KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„Układy reprogramowalne i SoC”
„Język VHDL (część 4)”

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

Stałe

- Stałe są definiowane przy pomocy słowa kluczowego CONSTANT.

```
CONSTANT name : type := value
```

```
CONSTANT set_bit : BIT := '1';
CONSTANT datamemory : memory := (('0','0','0','0'),
                                 ('0','0','0','1'),
                                 ('0','0','1','1'));
```

- Stała może być zadeklarowana wewnątrz PACKAGE, ENTITY lub ARCHITECTURE
  - Zadeklarowana w pakiecie jest naprawdę globalna, gdyż pakiet może byc uzywany przez wiele komponentów
  - Zadeklarowana w ENTITY (po deklaracji portów) jest globalna dla wszystkich architektur
  - Zadeklarowana w ARCHITECTURE jest dostępna tylko w kodzie tej architektury

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Sygnały

- SIGNAL służy do przekazywania wartości z i do obwodu a także pomiędzy jego elementami
  - Reprezentuje przewody
- Wszystkie porty są również traktowane jako sygnały

```
SIGNAL control: BIT := '0';
SIGNAL count: INTEGER RANGE 0 TO 100;
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
```

- Ważną cechą sygnału, użytego wewnątrz sekwencyjnej sekcji kodu jest to, że nie jest on uaktualniany natychmiast, ale dopiero po zakończeniu bieżącej aktywacji procesu
- Przypisania do sygnału wykonuje się przy pomocy operatora "<="
  - Wielokrotne przypisania do sygnału są dozwolone tylko wewnątrz jednego procesu
    - Brane jest wówczas pod uwagę ostatnie przypisanie

Przykład: zliczanie jedynek

```
1  ------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY count_ones IS
6     PORT ( din: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            ones: OUT INTEGER RANGE 0 TO 8);
8  END count_ones;
9 ------------------------------------
10 ARCHITECTURE not_ok OF count_ones IS
11    SIGNAL temp: INTEGER RANGE 0 TO 8;
12 BEGIN
13    PROCESS (din)
14    BEGIN
15       temp <= 0;
16       FOR i IN 0 TO 7 LOOP
17          IF (din(i)='1') THEN
18             temp <= temp + 1;
19          END IF;
20       END LOOP;
21       ones <= temp;
22    END PROCESS;
23 END not_ok;
24 ------------------------------------
```

- Przykład nie zadziała
- Występuje wielokrotne przypisanie do sygnału temp
- W celu osiagnięcia oczekiwanego efektu należy użyć zmiennej

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

VARIABLE

- Zmienne reprezentują jedynie lokalną informację
- Mogą być zadeklarowane i użyte tylko wewnątrz procesu, funkcji lub procedury
- Uaktualnienie wartości zmiennej jest natychmiastowe - można korzystać z nowej wartości zaraz po przypisaniu

```
VARIABLE name : type [range] [:= init_value];
```

```
VARIABLE control: BIT := '0';
VARIABLE count: INTEGER RANGE 0 TO 100;
VARIABLE y: STD_LOGIC_VECTOR (7 DOWNTO 0) := "10001000";
```

Przykład: zliczanie jedynek

```
1  ----------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------
5  ENTITY count_ones IS
6     PORT ( din: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            ones: OUT INTEGER RANGE 0 TO 8);
8  END count_ones;
9 ----------------------------------
10 ARCHITECTURE ok OF count_ones IS
11 BEGIN
12    PROCESS (din)
13       VARIABLE temp: INTEGER RANGE 0 TO 8;
14    BEGIN
15       temp := 0;
16       FOR i IN 0 TO 7 LOOP
17          IF (din(i)='1') THEN
18             temp := temp + 1;
19          END IF;
20       END LOOP;
21       ones <= temp;
22    END PROCESS;
23 END ok;
24 ----------------------------------
```

Sygnał a zmienna

	SIGNAL	VARIABLE
Zastosowanie	Reprezentuje połączenia w obwodzie	Reprezentuje lokalną informację
Zasięg	Globalny	Lokalny (widoczny wewnątrz procesu, procedury i funkcji
Zachowanie	Uaktualnienie nie jest natychmiastowe w kodzie sekwencyjnym	Uaktualniana natychmiast
Użycie	W PACKAGE, ENTITY lub ARCHITECTURE	Tylko w kodzie sekwencyjnym: procesie, funkcji lub procedurze

