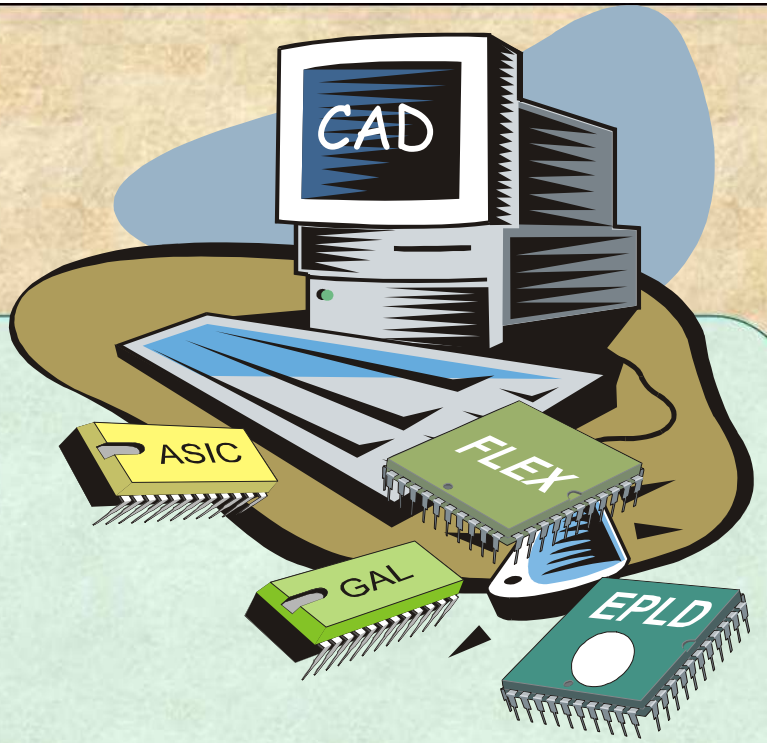
Język VHDL – podstawy



Mariusz Rawski

rawski@tele.pw.edu.pl

http://rawski.zpt.tele.pw.edu.pl/

Języki opisu sprzętu

- **V**ery high speed integrated **H**ardware **D**escription **L**anguage
- Przemysłowy standard języka HDL (IEEE Std 1076-1987, IEEE Std 1076-1993)
- VHDL jest językiem przenośnym może być stosowany przez systemy różnych firm
- Zawiera użyteczne konstrukcje semantyczne umożliwiające zwięzłą specyfikację złożonych układów cyfrowych
- Projekt może być opisany hierarchicznie (na wielu poziomach)
- Możliwe jest korzystanie z biblioteki gotowych elementów i tworzenie podukładów (tzw. komponentów)

- Powszechnie stosowany w specyfikacjach układów cyfrowego przetwarzania sygnałów:
  - U. Meyer Baese, **DSP with FPGAs**, Springer, Berlin 2001,
  - K. K. Parchi, T. Nishitani, **Digital Signal Processing for Multimedia Systems**, Marcel Dekker, Inc. New York 1999

Język VHDL

Jednostka projektowa

Deklaracja jednostki

```
entity halfadder is
    port (
        a, b    : in bit;
        s, carry : out bit
    );
end halfadder;
```

Opis architektury

```
architecture dataflow of
    halfadder is
begin
    s <= a xor b;
    carry <= a and b;
end dataflow;
```

Informacje o wejściach i wyjściach układu oraz o parametrach

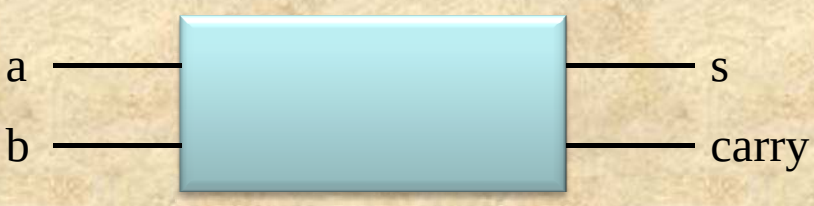
Funkcje i działanie układu

Taka sama nazwa jak nazwa pliku bez rozszerzenia nazwy

```
entity test is
  port (
        input_pin_name   : in bit
        output_pin_name : out bit
  );
end test
architecture test_body of test is
begin
  output_pin_name <= input_pin_name;
end test_body;
```

