KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

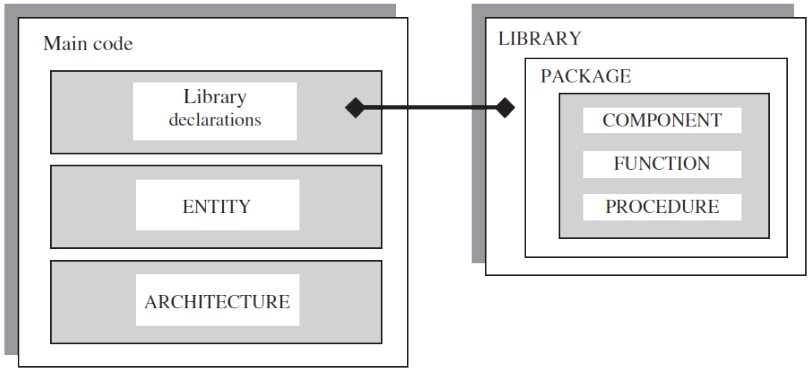
„Układy reprogramowalne i SoC”
„Język VHDL (część 5)”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Prezentacja dystrybuowana jest bezpłatnie

Pakiety i komponenty



- Często używane fragmenty kodu VHDL są zazwyczaj opisywane w formie komponentów, funkcji i procedur umieszczonych w pakietach, które składają się na bibliotekę.
- Składnia struktury pakietu wygląda następująco:

```
PACKAGE package_name IS
   (declarations)
END package_name;

[PACKAGE BODY package_name IS
   (FUNCTION and PROCEDURE descriptions)
END package_name;]
```

Prosty pakiet

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  PACKAGE my_package IS
6     TYPE state IS (st1, st2, st3, st4);
7     TYPE color IS (red, green, blue);
8     CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
9  END my_package;
10 ------------------------------------------------
```

Pakiet zawierający funkcję

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  PACKAGE my_package IS
6     TYPE state IS (st1, st2, st3, st4);
7     TYPE color IS (red, green, blue);
8     CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
9     FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
10 END my_package;
11 ------------------------------------------------
12 PACKAGE BODY my_package IS
13    FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
14    BEGIN
15       RETURN (s'EVENT AND s='1');
16    END positive_edge;
17 END my_package;
18 ------------------------------------------------
```

- Przykład użycia:

```
-----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------
ENTITY...
...
ARCHITECTURE...
...
-----------------------------------
```

Komponent

- Komponent pozwala nam na zadeklarowanie elementu i uzycie go jako części składowej w kodzie VHDL
- Aby użyć komponentu, należy go najpierw zadeklarować:

```
COMPONENT component_name IS
   PORT (
         port_name : signal_mode signal_type;
         port_name : signal_mode signal_type;
         ...);
END COMPONENT;
```

- Użycie komponentu:

```
label: component_name PORT MAP (port_list);
```

- Komponent można zadeklarować zarówno bezpośrednio w kodzie VHDL, jak i w pakiecie

Bezpośrednia deklaracja komponentu

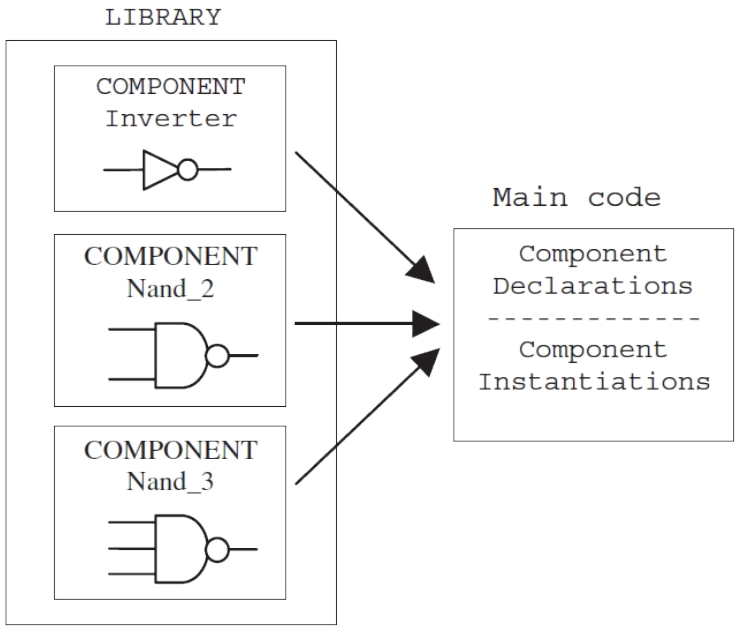
```
1  ------ File inverter.vhd: ------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------
5  ENTITY inverter IS
6     PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
7  END inverter;
8  ----------------------------------
9  ARCHITECTURE inverter OF inverter IS
10 BEGIN
11    b <= NOT a;
12 END inverter;
13 ------------------------------------------
```

