VHDL dla układów programowalnych

Zamiast „tworzyć” schemat układu cyfrowego, jego działanie można opisać przy użyciu tzw. języków opisu sprzętu HDL

(*Hardware Descripsion Language*)

Stanowi to „wsad” dla *syntezy komputerowej*...

Język VHDL może być stosowany do projektowania układów cyfrowych realizowanych w programowalnych układach scalonych typu CPLD i FPGA...

Literatura nieobowiązkowa...

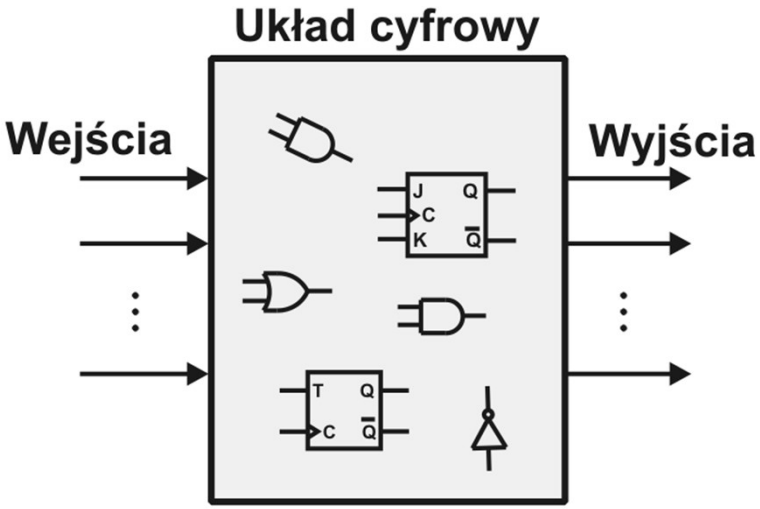
www.link.springer.com

- **Brock J. LaMeres**
  Introduction to Logic Circuits & Logic Design with VHDL
- **Brock J. LaMeres**
  Quick Start Guide to VHDL
- **Giuliano Donzellini, Luca Oneto, Domenico Ponta, Davide Anguita**
  Introduction to Digital Systems Design

- **Eduardo Augusto Bezerra, Djones Vinicius Lettnin**
  Synthesizable VHDL Design for FPGAs
- **Aiken Pang, Peter Membrey**
  Beginning FPGA: Programming Metal

VHDL – plik tekstowy z rozszerzeniem .vhd !!!

*.vhd



UklCyfr.vhd

```
entity UklCyfr is
port (
                    WEJŚCIA I WYJŚCIA

                );
end UklCyfr;


architecture Beh of UklCyfr is
begin

                    SPOSÓB DZIAŁANIA


end Beh;
```