Te same nazwy

Deklaracja jednostki

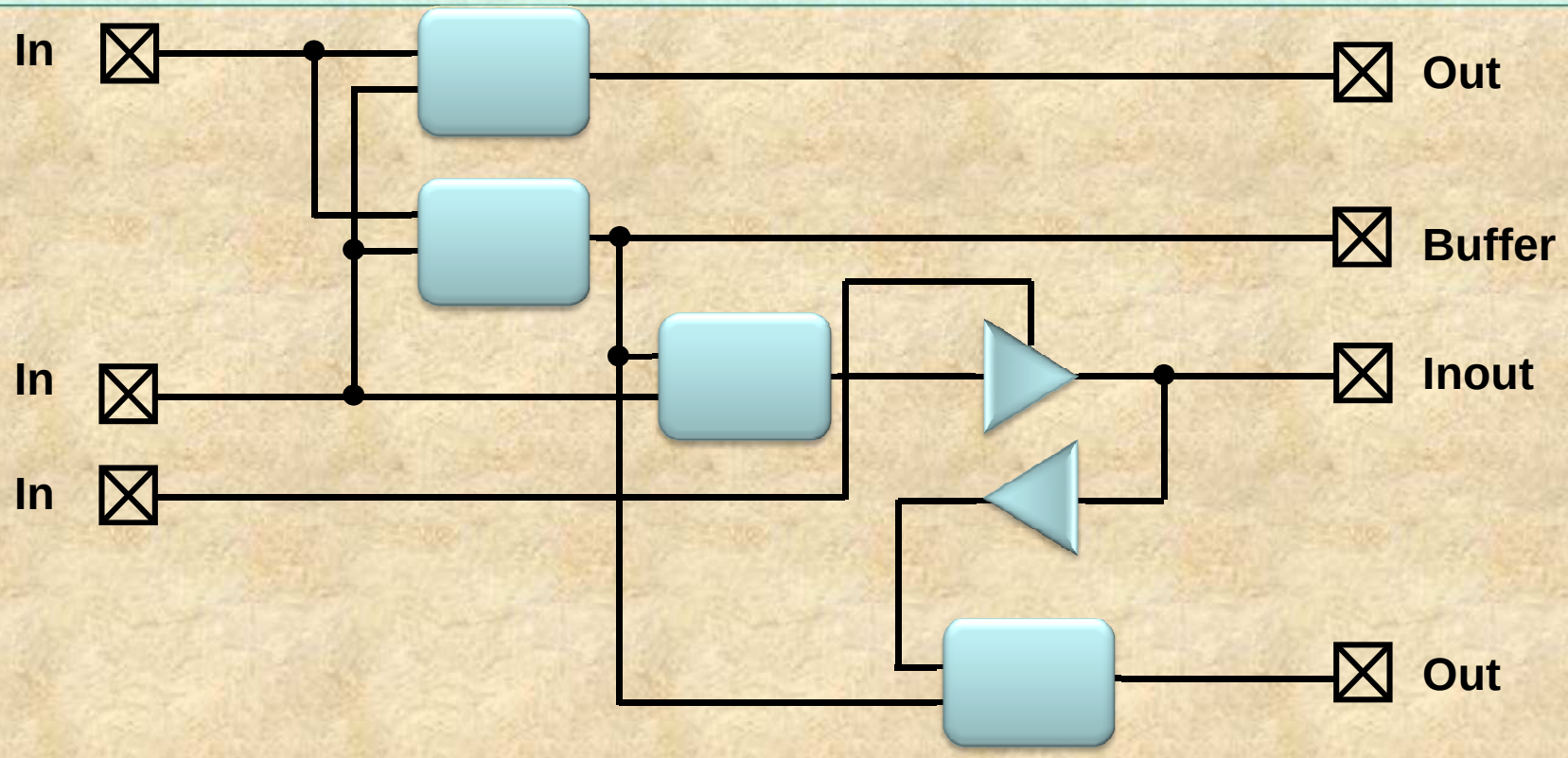


- Deklaracja wyprowadzeń
  - nazwa
  - tryb
  - typ danych

```
entity halfadder is
    port (
      a, b      : in bit;
      s, carry : out bit
    );
end halfadder;
```

- Typ portu
  - in - wejściowy
  - out - wyjściowy
  - inout - dwukierunkowy
  - buffer - wyjściowy z możliwością odczytu
- Typy danych:
  - bit,
  - bit_vector,
  - integer,
  - boolean

