# „Układy reprogramowalne i SoC"
# „Język VHDL (część 2)"

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

# Operatory

- VHDL posiada szereg predefiniowanych operatorów:
  - Przypisania
  - Logicznych
  - Arytmetycznych
  - Relacji
  - Przesunięć
  - Konkatenacji

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operatory przypisania

- Używane do nadawania wartości sygnałom, zmiennym i stałym
  - <= - dla sygnałów
  - := - dla zmiennych, stałych oraz parametrów GENERIC. Używany również do nadawania wartości początkowych.
  - => - do nadawania wartości poszczególnym elementom wektora.

```
SIGNAL x   : STD_LOGIC;
VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Leftmost bit is MSB
SIGNAL w   : STD_LOGIC_VECTOR(0 TO 7);     -- Rightmost bit is
                                           -- MSB

x <= '1';                    -- '1' is assigned to SIGNAL x using "<="
y := "0000";                 -- "0000" is assigned to VARIABLE y using ":="
w <= "10000000";             -- LSB is '1', the others are '0'
w <= (0 =>'1', OTHERS =>'0'); -- LSB is '1', the others are '0'
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operatory logiczne

- Dla operandów typu BIT, STD_LOGIC, STD_ULOGIC, BIT_VECTOR, STD_LOGIC_VECTOR lub STD_ULOGIC_VECTOR
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - XOR
  - XNOR (VHDL93)

```
y <= NOT a AND b;    -- (a'.b)
y <= NOT (a AND b);  -- (a.b)'
y <= a NAND b;       -- (a.b)'
```

- NOT ma wyższy priorytet niż pozostałe operatory

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operatory arytmetyczne

- Używane do operacji arytmetycznych na typach INTEGER, SIGNED, UNSIGNED i REAL
  - dodawanie (+)
  - odejmowanie (-)
  - mnożenie (*)
  - dzielenie (/)
  - potęgowanie (**)
  - modulo, znak taki jak znak drugiego argumentu (MOD)
    - -5 mod 3 = 1
    - 7 mod -4 = -1
  - modulo, znak taki jak znak pierwszego argumentu (REM)
    - -5 rem 3 = -2
    - 7 rem -4 = 3
  - wartość bezwzględna (ABS)

# Operatory arytmetyczne

- Wsparcie dla syntezy:
  - dodawanie, odejmowanie, mnożenie bez ograniczeń
  - dzielenie - tylko potęgi dwójki (przesunięcie bitowe)
  - pozostałe przeważnie niesyntezowalne

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operatory relacji

- Dla typów INTEGER, SIGNED, UNSIGNED i REAL
  - równe (=)
  - różne (/=)
  - mniejsze (<)
  - większe (>)
  - mniejsze lub równe (<=)
  - wieksze lub równe (>=)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operatory przesunięć

- Wprowadzone w VHDL93
  - <lewy operand> operacja <prawy operand>
  - lewy operand - BIT_VECTOR
  - prawy operand - INTEGER
  - sll - przesuwa logicznie w lewo (miejsca zwolnione zapisywane są wartością '0')
  - srl - przesuwa logicznie w prawo (miejsca zwolnione zapisywane są wartością '0')
  - sla - przesuwa arytmetycznie w lewo (miejsca zwolnione zapisywane są kopią prawego bitu)
  - sra - przesuwa arytmetycznie w prawo (miejsca zwolnione zapisywane są kopią lewego bitu)
  - rol - obraca logicznie w lewo (pierwszy bit staje się drugim, drugi - trzecim, itd.)
  - ror - obraca logicznie w prawo (pierwszy bit staje się ostatnim, drugi - pierwszym, itd.)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operator konkatenacji

- &
- Dla typów BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR, SIGNED, UNSIGNED

```
y <= "01000010";
z <= "010" & y & "11";
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Atrybuty tablic

- Typy tablicowe posiadają następujące predefiniowane atrybuty:
  - d'LOW: Najniższy indeks tablicy
  - d'HIGH: Najwyzszy adres tablicy
  - d'LEFT: Indeks skrajnego lewego elementu tablicy
  - d'RIGHT: Indeks skrajnego prawego elementu tablicy
  - d'LENGTH: Rozmiar wektora
  - d'RANGE: Zakres wektora
  - d'REVERSE_RANGE: Odwrócony zakres wektora

KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

# Atrybuty tablic - przykłady

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);

d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,
d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).