VHDL – rodzaje portów, wynikają z budowy bloków I/O układów programowalnych

Porty wejściowe

in

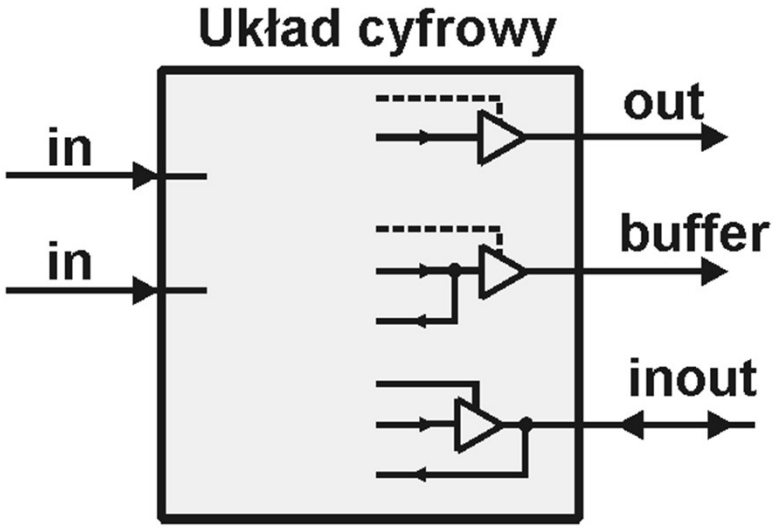
Porty wyjściowe

out

buffer

Porty wej-wyj

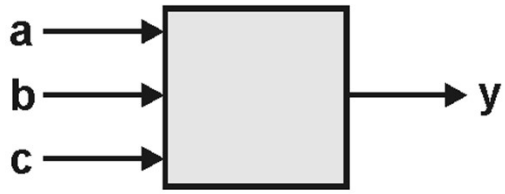
inout



std_logic

Oznacza, że pojedynczy sygnał cyfrowy może przyjąć jedną z wartości: ’U’, ’X’, ’0’, ’1’, ’Z’, ’W’, ’L’, ’H’, ’–’

```
port (   a :     in   std_logic;
         b, c :  in   std_logic;
         y :     out  std_logic   );
```



VHDL – typ wielobitowych sygnałów cyfrowych (np. magistrala)

std_logic_vector(… downto …)

std_logic_vector(… to …)

Oznacza, że sygnał cyfrowy jest wielobitowy oraz każdy jego bit może przyjąć jedną z wartości: ’U’, ’X’, ’0’, ’1’, ’Z’, ’W’, ’L’, ’H’, ’–’

```
port (   a :      in   std_logic_vector( 7 downto 0 );
         b, c :   in   std_logic_vector( 0 to 1 );
         y :      out  std_logic_vector( 31 downto 27 )  );
```

---

odwołanie do pojedynczego bitu: `y(31)`

odwołanie do kilku bitów: `a(7 downto 5)`

UWAGA!

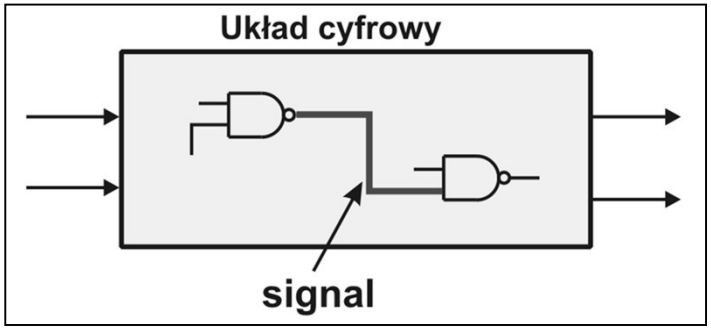
Stosowanie typów std_logic i std_logic_vector wymaga zadeklarowania użycia biblioteki zawierającej definicję tego typu.

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

entity UklCyfr is
port (     …
                    );
end UklCyfr;

architecture Beh of UklCyfr is
begin
            …

end Beh;
```

signal

Umożliwia opisanie połączenia pomiędzy dowolnymi cyfrowymi elementami wewnątrz części architecture kodu VHDL.

```
architecture Beh of UklCyfr is
   signal x : std_logic;
   signal z : std_logic_vector(3 downto 0);
begin

    x <=   ...   ;
    z <=   ...   ;

end Beh;
```



Operacje logiczne dokonywane są na pojedynczych bitach:

not and or nand nor xor

Można stosować nawiasy do ustalenia kolejności działań:

```
y <= a xor not ( b and c );
```

Operacje arytmetyczne dokonywane są na typach wektorowych:

+ – / * mod rem

Operacje relacji – wynik działania jest typu boolean

= /= < <= > >=

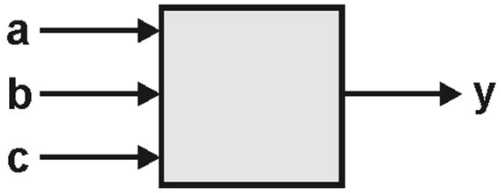
Działanie układu cyfrowego opisuje się w części architecture za pomocą instrukcji:

- ***WSPÓŁBIEŻNYCH*** **(przypisania do sygnału, process, wywołanie procedury, port map, generate, block, ...)**

- ***SEKWENCYJNYCH*** **(przypisania do sygnału w obrębie procesu, przypisanie do zmiennej, pętle loop, ...)**

VHDL – współbieżne przypisanie do sygnału

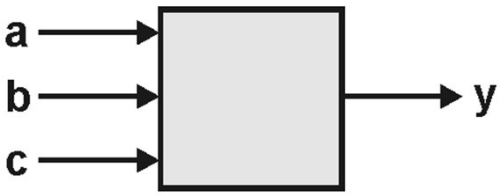
<=



$$y = f(a,b,c) = \bar{b}\bar{c} + \bar{a}b + abc$$

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

entity UklCyfr is
    port (  a, b, c : in std_logic;   y : out std_logic   );
end UklCyfr;

architecture Beh1 of UklCyfr is
begin

  y  <=   (not b and not c) or (not a and not b) or (a and b and c);

end Beh1;
```

