Projektowanie Systemów Cyfrowych

Projekt

Arytmometr 16-bitowy z 8-bitowymi szynami wejścia i wyjścia realizujący operacje:

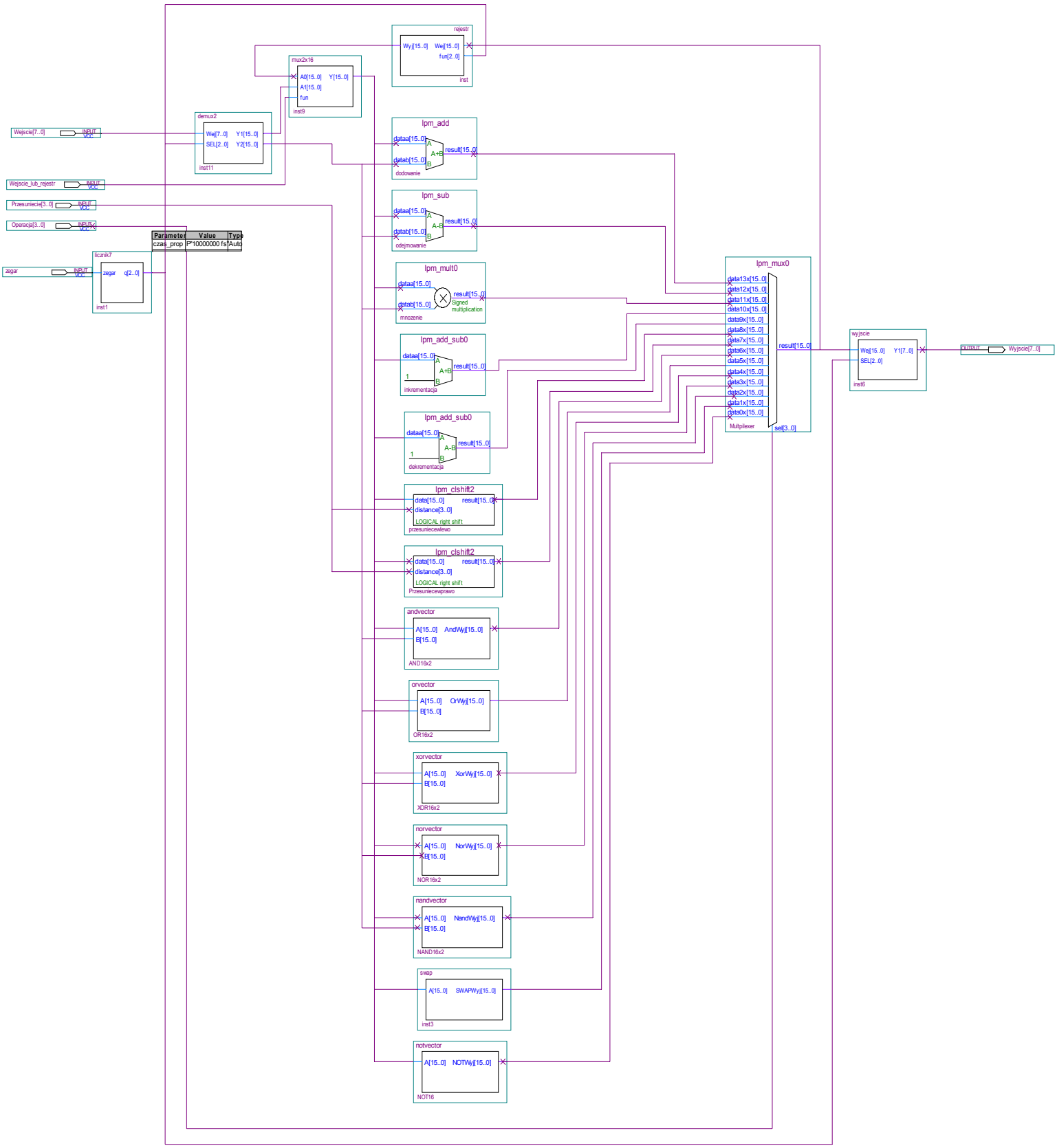
a) logiczne: AND, OR, XOR, NOT, NOR, NAND, SHL, SHR I SWAP
b) arytmetyczne: dodawanie, odejmowanie, inkrementacja, dekrementacja i mnożenie

Zespół:

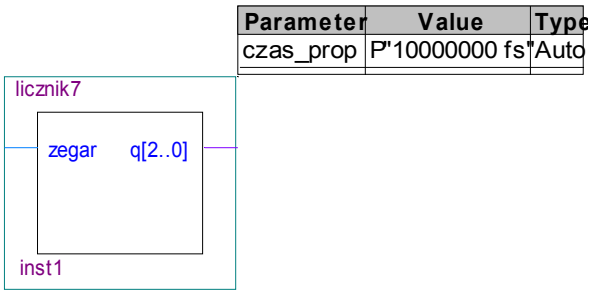
Jacek Donchor
Grzegorz Tkacz
Paweł Wacnik

1. Opis elementów, symbole, pliki vhdl i schematy

a) główny



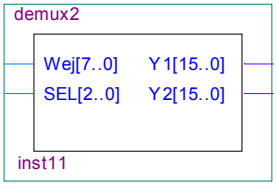
b) Licznik, element ten po wystąpieniu zbocza narastającego na zegarze (co 10 ns) inkrementuje wyjście licznika, po osiągnięciu wartości siedem, jest zerowany i następuje ponowienie cyklu. Licznik odpowiada za synchronizacje układu, tzn. aby w odpowiednich taktach zegara były realizowanie odpowiednie działania (wczytywanie danych, wykonywanie operacji, zapis do rejestru, wyświetlenie).



licznik7.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity licznik7 is
generic (czas_prop: Time := 10 ns);
port          (zegar: in bit;
               q: out natural range  7 downto 0);
end licznik7;