SIGNAL x: STD_LOGIC_VECTOR (0 TO 7);
-- All four LOOP statements below are synthesizable and equivalent.
FOR i IN RANGE (0 TO 7) LOOP ...
FOR i IN x'RANGE LOOP ...
FOR i IN RANGE (x'LOW TO x'HIGH) LOOP ...
FOR i IN RANGE (0 TO x'LENGTH-1) LOOP ...
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Atrybuty sygnałów

- s’EVENT: Zwraca true, jeżeli w obecnej chwili nastąpiło zdarzenie na sygnale s
- s’STABLE: Zwraca true, jeżeli w obecnej chwili nie było zdarzenia na sygnale s
- s’ACTIVE: Zwraca true, jeżeli s = ‘1’
- s’QUIET <time>: Zwraca true, jeżeli nie było zdarzenia na sygnale s w podanym czasie
- s’LAST_EVENT: Zwraca czas od ostatniego zdarzenia na sygnale s
- s’LAST_ACTIVE: Zwraca czas od ostatniego momentu, gdy s miało wartość ‘1’
- s’LAST_VALUE: Zwraca wartość s sprzed ostatniego zdarzenia na sygnale s

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Atrybuty sygnałów

- Większość atrybutów sygnałów służy jedynie do symulacji, jedynie dwa pierwsze są używane przy syntezie

```
IF (clk'EVENT AND clk='1')... -- EVENT attribute used
-- with IF
IF (NOT clk'STABLE AND clk='1')... -- STABLE attribute used
-- with IF
WAIT UNTIL (clk'EVENT AND clk='1'); -- EVENT attribute used
-- with WAIT
IF RISING_EDGE(clk)... -- call to a function
```

# Atrybuty zdefiniowane przez użytkownika

- Oprócz korzystania z predefiniowanych atrybutów użytkownik może również definiować własne
- Deklaracja atrybutu:

```
ATTRIBUTE attribute_name: attribute_type;
```

- Specyfkacja atrybutu:

```
ATTRIBUTE attribute_name OF target_name: class IS value;
```

- Gdzie:
  - attribute_type: dowolny typ danych
  - class: TYPE, SIGNAL, FUNCTION, itd.
  - value: ‘0’, 27, “00 11 10 01”, itd.

# Atrybuty zdefiniowane przez użytkownika - przykłady

```
ATTRIBUTE number_of_inputs: INTEGER;                -- declaration
ATTRIBUTE number_of_inputs OF nand3: SIGNAL IS 3; -- specification
...
inputs <= nand3'number_of_inputs;       -- attribute call, returns 3
```

- Wielu producentów narzędzi do syntezy (w tym Xilinx) stosuje atrybut *enum_encoding*, pozwalający na wybór kodowania typów wyliczeniowych:

```
ATTRIBUTE enum_encoding OF color: TYPE IS "11 00 10 01";
```

- Najczęstszym zastosowaniem tego atrybutu jest synteza automatów skończonych

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przeciążenie operatorów

- Predefiniowane operatory mogą operować tylko na niektórych typach danych
- Jeżeli nam to nie wystarcza, możemy zdefiniować własne:

```
--------------------------------------
FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS
BEGIN
   IF (b='1') THEN RETURN a+1;
   ELSE RETURN a;
   END IF;
END "+";
--------------------------------------


------------------------------
SIGNAL inp1, outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;
   (...)
outp <= 3 + inp1 + inp2;
   (...)
------------------------------
```

# GENERIC

- GENERIC służy do parametryzacji kodu VHDL

```
GENERIC (parameter_name : parameter_type := parameter_value);
```

- Przykład: zdefiniować parametr o nazwie n, typu INTEGER, z domyślną wartością 8:

```
ENTITY my_entity IS
   GENERIC (n : INTEGER := 8);
   PORT (...);
END my_entity;
ARCHITECTURE my_architecture OF my_entity IS
   ...
END my_architecture;
```

- W entity można mieć więcej niż jeden parametr GENERIC:

```
GENERIC (n: INTEGER := 8; vector: BIT_VECTOR := "00001111");
```