VHDL – współbieżne przypisanie warunkowe when-else

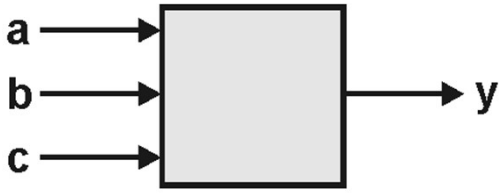


Tablica prawdy

abc	y
000	1
001	1
010	0
011	0
100	1
101	0
110	0
111	1

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

entity UklCyfr is
  port (  abc : in std_logic_vector(2 downto 0);
          y : out std_logic  );
  end UklCyfr;

architecture Beh2 of UklCyfr is
begin

 y  <=  '1' when abc = "000" else
        '1' when abc = "001" else
        '0' when abc = "010" else
        '0' when abc = "011" else
        '1' when abc = "100" else
        '0' when abc = "101" else
        '0' when abc = "110" else
        '1';

end Beh2;
```

VHDL – współbieżne przypisanie selektywne with-select



Tablica prawdy

abc	y
000	1
001	1
010	0
011	0
100	1
101	0
110	0
111	1

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

entity UklCyfr is
  port (  abc : in std_logic_vector(2 downto 0);
          y : out std_logic    );
  end UklCyfr;

architecture Beh3 of UklCyfr is
begin

 with abc select
   y  <= '1' when "000",
         '1' when "001",
         '0' when "010",
         '0' when "011",
         '1' when "100",
         '0' when "101",
         '0' when "110",
         '1' when others;

end Beh3;
```

VHDL – współbieżna instrukcja process, zawiera instrukcje interpretowane jako sekwencyjne działanie układu cyfrowego

```
process( … )
…

begin
…


end process;
```

– w nawiasie lista wrażliwości

– deklaracje typów, stałych i zmiennych

– wewnątrz procesu instrukcje wykonywane (interpretowane!) są sekwencyjnie !!!

lista wrażliwości – zawiera nazwy sygnałów, zmiana stanu dowolnego sygnału powoduje „rozpoczęcie wykonywania” procesu:

process(a, clk, q(3))

VHDL – współbieżne instrukcja process

```
architecture Beh of UklCyfr is
begin

asd:    process( … )
        begin
        …
        end process;

wpis:   process( … )
        begin
        …
        end process;

odczyt: process( … )
        begin
        …
        end process;

end Beh;
```

Dla opisu sekwencyjnego działania układu lub jego części...

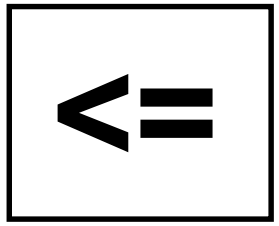
Pomiędzy sobą procesy są współbieżne

Wewnątrz procesów instrukcje „wykonywane” są po kolei

Przekazywanie „wyników” pomiędzy procesami za pomocą signal, po wykonaniu się procesu

Procesy mogą mieć etykiety

VHDL – sekwencyjna instrukcja przypisania do sygnału




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

entity UklCyfr is
  port (  a, b, c : in std_logic;
          y0, y1 : out std_logic    );
end UklCyfr;

architecture Beh4 of UklCyfr is
begin

    process( a, b, c )
    begin
        y0 <= not b;
        y1 <=  a nor c;
    end process;

end Beh4;
```

VHDL – sekwencyjna instrukcja przypisania warunkowego if

```
...

process( abc )
begin
        if     abc = ”000” then y <= ’1’ ;
        elsif abc = ”001” then y <= ’1’ ;
        elsif abc = ”010” then y <= ’0’ ;
        elsif abc = ”011” then y <= ’0’ ;
        elsif abc = ”100” then y <= ’1’ ;
        elsif abc = ”101” then y <= ’0’ ;
        elsif abc = ”110” then y <= ’0’ ;
        else y <= ’1’ ;
        end if ;

end process;