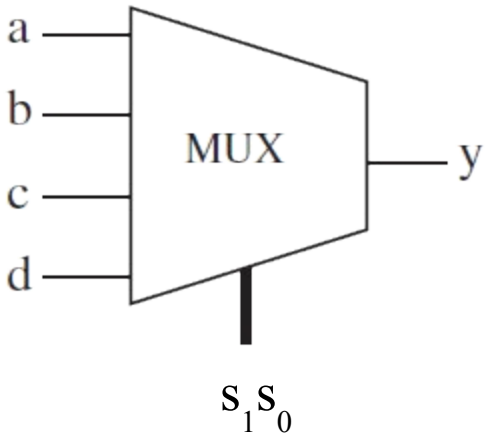
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

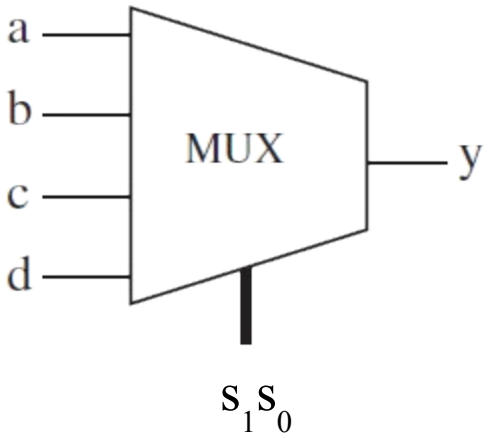
Przykład: multiplekser

```
1  -- Solution 1: using a SIGNAL (not ok) --
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 not_ok OF mux IS
11    SIGNAL sel : INTEGER RANGE 0 TO 3;
12 BEGIN
13    PROCESS (a, b, c, d, s0, s1)
14    BEGIN
15       sel <= 0;
16       IF (s0='1') THEN sel <= sel + 1;
17       END IF;
18       IF (s1='1') THEN sel <= sel + 2;
19       END IF;
20       CASE sel IS
21          WHEN 0 => y<=a;
22          WHEN 1 => y<=b;
23          WHEN 2 => y<=c;
24          WHEN 3 => y<=d;
25       END CASE;
26    END PROCESS;
27 END not_ok;
28 ---------------------------------------
```



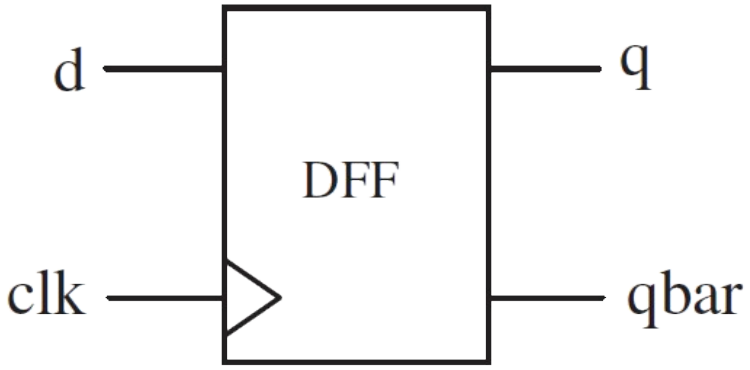
Przykład: multiplekser

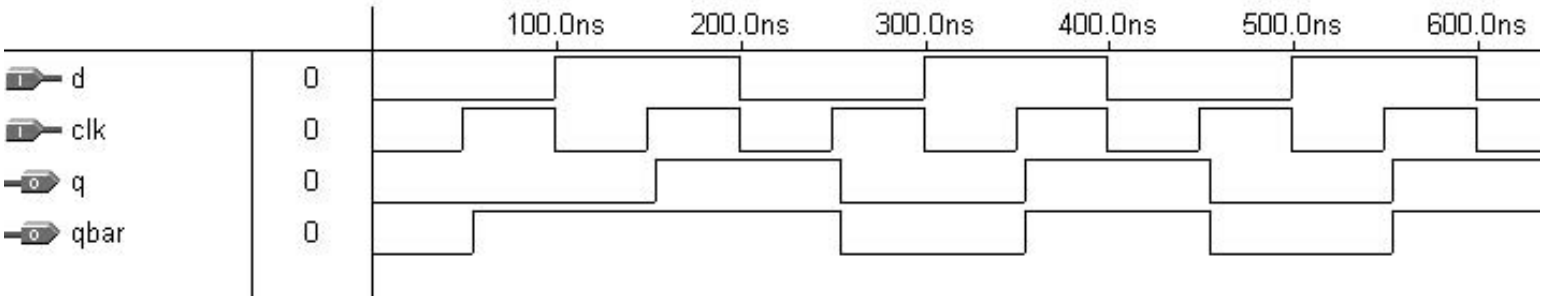
```
1  -- Solution 2: using a VARIABLE (ok) ----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY mux IS
6     PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
7            y: OUT STD_LOGIC);
8  END mux;
9  ---------------------------------------
10 ARCHITECTURE ok OF mux IS
11 BEGIN
12    PROCESS (a, b, c, d, s0, s1)
13       VARIABLE sel : INTEGER RANGE 0 TO 3;
14    BEGIN
15       sel := 0;
16       IF (s0='1') THEN sel := sel + 1;
17       END IF;
18       IF (s1='1') THEN sel := sel + 2;
19       END IF;
20       CASE sel IS
21          WHEN 0 => y<=a;
22          WHEN 1 => y<=b;
23          WHEN 2 => y<=c;
24          WHEN 3 => y<=d;
25       END CASE;
26    END PROCESS;
27 END ok;
28 ---------------------------------------
```