Tryby portu

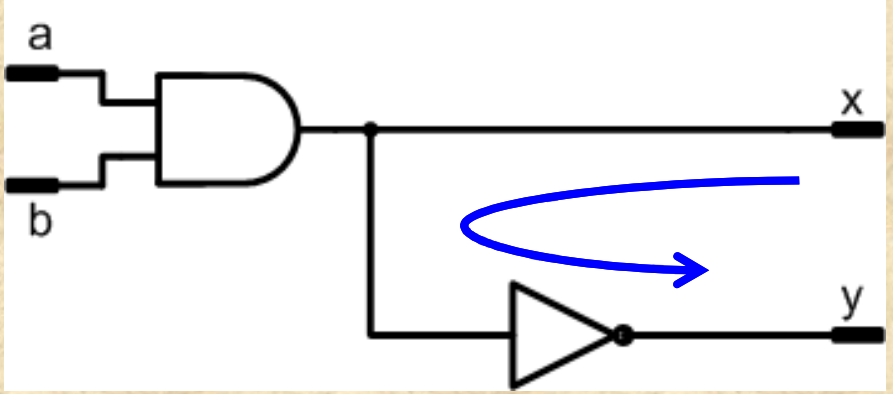


- **in** – jednokierunkowy port wejściowy
- **out** – jednokierunkowy port wyjściowy
- **buffer** – port wyjściowy, który pozwala na odczytywanie pojawiających sie na nim wartości wewnątrz projektowanego układu umożliwiając stosowanie wewnętrznego sprzężenia zwrotnego (UWAGA: może sprawiać problemy z kompatybilnością)
- **inout** – port dwukierunkowy umożliwiający stosowanie wewnętrznego sprzężenia zwrotnego

Tryby portu

```
library ieee;
use ieee.std_logic_1164.all;

entity port_mode is
      port (
         a, b    : in  std_logic;
         x, y    : out std_logic
      );
end port_mode;
architecture data_flow of port_mode is
begin
      x <= a and b;
      y <= not x;
end data_flow;
```



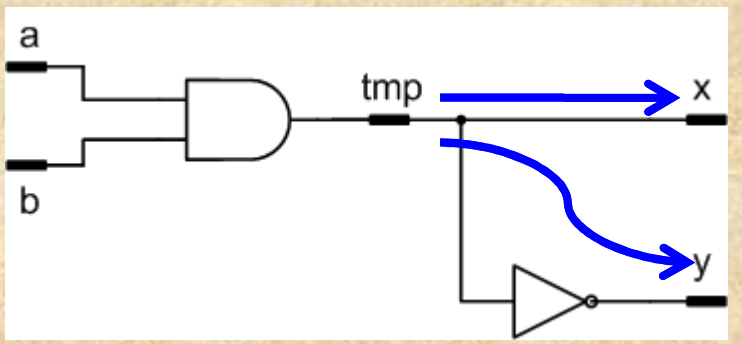
Error (10309): VHDL Interface Declaration error in port_mode.vhd(13): interface object "x" of mode out cannot be read. Change object mode to buffer or inout.

- Sygnał ‘x’ jest użyty do obliczenia sygnału ‘y’, stąd też w VHDL’u interpretowany jest on jako sygnał zewnętrzny wchodzący do układu.
- To narusza tryb **out** portu ‘x’.
- Rozwiązaniem jest zmiana trybu na **inout**, co nie jest najlepszym wyjściem, gdyż ‘x’ nie jest portem dwukierunkowym.

Tryby portu

```
library ieee;
use ieee.std_logic_1164.all;

entity port_mode is
      port (
         a, b    : in  std_logic;
         x, y    : out std_logic
      );
end port_mode;
architecture data_flow of port_mode is
      signal tmp : std_logic;
begin
      tmp <= a and b;
      x <= tmp
      y <= not tmp;
end data_flow;
```



- Lepszym rozwiązaniem jest użycie wewnętrznego sygnału reprezentującego pośredni wynik.

Blok architektury

```
architecture dataflow of halfadder is
begin
    s <= a xor b;
    carry <= a and b;
end dataflow;
```

- Architektura opisuje sposób działania jednostki (entity) lub jej budowę (strukturę).
- Style specyfikacji
  - **funkcjonalny** (behavioralny) – specyfikowane jest działanie układu bez podania szczegółów realizacyjnych
  - **przepływ danych** (dataflow) – opis w postaci tzw. instrukcji przypisania reprezentowanych wyrażeniami logicznymi
  - **strukturalny** – specyfikowana jest budowa układu. W opisie podane są komponenty oraz połączenia oraz połączenia między nimi.

Blok architektury

architecture nazwa_arch **of** nazwa_jednostki **is**

Definicje i deklaracje:
(typów, podtypów, stałych, sygnałów, komponentów, konfiguracji)

begin

– **instrukcje przypisania**
– **procesy**
– **komponenty**

end nazwa_arch;

Moduły systemowe wspomagające specyfikację…

dostępne dla wielu jednostek projektowych to **biblioteki** i **pakiety**.