architecture behawior_licznika of licznik7 is
begin
  licz: process (zegar)
    variable wartosc: natural := 0;
  begin
    if ((zegar = '1')) then
      wartosc := (wartosc) mod 8;
      q <= wartosc after czas_prop;
      wartosc:=wartosc+1;
    end if;
end process licz;
end behawior_licznika;
```

c) Demultiplekser, element ten przyjmuje 8-bitowe słowa na wejściu i dla określonej wartości licznika, podanego na wejście SEL, tworzy 2 słowa 16-bitowe. Dla SEL=0 wchodzi LSB Y1, dla SEL=1 wchodzi MSB Y1, dla SEL=2 wchodzi LSB Y2, dla , dla SEL=3 wchodzi MSB Y2.

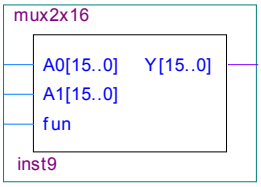


demux2.vhd

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

entity demux2 is

  port (
    Wej   : in  std_logic_vector(7 downto 0);
    SEL : in  integer range 7 downto 0;
    Y1  : out  std_logic_vector(15 downto 0);
    Y2  : out  std_logic_vector(15 downto 0));

end demux2;

architecture behave of demux2 is

begin
process(Wej,SEL)
begin

if (SEL=0)    then

   Y1(7) <= Wej(7);
   Y1(6) <= Wej(6);
   Y1(5) <= Wej(5);
   Y1(4) <= Wej(4);
   Y1(3) <= Wej(3);
   Y1(2) <= Wej(2);
   Y1(1) <= Wej(1);
   Y1(0) <= Wej(0);

end if;


if  SEL=1 then
   Y1(15) <= Wej(7);
   Y1(14) <= Wej(6);
   Y1(13) <= Wej(5);
   Y1(12) <= Wej(4);
   Y1(11) <= Wej(3);
   Y1(10) <= Wej(2);
   Y1(9) <= Wej(1);
   Y1(8) <= Wej(0);

end if;

if SEL=2 then

   Y2(7) <= Wej(7);
   Y2(6) <= Wej(6);
```

```
   Y2(5) <= Wej(5);
   Y2(4) <= Wej(4);
   Y2(3) <= Wej(3);
   Y2(2) <= Wej(2);
   Y2(1) <= Wej(1);
   Y2(0) <= Wej(0);

end if;
if SEL=3 then
   Y2(15) <= Wej(7);
   Y2(14) <= Wej(6);
   Y2(13) <= Wej(5);
   Y2(12) <= Wej(4);
   Y2(11) <= Wej(3);
   Y2(10) <= Wej(2);
   Y2(9) <= Wej(1);
   Y2(8) <= Wej(0);

end if;

end process;
end behave;
```

d) Multiplekser 16-bitowy który dla fun=0 wybiera na wejście zawartość rejestru, który zawiera ostatni stan na wyjściu arytmometru, dla fun=1 wybiera na wejściu dane wychodzące z wyjscia Y1 demultipleksera z podpunktu c.

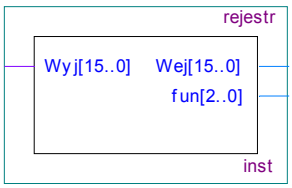


mux2x16.vhd

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

entity mux2x16 is
port(A0: in std_logic_vector(15 downto 0);
                        A1: in std_logic_vector(15 downto 0);
                        fun: in integer range 1 downto 0;
                        Y: out std_logic_vector(15 downto 0));
end mux2x16;

architecture mux2x16_arch of mux2x16 is
begin
```

```
with fun select
Y <= A0 when 0,
             A1 when 1,

             null when others;
end mux2x16_arch;
```

e) Rejestr który na wejściu dla wartości licznika podawanej na wejście fun=5 zapamiętuje 16-bitowe które jest wynikiem operacji arytmetycznej lub logicznej, dla fun=0 zawartość rejestru jest wyprowadzone na jego wyjście i wprowadzone do multiplekser z punktu d.

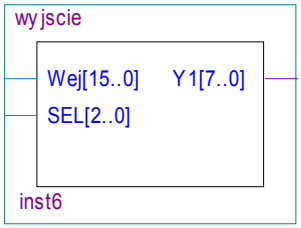


rejestr.vhd

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

entity rejestr is

  port (Wej   : in  std_logic_vector(15 downto 0);
                                    fun : in integer range 7 downto 0;
                          Wyj  : out  std_logic_vector(15 downto 0));
end rejestr;

architecture behave of rejestr is

begin
process(Wej,fun)
variable wnetrze:std_logic_vector(15 downto 0);
begin

if fun=5 then

   wnetrze(15) := Wej(15);
   wnetrze(14) := Wej(14);
   wnetrze(13) := Wej(13);
   wnetrze(12) := Wej(12);
   wnetrze(11) := Wej(11);
   wnetrze(10) := Wej(10);
   wnetrze(9) := Wej(9);
   wnetrze(8) := Wej(8);

   wnetrze(7) := Wej(7);
   wnetrze(6) := Wej(6);
```

```
   wnetrze(5) := Wej(5);
   wnetrze(4) := Wej(4);
   wnetrze(3) := Wej(3);
   wnetrze(2) := Wej(2);
   wnetrze(1) := Wej(1);
   wnetrze(0) := Wej(0);
 end if;

if fun=0 then
   Wyj(15) <= wnetrze(15);
   Wyj(14) <= wnetrze(14);
   Wyj(13) <= wnetrze(13);
   Wyj(12) <= wnetrze(12);
   Wyj(11) <= wnetrze(11);
   Wyj(10) <= wnetrze(10);
   Wyj(9) <= wnetrze(9);
   Wyj(8) <= wnetrze(8);

   Wyj(7) <= wnetrze(7);
   Wyj(6) <= wnetrze(6);
   Wyj(5) <= wnetrze(5);
   Wyj(4) <= wnetrze(4);
   Wyj(3) <= wnetrze(3);
   Wyj(2) <= wnetrze(2);
   Wyj(1) <= wnetrze(1);
   Wyj(0) <= wnetrze(0);
 end if;
end process;
end behave;
```

f) Wyjście. Jest to multiplekser który na wyprowadza 16-bitowy wynik obliczenia na wyjście 8-bitowe. Jest to realizowane w dwóch taktach zegara. Dla SEL=6 jest Y1 przyjmuje słowo MSB, a dla SEL=7 przyjmuje słowo LSB.