Przykład: przerzutnik z wyjściem prostym i zanegowanym

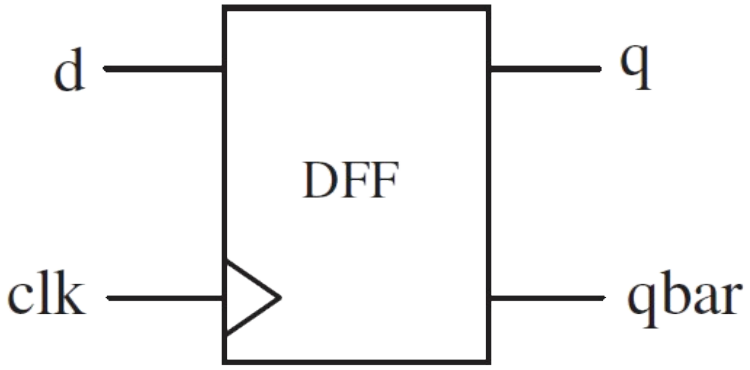
```
1  ---- Solution 1: not OK ---------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk: IN STD_LOGIC;
7            q: BUFFER STD_LOGIC;
8            qbar: OUT STD_LOGIC);
9  END dff;
10 ---------------------------------------
11 ARCHITECTURE not_ok OF dff IS
12 BEGIN
13    PROCESS (clk)
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          q <= d;
17          qbar <= NOT q;
18       END IF;
19    END PROCESS;
20 END not_ok;
21 ---------------------------------------
```