...
```

abc	y
000	1
001	1
010	0
011	0
100	1
101	0
110	0
111	1

VHDL – sekwencyjna instrukcja przypisania selektywnego case

```
...

process( abc )
begin

        case abc is
              when ”000” => y <= ’1’ ;
              when ”001” => y <= ’1’ ;
              when ”010” => y <= ’0’ ;
              when ”011” => y <= ’0’ ;
              when ”100” => y <= ’1’ ;
              when ”101” => y <= ’0’ ;
              when ”110” => y <= ’0’ ;
              when others => y <= ’1’;
        end case;

end process;

...
```

abc	y
000	1
001	1
010	0
011	0
100	1
101	0
110	0
111	1

BLOKI KOMBINACYJNE

Multiplekser

Demultiplekser

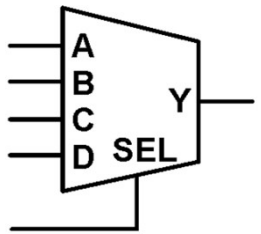
Konwerter kodu

Sumator

Komparator

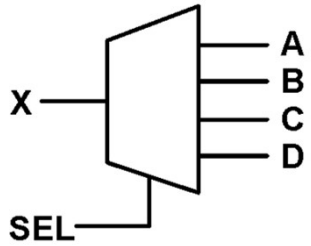
VHDL – multiplekser

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

entity MUX is
  port (    A, B, C, D : in std_logic;
            SEL : in std_logic_vector(1 downto 0);
            Y : out std_logic    );
end MUX;

architecture Bech of MUX is
begin

    Y <=  A when SEL = ”00” else
          B when SEL = ”01” else
          C when SEL = ”10” else
          D;

end Bech;
```



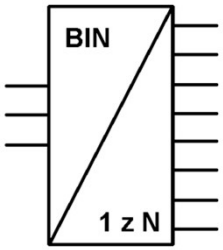
VHDL – demultiplekser

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

entity MUX is
  port (   X : in std_logic;
           SEL : in std_logic_vector(1 downto 0);
           A, B, C, D : out std_logic    );
end MUX;

architecture Bech of MUX is
begin

   A <=  X  when SEL = ”00”  else ’0’;
   B <=  X  when SEL = ”01”  else ’0’;
   C <=  X  when SEL = ”10”  else ’0’;
   D <=  X  when SEL = ”11”  else ’0’;

end Bech;
```



VHDL – konwerter kodu (binarnego na 1 z N)

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

entity KONW is
  port (    A : in   std_logic_vector (2 downto 0);
            Y : out std_logic_vector (7 downto 0)   );
end KONW;

architecture Bech of KONW is
begin
      process( A )
      begin
            case A is
              when ”000”  =>  Y <= ”00000001”;
              when ”001”  =>  Y <= ”00000010”;
              when ”010”  =>  Y <= ”00000100”;
              when ”011”  =>  Y <= ”00001000”;
              when ”100”  =>  Y <= ”00010000”;
              when ”101”  =>  Y <= ”00100000”;
              when ”110”  =>  Y <= ”01000000”;
              when others =>  Y <= ”10000000”;
            end case;
       end process;
end Bech;
```



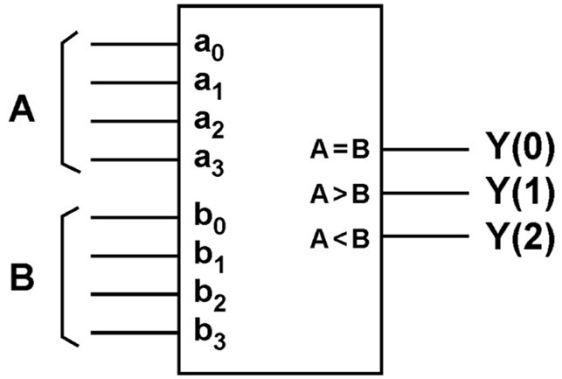
VHDL – sumator (dwóch liczb czterobitowych)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;     -- funkcje arytmetyczne dla typu std_logic_vector