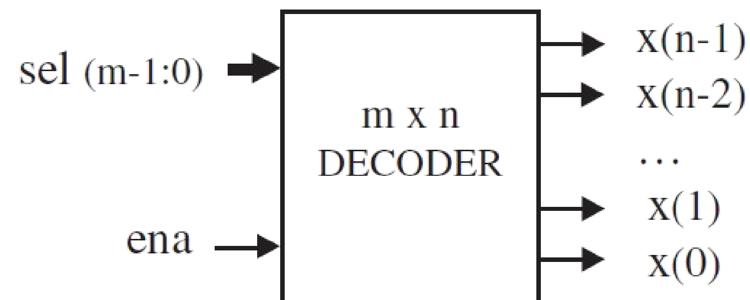
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przykład: dekoder

```
1  ---------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY decoder IS
6     PORT ( ena : IN STD_LOGIC;
7            sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            x :   OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END decoder;
10  ---------------------------------------------
11 ARCHITECTURE generic_decoder OF decoder IS
12 BEGIN
13    PROCESS (ena, sel)
14       VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
15       VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
16    BEGIN
17    temp1 := (OTHERS => '1');
18    temp2 := 0;
19    IF (ena='1') THEN
20       FOR i IN sel'RANGE LOOP -- sel range is 2 downto 0
21          IF (sel(i)='1') THEN -- Bin-to-Integer conversion
22             temp2 := 2*temp2+1;
23          ELSE
24             temp2 := 2*temp2;
25         END IF;
26      END LOOP;
27    temp1(temp2):='0';
28    END IF;
29    x <= temp1;
30 END PROCESS;
31 END generic_decoder;
32 ---------------------------------------------
```



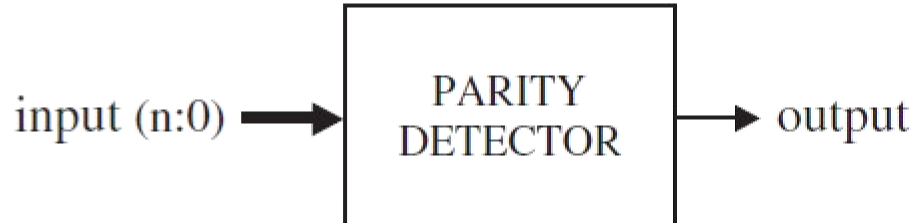

| ena | sel | x |
|---|---|---|
| 0 | 00 | 1111 |
| 1 | 00 | 1110 |
| | 01 | 1101 |
| | 10 | 1011 |
| | 11 | 0111 |

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

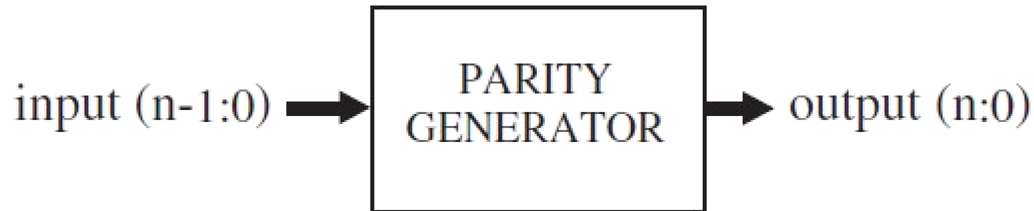
# Przykład: kontroler parzystości




```
1  ---------------------------------------------
2  ENTITY parity_det IS
3     GENERIC (n : INTEGER := 7);
4     PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
5            output: OUT BIT);
6  END parity_det;
7  ---------------------------------------------
8  ARCHITECTURE parity OF parity_det IS
9  BEGIN
10    PROCESS (input)
11       VARIABLE temp: BIT;
12    BEGIN
13       temp := '0';
14       FOR i IN input'RANGE LOOP
15          temp := temp XOR input(i);
16       END LOOP;
17    output <= temp;
18 END PROCESS;
19 END parity;
20 ---------------------------------------------
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przykład: generator parzystości




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

# Problem

```
SIGNAL a : BIT := '1';
SIGNAL b : BIT_VECTOR (3 DOWNTO 0) := "1100";
SIGNAL c : BIT_VECTOR (3 DOWNTO 0) := "0010";
SIGNAL d : BIT_VECTOR (7 DOWNTO 0);
SIGNAL e : INTEGER RANGE 0 TO 255;
SIGNAL f : INTEGER RANGE -128 TO 127;

x1 <= a & c;                        -> x1 <= ________
x2 <= c & b;                        -> x2 <= ________
x3 <= b XOR c;                      -> x3 <= ________
x4 <= a NOR b(3);                   -> x4 <= ________
x5 <= b sll 2;                      -> x5 <= ________
x6 <= b sla 2;                      -> x6 <= ________
x7 <= b rol 2;                      -> x7 <= ________
x8 <= a AND NOT b(0) AND NOT c(1); -> x8 <= ________
d <= (5=>'0', OTHERS=>'1');         -> d  <= ________