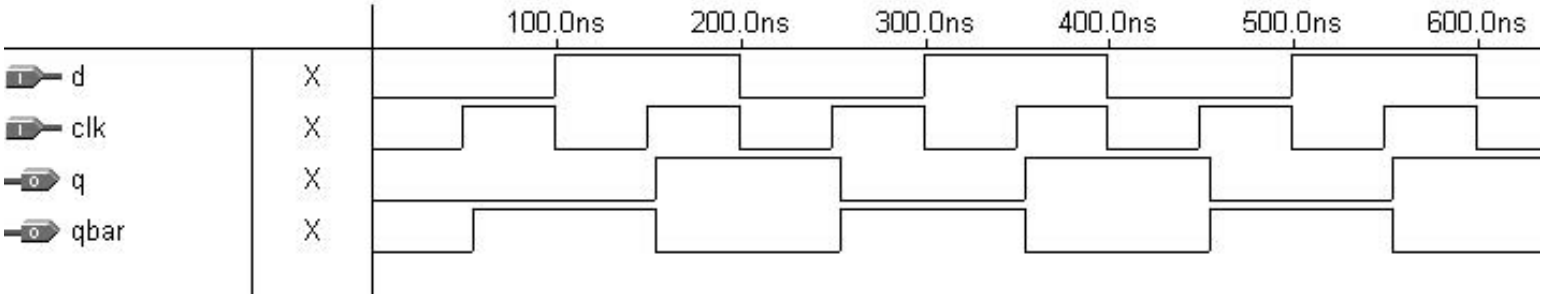




Przykład: przerzutnik z wyjściem prostym i zanegowanym

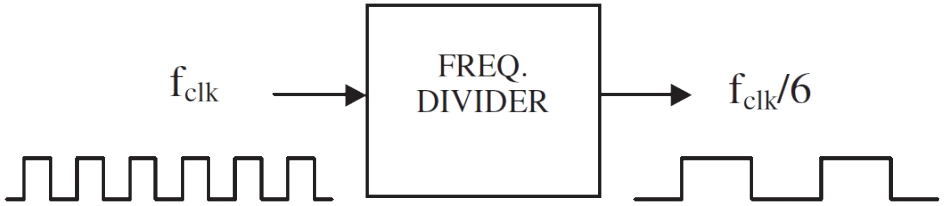
```
1  ---- Solution 2: OK ------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk: IN STD_LOGIC;
7            q: BUFFER STD_LOGIC;
8            qbar: OUT STD_LOGIC);
9  END dff;
10 ---------------------------------------
11 ARCHITECTURE ok OF dff IS
12 BEGIN
13    PROCESS (clk)
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          q <= d;
17       END IF;
18    END PROCESS;
19    qbar <= NOT q;
20 END ok;
21 ---------------------------------------
```

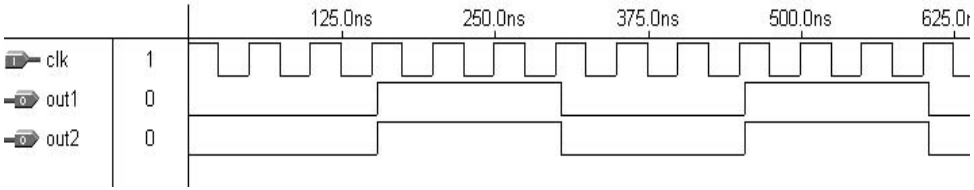




Przykład: dzielnik częstotliwości

```
1  ---------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY freq_divider IS
6     PORT ( clk : IN STD_LOGIC;
7            out1, out2 : BUFFER STD_LOGIC);
8  END freq_divider;
9  ---------------------------------------
10 ARCHITECTURE example OF freq_divider IS
11    SIGNAL count1 : INTEGER RANGE 0 TO 7;
12 BEGIN
13    PROCESS (clk)
14       VARIABLE count2 : INTEGER RANGE 0 TO 7;
15    BEGIN
16       IF (clk'EVENT AND clk='1') THEN
17          count1 <= count1 + 1;
18          count2 := count2 + 1;
19          IF (count1 = ? ) THEN
20             out1 <= NOT out1;
21             count1 <= 0;
22          END IF;
23          IF (count2 = ? ) THEN
24             out2 <= NOT out2;
25             count2 := 0;
26        END IF;
27       END IF;
28    END PROCESS;
29 END example;
30 ---------------------------------------
```





- Co wpisać zamiast '?'

Sygnały i zmienne a przerzutniki

- Sygnał generuje przerzutnik kiedy wykonywane jest do niego przypisanie na zboczu innego sygnału - podczas synchronicznego przypisania.
- Może to nastąpić wewnątrz procesu, funkcji lub procedury
- Zmienna nie generuje przerzutnika, jeżeli jej wartość nie opuszcza procesu.
  - Jeżeli na zboczu jakegoś sygnału do zmiennej jest przypisywana wartość, a następnie zmienna jest przypisana do sygnału, przerzutnik może zostać wygenerowany.
- Zmienna wygeneruje przerzutnik, gdy jest użyta przed przypisaniem do niej wartości

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Przykłady

```
-- output1 and output2 will both be stored
-- (that is, infer flip-flops)
PROCESS (clk)
BEGIN
   IF (clk'EVENT AND clk='1') THEN
      output1 <= temp; -- output1 stored
      output2 <= a; -- output2 stored
   END IF;
END PROCESS;

-- only output1 will be stored
-- (output2 will make use of logic gates)
PROCESS (clk)
BEGIN
   IF (clk'EVENT AND clk='1') THEN
       output1 <= temp; -- output1 stored
   END IF;
   output2 <= a; -- output2 not stored
END PROCESS;
```

```
-- temp (a variable) will cause x
-- (a signal) to be stored.
PROCESS (clk)
   VARIABLE temp: BIT;
BEGIN
   IF (clk'EVENT AND clk='1') THEN
       temp := a;
   END IF;
   x <= temp; -- temp causes x to be stored
END PROCESS;
```