entity SUM is
  port (    A, B : in std_logic_vector(3 downto 0);
            Y : out std_logic_vector(4 downto 0)      );
end SUM;

architecture Bech of SUM1 is
begin

    Y <= (’0’&A) + (’0’&B);

end Bech;
```

& – konkatenacja, czyli składanie pojedynczych bitów, wektorów lub sygnałów

VHDL – komparacja (dwóch liczb czterobitowych A i B)

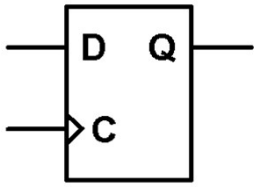
```
library ieee;
use ieee.std_logic_1164.all;

entity KOMP is
   port( A, B : in std_logic_vector(3 downto 0);  Y : out std_logic_vector(2 downto 0) );
end KOMP;

architecture Bech of KOMP is
begin

    Y(0) <= '1' when A = B else '0';

    Y(1) <= '1' when A > B else '0';

    Y(2) <= '1' when A < 12 else '0';

end Bech;
```



ELEMENTY PAMIĘCIOWE

Czyli przerzutniki i zatrzaski...

VHDL – synchroniczny przerzutnik D

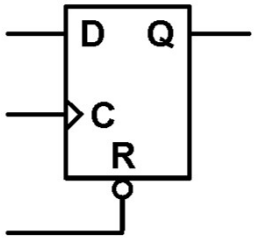



```
library ieee;
use ieee.std_logic_1164.all;

entity D_FF is
  port (   D, C : in std_logic;
           Q : out std_logic    );
  end D_FF;

architecture Bech of D_FF is
begin

    process( C )
    begin
       if rising_edge(C) then          -- detekcja zbocza narastającego

            Q <= D ;                   -- „nie ma” else !!!

       end if;
    end process;

end Bech;
```

zbocze opadające: falling_edge

VHDL – synchroniczny przerzutnik D z kasowaniem asynchronicznym

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

entity D_FF is
  port (   D, C, R : in std_logic;
           Q : out std_logic    );
  end D_FF;

architecture Bech of D_FF is
begin
    process( C, R )
    begin
      if R = '0' then
           Q <= '0';
      elsif rising_edge(C) then
           Q <= D;
      end if;
    end process;
end Bech;
```

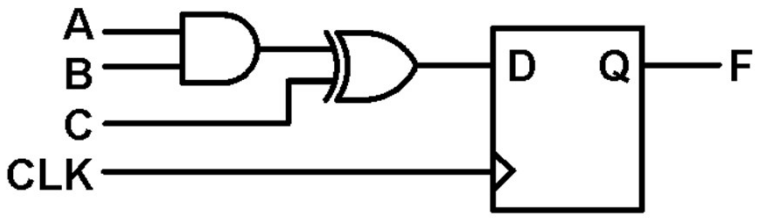


wszystkie przypisania *ASYNCHRONICZNE*

---

wszystkie przypisania *SYNCHRONICZNE*

VHDL – pewien układ synchroniczny...

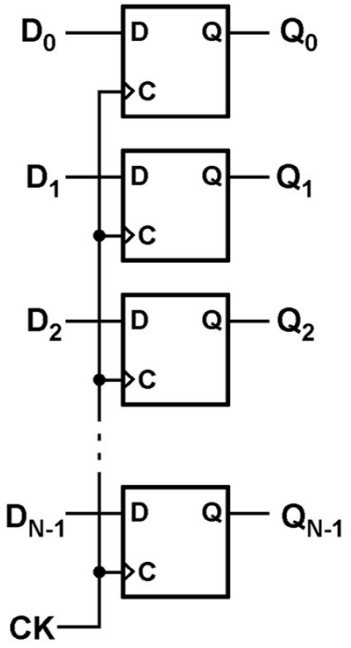



```
library ieee;
use ieee.std_logic_1164.all;

entity US is
  port (    A,B,C,CLK : in std_logic;
            F : out std_logic    );
  end US;

architecture Bech of US is
begin

    process( CLK )
    begin
        if rising_edge(CLK) then

            F <= ( A and B ) xor C;

        end if;
    end process;

end Bech;
```