c'LOW           -> ______
d'HIGH          -> ______
c'LEFT          -> ______
d'RIGHT         -> ______
c'RANGE         -> ______
d'LENGTH        -> ______
c'REVERSE_RANGE -> ______
```

# Kod współbieżny w VHDL

- W języku VHDL można pisać kod współbieżny i sekwencyjny.
- Występują dwie instrukcje współbieżne: WHEN i GENERATE
- W kodzie współbieżnym mogą również wystąpić przypisania z użyciem operatorów oraz przypisanie blokowe
- Jeżeli w kodzie występują tylko instrukcje współbieżne, wynik nie zależy od ich kolejności

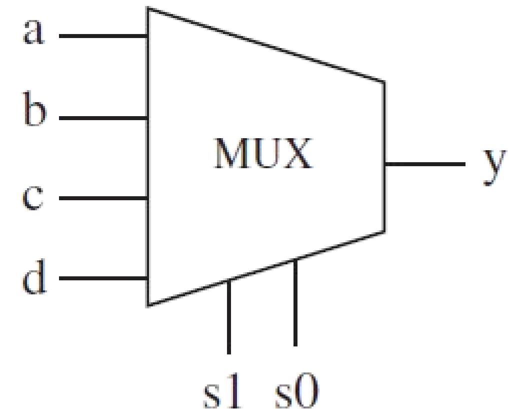
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Multiplekser w postaci kodu współbieżnego

```
1  ---------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY mux IS
6     PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
7     y: OUT STD_LOGIC);
8  END mux;
9 ---------------------------------------
10 ARCHITECTURE pure_logic OF mux IS
11 BEGIN
12   y <= (a AND NOT s1 AND NOT s0) OR
13         (b AND NOT s1 AND s0) OR
14         (c AND s1 AND NOT s0) OR
15         (d AND s1 AND s0);
16 END pure_logic;
17 ---------------------------------------
```



# Instrukcja WHEN

- Instrukcja WHEN może mieć dwie postacie:
  - WHEN/ELSE

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

  - WITH/SELECT/WHEN

```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

- W drugiej postaci wszystkie warianty muszą być obsłużone, więc często przydatne jest słowo kluczowe OTHERS
- Przydatne jest także słowo kluczowe UNAFFECTED, gdy nie chcemy wykonać przypisania w pewnych wariantach

# Instrukcja WHEN - przykład

```
------ With WHEN/ELSE -------------------------
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN ctl='1' ELSE
        "010";

---- With WITH/SELECT/WHEN --------------------
WITH control SELECT
   output <= "000" WHEN reset,
   "111" WHEN set,
   UNAFFECTED WHEN OTHERS;
-----------------------------------------------
```

- WHEN value może przyjąć jedną z trzech form:

```
WHEN value                -- single value
WHEN value1 to value2     -- range, for enumerated data types
                          -- only
WHEN value1 | value2 |... -- value1 or value2 or ...
```

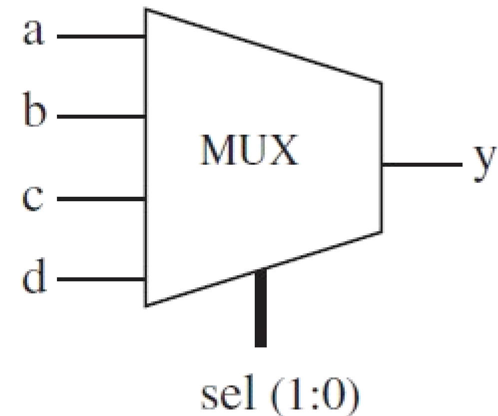
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

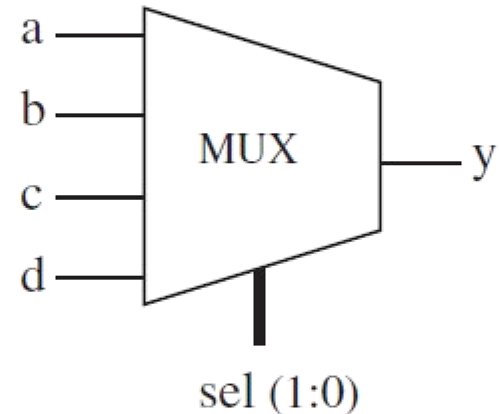
# Przykład - multiplekser (2)

```
1  ------- Solution 1: with WHEN/ELSE --------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY mux IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7            sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8            y: OUT STD_LOGIC);
9  END mux;
10 ---------------------------------------------
11 ARCHITECTURE mux1 OF mux IS
12 BEGIN
13    y <= a WHEN sel="00" ELSE
14         b WHEN sel="01" ELSE
15         c WHEN sel="10" ELSE
16         d;
17 END mux1;
18 ---------------------------------------------
```