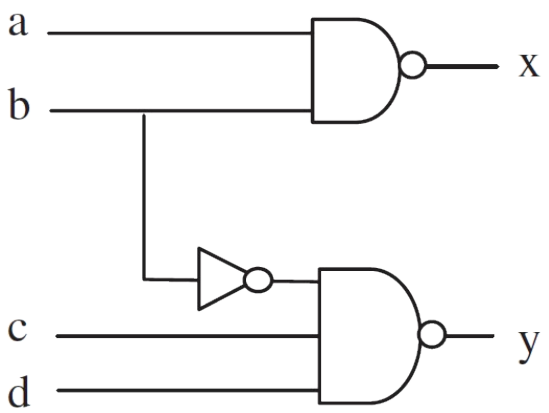
```
1  ------ File nand_2.vhd: --------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------
5  ENTITY nand_2 IS
6     PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
7  END nand_2;
8  ----------------------------------
9  ARCHITECTURE nand_2 OF nand_2 IS
10 BEGIN
11    c <= NOT (a AND b);
12 END nand_2;
13 ------------------------------------------
```

```
1  ----- File nand_3.vhd: ---------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------
5  ENTITY nand_3 IS
6     PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
7  END nand_3;
8  ----------------------------------
9  ARCHITECTURE nand_3 OF nand_3 IS
10 BEGIN
11    d <= NOT (a AND b AND c);
12 END nand_3;
13 ------------------------------------------
```

Bezpośrednia deklaracja komponentu (c.d.)

```
1  ----- File project.vhd: --------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY project IS
6     PORT (a, b, c, d: IN STD_LOGIC;
7           x, y: OUT STD_LOGIC);
8  END project;
9  ------------------------------------
10 ARCHITECTURE structural OF project IS
11    -------------
12    COMPONENT inverter IS
13       PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
14    END COMPONENT;
15    -------------
16    COMPONENT nand_2 IS
17       PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
18    END COMPONENT;
19    -------------
20    COMPONENT nand_3 IS
21       PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
22    END COMPONENT;
23    -------------
24    SIGNAL w: STD_LOGIC;
25 BEGIN
26    U1: inverter PORT MAP (b, w);
27    U2: nand_2 PORT MAP (a, b, x);
28    U3: nand_3 PORT MAP (w, c, d, y);
29 END structural;
30 ----------------------------------------
```