REJESTRY

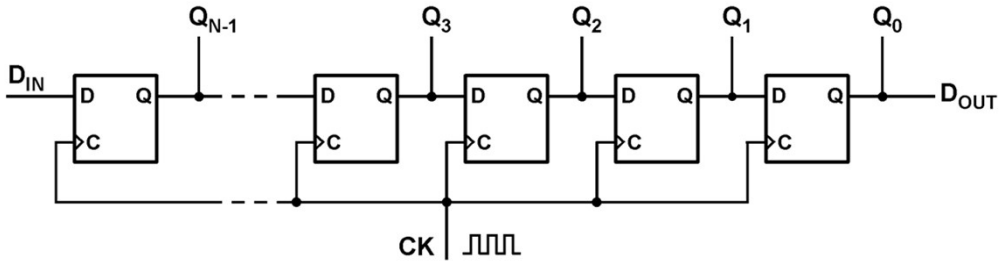
Czyli układy zbudowane z przerzutników połączonych równolegle lub szeregowo...

VHDL – rejestr równoległy




```
-- REJESTR 8-BITOWY

library ieee;
use ieee.std_logic_1164.all;

entity REG is
  port (   D : in std_logic_vector(7 downto 0);
           CK : in std_logic;
           Q : out std_logic_vector(7 downto 0)   );
end REG;

architecture Bech of REG is
begin

    process( CK )
    begin
      if rising_edge(CK) then Q <= D; end if;
    end process;

end Bech;
```

VHDL – rejestr przesuwający (szeregowo-równoległy)




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

entity SH_REG is
  port (    D : in std_logic;
            CK : in std_logic;
            Q : inout std_logic_vector(7 downto 0)   );
end SH_REG;

architecture Bech of SH_REG is
begin
    process( CK )
    begin
      if rising_edge(CK) then
           Q <= D&Q(7 downto 1);    -- przesuw w prawo
      end if;
    end process;
end Bech;
```

```
Q(7) <= D;
Q(6) <= Q(7);
Q(5) <= Q(6);
Q(4) <= Q(5);
Q(3) <= Q(4);
Q(2) <= Q(3);
Q(1) <= Q(2);
Q(0) <= Q(1);
```

```
Q <= Q(6 downto 0)&D;    -- przesuw w lewo
```

LICZNIKI

Czyli układy, których stan sygnału wyjściowego odpowiada liczbie wystąpień impulsów na wejściu (zazwyczaj zegarowym)...

VHDL – synchroniczny licznik binarny *modulo 16* , zliczający w górę

```
-- przerzutnik  D --
library ieee;
use ieee.std_logic_1164.all;

entity D_FF is
  port (     D : in std_logic;
             C : in std_logic;
             Q : out std_logic    );
  end D_FF;

architecture Bech of D_FF is
begin
    process( C )
    begin
      if rising_edge(C) then
             Q <= D;
      end if;
    end process;
end Bech;
```

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

entity CNT_mod16 is
  port (    C : in std_logic;
            Q : inout std_logic_vector( 3 downto 0 ) );
  end CNT_mod16 ;

architecture Bech of CNT_mod16 is
begin
    process( C )
    begin
      if rising_edge(C) then
            Q <= Q + 1;          -- DODAWANIE !!!
      end if;
    end process;
end Bech;
```

Licznik binarny „powstaje z przerzutnika D”

VHDL – synchroniczny licznik binarny modulo 16, zliczający w dół

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

entity CNT_mod16 is
  port (    C : in std_logic;
            Q : inout std_logic_vector( 3 downto 0 )   );
  end CNT_mod16 ;

architecture Bech of CNT_mod16 is
begin

    process( C )
    begin
      if rising_edge(C) then
            Q <= Q – 1;          -- ODEJMOWANIE !!!
      end if;
    end process;

end Bech;
```

VHDL – synchroniczny licznik binarny modulo 16, zliczający w górę, z kasowaniem asynchronicznym

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

entity CNT_mod16 is
  port (   R, C : in std_logic;
           Q : inout std_logic_vector( 3 downto 0 )   );
  end CNT_mod16 ;

architecture Bech of CNT_mod16 is
begin

    process( C, R )
    begin
      if R = '0' then Q <= "0000";          -- KASOWANIE ASYNCHRONICZNE
      elsif rising_edge(C) then Q <= Q + 1;
      end if;
    end process;

end Bech;
```

