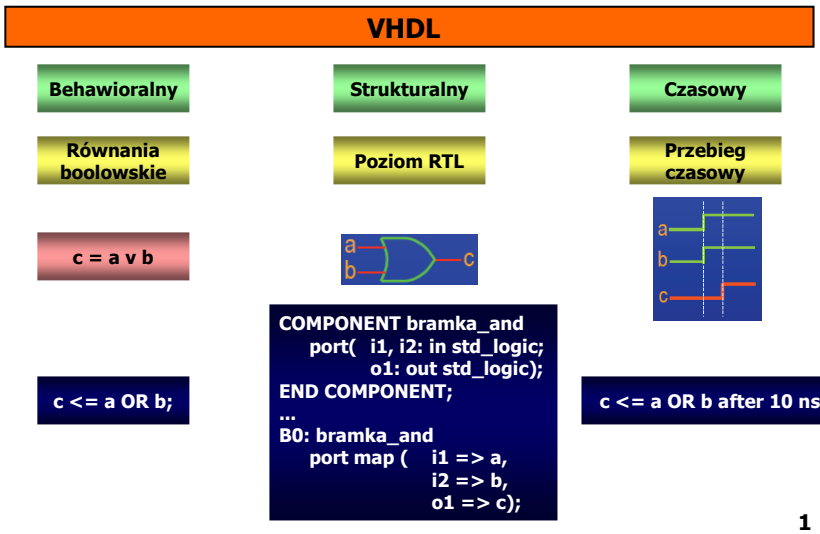
Style opisu VHDL

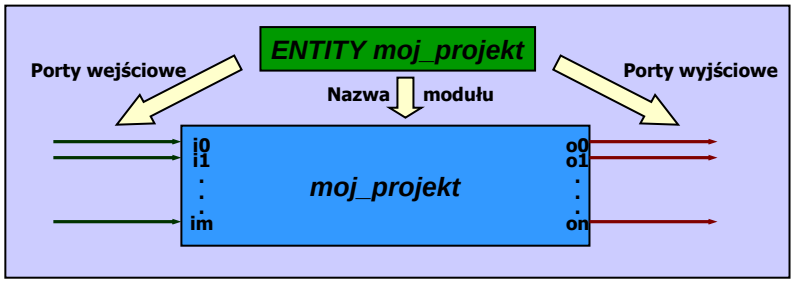


1

Entity – jednostka projektowa

Każdy moduł projektu w VHDL musi zawierać jednostkę projektową *entity*.

Jednostka projektowa (entity) w VHDL oznacza konstrukcję służącą do specyfikacji interfejsu dokomunikacji ze światem zewnętrznym.



2

Entity – jednostka projektowa

Każda jednostka projektowa (*entity*) w projekcie musi posiadać unikalną nazwę

Wejścia i wyjścia w projekcie noszą nazwę portów (PORTS)

Typy portów:

`in` – port wejściowy

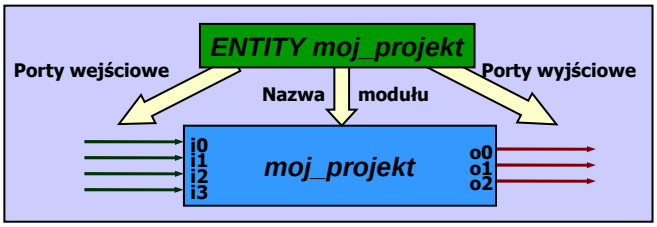
`out` – port wyjściowy

`inout` – port dwukierunkowy, wykorzystywany w sytuacji, gdy dane przekazywane są przez moduł bez ich zmiany

Entity – jednostka projektowa

Deklaracja `entity`

```
entity moj_projekt is
  port(
       i0, i1, i2, i3: in bit;
       o0, o1, o2: out bit
       );
end moj_projekt;
```



Generic – parametry ogólne

Wykorzystywana do parametryzacji projektu – jednostki projektowej

Składa się z ogólnej listy interfejsu (*generic interface list*), w której zdefiniowane są parametry jednostki projektowej (*formal generic constants*)

Elementy składowe ogólnej listy interfejsu mogą być jedynie stałymi rodzaju `in` (rodzaj jest pomijany w części deklaracyjnej)

Użycie *generic*

Składnia `generic`

```
entity projekt_testowy is
  generic(N: integer);
  port(
       a_in, b_in: in bit_vector(N-1 downto 0);
       stat: in bit;
       y_out: out bit_vector (N-1 downto 0);
       );
end projekt_testowy;
```

Brak określenia portu

Deklaracja `generic` jest widoczna dla wszystkich architektur opartych na danej jednostce projektowej (*entity*)

Architecture – implementacja jednostki projektowej

Ciało architektury (`architecture`) definiuje sposób działania jednostki projektowej od pobrania danych z portów wejściowych do wygenerowania danych na portach wyjściowych.