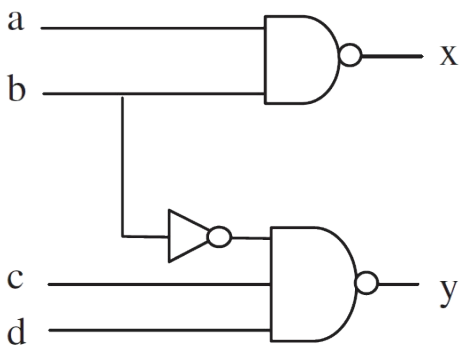


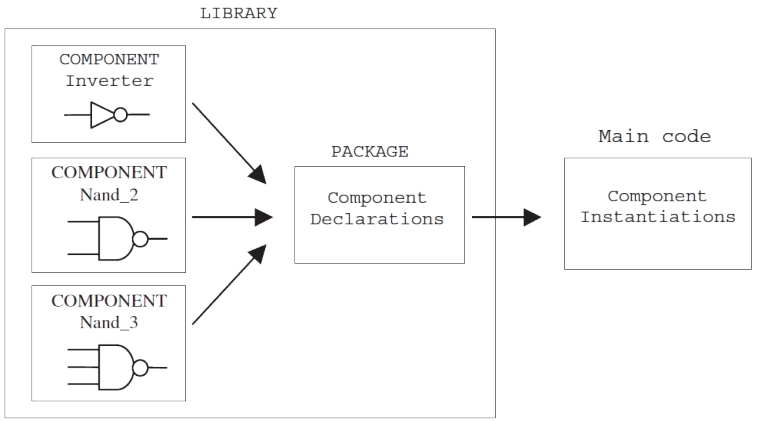


Komponent w pakiecie

```
1  ----- File my_components.vhd: --------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------
5  PACKAGE my_components IS
6     ------ inverter: -------
7     COMPONENT inverter IS
8        PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
9     END COMPONENT;
10    ------ 2-input nand: ---
11    COMPONENT nand_2 IS
12       PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
13    END COMPONENT;
14    ------ 3-input nand: ---
15    COMPONENT nand_3 IS
16       PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
17    END COMPONENT;
18 ------------------------
19 END my_components;
20 -----------------------------------------
```

```
1  ----- File project.vhd: --------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_components.all;
5  --------------------------------
6  ENTITY project IS
7     PORT ( a, b, c, d: IN STD_LOGIC;
8            x, y: OUT STD_LOGIC);
9  END project;
10 --------------------------------
11 ARCHITECTURE structural OF project IS
12    SIGNAL w: STD_LOGIC;
13 BEGIN
14 U1: inverter PORT MAP (b, w);
15 U2: nand_2 PORT MAP (a, b, x);
16 U3: nand_3 PORT MAP (w, c, d, y);
17 END structural;
18 -----------------------------------------
```





PORT MAP

- Są dwa sposoby dołączania portów komponentów do sygnałów:
  - Odwzorowanie pozycyjne
    - sygnały x oraz y odpowiadają portom a oraz b

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
U1: inverter PORT MAP (x, y);
```

  - Odwzorowanie przez nazwę
    - podajemy nazwę każdego portu, do którego dołączamy sygnał
    - możemy także zostawić niepodłączone porty

```
U1: inverter PORT MAP (x=>a, y=>b);
U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

GENERIC MAP

- GENERIC MAP służy do przekazania parametrów do używanej ENTITY
- Musi wystąpić przed PORT MAP
- Możliwe odwzorowanie pozycyjne lub przez nazwę

```
label: compon_name GENERIC MAP (param. list) PORT MAP (port list);
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

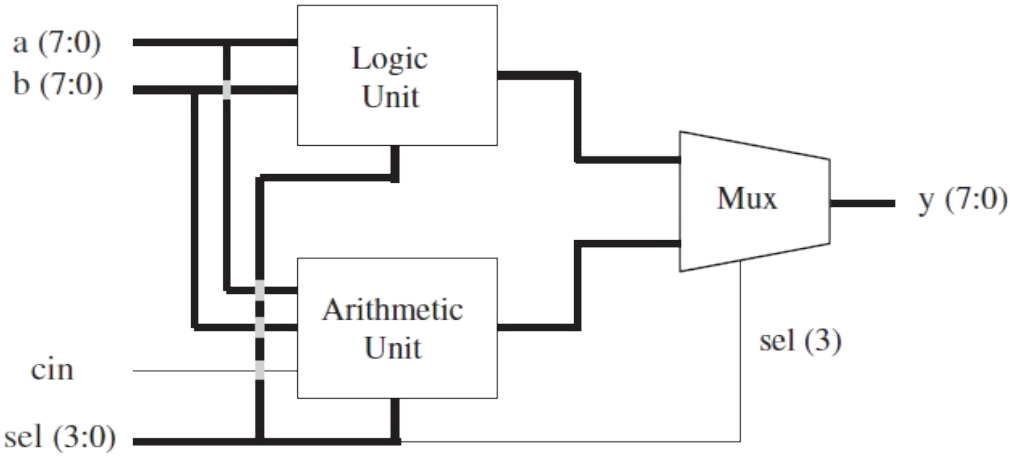
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Przykład użycia sparametryzowanego komponentu

```
1  ------ File parity_gen.vhd (component): --------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------
5  ENTITY parity_gen IS
6     GENERIC (n : INTEGER := 7); -- default is 7
7     PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
8            output: OUT BIT_VECTOR (n+1 DOWNTO 0));
9  END parity_gen;
10 ---------------------------------
11 ARCHITECTURE parity OF parity_gen IS
12 BEGIN
13    PROCESS (input)
14        VARIABLE temp1: BIT;
15        VARIABLE temp2: BIT_VECTOR (output'RANGE);
16    BEGIN
17       temp1 := '0';
18       FOR i IN input'RANGE LOOP
19          temp1 := temp1 XOR input(i);
20          temp2(i) := input(i);
21       END LOOP;
22       temp2(output'HIGH) := temp1;
23       output <= temp2;
24    END PROCESS;
25 END parity;
26 ---------------------------------------------
```