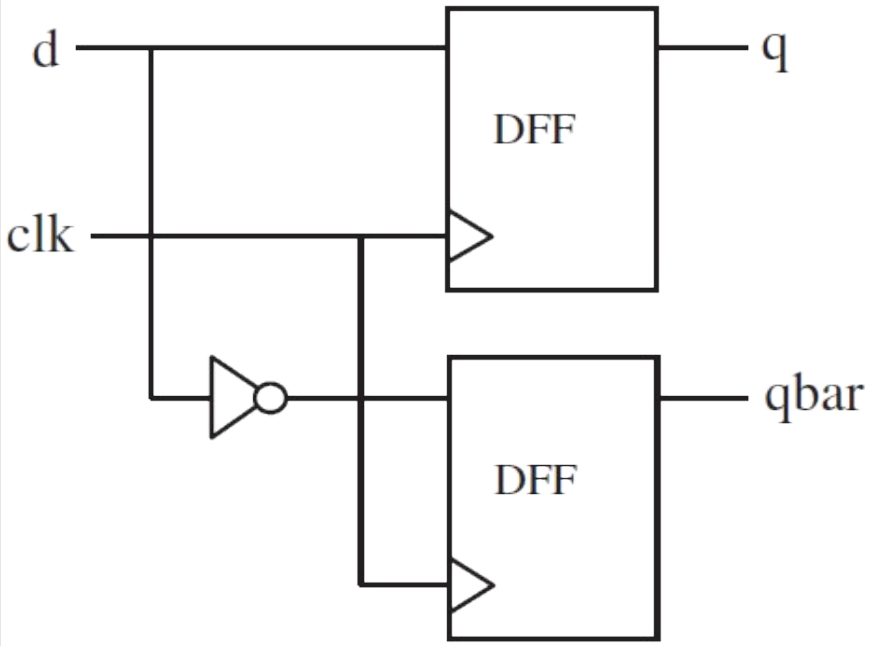
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

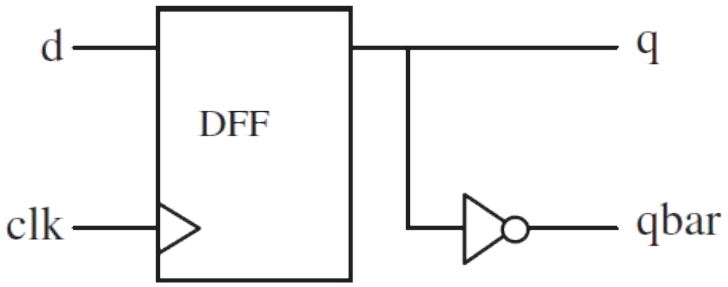
Przerzutnik z wyjściem prostym i zanegowanym

```
1  ---- Solution 1: Two DFFs ---------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk: IN STD_LOGIC;
7            q: BUFFER STD_LOGIC;
8            qbar: OUT STD_LOGIC);
9  END dff;
10 ---------------------------------------
11 ARCHITECTURE two_dff OF dff IS
12 BEGIN
13    PROCESS (clk)
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          q <= d; -- generates a register
17          qbar <= NOT d; -- generates a register
18       END IF;
19    END PROCESS;
20 END two_dff;
21 ---------------------------------------
```



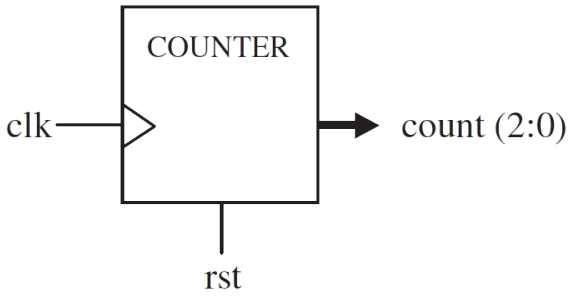
Przerzutnik z wyjściem prostym i zanegowanym

```
1 ---- Solution 2: One DFF ---------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6    PORT ( d, clk: IN STD_LOGIC;
7           q: BUFFER STD_LOGIC;
8           qbar: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------
11 ARCHITECTURE one_dff OF dff IS
12 BEGIN
13    PROCESS (clk)
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          q <= d; -- generates a register
17       END IF;
18    END PROCESS;
19    qbar <= NOT q; -- uses logic gate (no register)
20 END one_dff;
21 ---------------------------------------
```