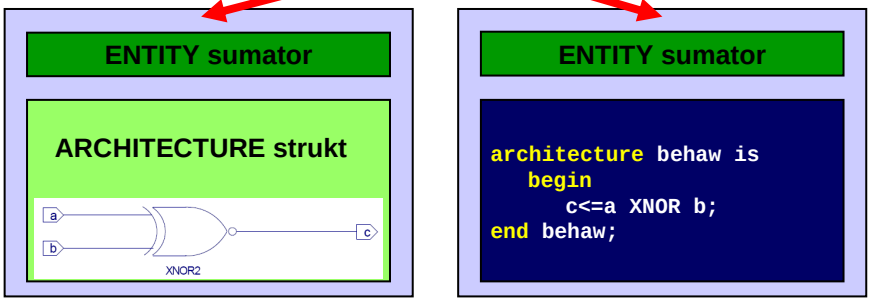
Istnieje możliwość przyporządkowania wielu architektur do jednej jednostki projektowej:

- cecha użyteczna w podejściu top down – dedykowana architektura opisuje kolejne stopnie redefinicji na różnych poziomach abstrakcji
- Potrzeba opisu różnych wariantów tego samego modułu

7

Wielość architektur

```
entity sumator is
  port(
       a, b: in bit;
       c: out bit
       );
end sumator;
```

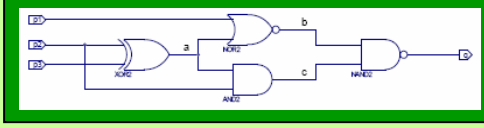


8

Deklaracja architecture

ENTITY sumator

ARCHITECTURE struk



Struktura układu

```
architecture struk of test_proj is
  signal a, b, c: bit;

  begin
      a <= p2 XOR p3;
      b <= p1 NOR a;
      c <= p2 AND a;
      q <= b NAND c;
  end struk;
```

Kod VHDL

9

Składnia architecture

Słowa kluczowe

Nazwa architektury

Nazwa jednostki projektowej

Deklaracja sygnałów

Część deklaracyjna

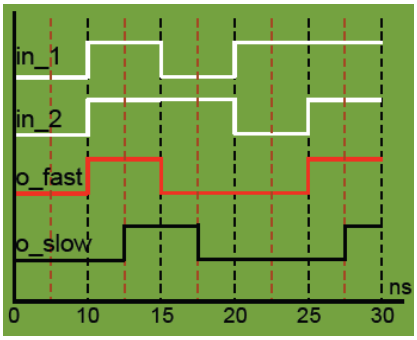
Ciało architektury

```
architecture struk of test_proj is
  signal a, b, c: bit;

  begin
      a <= p2 XOR p3;
      b <= p1 NOR a;
      c <= p2 AND a;
      q <= b NAND c;
  end struk;
```

10

Przykład prostego projektu

```
entity and2_gate is
  port(in_1, in_2: in bit;
       o_slow:     out bit;
       o_fast:     out bit);
end and2_gate;

architecture gates of and2_gate is
  begin
    o_slow <= in_1 and in_2 after 5;
    o_fast <= in_1 and in_2;
end gates;
```



11

Elementy leksykalne VHDL

- komentarze – rozpoczynają się od „--” i obowiązują do końca linii,
- słowa kluczowe – posiadają specjalne przeznaczenie, nie mogą być używane jako identyfikatory,
- identyfikatory
- symbole specjalne
- liczby
- znaki
- łańcuchy znaków oraz bitów

Wielkość liter w VHDL w wersji 1987 nie ma znaczenia

12

Identyfikatory

Przeznaczenie:

- **nazwy obiektów w modelu VHDL**

Wymagania:

- **powinny określać przeznaczenie lub funkcje obiektu**
- **dowolna długość**
- **rozpoczynają się literą alfabetu**
- **następujące po niej symbole należą do zbioru: 'A' – 'Z', 'a' – 'z', '0' – '9', '_'**
- **nie mogą kończyć się znakiem podkreślenia**
- **nie mogą zawierać kolejno dwóch znaków podkreślenia**