- Biblioteka to zbiór zanalizowanych pakietów i jednostek projektowych przechowywanych w danym systemie operacyjnym i dostępnych bezpośrednio wg nazwy logicznej (nie jest to nazwa katalogu w tym systemie).
- Pakiet jest modułem do deklarowania i definiowania obiektów, typów, komponentów, funkcji i procedur przeznaczonych do stosowania w wielu jednostkach projektowych.
- Dostęp do pakietu osiąga się deklaracją:
  - library library_name;
  - use library_name.package_name.all;
- Biblioteka systemowa:
  - library ieee;
  - use ieee.std_logic_1164.all;

Biblioteki

- Biblioteki systemowa IEEE
  - IEEE to najważniejsza biblioteka zawierająca pakiety ze standardowymi typami danych. Odwołanie do tej biblioteki a także polecenie use umożliwiające dostęp do standardowego pakietu: **std_logic_1164** musi poprzedzać każdą jednostkę projektową.
  - Dostęp do pakietów definiowanych przez użytkownika osiąga się poleceniem:

```
library ieee;
use ieee.std_logic_1164.all;
```

- Biblioteka robocza – work
  - Bieżące jednostki są kompilowane do biblioteki roboczej o domyślnej nazwie logicznej WORK.
  - Dostęp do pakietów definiowanych przez użytkownika osiąga się poleceniem:

```
library work;
use work.user_package_name.all;
```

Elementy strukturalne języka VHDL

- Słowa kluczowe
- Identyfikatory
- Obiekty danych
- Operatory
- Atrybuty
- Instrukcje

Komentarze

```
-------------------------------------
-- Plik          : port_mode.vhd
-- Versja        : 1.0
-- Data          : 11.11.2007
-- Opis          : Przyklad ilustrujący tryby
--                 portow
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity port_mode is
      port(
         a, b    : in  std_logic;        -- wejscia ukladu
         x, y    : out std_logic         -- wyjscia ukladu
      );
end port_mode;
-- Architektura ukladu
architecture data_flow of port_mode is
      signal tmp : std_logic; -- sygnal wewnetrzny
begin
      -- wysterowanie sygnalu wewnetrznego
      tmp <= a and b;
      -- wykorzystanie sygnalu wewnetrznego
      x <= tmp;
      y <= not tmp;
end data_flow;
```

- Komentarz rozpoczynają dwa znaki '--' z następującym po nich tekstem komentarza
- Wszystko po symbolach '--' jest ignorowane
- Komentarze stosuje się dla celów dokumentacji i nie ma on wpływu na kod
- Dobrym zwyczajem jest dołączanie na początku pliku „nagłówka" komentarza z informacją dotyczącą projektu

Słowa kluczowe

abs	downto	library	postponed	Srl
access	else	linkage	procedure	Subtype
after	elsif	literal	process	Then
alias	end	loop	pure	To
all	entity	map	range	Transport
and	exit	mod	record	Type
architecture	file	nand	register	Unaffected
array	for	new	reject	Units
assert	function	Next	rem	Until
attribute	generate	Nor	report	Use
begin	generic	Not	return	Variable
block	group	Null	rol	Wait
body	guarded	Of	ror	When
buffer	if	On	select	While
bus	impure	Open	severity	With
case	in	Or	signal	Xnor
component	inertial	Others	shared	Xor
configuration	inout	Out	sla	
constant	is	Package	sll	
disconnect	label	Port	sra	

Key Words

- begin, end..
- if, then, case...
- and, or, nand, xor..
- signal, component, in, out

Identyfikatory

- Identyfikator jest nazwą obiektu w kodzie VHDL
- Podstawowe zasady tworzenia identyfikatorów:
  - Identyfikatory mogą się składać jedynie z liter, cyfr i znaków podkreślenia '_'
  - Nazwa musi się rozpoczynać od litery
  - Nazwa nie może się kończyć znakiem '_'
  - Niedozwolone są dwa znaki podkreślenia '__' w nazwie
  - Wielkość liter nie ma znaczenia
- Nazwa powinna być znacząca

```
Ta_Sama_Nazwa
TA_SAMA_NAZWA
ta_sama_nazwa
tA_SaMa_NaZwa
```

```
16_bit_bus   : integer; -- błędnie
_bus_16_bit_ : integer; -- błędnie
bus_16_bit   : integer; -- O.K.
bus__16_bit  : integer; -- błędnie
bus$16$bit   : integer; -- błędnie
Bus16Bit     : integer; -- O.K.
```

Liczby, znaki i napisy

- **Liczby**
  - W języku VHDL można się posługiwać liczbami:
    - całkowitymi – 0, 128, 98E3
    - rzeczywistymi – 0.0, 1.238, 9.8E3
  - Liczby mogą być przedstawione w różnych systemach:
    - dziesiętnie – 45,
    - binarnie – 2#101101#
    - heksadecymalnie – 16#2D#
  - Można wykorzystać znaki podkreślenia dla zwiększenia czytelności:
    - 2#0110_0101_0011# ≡ 2#011001010011#
- **Znaki**
  - W języku VHDL znak ujmuje się w apostrofy: ‘a’, ‘3’
    - 1 to liczba, zaś ‘1’ to znak
- **Napis**
  - W języku VHDL napis ujmuje się w znaki cudzysłowów : ”napis”
    - ”100101” to napis, zaś 100101 to liczba