# Przykład - multiplekser (2) - drugie rozwiązanie

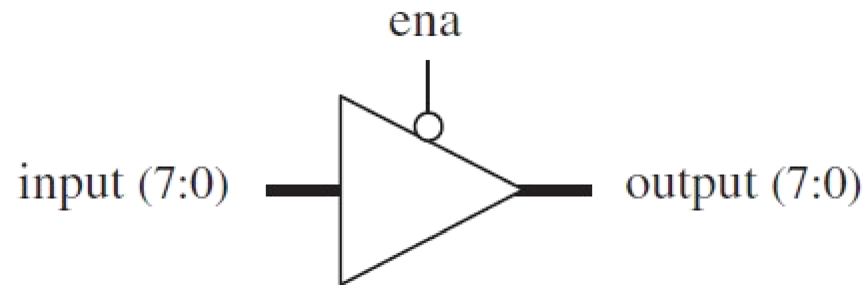
```
1  --- Solution 2: with WITH/SELECT/WHEN -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY mux IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7              sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8              y: OUT STD_LOGIC);
9  END mux;
10 ------------------------------------------
11 ARCHITECTURE mux2 OF mux IS
12 BEGIN
13    WITH sel SELECT
14       y <= a WHEN "00", -- notice "," instead of ";"
15       b WHEN "01",
16       c WHEN "10",
17       d WHEN OTHERS; -- cannot be "d WHEN "11" "
18 END mux2;
19 -------------------------------------------
```



# Przykład - multiplekser (3)

```
1 ---------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY mux IS
6    PORT ( a, b, c, d: IN STD_LOGIC;
7           sel: IN INTEGER RANGE 0 TO 3;
8           y: OUT STD_LOGIC);
9 END mux;
10 ---- Solution 1: with WHEN/ELSE ---------------
11 ARCHITECTURE mux1 OF mux IS
12 BEGIN
13    y <= a WHEN sel=0 ELSE
14         b WHEN sel=1 ELSE
15         c WHEN sel=2 ELSE
16         d;
17 END mux1;
18 -- Solution 2: with WITH/SELECT/WHEN --------
19 ARCHITECTURE mux2 OF mux IS
20 BEGIN
21    WITH sel SELECT
22       y <= a WHEN 0,
23       b WHEN 1,
24       c WHEN 2,
25       d WHEN 3; -- here, 3 or OTHERS are equivalent,
26 END mux2;       -- for all options are tested anyway
27 ---------------------------------------------
```

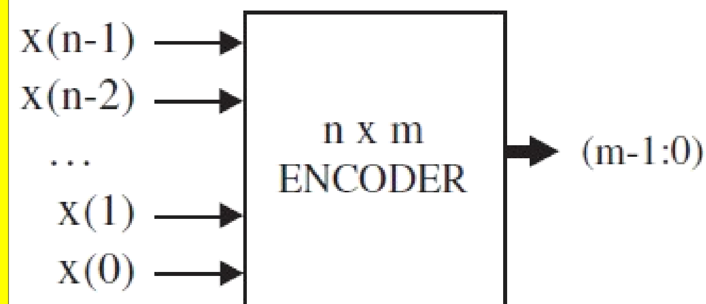
# Przykład - bufor trójstanowy




```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ----------------------------------------------
4  ENTITY tri_state IS
5     PORT ( ena: IN STD_LOGIC;
6            input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
8  END tri_state;
9  ----------------------------------------------
10 ARCHITECTURE tri_state OF tri_state IS
11 BEGIN
12    output <= input WHEN (ena='0') ELSE
13              (OTHERS => 'Z');
14 END tri_state;
15 ----------------------------------------------
```

# Przykład - koder

```
1  ---- Solution 1: with WHEN/ELSE -------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6    PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7           y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  ---------------------------------------------
10 ARCHITECTURE encoder1 OF encoder IS
11 BEGIN
12    y <= "000" WHEN x="00000001" ELSE
13         "001" WHEN x="00000010" ELSE
14         "010" WHEN x="00000100" ELSE
15         "011" WHEN x="00001000" ELSE
16         "100" WHEN x="00010000" ELSE
17         "101" WHEN x="00100000" ELSE
18         "110" WHEN x="01000000" ELSE
19         "111" WHEN x="10000000" ELSE
20         "ZZZ";
21 END encoder1;
22 ---------------------------------------------
```