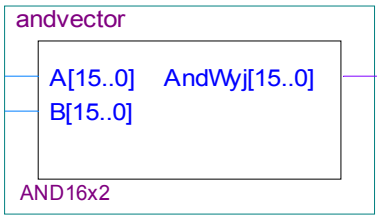


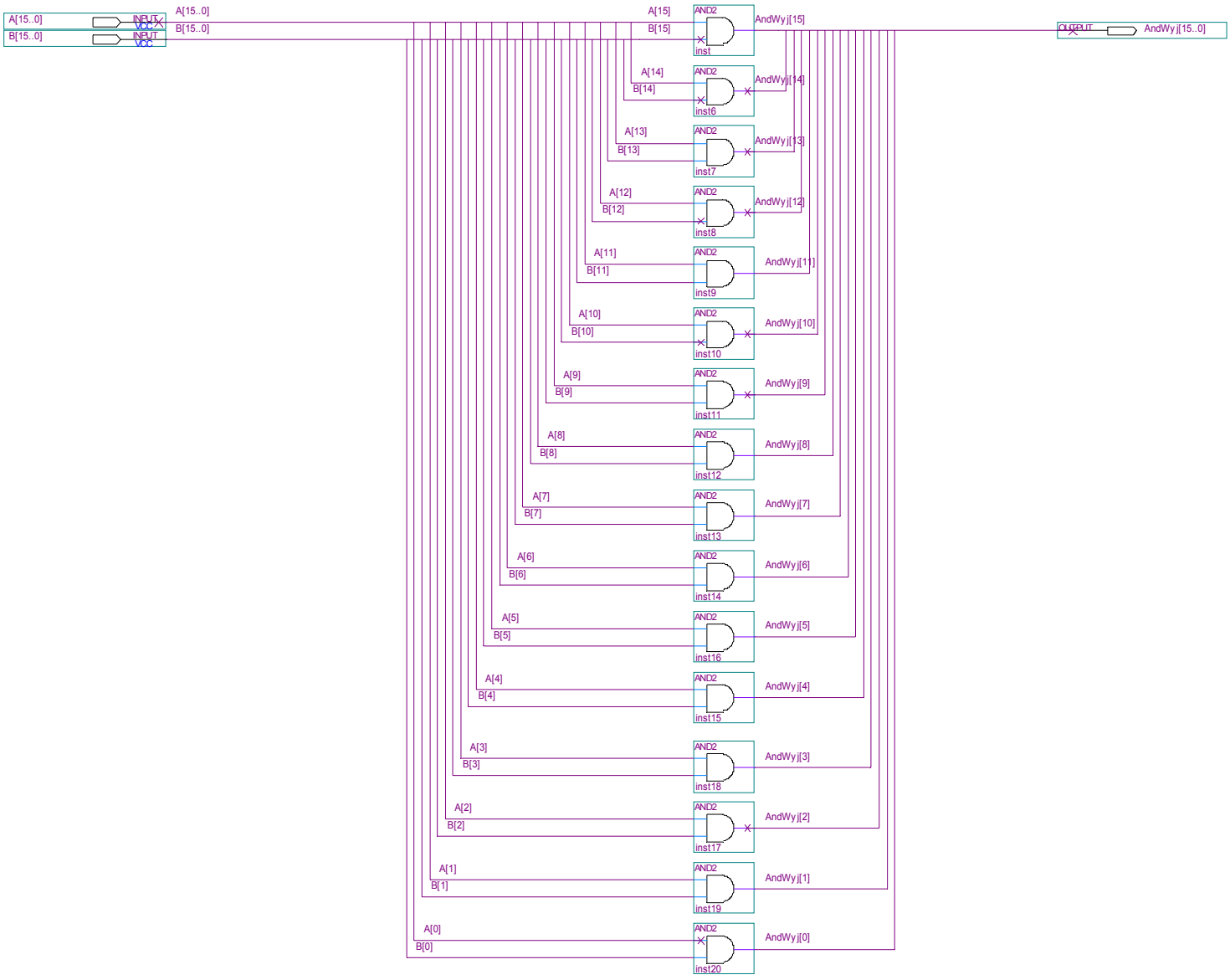
wyjscie.vhd

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

entity wyjscie is
```

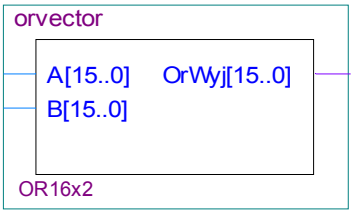
```
  port (  Wej   : in  std_logic_vector(15 downto 0);
                      SEL : in  natural range 7 downto 0;
                      Y1  : out  std_logic_vector(7 downto 0));
end wyjscie;

architecture behave of wyjscie is

begin
process(Wej,SEL)
begin

if  SEL=6 then

   Y1(7) <= Wej(15);
   Y1(6) <= Wej(14);
   Y1(5) <= Wej(13);
   Y1(4) <= Wej(12);
   Y1(3) <= Wej(11);
   Y1(2) <= Wej(10);
   Y1(1) <= Wej(9);
   Y1(0) <= Wej(8);

end if;

if SEL=7 then
   Y1(7) <= Wej(7);
   Y1(6) <= Wej(6);
   Y1(5) <= Wej(5);
   Y1(4) <= Wej(4);
   Y1(3) <= Wej(3);
   Y1(2) <= Wej(2);
   Y1(1) <= Wej(1);
   Y1(0) <= Wej(0);

end if;

if (SEL=0 or SEL=1 or SEL=2 or SEL=3 or SEL=4 or SEL=5) then
Y1<=x"00";
end if ;