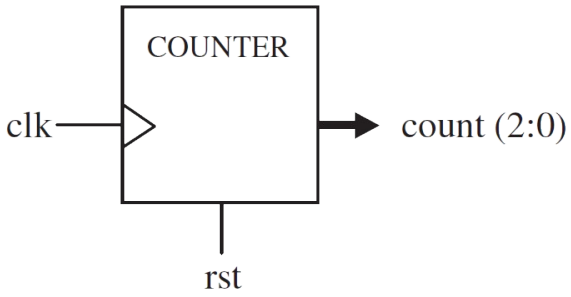
Przykład: licznik

```
1  ------ Solution 1: With a VARIABLE --------
2  ENTITY counter IS
3     PORT ( clk, rst: IN BIT;
4            count: OUT INTEGER RANGE 0 TO 7);
5  END counter;
6  ----------------------------------------
7  ARCHITECTURE counter OF counter IS
8  BEGIN
9     PROCESS (clk, rst)
10       VARIABLE temp: INTEGER RANGE 0 TO 7;
11    BEGIN
12       IF (rst='1') THEN
13          temp:=0;
14       ELSIF (clk'EVENT AND clk='1') THEN
15          temp := temp+1;
16       END IF;
17       count <= temp;
18    END PROCESS;
19 END counter;
20 ----------------------------------------
```



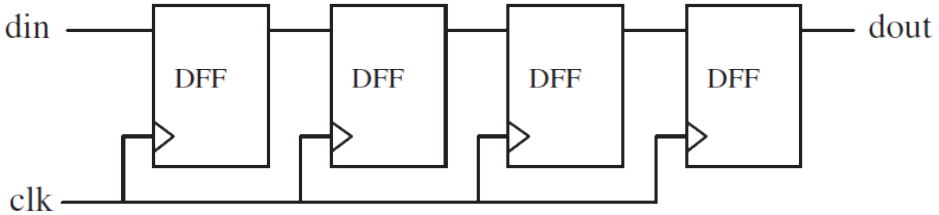
Przykład: licznik

```
1  ------ Solution 2: With SIGNALS only -------
2  ENTITY counter IS
3     PORT ( clk, rst: IN BIT;
4            count: BUFFER INTEGER RANGE 0 TO 7);
5  END counter;
6  ----------------------------------------
7  ARCHITECTURE counter OF counter IS
8  BEGIN
9     PROCESS (clk, rst)
10       BEGIN
11          IF (rst='1') THEN
12             count <= 0;
13          ELSIF (clk'EVENT AND clk='1') THEN
14             count <= count + 1;
15       END IF;
16    END PROCESS;
17 END counter;
18 ----------------------------------------
```



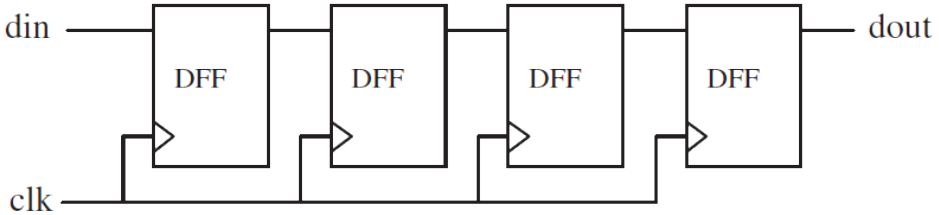
Przykład: rejestr przesuwny

```
1  -------- Solution 1: ----------------
2  ENTITY shift IS
3     PORT ( din, clk: IN BIT;
4            dout: OUT BIT);
5  END shift;
6  ------------------------------------
7  ARCHITECTURE shift OF shift IS
8  BEGIN
9     PROCESS (clk)
10       VARIABLE a, b, c: BIT;
11    BEGIN
12       IF (clk'EVENT AND clk='1') THEN
13          dout <= c;
14          c := b;
15          b := a;
16          a := din;
17       END IF;
18    END PROCESS;
19 END shift;
20 ------------------------------------
```