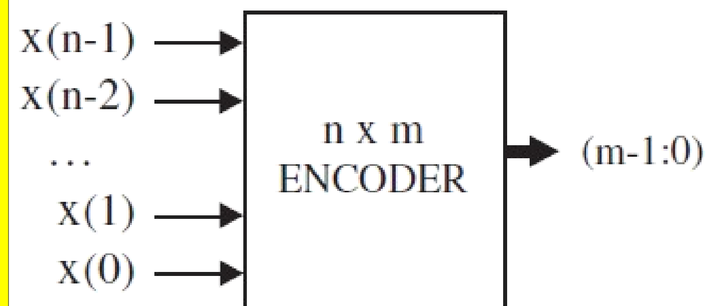

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

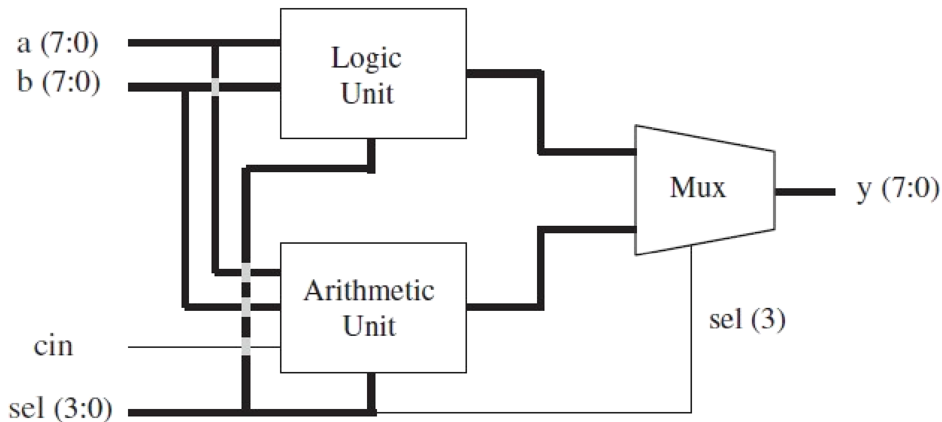
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przykład - koder (2 wersja)

```
1  ---- Solution 2: with WITH/SELECT/WHEN ------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9 ---------------------------------------------
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12    WITH x SELECT
13       y <= "000" WHEN "00000001",
14            "001" WHEN "00000010",
15            "010" WHEN "00000100",
16            "011" WHEN "00001000",
17            "100" WHEN "00010000",
18            "101" WHEN "00100000",
19            "110" WHEN "01000000",
20            "111" WHEN "10000000",
21            "ZZZ" WHEN OTHERS;
22 END encoder2;
23 ---------------------------------------------
```



# Przykład - ALU




| 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 | |

# ALU - implementacja

```
1  ---------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_unsigned.all;
5  ---------------------------------------------
6  ENTITY ALU IS
7     PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9           cin: IN STD_LOGIC;
10          y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 ---------------------------------------------
13 ARCHITECTURE dataflow OF ALU IS
14    SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16    ----- Arithmetic unit: ------
17    WITH sel(2 DOWNTO 0) SELECT
18       arith <= a WHEN "000",
19       a+1 WHEN "001",
20       a-1 WHEN "010",
21       b WHEN "011",
22       b+1 WHEN "100",
23       b-1 WHEN "101",
24       a+b WHEN "110",
25       a+b+cin WHEN OTHERS;
26    ----- Logic unit: -----------
27    WITH sel(2 DOWNTO 0) SELECT
28       logic <= NOT a WHEN "000",
29                NOT b WHEN "001",
30                a AND b WHEN "010",
31                a OR b WHEN "011",
32                a NAND b WHEN "100",
33                a NOR b WHEN "101",
34                a XOR b WHEN "110",
35                NOT (a XOR b) WHEN OTHERS;
36    -------- Mux: ---------------
37    WITH sel(3) SELECT
38       y <= arith WHEN '0',
39       logic WHEN OTHERS;
40 END dataflow;
41 ---------------------------------------------
```

# Instrukcja GENERATE

- Instrukcja współbiezna, pozwalająca na wielokrotne powtórzenie sekcji kodu, generując wiele wersji tego samego przypisania. Posiada dwie formy:
  - FOR/GENERATE

```
label: FOR identifier IN range GENERATE
   (concurrent assignments)
END GENERATE;
```