Formatowanie kodu

```
library ieee;
use ieee.std_logic_1164.all;

entity port_mode is
      port(
         a, b    : in  std_logic;
         x, y    : out std_logic
      );
end port_mode;

architecture data_flow of port_mode is
      signal tmp : std_logic;
begin
      tmp <= a and b;
      x <= tmp;
      y <= not tmp;
end data_flow;
```

```
library ieee; use ieee.std_logic_1164.all;
entity port_mode is port( a, b : in
std_logic; x, y    : out std_logic ); end
port_mode; architecture data_flow of
port_mode is signal tmp : std_logic; begin
tmp <= a and b; x <= tmp;
y <= not tmp; end data_flow;
```

- VHDL jest językiem niewrażliwym na wielkość liter a „białe znaki” mogą być swobodnie umieszczane pomiędzy elementami leksykalnymi
- Pomimo, że formatowanie kodu nie wpływa na jego zawartość i efektywność projektu ma ogromne znacznie dla czytelności
- Obydwa kody opisują dokładnie ten sam projekt
- Odpowiednio sformatowany i udokumentowany kod pozwala na łatwiejsze zrozumienie opisywanego projektu i lokalizowanie ewentualnych błędów.

Obiekty

- Obiekty w języku VHDL to elementy posiadające nazwę i przechowujące wartość określonego typu. Istnieje cztery rodzaje obiektów:
  - sygnały (signal),
  - zmienne (variable),
  - stałe (constant),
  - pliki (file),
  - aliasy (alias).

- Zasięg obiektów:
  - obiekty zadeklarowane w pakietach są widoczne w wszystkich kodach VHDL wykorzystujących te pakiety,
  - obiekty zadeklarowane w interfejsie (entity) jednostki projektowej są dostępne dla wszystkich architektur z nią związanych,
  - obiekty zadeklarowane w bloku architektury (architecture) są dostępne dla wszystkich konstrukcji wewnątrz tej architektury,
  - obiekty zadeklarowane w procesie dostępne są jedynie wewnątrz procesu,

Sygnały

- Najczęściej spotykane obiekty w kodzie VHDL.
- Sygnały są wykorzystywane do komunikacji między komponentami VHDL
- Rzeczywiste, fizyczne połączenia w systemie często są reprezentowane przez sygnały VHDL
- Sygnał musi zostać zadeklarowany w sekcji deklaracji w bloku architektury. Sposób deklaracji jest następujący:

  ```
  signal nazwa_sygnalu1, nazwa_sygnalu_2, … : typ_danych;
  ```

- Przykład:

  ```
  signal a, b, c : std_logic;
  ```

- Przypisanie wartości:

  ```
  nazwa_sygnalu1 <= wartość;
  ```

- Istnieje możliwości podania początkowej wartości sygnału, co wykorzystuje się w symulacji jednak nie jest syntezowane:

  ```
  signal a, b, c : std_logic := '0';
  ```

- Wszystkie przypisania wartości sygnałom odbywają się z opóźnieniem zależnym od technologii, w której system jest realizowany

Zmienne

- Reprezentuje symboliczny element pamięciowy, w którym można przechowywać i modyfikować wartości.
- Nie istnieje bezpośredniego odwzorowania zmiennych w realizowanym układzie cyfrowym.
- Zmienna może być deklarowana i wykorzystywana jedyne w procesach.
- Głównym zastosowanie zmiennych jest abstrakcyjny opis funkcjonowania systemu.
- Sposób deklaracji jest następujący:

  variable nazwa_zmiennej1, nazwa_zmiennej2, … : typ_danych;

- Przypisanie wartości:

  nazwa_zmiennej := wartość;

- Przypisanie wartości zmiennym odbywa się bez opóźnienia, stąd nazywane jest też natychmiastowym (*immediate assigment*).

Stałe

- Stała reprezentuje obiekt przechowujący wartość, która nie może być zmieniona
- Sposób deklaracji jest następujący:

  constant nazwa_stalej : typ_danych := warotść;
- Przykład:

  constant BUS_WIDTH : integer := 32;

  constant BUS_BYTES : integer := BUS_WIDTH / 8;
- Stosowanie stałych zwiększa czytelność kodu i ułatwia modyfikacje.