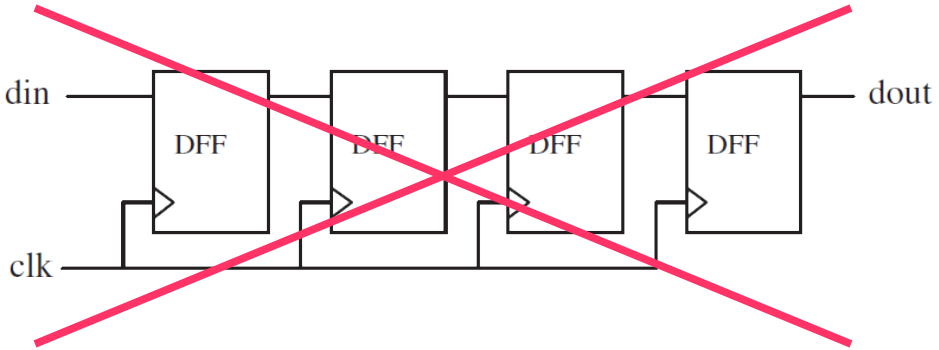
Przykład: rejestr przesuwny

```
1  -------- Solution 2: ----------------
2  ENTITY shift IS
3     PORT ( din, clk: IN BIT;
4            dout: OUT BIT);
5  END shift;
6  ------------------------------------
7  ARCHITECTURE shift OF shift IS
8     SIGNAL a, b, c: BIT;
9  BEGIN
10    PROCESS (clk)
11    BEGIN
12       IF (clk'EVENT AND clk='1') THEN
13          a <= din;
14          b <= a;
15          c <= b;
16          dout <= c;
17       END IF;
18    END PROCESS;
19 END shift;
20 ------------------------------------
```



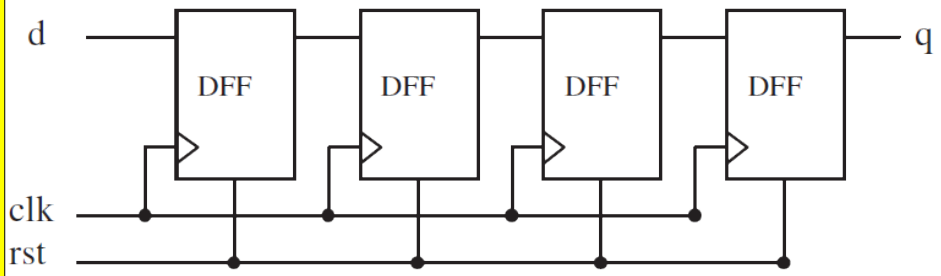
Przykład: rejestr przesuwny

```
1  -------- Solution 3: ----------------
2  ENTITY shift IS
3     PORT ( din, clk: IN BIT;
4            dout: OUT BIT);
5  END shift;
6  ------------------------------------
7  ARCHITECTURE shift OF shift IS
8  BEGIN
9     PROCESS (clk)
10       VARIABLE a, b, c: BIT;
11    BEGIN
12       IF (clk'EVENT AND clk='1') THEN
13          a := din;
14          b := a;
15          c := b;
16          dout <= c;
17       END IF;
18    END PROCESS;
19 END shift;
20 ------------------------------------
```



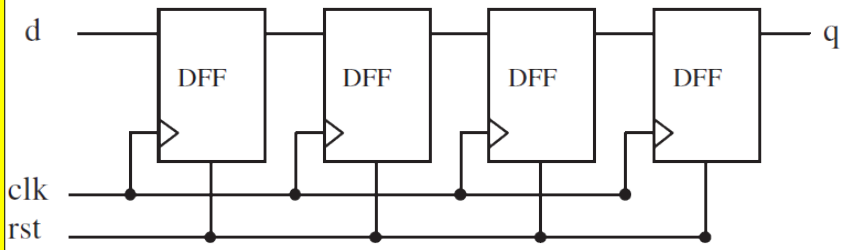
Przykład: rejestr przesuwny z resetem

```
1  ---- Solution 1: With an internal SIGNAL ---
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY shiftreg IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END shiftreg;
9  ----------------------------------------
10 ARCHITECTURE behavior OF shiftreg IS
11    SIGNAL internal: STD_LOGIC_VECTOR (3 DOWNTO 0);
12 BEGIN
13    PROCESS (clk, rst)
14    BEGIN
15       IF (rst='1') THEN
16          internal <= (OTHERS => '0');
17       ELSIF (clk'EVENT AND clk='1') THEN
18          internal <= d & internal(3 DOWNTO 1);
19       END IF;
20    END PROCESS;
21    q <= internal(0);
22 END behavior;
23 ----------------------------------------
```