Symbole specjalne

Przeznaczenie:

oznaczenie operacji

& ( ) + - / < = >

ograniczenie fragmentów konstrukcji językowej

" # ` [ ]

interpunkcja składniowa

, . : ;

Symbole złożone z dwóch znaków (nie mogą zawierać spacji):

=> ** := /= >= <= <>

Typy danych

Ścisła kontrola typów danych

Typy wbudowane:

- **wyliczeniowe**
- **całkowite**
- **zmiennoprzecinkowe**
- **fizyczne**
- **tablicowe**

Typy danych

Typy wyliczeniowe:

- **definiowane przez użytkownika, określające wyższy poziom abstrakcji**

PRZYKŁAD:

Przy modelowaniu jednostki sterującej procesora zamiast kodów binarnych określających jego stan wygodniej (czytelniej) jest stosować nazwy opisowe:

```
type stan_proc is (idle, add, store, compare, move, wait);

                  ......................

variable  operacja:stan_proc := wait;
```

Typy danych

Typ boolowski:

- **wyliczeniowy typ predefiniowany**
- **wykorzystywany do reprezentacji wartości zmiennych warunkowych sterujących wykonaniem modeli behawioralnych**

DEFINICJA:

```
type boolean is (false, true);
```

Zastosowanie:

- **relacje równości i nierówności (porównanie operandów tego samego typu daje w wyniku wartość boolowską)**
- **relacje mniejszości i większości – ostre i nieostre (stosowane do typów porządkowych – całkowitych i znakowych)**

Operatory logiczne

Operatory logiczne na argumentach boolowskich:

- **AND, OR, NAND, NOR, XOR, XNOR, NOT**

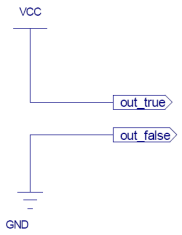
Wynik: boolowski

A	B	A and B	A nand B	A or B	A nor B	A xor B	A xnor B
false	false	false	true	false	true	false	true
false	true	false	true	true	false	true	false
true	false	false	true	true	false	true	false
true	true	true	false	true	false	false	true

Synteza danych logicznych

Typ boolowski jest typem syntezowalnym

- **„true" jest równoznaczne z logiczną „1", reprezentowaną przez wysoki poziom napięcia (V_{cc})**
- **„false" jest równoznaczne z logicznym „0", reprezentowanym przez niski poziom napięcia (GND)**




```
entity bool_synt is
      port( out_true: out bit;
            out_false: out bit);
end bool_synt;

architecture test of bool_synt is
begin
      out_true <= true;
      out_false <= false;
end test;
```

19

Typ bitowy

- **Typ predefiniowany VHDL**
- **Najczęściej używany typ danych**
- **Typ syntezowalny**

DEFINICJA:

```
type bit is ('0', '1');
```

Konsekwencje:

- **Symbole „0" i „1" należą do typu *bit* oraz *character***
- **Operacje logiczne analogiczne do typu boolean**

```
'0' or '1' = '1';   -- poprawnie

'0' xor false       -- błąd
```

20

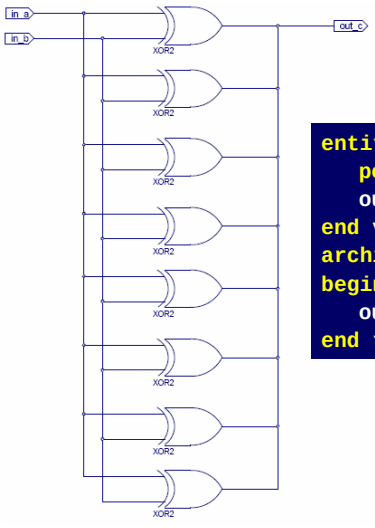
Typ bitowy a boolowski

- Wartości boolowskie stosowane są do modelowania abstrakcyjnych uwarunkowań
- Typ bitowy reprezentuje sprzętowy poziom logiczny
- Operacje logiczne na typie bitowym wykorzystują logikę dodatnią

Typ bit_vector

- Typ predefiniowany
- W VHDL zdefiniowany jako tablica bitów
- Wykorzystywany do reprezentacji magistral – uproszczona notacja dla funkcji logicznych na magistralach (np. operacja XOR wykonywana na zawartości 8-bitowych rejestrów zamiast ośmiu instrukcji bitowych)