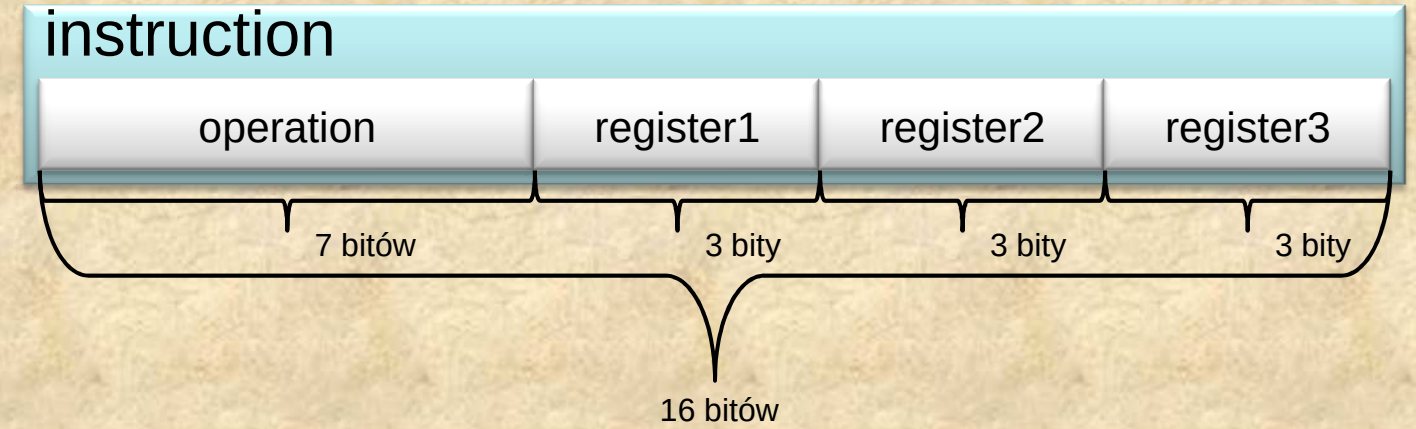
```
architecture behavior of even_detector_beh1 is
   -- deklaracja sygnalu
   signal odd : std_logic;
begin
process (x)
      variable tmp : std_logic;
   begin
      tmp := '0';
      for i in 2 downto 0 loop
         tmp := tmp xor x(i);
      end loop;
      odd <= tmp;
   end process;
end behavior ;
```

```
architecture behavior of even_detector_beh1 is
   -- deklaracja sygnalu
   signal odd : std_logic;
begin
process (x)
      variable tmp : std_logic;
      constant BUS_WIDTH : integer := 3;
   begin
      tmp := '0';
      for i in BUS_WIDTH - 1 downto 0 loop
         tmp := tmp xor x(i);
      end loop;
      odd <= tmp;
   end process;
end behavior ;
```

Aliasy

- Aliasy nie są obiektami danych w dosłownym znaczeniu tego słowa.
- Stanowią alternatywną nazwę dla istniejących obiektów.
- Jedna z form aliasu sygnałów jest wyjątkowo pomocna w syntezie. Rozważmy 16 bitowy rejestr przechowujący instrukcję procesora składającą się z kodu operacji i trzech rejestrów. Można wykorzystać aliasy do identyfikacji poszczególnych pól składowych instrukcji.

```
signal instruction  : std_logic_vector (15 downto 0);
alias  operation    : std_logic_vector (6 downto 0) is instruction (15 downto 9) ;
alias  register1    : std_logic_vector (2 downto 0) is instruction (8 downto 6) ;
alias  register2    : std_logic_vector (2 downto 0) is instruction (5 downto 3) ;
alias  register3    : std_logic_vector (2 downto 0) is instruction (2 downto 0);
```



Typy danych

- Typy skalarne
  - **integer** – liczby całkowite
  - **real** – liczby rzeczywiste
  - **bit** – typ bitowy (0, 1)
  - **boolean** – typ boolowski (prawda, fałsz)
  - **std_ulogic**, **std_logic** – typy wielowartościowe
- Typy złożone
  - **string** – typ łancuchowy
  - **bit_vector** – typ bitwektorowy
  - **std_logic_vector** – wektor typu std_logic
- Typy wektorowo - skalarne
  - **signed**
  - **unsigned**

Atrybuty

- Dostarczają dodatkowych informacji o obiektach (np. sygnałach, zmiennych, typach lub komponentach)
- Pewna liczba atrybutów jest predefiniowana

- Składnia
  `obiekt'atrybut[(parametr)];`

- Przykładowe atrybuty
  - 'EVENT – równy TRUE, gdy zachodzi zmiana wartości sygnału,
  - 'STABLE – równy TRUE, gdy nie zachodzi zmiana wartości sygnału
  - 'LEFT - zwraca lewą granicę zakresu
  - 'RIGHT - zwraca prawą granicę zakresu
  - 'RANGE - zwraca zakres typu