Zamiast	Q <= "0000";
można	Q <= (others => '0');

VHDL – synchroniczny licznik binarny modulo 16, dwukierunkowy

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

entity CNT_mod16 is
  port (    DIR, C : in std_logic;
            Q : inout std_logic_vector( 3 downto 0 )   );
  end CNT_mod16 ;

architecture Bech of CNT_mod16 is
begin
    process( C )
    begin
      if rising_edge(C) then
        if DIR = '0' then
          Q <= Q + 1;          -- DODAWANIE
        else
          Q <= Q – 1;          -- ODEJMOWANIE
        end if;
      end if;
    end process;
end Bech;
```

VHDL – synchroniczny licznik *modulo 10*, zliczający w górę

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

entity CNT_mod10 is
  port (    C : in std_logic;
            Q : inout std_logic_vector( 3 downto 0 )  );
  end CNT_mod10 ;

architecture Bech of CNT_mod10 is
begin
    process( C )
    begin
      if rising_edge(C) then
         if Q < 9 then
            Q <= Q + 1;          -- dodawanie dopóki Q mniejsze od 9
         else
            Q <= ”0000”;         -- zerowanie jeśli Q równe 9
         end if;
      end if;
    end process;
end Bech;
```

VHDL – synchroniczny licznik modulo 10, zliczający w dół (wstecz)

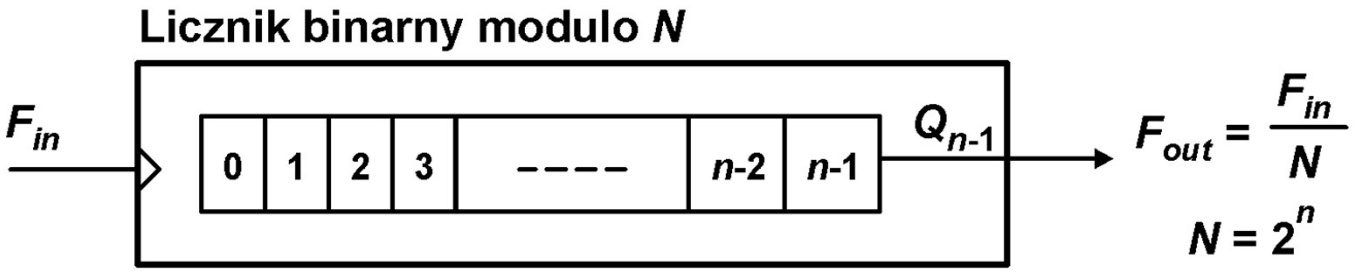
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CNT_mod10 is
  port (    C : in std_logic;
            Q : inout std_logic_vector(3 downto 0)  );
  end CNT_mod16 ;

architecture Bech of CNT_mod10 is
begin
    process( C )
    begin
      if rising_edge(C) then
         if Q > 0 then
             Q <= Q – 1;          -- odejmowanie dopóki Q większe od 0
         else
             Q <= ”1001”;         -- ustawianie 9 jeśli Q równe 0
         end if;
      end if;
    end process;
end Bech;
```

DZIELNIKI CZĘSTOTLIWOŚCI

Czyli układy, których sygnał wyjściowy ma kilkukrotnie mniejszą częstotliwość niż sygnał wejściowy (zazwyczaj zegarowy)...

Najprostszym dzielnikiem jest układ licznika binarnego, którego cykl liczenia jest równy wielokrotności liczby 2 (Q <= Q+1;).

Wówczas wyjściem jest najbardziej znaczący bit (MSB) a współczynnik wypełnienia jest zawsze równy ½.



Dla podziału będącego liczbą całkowitą (a nie binarną), można stosować licznik ze skróconym cyklem liczenia.

Należy ograniczyć liczbę stanów do wartości podziału *N*, czyli

```
if Q < N-1  then
   Q <= Q + 1;      -- dodawanie dopóki Q mniejsze od podziału
else
   Q <= (others => '0');
end if;
```

Wyjściem jest najbardziej znaczący bit (MSB) ale współczynnik wypełnienia nie jest równy ½.