  - IF/GENERATE

```
label: IF condition GENERATE
   (concurrent assignments)
END GENERATE;
```

  ELSE jest niedozwolone
- Instrukcja GENERATE musi być opatrzona etykietą.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# GENERATE - przykłady

```
SIGNAL x: BIT_VECTOR (7 DOWNTO 0);
SIGNAL y: BIT_VECTOR (15 DOWNTO 0);
SIGNAL z: BIT_VECTOR (7 DOWNTO 0);
...
G1: FOR i IN x'RANGE GENERATE
   z(i) <= x(i) AND y(i+8);
END GENERATE;
```

- Aby kod był syntezowalny, oba limity muszą być statyczne

```
NotOK: FOR i IN 0 TO choice GENERATE
   (concurrent statements)
END GENERATE;
```

- Nie należy wykonywać wielokrotnych przypisań do tego samego sygnału:

```
OK: FOR i IN 0 TO 7 GENERATE
   output(i)<='1' WHEN (a(i) AND b(i))='1' ELSE '0';
END GENERATE;

NotOK: FOR i IN 0 TO 7 GENERATE
   accum <="11111111" WHEN (a(i) AND b(i))='1' ELSE "00000000";
END GENERATE;

NotOK: For i IN 0 to 7 GENERATE
   accum <= accum + 1 WHEN x(i)='1';
END GENERATE;
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przykład - układ przesuwający wektor

```
---------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY shifter IS
6     PORT ( inp: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
7            sel: IN INTEGER RANGE 0 TO 4;
8            outp: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END shifter;
10 ---------------------------------------------
11 ARCHITECTURE shifter OF shifter IS
12    SUBTYPE vector IS STD_LOGIC_VECTOR (7 DOWNTO 0);
13    TYPE matrix IS ARRAY (4 DOWNTO 0) OF vector;
14 SIGNAL row: matrix;
15 BEGIN
16    row(0) <= "0000" & inp;
17    G1: FOR i IN 1 TO 4 GENERATE
18       row(i) <= row(i-1)(6 DOWNTO 0) & '0';
19    END GENERATE;
20    outp <= row(sel);
21 END shifter;
22 ---------------------------------------------
```

row(0): 0 0 0 0 1 1 1 1

row(1): 0 0 0 1 1 1 1 0

row(2): 0 0 1 1 1 1 0 0

row(3): 0 1 1 1 1 0 0 0

row(4): 1 1 1 1 0 0 0 0

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Instrukcja BLOCK

- Występują dwa rodzaje instrukcji BLOCK:
  - prosty
  - chroniony
- Prosta instrukcja BLOCK to metoda na lokalny podział kodu
- Pozwala na powiązanie zestawu instrukcji współbieżnych w jeden blok, zwiększając czytelność kodu.

```
label: BLOCK
   [declarative part]
BEGIN
   (concurrent statements)
END BLOCK label;
```

# BLOCK - przykłady

```
------------------------
ARCHITECTURE example ...
BEGIN
   ...
   block1: BLOCK
   BEGIN
      ...
   END BLOCK block1
   ...
   block2: BLOCK
   BEGIN
      ...
   END BLOCK block2;
   ...
END example;
------------------------
b1: BLOCK
   SIGNAL a: STD_LOGIC;
BEGIN
   a <= input_sig WHEN ena='1' ELSE 'Z';
END BLOCK b1;
```

- Bloki można zagnieżdżać:

```
label1: BLOCK
   [declarative part of top block]
BEGIN
   [concurrent statements of top block]
   label2: BLOCK
      [declarative part nested block]
   BEGIN
      (concurrent statements of nested block)
   END BLOCK label2;
   [more concurrent statements of top block]
END BLOCK label1;
```

# BLOCK chroniony

- Chroniona instrukcja BLOCK zawiera dodatkowy warunek. Instrukcje wewnątrz bloku poprzedzone słowem kluczowym GUARDED są wykonywane tylko wtedy, gdy warunek ten jest spełniony

```
label: BLOCK (guard expression)
   [declarative part]
BEGIN
   (concurrent guarded and unguarded statements)
END BLOCK label;
```

# Przykład - latch zaimplementowany jako blok chroniony

```
1  -------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -------------------------------
5  ENTITY latch IS
6     PORT (d, clk: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END latch;
9  -------------------------------
10 ARCHITECTURE latch OF latch IS
11 BEGIN
12    b1: BLOCK (clk='1')
13    BEGIN
14       q <= GUARDED d;
15    END BLOCK b1;
16 END latch;
17 -------------------------------
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