Operatory

Operacja	Opis	Typ argumentu a	Typ argumentu b	Typ wyniku
a ** b	potęgowanie	integer	integer	integer
abs a	wartość bezwzględna	integer		integer
not a	negacja	boolean, bit, bit_vetctor		boolean, bit, bit_vetctor
a * b	mnożenie	integer	integer	integer
a / b	dzielenie			
a **mod** b	wartość modulo			
a **rem** b	reszta z dzielenia			
+ a		integer		integer
– a	negacja			
a + b	dodawanie	integer	integer	integer
a – b	odejmowanie			
a & b	złączenie	tablica 1-D, element	tablica 1-D, element	tablica 1-D
a **sll** b	przesunięcie w lewo – logiczne	bit_vector	integer	bit_vector
a **srl** b	przesunięcie w prawo– logiczne			
a **sla** b	przesunięcie w lewo – arytmetyczne			
a **sra** b	przesunięcie w prawo– arytmetyczne			
a **rol** b	rotacja w lewo			
a **ror** b	rotacja w prawo			
a = b	równe	dowolne	jak a	boolean
a /= b	różne			
a < b	mniejsze	skalar lub tablica 1-D	jak a	boolean
a <= b	mniejsze lub równe			
a > b	większe			
a >= b	większe lub równe			
a **and** b	and	boolean, bit, bit_vetctor	jak a	jak a
a **or** b	or			
a **xor** b	xor			
a **nand** b	nand			
a **nor** b	nor			
a **xnor** b	xnor			

Operatory

- W trakcie syntezy operatory VHDL zostaną zrealizowane jako fizyczne komponenty.
- Złożoność realizacji sprzętowej tych operatorów jest bardzo różna. Wiele z nich jak np. mnożenie czy dzielenie jest bardzo trudno zrealizować.
- Kolejność wykonania operatorów:

Priorytet	Operatory
Najwyższy	**** abs not**
	*** / mod rem**
	+ - (identycznoć, negacja)
	& + - (suma i różnica)
	sll srl sla sra rol ror
	= /= < <= > >=
Najniższy	**and or nand nor xor xnor**

- W przypadku równoprawnych operacji wykonywanie odbywa się z lewej do prawej.

a + b – 1 > c or a < d

- Wykorzystując nawiasy można ustalić kolejność wykonywania operacji

(a and b) or (a and d)

Typy danych pakietu IEEE std_logic_1164

- Najbardziej użyteczne typy danych pakietu std_logic_1164 to:
  - std_logic,
  - std_logic_vector
- Typ std_logic posiada 9 możliwych wartości:
  - '0' i '1' – „silna" logiczna wartości 0 i 1,
  - 'L' i 'H' – „słaba" logiczna wartość 0 i 1,
  - 'X' i 'W' – wartość nieznana i „słaba" wartość nieznana,
  - 'U' – wartość niezainicjowana (wykorzystana w symulacji),
  - '–' – wartość nieokreślona (don't care),
  - 'Z' – stan wysokiej impedancji.
- Typ std_logic_vector jest kolekcją elementów typu std_logic. Sygnał 8 bitowy jest reprezentowany przez:

```
signal a : std_logic_vector (7 downto 0);
signal b : std_logic_vector (0 to 7);
```

- Dla 'a' najbardziej znaczący bit MSB ma indeks 7 najmniej znaczący 0, dla 'b' najbardziej znaczący bit MSB ma indeks 0 najmniej znaczący 7
- Możliwy jest dostęp do poszczególnych elementów:
  - a(7) – najbardziej znaczący bit MSB
  - a(3 downto 0) – najmłodsze 4 bity

Przeciążone operatory i funkcje konwersji

Operacja	Typ argumentu a	Typ argumentu b	Typ wyniku
not a	std_logic, std_logic_vector		jak a
a **and** b	std_logic, std_logic_vector	jak a	jak a
a **or** b			
a **xor** b			
a **nand** b			
a **nor** b			
a **xnor** b			

Operacja	Typ argumentu a	Typ wyniku
to_bit(a)	std_logic	bit
to_stdulogic(a)	bit	std_logic
to_bitvector(a)	std_logic_vector	bit_vector
to_stdlogicvector(a)	bit_vector	std_logic_vector

- Dla następujących deklaracji sygnałów:

```
signal s1, s2, s3          : std_logic_vector (7 downto 0);
signal b1, b2 : bit_vector (7 downto 0);
```

- Błędne konstrukcje spowodowane niezgodnością typów

```
s1 <= b1;
b2 <= s1 and s2;
s3 <= b1 or s2;
```