Przykład



Układ zsyntezowany

Kod VHDL

```
entity ve_xor is
   port(in_a, in_b: in bit_vector(7 downto 0);
   out_c: out bit_vector(7 downto 0));
end ve_xor;
architecture test of ve_xor is
begin
   out_c <= in_a XOR in_b;
end test;
```

Typ znakowy

- Typ wyliczeniowy predefiniowany VHDL
- Zdefiniowany w paczce (package) STANDARD

DEFINICJA:

```
type character is (
    NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
    BS, HT, LF, VT, FF, CR, SO, SI,
    DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
    CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,
    ' ', '!', '"', '#', '$', '%', '&', ''',
    '(', ')', '*', '+', ',', '-', '.', '/',
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', ':', ';', '<', '=', '.', '?',
    '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
    'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
    'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
    'X', 'Y', 'Z', '[', '\', ']', '^', '_',
    '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
    'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
    'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
    'x', 'y', 'z', '{', '|', '}', '~', DEL);
```

23

Typ znakowy

- Zapisujemy w apostrofach
- Każdy znak reprezentuje liczbę 8 bitową (format ASCII), syntezowalną do magistrali 8-bitowej

PRZYKŁAD:

```
variable znak: character;
...

znak:='T';
```

Typ całkowity (integer)

- typ syntezowalny z ograniczeniami
- automatyczna synteza -> zbyt szerokie magistrale
- alternatywa: tablice bitów (*bit_vector*)

24

Typ zmiennoprzecinkowy (real)

- !!!nie syntezowalny!!!
- wykorzystywany w symulacjach

Podsumowanie

- typ całkowity: INTEGER
- podtypy całkowite: NATURAL, POSITIVE
- typ zmiennoprzecinkowy: REAL
- typ fizyczny: TIME
- typy tablicowe: STRING, BIT_VECTOR
- typy wyliczeniowe: BIT, BOOLEAN, SEVERITY_LEVEL, CHARACTER

Typy fizyczne:

- Reprezentacja wielkości występujących w świecie fizycznym (np. masa, długość, czas, prąd itp.)
- Definicja typu fizycznego zawiera podstawową jednostkę miary i może zawierać wtórne jednostki miary (powstałe przez całkowite pomnożenie jednostki podstawowej)

Typy TIME i SEVERITY_LEVEL

```
type time is range 0 to
1E3
units
   fs;
   ps = 1000 fs;
   ns = 1000 ps;
   us = 1000 ns;
   ms = 1000 us;
   sec = 100 ms;
   min = 60 sec;
   hr = 60 min;
end units;
```

```
type severity_level is
(note, warning, error,
failure);
```

Jednostka podstawowa

Przykład zastosowania:

```
entity tik is
  port(clk: out bit);
end tik;

architecture tak of tik is
begin
process
begin
clk <= '0';
wait for 5ns; -- instrukcja nie syntezowalna
clk <= '1';
wait for 5ns;
end process;
end tak;
```

Biblioteki:

- Biblioteka IEEE: umożliwia wykorzystanie pakietów definiujących wszystkie niezbędne typy danych w VHDL
- Biblioteki należy zadeklarować przed jednostką projektową

```
Library IEEE;
Use std_logic_1164.all;
Library test;
Use test.components.all;

Entity projekt is
....
....
....
```


Biblioteka IEEE:

- Pakiety standardowe definiują najczęściej stosowane typy danych i operatory
- Wielowartościowy system logiki – Std_Logic_1164
  - std_ulogic
  - std_ulogic_vector
  - std_logic
  - std_logic_vector
- Należy zawsze dołączać bibliotekę standardową IEEE

```
Library IEEE;
Use std_logic_1164.all;
Library test;
Use test.components.all;

Entity projekt is
....
....
```

Wady typu bitowego:

- Zawiera jedynie dwa elementy: '0' oraz '1'
- Nie jest wystarczający do modelowania rzeczywistego sprzętu
  - Brak możliwości oznaczenia:
    - Wartości niezainicjalizowanych
    - Wartości niezdefiniowanych (magistrala sterowana równocześnie co najmniej dwoma sygnałami)
    - Wyjść trójstanowych

Typy std_logic i std_ulogic:

- Zaprojektowane w celu modelowania sygnałów elektrycznych
- Reprezentacja sygnałów wymuszanych przez drivery aktywne (forcing strength), drivery rezystancyjne (pull-up i pull-down weak strngth), wyjścia trójstanowe

```
type std_ulogic is ( 'U',    -niezainicjalizowany
                     'X',    -wymuszony stan nieokreślony
                     '0',    -wymuszone 0
                     '1',    -wymuszona 1
                     'Z',    -wysoka impedancja
                     'W',    -słaby stan nieokreślony
                     'L',    -słabe zero
                     'H',    -słaba jedynka
                     '-');   -don't care
```

Przykład – bufor trójstanowy:

- *dane_wy* przyjmuje wartość *dane_we* dla *enable*='1'
- Dla 8 pozostałych wartości sygnału *enable* ('U', 'X', 'Z', 'W', 'L', 'H', '-') sygnał *dane_wy* zachowuje swoją wartość (eliminuje to wpływ błędnego sygnału *enable* na wartość sygnału *dane_wy*)

```
entity tri_state is
  port(dane_we, enable: in std_logic;
       dane_wy        : out std_logic);
end tri_state;

architecture test of tri_state is
begin
  dane_wy<=dane_we when enable='1' else 'Z';
end test;
```

33

std_logic vs. std_ulogic

- Oba typy zawierają 8 wartości
- std_logic – typ rozstrzygalny („rozwiązywalny")
  - możliwość wykorzystania więcej niż jednego drivera
  - wartość wyjścia oparta na funkcji rozstrzygającej zgodnie z tabelą rozstrzygającą

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

34

Zasady rozstrzygania sygnałów

- W systemie cyfrowym, dwa sygnały wysterowujące tę samą linię magistrali podlegają prawom teorii obwodów
  - sygnał wysterowany do pośredniej wartości zależy od pojemności driverów pozostających w konflikcie, przy czym sygnał ten może nie reprezentować żadnego ważnego stanu logicznego
  - możliwe jest połączenie wyjść, pod warunkiem, że tylko jeden sygnał sterujący jest aktywny, pozostałe znajdują się w stanie wysokiej impedancji

Zasady użycia typów std_logic i std_ulogic

- Sygnał typu std_ulogic nie może być wysterowany więcej niż jednym sygnałem (w przeciwnym wypadku generowany jest błąd)
- Rozwiązanie powyższe zalecane w projektach bez buforów trójstanowych
- Sygnał wyjściowy std_logic umożliwia połączenie wielu sygnałów sterujących w systemach z buforami trójstanowymi
- Tylko jeden sygnał może być w danej chwili aktywny

Przykład:

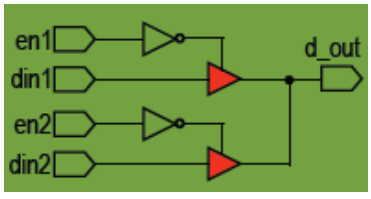
```
library IEEE;
use IEEE.std_logic_1164.all;

entity bad_tri_state is
  port(data_in1, data_in2, en1, en2: in std_ulogic;
       data_out: out std_ulogic);
end bad_tri_state;

architecture example of bad_tri_state is
  begin
    data_out <= data_in1 when en1 = '1' else 'Z';
    data_out <= data_in2 when en1 = '1' else 'Z';
end example;
```

```
** Error:
Signal (/BAD_TRI_STATE/DATA_OUT) is not a
resolved signal and has more than 1 source:
Driver from process /BAD_TRI_STATE/_P1
Driver from process /BAD_TRI_STATE/_P2
```

37

Przykład:

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

entity res_tri_state is
  port(din1, din2, en1, en2: in std_logic;
       d_out: out std_logic);
end res_tri_state;

architecture example of res_tri_state is
  begin
    -- en1 and en2 never simultaneously set
    d_out <= din1 when en1 = '1' else 'Z';
    d_out <= din2 when en1 = '1' else 'Z';
end example;
```



38

Typ std_ulogic_vector, std_logic_vector

- Nieograniczona tablica o wartościach standard logic
- Wykorzystanie podobne do bit_vector
  - Bardziej precyzyjna reprezentacja wartości cyfrowych sygnałów elektrycznych

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;

subtype std_ulogic_word is std_ulogic_vector (0 to 31);
```

```
function resolved( s: std_ulogic_vector) return std_ulogic;
subtype std_logic is resolved std_ulogic;
type std_logic_vector is array (natural range <>) of std_logic;
```

39