end process;
end behave;
```

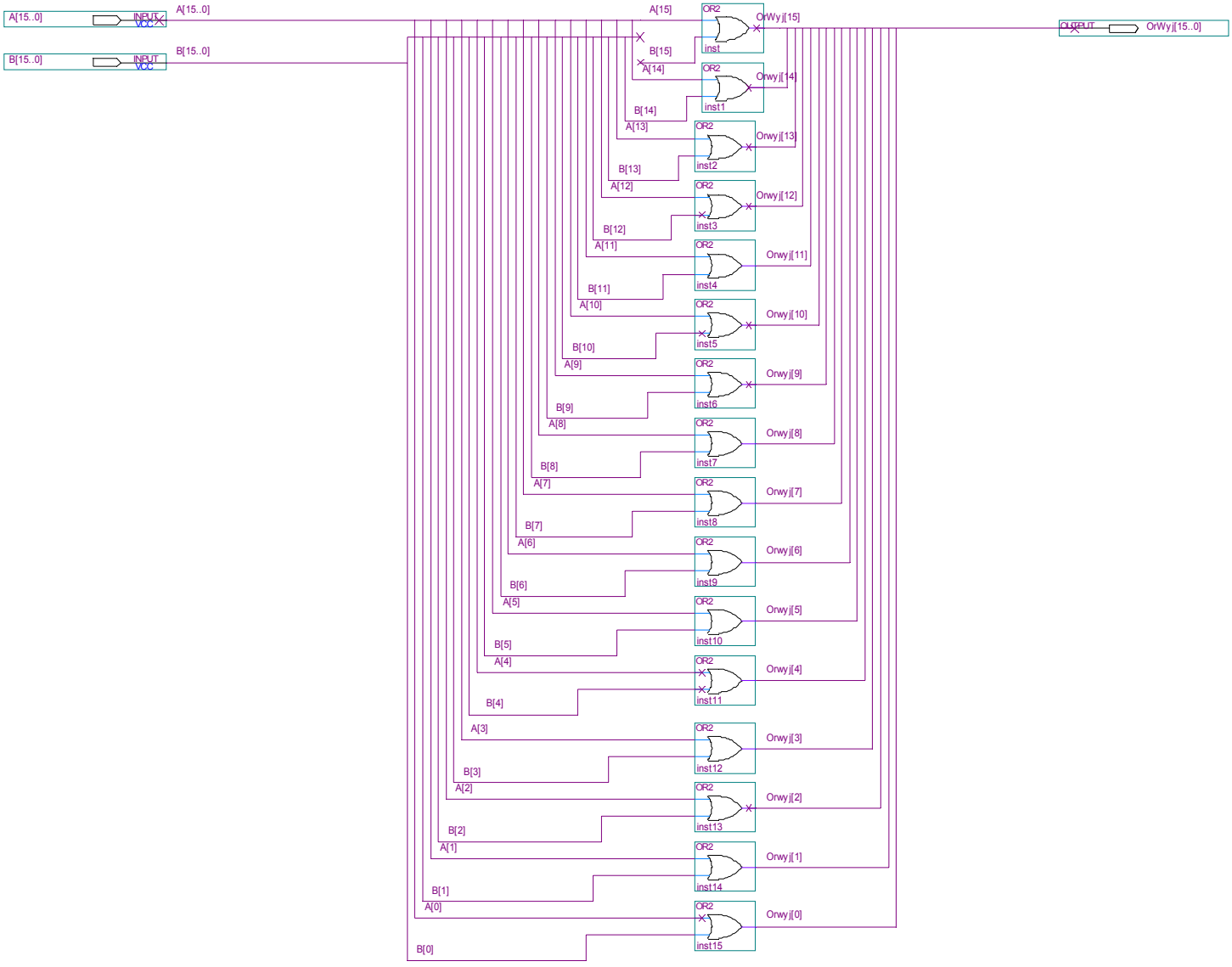
g) Realizacja operacji AND



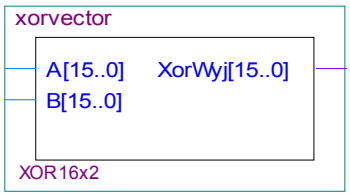


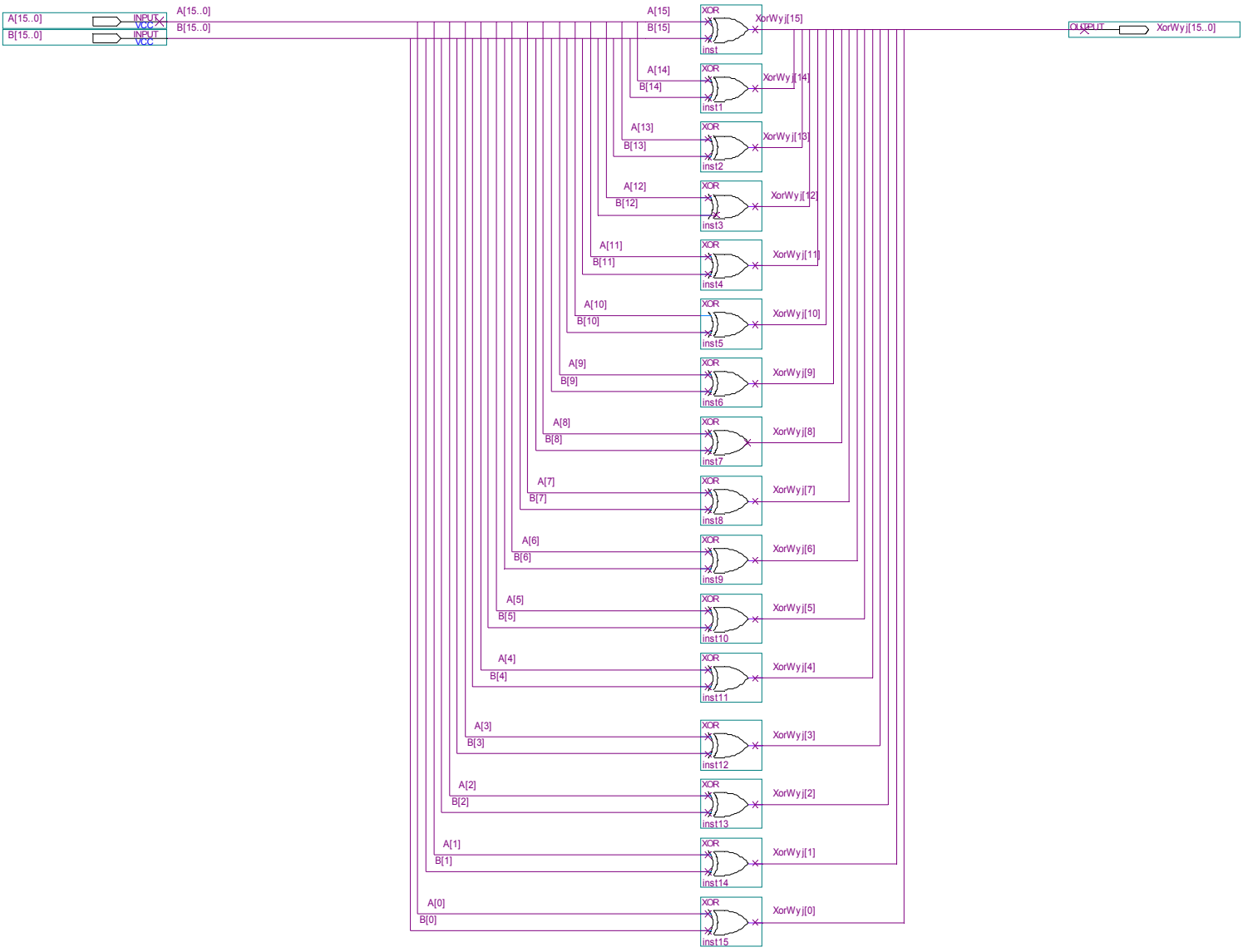
h) Realizacja operacji OR



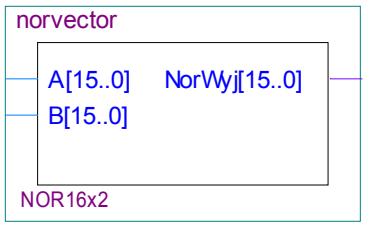


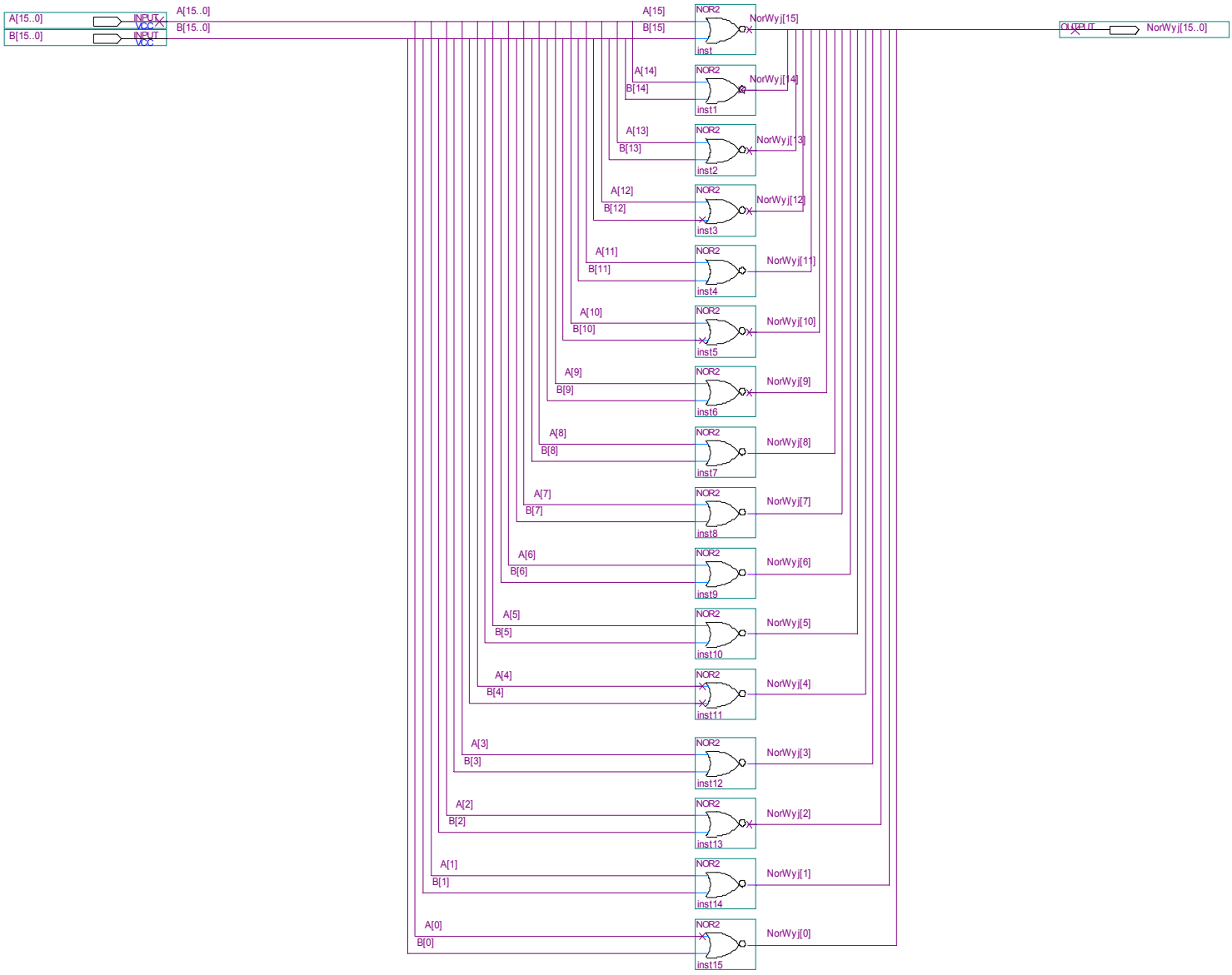