- Poprawne konstrukcje wykorzystujące funkcje konwersji

```
s1 <= to_stdlogicvector(b1);
b2 <= to_bitvector(s1 and s2);
s3 <= to_stdlogicvector(b1) or s2;  lub  s3 <= to_stdlogicvector( b1 or to_bitvector(s2));
```

Operatory relacyjne

- Dla tablicy 1-D możliwe jest stosowanie operatorów relacyjnych
- Obydwa argumenty muszą mieć ten sam typ elementów
- Porównanie następuje element po elemencie zaczynając od lewej strony
- Jeśli jedna z tablic osiągnie wcześniej koniec jest uznawana z element mniejszy

  ” 011” = ” 011” ”011” > ” 010 ” ” 011 ” > ” 00011” ” 0110 ” > ” 011”

- Dla:

```
signal s1 : std_logic_vector (7 downto 0);
signal s2 : std_logic_vector (3 downto 0);

if (s1 = s2) then
…
else
….
```

wyrażenie s1 = s2 jest zawsze fałszywe. Tego typu błąd jest trudny do zlokalizowania.

Operatory konkatenacji i agregacja

- Bardzo przydatny operator dla manipulacji na tablicach 1-D
- Umożliwia łączenie elementów, małych tablic i fragmentów tablic w większe tablice.
- Przesunięcie w prawo o dwie pozycje i dołączenie dwóch zer na początku:

  y <= ”00 ” & a(7 downto 2);

- Przesunięcie w prawo o dwie pozycje i dołączenie dwóch bitów o wartości takiej jak MSB (arithmetic shift):

  y <= a(7) & a(7) & a(7 downto 2);

- Rotacja w praco o dwie pozycje:

  y <= a(1 downto 0) & a(7 downto 2);

- Agregacja nie jest operatorem

  y <= "10100000”;

  y <= (‘1’, ‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’);

  y <= (7 => ‘1’, 6 =>‘0’, 5 =>‘1’, 4 =>‘0’, 3 =>‘0’, 2 =>‘0’, 1 =>‘0’, 0 =>‘0’);

  y <= (7|5 => ‘1’, 6|4|3|2|1|0 =>‘0’);

  y <= (7|5 => ‘1’, others =>‘0’);

- Wyzerowanie całego wektora

  y <= (others => ‘0’);

Typy danych pakietu IEEE numeric_std

- Definiuje typy **signed** i **unsigned** jako tablica elementów **std_logic**
- Dla unsigned tablica jest interpretowana w kodzie NKB
- Dla signed tablica jest interpretowana w kodzie U2
- Deklaracja obiektów nowego typu jest identyczna jak std_logic_vector:

```
signal s1 : signed(7 downto 0);
```

- Dla celów operacji na obiektach nowego typu przeciążono operatory: **abs**, *****, **/**, **mod**, **rem**, **+**, **–**
- Operatory relacyjne także zostały przeciążone.

  Przykład: ” 011” > ”1000” dla sygnałów typu:
  - std_logic_vector ⟹ false – pierwszy element ”011” jest mniejszy niż ”1000”
  - unsigned ⟹ false – 3 > 8
  - signed ⟹ true – 3 > – 8

Funkcje pakietu IEEE numeric_std

Operacja	Opis	Typ argumentu a	Typ argumentu b	Typ wyniku
shift_left(a, b)	przesunięcie w lewo	unsigned, signed	naturalny	jak a
shift_right(a, b)	przesunięcie w prawo			
rotate_left(a, b)	obrót w lewo			
rotate_right(a, b)	obrót w prawo			
resize(a, b)	zmiana rozmiaru	unsigned, signed	naturalny	jak a
std_match(a, b)	porównanie z uwzględnieniem '–'	unsigned, signed, std_logic_vector, std_logic	jak a	boolean
to_integer(a)	konwersje typów danych	unsigned, signed		integer
to_unsigned(a)		natural	natural	unsigned
to_signed(a)		integer	natural	signed

Konwersja typów

Typ argumentu a	Docelowy typ danych	Funkcja konwersji/rzutowanie
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

Kodowanie w VHDL – porady

- Używaj typów **std_logic** i **std_logic_vector** zamiast **bit** i **bit_vector.**
- Używaj pakietu **numeric_std** i typów **unsigned** i **signed** do syntezy operatorów arytmetycznych.
- Używaj jedynie malejącego zakresu tablic (downto) dla typów **unsigned**, **signed** i **std_logic_vector.**
- Używaj nawiasów do sprecyzowania zamierzonej kolejności wykonania operacji.
- Nie używaj typów definiowanych przez użytkownika, chyba, że istnieje po temu poważny powód.
- Nie używaj bezpośredniego przypisania := dla nadania wartości początkowej sygnałom.
- Używaj argumentów o takim samym rozmiarze dla operatorów relacyjnych