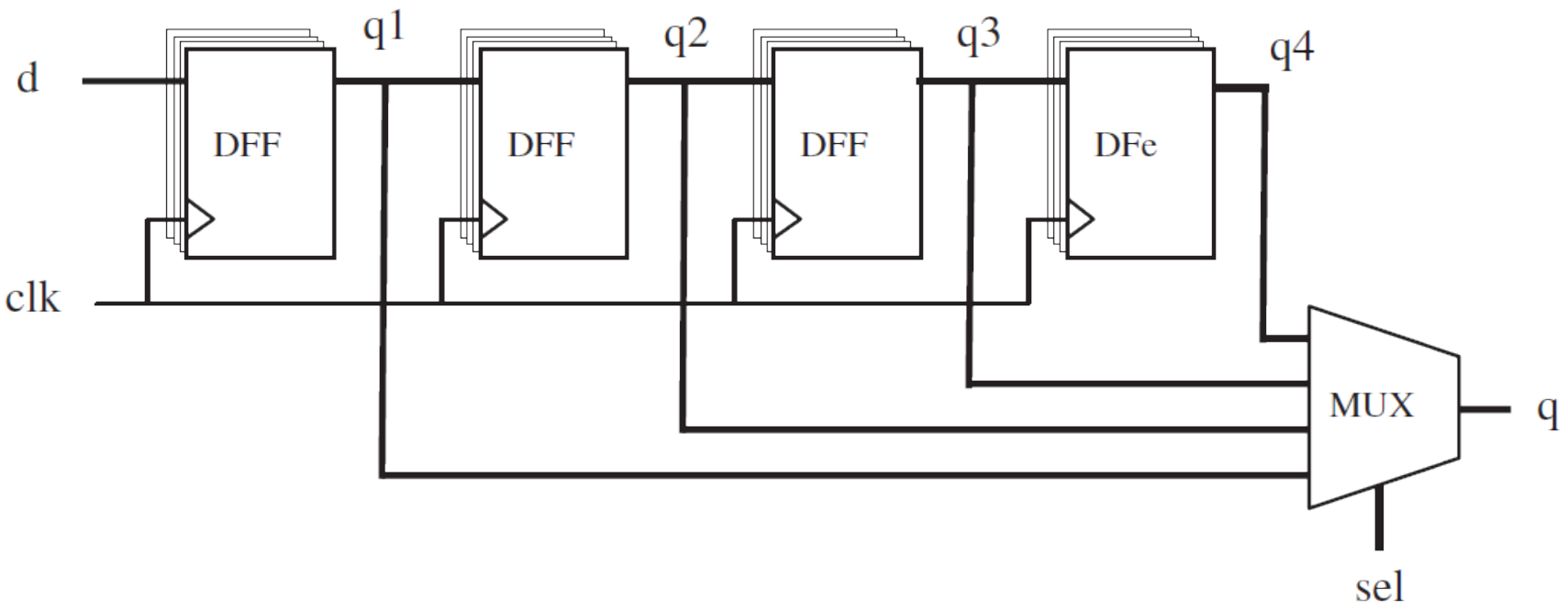


Przykład: rejestr przesuwny z resetem

```
1  -- Solution 2: With an internal VARIABLE ---
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY shiftreg IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END shiftreg;
9  ----------------------------------------
10 ARCHITECTURE behavior OF shiftreg IS
11 BEGIN
12    PROCESS (clk, rst)
13       VARIABLE internal: STD_LOGIC_VECTOR (3 DOWNTO 0);
14    BEGIN
15       IF (rst='1') THEN
16          internal := (OTHERS => '0');
17       ELSIF (clk'EVENT AND clk='1') THEN
18          internal := d & internal(3 DOWNTO 1);
19       END IF;
20       q <= internal(0);
21    END PROCESS;
22 END behavior;
23 ----------------------------------------
```



Problem: programowalny układ opóźniający



Który program działa poprawnie?

```
-------------------------------------
ENTITY dff IS
   PORT ( d, clk: IN BIT;
          q: BUFFER BIT;
          qbar: OUT BIT);
END dff;
-------- Solution 1 -------------------
ARCHITECTURE arch1 OF dff IS
BEGIN
   PROCESS (clk)
      VARIABLE temp: BIT;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := d;
         q <= temp;
         qbar <= NOT temp;
      END IF;
   END PROCESS;
END arch1;
```

```
-------- Solution 2 -------------------
ARCHITECTURE arch2 OF dff IS
BEGIN
   PROCESS (clk)
      VARIABLE temp: BIT;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := d;
         q <= temp;
         qbar <= NOT q;
      END IF;
   END PROCESS;
END arch2;
```

```
-------- Solution 3 -------------------
ARCHITECTURE arch3 OF dff IS
BEGIN
   PROCESS (clk)
      VARIABLE temp: BIT;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := d;
         q <= temp;
      END IF;
   END PROCESS;
   qbar <= NOT q;
END arch3;
-------------------------------------
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„Układy reprogramowalne i SoC”
„Język VHDL (część 4)”

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