i) Realizacja operacji XOR



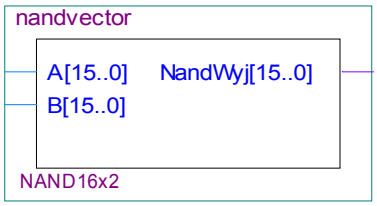


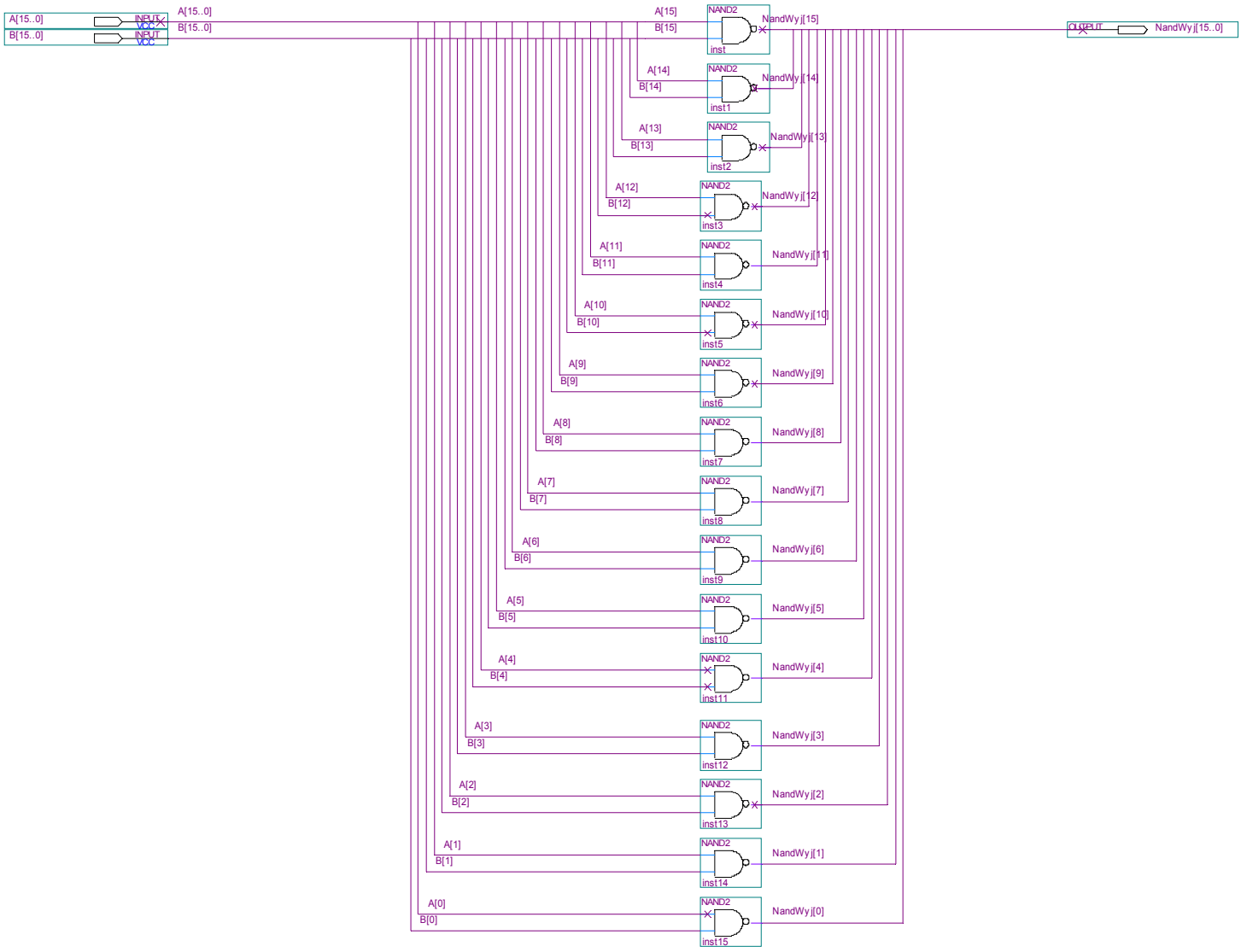
j) Realizacja operacji NOR



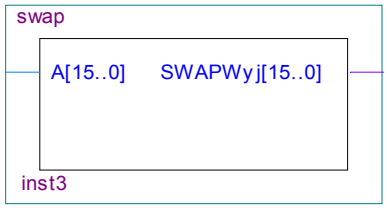


k) Realizacja operacji NAND





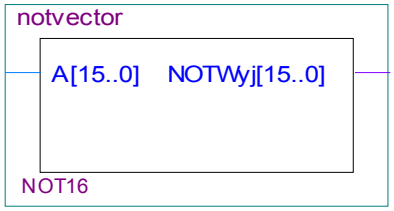
l) Realizacja operacji SWAP



swap.vhd