```
1  ------ File my_code.vhd (actual project): ------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------
5  ENTITY my_code IS
6     GENERIC (n : POSITIVE := 2); -- 2 will overwrite 7
7     PORT ( inp: IN BIT_VECTOR (n DOWNTO 0);
8            outp: OUT BIT_VECTOR (n+1 DOWNTO 0));
9  END my_code;
10 ---------------------------------
11 ARCHITECTURE my_arch OF my_code IS
12 ------------------------
13    COMPONENT parity_gen IS
14       GENERIC (n : POSITIVE);
15       PORT (input: IN BIT_VECTOR (n DOWNTO 0);
16             output: OUT BIT_VECTOR (n+1 DOWNTO 0));
17    END COMPONENT;
18 ------------------------
19 BEGIN
20    C1: parity_gen GENERIC MAP(n) PORT MAP(inp, outp);
21 END my_arch;
22 -------------------------------------------------
```

Przykład: ALU złożone z komponentów



sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

Przykład: ALU złożone z komponentów

```
1  -------- COMPONENT arith_unit: --------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_unsigned.all;
5  ----------------------------------------
6  ENTITY arith_unit IS
7     PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8            sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
9            cin: IN STD_LOGIC;
10           x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END arith_unit;
12 ----------------------------------------
13 ARCHITECTURE arith_unit OF arith_unit IS
14    SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16    WITH sel SELECT
17       x <= a WHEN "000",
18            a+1 WHEN "001",
19            a-1 WHEN "010",
20            b WHEN "011",
21            b+1 WHEN "100",
22            b-1 WHEN "101",
23            a+b WHEN "110",
24            a+b+cin WHEN OTHERS;
25 END arith_unit;
26 ----------------------------------------------------
```

Przykład: ALU złożone z komponentów

```
1 -------- COMPONENT logic_unit: -------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY logic_unit IS
6    PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7           sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8           x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END logic_unit;
10 ---------------------------------------
11 ARCHITECTURE logic_unit OF logic_unit IS
12    BEGIN
13       WITH sel SELECT
14          x <= NOT a WHEN "000",
15               NOT b WHEN "001",
16               a AND b WHEN "010",
17               a OR b WHEN "011",
18               a NAND b WHEN "100",
19               a NOR b WHEN "101",
20               a XOR b WHEN "110",
21               NOT (a XOR b) WHEN OTHERS;
22 END logic_unit;
23 -----------------------------------------------
```

```
1  -------- COMPONENT mux: --------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY mux IS
6     PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            sel: IN STD_LOGIC;
8            x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END mux;
10 ---------------------------------------
11 ARCHITECTURE mux OF mux IS
12 BEGIN
13    WITH sel SELECT
14       x <= a WHEN '0',
15       b WHEN OTHERS;
16 END mux;
17 -----------------------------------------------
```

Przykład: ALU złożone z komponentów

```
1  -------- Project ALU (main code): -------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY alu IS
6     PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7            cin: IN STD_LOGIC;
8            sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9            y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
10 END alu;
11 ---------------------------------------
12 ARCHITECTURE alu OF alu IS
13 ----------------------
14 COMPONENT arith_unit IS
15    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
16           cin: IN STD_LOGIC;
17           sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
18           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
19 END COMPONENT;
20 ----------------------
21 COMPONENT logic_unit IS
22    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23           sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
24           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
25 END COMPONENT;
26 ----------------------
27 COMPONENT mux IS
28    PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
29           sel: IN STD_LOGIC;
30           x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
31 END COMPONENT;
32 ----------------------
33    SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);
34 ----------------------
35 BEGIN
36    U1: arith_unit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
37    U2: logic_unit PORT MAP (a, b, sel(2 DOWNTO 0), x2);
38    U3: mux PORT MAP (x1, x2, sel(3), y);
39 END alu;
40 -------------------------------------------------
```

Funkcje i procedury

- Funkcje i procedury są wspólnie określane mianem podprogramów
- Używają instrukcji sekwencyjnych (IF, CASE oraz LOOP)
- Głównym celem stosowania funkcji i procedur jest tworzenie bibliotek, aczkolwiek mogą one być również używane na bieżąco w tworzonym kodzie.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

FUNCTION

- Funkcja jest częścią kodu sekwencyjnego
- Służy do rozwiązywania często spotykanych problemów, takich jak konwersje typów, operacje logiczne, obliczenia arytmetyczne, definiowanie nowych operatorów
- Stosowanie funkcji skraca główny kod, powodując że jest łatwiejszy do zrozumienia
- Definicja funkcji:

```
FUNCTION function_name [<parameter list>] RETURN data_type IS
   [declarations]
BEGIN
   (sequential statements)
END function_name;

<parameter list> = [CONSTANT] constant_name: constant_type;
<parameter list> = SIGNAL signal_name: signal_type;
```

FUNCTION

- Parametry mogą być stałymi lub sygnałami
- Typy parametrów mogą być dowolne
  - Nie należy umieszczać specyfikacji zakresów (RANGE, TO/DOWNTO) w deklaracji parametrów

```
FUNCTION f1 (a, b: INTEGER; SIGNAL c: STD_LOGIC_VECTOR)
   RETURN BOOLEAN IS
BEGIN
   (sequential statements)
END f1;
```

- Funkcja jest wywoływana jako element wyrażenia.
- Wyrażenie może występować samodzielnie bądź jako element instrukcji (sekwencyjnej lub współbieżnej)

```
x <= conv_integer(a);     -- converts a to an integer
                          -- (expression appears by itself)
y <= maximum(a, b);       -- returns the largest of a and b
                          -- (expression appears by itself)
IF x > maximum(a, b) ...  -- compares x to the largest of a, b
                          -- (expression associated to a
                          -- statement)
```

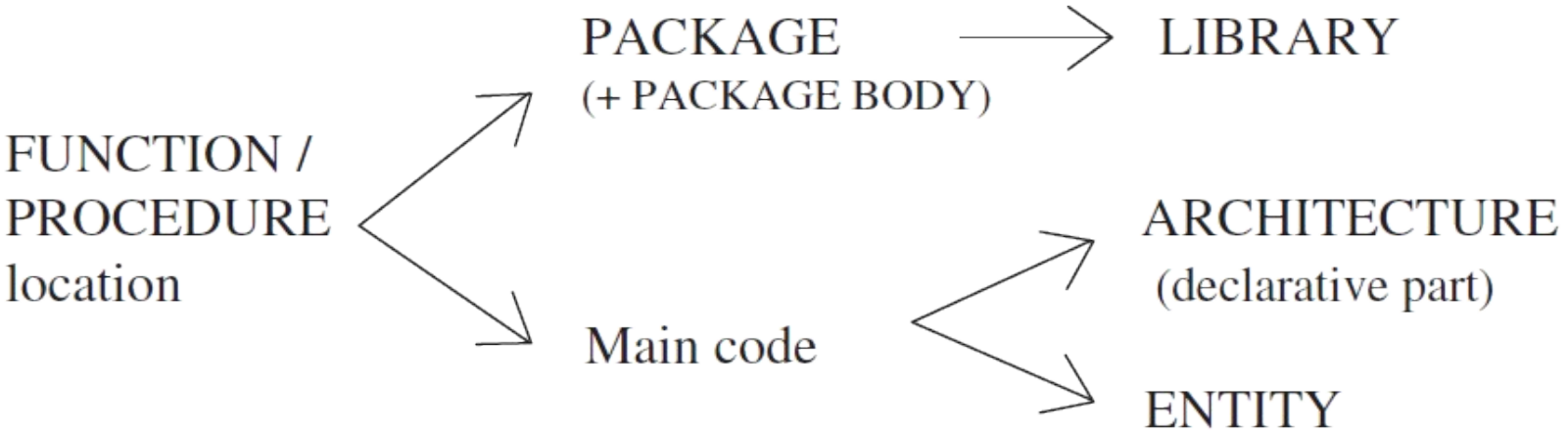
Przykłady funkcji

```
------ Function body: -------------------------------
FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
BEGIN
   RETURN (s'EVENT AND s='1');
END positive_edge;
------ Function call: -------------------------------
...
IF positive_edge(clk) THEN...
...
-------------------------------------------------------
```