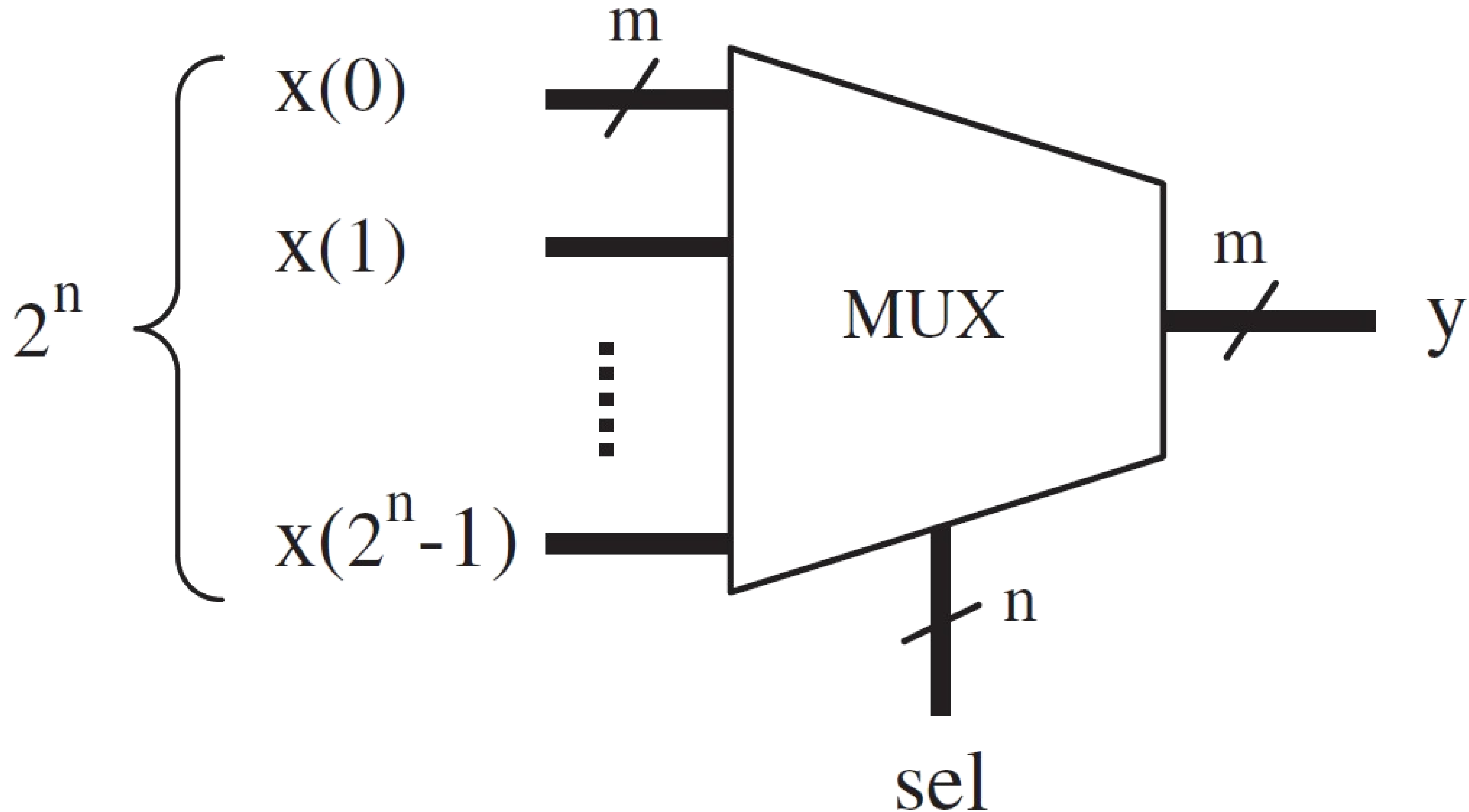
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Przykład - flip-flop zaimplementowany jako blok chroniony

```
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 dff OF dff IS
11 BEGIN
12    b1: BLOCK (clk'EVENT AND clk='1')
13    BEGIN
14       q <= GUARDED '0' WHEN rst='1' ELSE d;
15    END BLOCK b1;
16 END dff;
17 ------------------------------
```

# Problem

- Zaimplementować uogólniony multiplekser



# „Układy reprogramowalne i SoC”
# „Język VHDL (część 2)”

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