```
library ieee;

use ieee.std_logic_1164.all;

entity swap is

  port ( A  : in  std_logic_vector(15 downto 0);
                  SWAPWyj  : out  std_logic_vector(15 downto 0));

end swap;

architecture behave of swap is

begin
process(A)
begin

   SWAPWyj(15) <= A(7);
   SWAPWyj(14) <= A(6);
   SWAPWyj(13) <= A(5);
   SWAPWyj(12) <= A(4);
   SWAPWyj(11) <= A(3);
   SWAPWyj(10) <= A(2);
   SWAPWyj(9) <= A(1);
   SWAPWyj(8) <= A(0);

   SWAPWyj(7) <= A(15);
   SWAPWyj(6) <= A(14);
   SWAPWyj(5) <= A(13);
   SWAPWyj(4) <= A(12);
   SWAPWyj(3) <= A(11);
   SWAPWyj(2) <= A(10);
   SWAPWyj(1) <= A(9);
   SWAPWyj(0) <= A(8);

end process;
end behave;
```

ł) Realizacja operacji NOT



notvector.vhd

```
library ieee;

use ieee.std_logic_1164.all;

entity notvector is

 port (
   A  : in  std_logic_vector(15 downto 0);

   notvectorWyj  : out  std_logic_vector(15 downto 0));


end notvector;

architecture behave of notvector is

begin
process(A)
begin

  notvectorWyj<=(not A);