```
------ Function body: -------------------------------
FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
      RETURN INTEGER IS
   VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
BEGIN
   IF (vector(vector'HIGH)='1') THEN result:=1;
   ELSE result:=0;
   END IF;
   FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
      result:=result*2;
      IF(vector(i)='1') THEN result:=result+1;
      END IF;
   END LOOP;
   RETURN result;
END conv_integer;
------ Function call: -------------------------------
...
y <= conv_integer(a);
...
-------------------------------------------------------
```

Miejsce definicji funkcji

- Funkcja może być umieszczona w pakiecie lub głównym kodzie (ENTITY lub ARCHITECTURE)
- Jeżeli funkcja zdefiniowana jest w pakiecie, część deklaracyjna zawiera deklarację funkcji, a ciało pakietu jej definicję



Funkcja zdefiniowana w głównym kodzie (wariant 1)

```
1  ------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END dff;
9  ------------------------------------------
10 ARCHITECTURE my_arch OF dff IS
11 ---------------------------------------
12    FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
13       RETURN BOOLEAN IS
14    BEGIN
15       RETURN s'EVENT AND s='1';
16    END positive_edge;
17 ---------------------------------------
18 BEGIN
19    PROCESS (clk, rst)
20    BEGIN
21       IF (rst='1') THEN q <= '0';
22       ELSIF positive_edge(clk) THEN q <= d;
23       END IF;
24    END PROCESS;
25 END my_arch;
26 ------------------------------------------
```

Funkcja zdefiniowana w głównym kodzie (wariant 2)

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8     FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
9        RETURN BOOLEAN IS
10    BEGIN
11       RETURN s'EVENT AND s='1';
12    END positive_edge;
13 ----------------------------------------
14 END dff;
15 ----------------------------------------
16 ARCHITECTURE my_arch OF dff IS
17 ----------------------------------------
18 BEGIN
19    PROCESS (clk, rst)
20    BEGIN
21       IF (rst='1') THEN q <= '0';
22       ELSIF positive_edge(clk) THEN q <= d;
23       END IF;
24    END PROCESS;
25 END my_arch;
----------------------------------------
```

Funkcja zdefiniowana w pakiecie

```
1  ------- Package: ----------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  PACKAGE my_package IS
6     FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
         RETURN BOOLEAN;
7  END my_package;
8  ---------------------------------------------
9  PACKAGE BODY my_package IS
10    FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
11       RETURN BOOLEAN IS
12    BEGIN
13       RETURN s'EVENT AND s='1';
14    END positive_edge;
15 END my_package;
16 ---------------------------------------------
```

```
1 ------ Main code: ----------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE work.my_package.all;
5 ---------------------------------------------
6 ENTITY dff IS
7    PORT ( d, clk, rst: IN STD_LOGIC;
8    q: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------------
11 ARCHITECTURE my_arch OF dff IS
12 BEGIN
13    PROCESS (clk, rst)
14    BEGIN
15       IF (rst='1') THEN q <= '0';
16       ELSIF positive_edge(clk) THEN q <= d;
17       END IF;
18    END PROCESS;
19 END my_arch;
20 ---------------------------------------------
```

Funkcja conv_integer()

```
1  --------- Package: -------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----------------------------------------
5  PACKAGE my_package IS
6     FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
7        RETURN INTEGER;
8  END my_package;
9  -----------------------------------------
10 PACKAGE BODY my_package IS
11    FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
12          RETURN INTEGER IS
13       VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
14    BEGIN
15       IF (vector(vector'HIGH)='1') THEN result:=1;
16       ELSE result:=0;
17       END IF;
18       FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
19          result:=result*2;
20          IF(vector(i)='1') THEN result:=result+1;
21          END IF;
22       END LOOP;
23       RETURN result;
24    END conv_integer;
25 END my_package;
26 -----------------------------------------
```

```
1  -------- Main code: -------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_package.all;
5  -----------------------------------------
6  ENTITY conv_int2 IS
7     PORT ( a: IN STD_LOGIC_VECTOR(0 TO 3);
8            y: OUT INTEGER RANGE 0 TO 15);
9  END conv_int2;
10 -----------------------------------------
11 ARCHITECTURE my_arch OF conv_int2 IS
12 BEGIN
13    y <= conv_integer(a);
14 END my_arch;
15 -----------------------------------------
```