end process;
end behave;
```

Pozostałe elementy nie zostały opisane ponieważ należą do wbudowanych bibliotek środowiska Quartus, i były tworzone poprzez MegaWizard Plug-In Menager.

2. Wykresy wektorowe

Na wejściu dla wartości licznika:
0- wchodzi 8 młodszych bitów pierwszego słowa
1- wchodzi 8 starszych bitów pierwszego słowa
2- wchodzi 8 młodszych bitów drugiego słowa
3- wchodzi 8 starszych bitów drugiego słowa

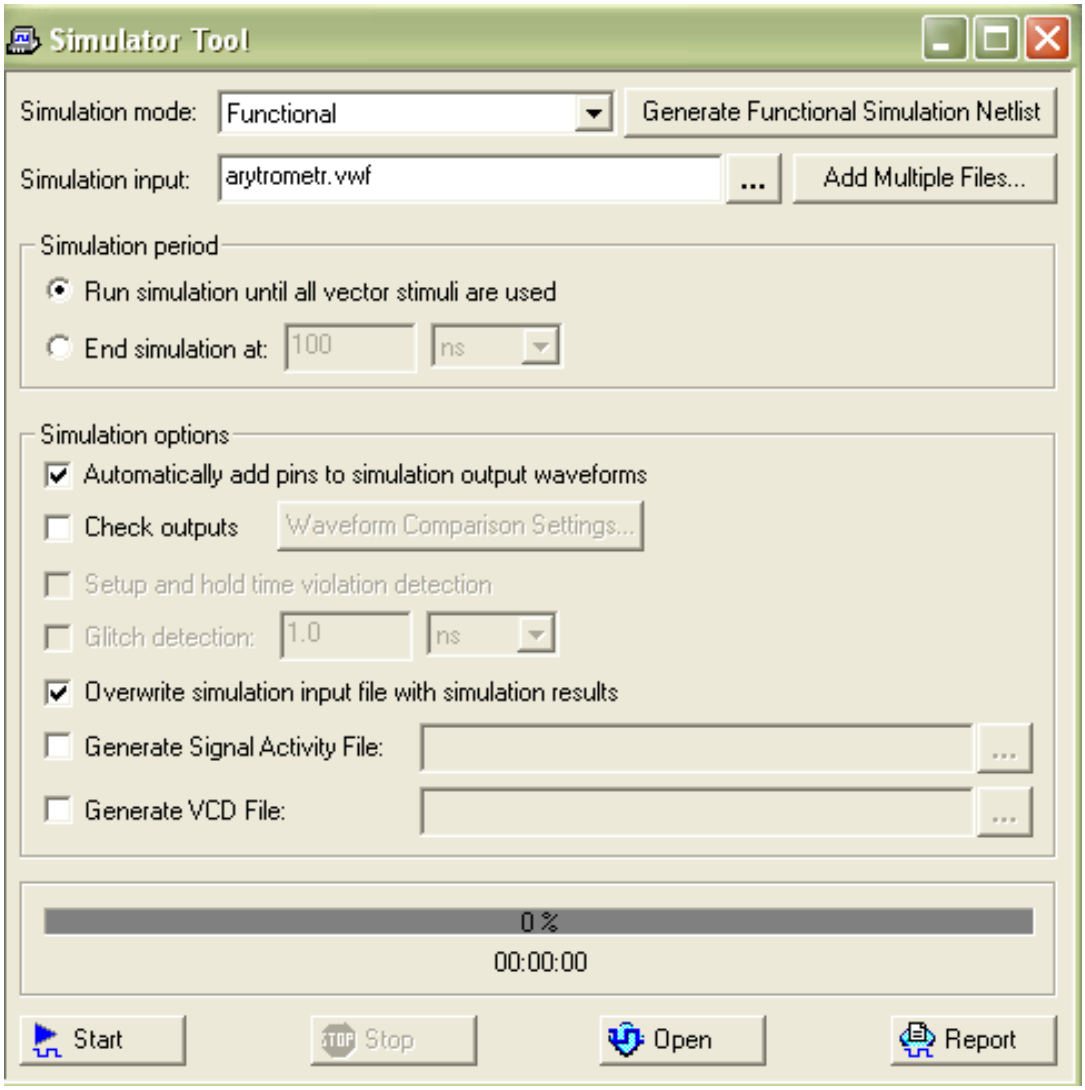
Dla wartości licznika 4 i 5 dokonywana są operacje logiczne lub arytmetyczne

Na wyjściu dla wartości licznika:

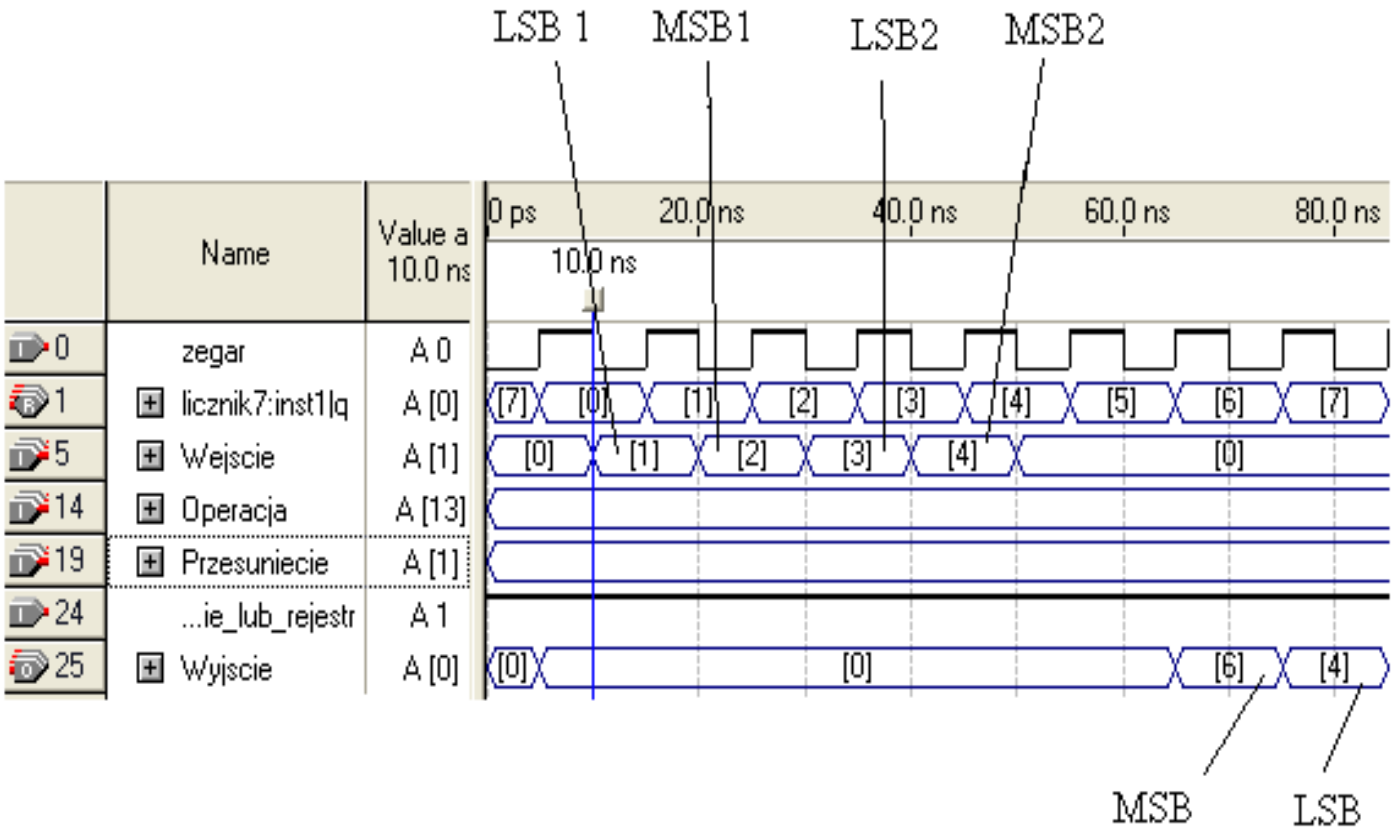