Przeciążony operator dodawania

```
1  -------- Package: ---------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  PACKAGE my_package IS
6     FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
7        RETURN STD_LOGIC_VECTOR;
8  END my_package;
9  ---------------------------------------------
10 PACKAGE BODY my_package IS
11    FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
12          RETURN STD_LOGIC_VECTOR IS
13       VARIABLE result:
14          STD_LOGIC_VECTOR(a'LEFT downto a'RIGHT);
15       VARIABLE carry: STD_LOGIC;
16    BEGIN
17       carry := '0';
18       FOR i IN a'REVERSE_RANGE LOOP
19          result(i) := a(i) XOR b(i) XOR carry;
20          carry := (a(i) AND b(i)) OR
21             (a(i) AND carry) OR (b(i) AND carry);
22       END LOOP;
23       RETURN result;
24    END "+";
25 END my_package;
26 ---------------------------------------------
```

```
1  --------- Main code: ------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_package.all;
5  ---------------------------------------------
6  ENTITY add_bit IS
7  PORT ( a: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
8         y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
9  END add_bit;
10 ---------------------------------------------
11 ARCHITECTURE my_arch OF add_bit IS
12    CONSTANT b: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
13    CONSTANT c: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
14 BEGIN
15    y <= a + b + c; -- overloaded "+" operator
16 END my_arch;
17 ---------------------------------------------
```

Przesunięcie arytmetyczne w lewo

```
1  ------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY shift_left IS
6     GENERIC (size: INTEGER := 4);
7     PORT ( a: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
8            x, y, z: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
9  END shift_left;
10 ------------------------------------------
11 ARCHITECTURE behavior OF shift_left IS
12 ------------------------------------------
13    FUNCTION slar (arg1: STD_LOGIC_VECTOR; arg2: NATURAL)
14          RETURN STD_LOGIC_VECTOR IS
15       VARIABLE input: STD_LOGIC_VECTOR(size-1 DOWNTO 0) := arg1;
16       CONSTANT size : INTEGER := arg1'LENGTH;
17       VARIABLE copy: STD_LOGIC_VECTOR(size-1 DOWNTO 0)
18                                 := (OTHERS => arg1(arg1'RIGHT));
19       VARIABLE result: STD_LOGIC_VECTOR(size-1 DOWNTO 0);
20    BEGIN
21       IF (arg2 >= size-1) THEN result := copy;
22       ELSE result := input(size-1-arg2 DOWNTO 1) &
23                      copy(arg2 DOWNTO 0);
24       END IF;
25       RETURN result;
26    END slar;
27 ------------------------------------------
28 BEGIN
29    x <= slar(a, 0);
30    y <= slar(a, 1);
31    z <= slar(a, 2);
32 END behavior;
33 ------------------------------------------
```

Mnożarka

```
1  --------- Package: ----------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ------------------------------------------
6  PACKAGE pack IS
7     FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
8  END pack;
9  ------------------------------------------
10 PACKAGE BODY pack IS
11    FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED IS
12       CONSTANT max: INTEGER := a'LENGTH + b'LENGTH - 1;
13       VARIABLE aa: UNSIGNED(max DOWNTO 0) :=
14                           (max DOWNTO a'LENGTH => '0')
15                           & a(a'LENGTH-1 DOWNTO 0);
16       VARIABLE prod: UNSIGNED(max DOWNTO 0) := (OTHERS => '0');
17    BEGIN
18       FOR i IN 0 TO b'LENGTH-1 LOOP
19          IF (b(i)='1') THEN prod := prod + aa;
20          END IF;
21          aa := aa(max-1 DOWNTO 0) & '0';
22       END LOOP;
23       RETURN prod;
24    END mult;
25 END pack;
26 ------------------------------------------------------
```

```
1  -------- Main code: ----------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  USE work.pack.all;
6  ------------------------------------------
7  ENTITY multiplier IS
8     GENERIC (size: INTEGER := 4);
9     PORT ( a, b: IN UNSIGNED(size-1 DOWNTO 0);
10           y: OUT UNSIGNED(2*size-1 DOWNTO 0));
11 END multiplier;
12 ------------------------------------------
13 ARCHITECTURE behavior OF multiplier IS
14 BEGIN
15    y <= mult(a,b);
16 END behavior;
17 ------------------------------------------------------
```

PROCEDURE

- Procedury są bardzo podobne do funkcji i mają ten sam podstawowy cel
- Procedura może przekazać więcej niż jedną wartość

```
PROCEDURE procedure_name [<parameter list>] IS
   [declarations]
BEGIN
   (sequential statements)
END procedure_name;

<parameter list> = [CONSTANT] constant_name: mode type;
<parameter list> = SIGNAL signal_name: mode type;
<parameter list> = VARIABLE variable_name: mode type;
```

- Procedura może mieć dowolną liczbę parametrów w trybie IN, OUT oraz INOUT, które mogą być sygnałami, zmiennymi lub stałymi
- Dla parametrów wejściowych domyślnym trybem jest CONSTANT, dla wyjściowych (OUT i INOUT) VARIABLE

PROCEDURE

- W przeciwieństwie do funkcji procedura może przyjmować sygnały jako argumenty

```
PROCEDURE my_procedure ( a: IN BIT; SIGNAL b, c: IN BIT;
                         SIGNAL x: OUT BIT_VECTOR(7 DOWNTO 0);
                         SIGNAL y: INOUT INTEGER RANGE 0 TO 99) IS
BEGIN
   ...
END my_procedure;
```

- Wywołanie procedury może być samodzielną instrukcją

```
compute_min_max(in1, in2, 1n3, out1, out2);
                -- statement by itself
divide(dividend, divisor, quotient, remainder);
                -- statement by itself
IF (a>b) THEN compute_min_max(in1, in2, 1n3, out1, out2);
                -- procedure call associated to another statement
```

Procedura w kodzie głównym

```
1  ----------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------------
5  ENTITY min_max IS
6     GENERIC (limit : INTEGER := 255);
7     PORT ( ena: IN BIT;
8            inp1, inp2: IN INTEGER RANGE 0 TO limit;
9            min_out, max_out: OUT INTEGER RANGE 0 TO limit);
10 END min_max;
11 ----------------------------------------------------
12 ARCHITECTURE my_architecture OF min_max IS
13 -------------------------
14    PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
15       SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
16    BEGIN
17       IF (in1 > in2) THEN
18          max <= in1;
19          min <= in2;
20       ELSE
21          max <= in2;
22          min <= in1;
23       END IF;
24    END sort;
25 -------------------------
26 BEGIN
27    PROCESS (ena,inp1,inp2)
28    BEGIN
29       IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
30       END IF;
31    END PROCESS;
32 END my_architecture;
33 ----------------------------------------------------
```

Procedura w pakiecie

```
1  ------------ Package: -------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  PACKAGE my_package IS
6     CONSTANT limit: INTEGER := 255;
7     PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
8                     SIGNAL min, max: OUT INTEGER RANGE 0 TO limit);
9  END my_package;
10 ------------------------------------
11 PACKAGE BODY my_package IS
12    PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
13       SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
14    BEGIN
15       IF (in1 > in2) THEN
16          max <= in1;
17          min <= in2;
18       ELSE
19          max <= in2;
20          min <= in1;
21       END IF;
22    END sort;
23 END my_package;
24 ------------------------------------------------
```

```
1  --------- Main code: ----------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_package.all;
5  ------------------------------------
6  ENTITY min_max IS
7     GENERIC (limit: INTEGER := 255);
8     PORT ( ena: IN BIT;
9            inp1, inp2: IN INTEGER RANGE 0 TO limit;
10           min_out, max_out: OUT INTEGER RANGE 0 TO limit);
11 END min_max;
12 ------------------------------------
13 ARCHITECTURE my_architecture OF min_max IS
14 BEGIN
15    PROCESS (ena,inp1,inp2)
16    BEGIN
17       IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
18       END IF;
19    END PROCESS;
20 END my_architecture;
21 ------------------------------------------------
```

ASSERT

- ASSERT jest niesyntezowalną instrukcją, służącą do wypisywania komunikatów (np. na ekranie) gdy pojawią się problemy podczas symulacji
- Zależnie od powagi problemu, symulacja może zostać zakończona

```
ASSERT condition
[REPORT "message"]
[SEVERITY severity_level];
```

- severity_level może mieć wartość Note, Warning, Error (domyśnie) lub Failure
- Komunikat jest wypisywany, gdy warunek ma wartość FALSE

```
ASSERT a'LENGTH = b'LENGTH
   REPORT "Error: vectors do not have same length!"
   SEVERITY failure;
```

Problemy

- Konwersja do STD_LOGIC_VECTOR
  - Napisać funkcję konwertującą INTEGER do STD_LOGIC_VECTOR o nazwie conv_std_logic. Przetestować funkcję.
- Mnożarka liczb ze znakiem
  - Napisać funkcję mnożącą operującą na argumentach typu SIGNED

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„Układy reprogramowalne i SoC”
„Język VHDL (część 5)”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Prezentacja dystrybuowana jest bezpłatnie