6- wychodzi 8 starszych bitów wyniku
7- wychodzi 8 młodszych bitów wyniku

Pin wejście_lub_rejestr odpowiada za wybór między wejściem arytmometru (wejscie_lub_rejestr=1) a wyjściem arytmometru (wejscie_lub_rejestr=0) kierowanym do bloku obliczeniowego.

Symulacje są prowadzone w trybie funkcjonalnym. Ustawienia na screenie:

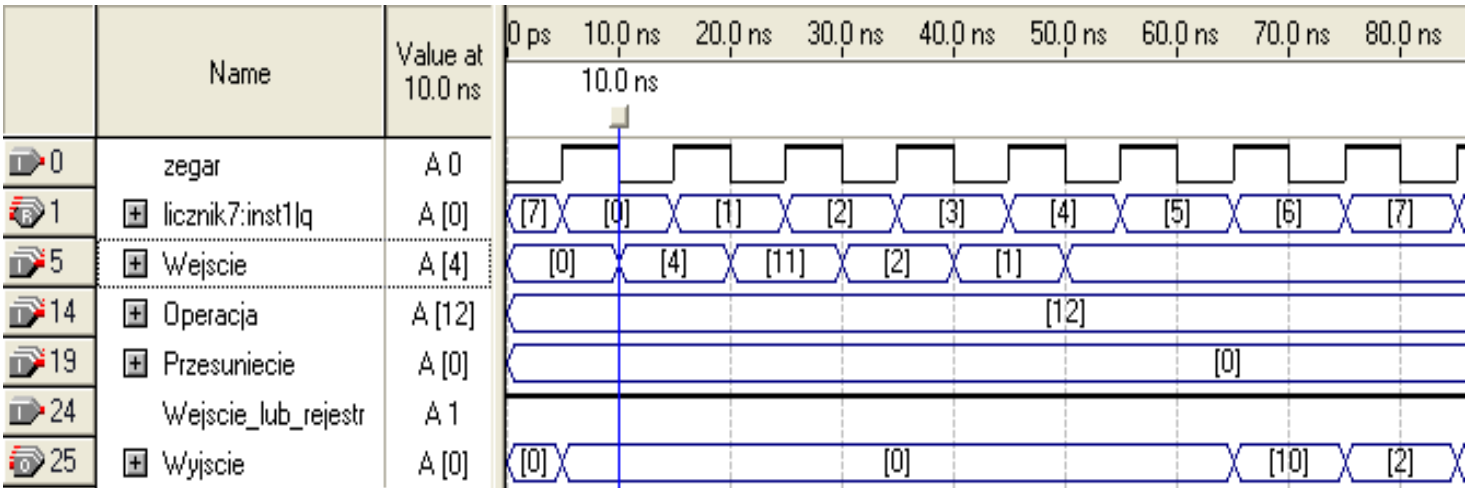


a) dodawanie

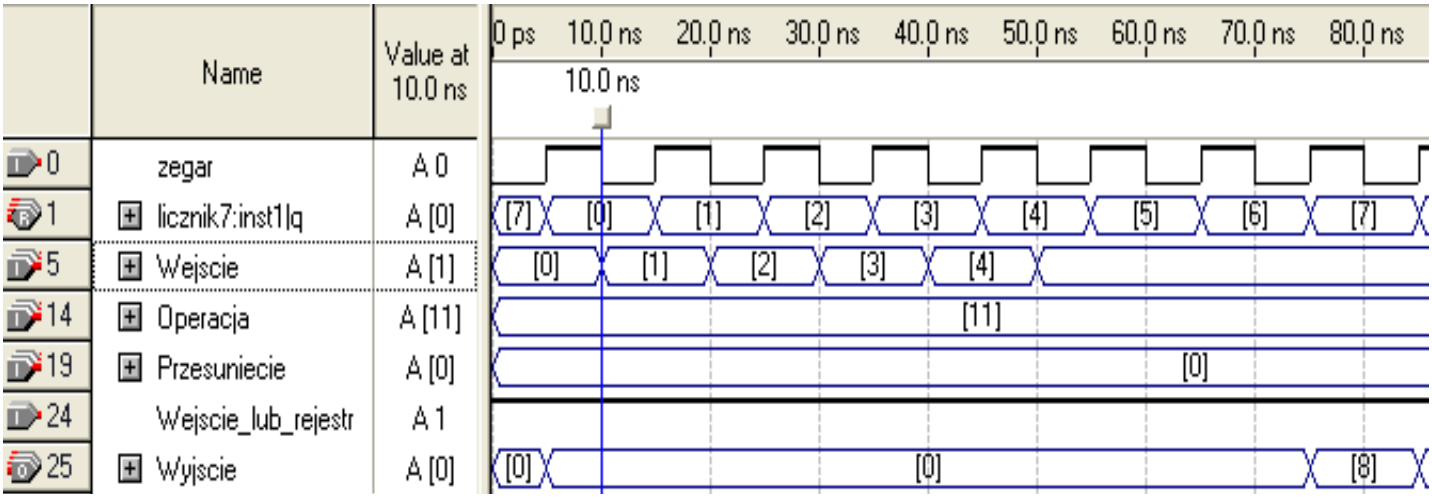


Dla pozostałych przypadków opis jest podobny.

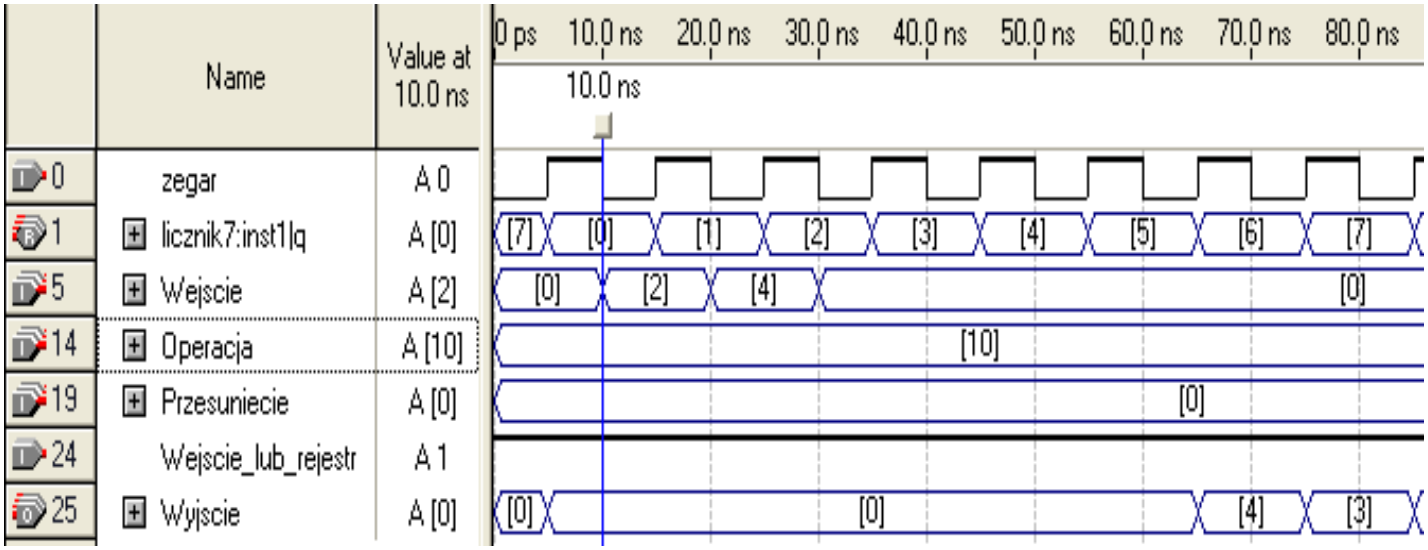
b) odejmowanie



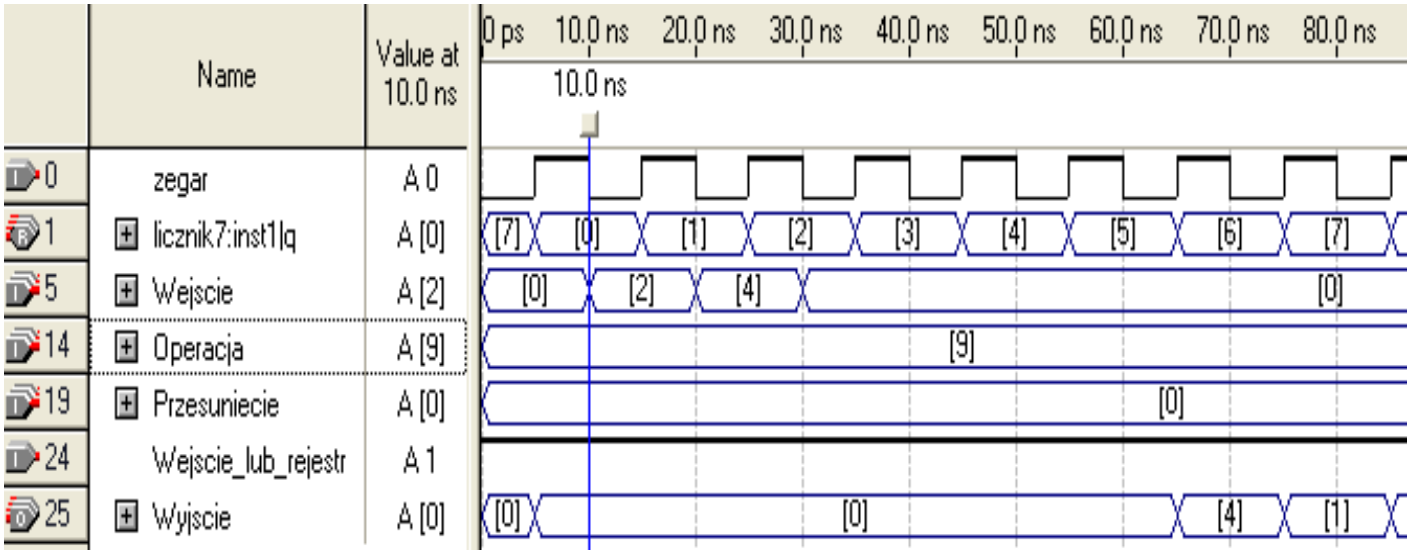
c) mnożenie (wyświetlana jest tylko zawartość MSB)



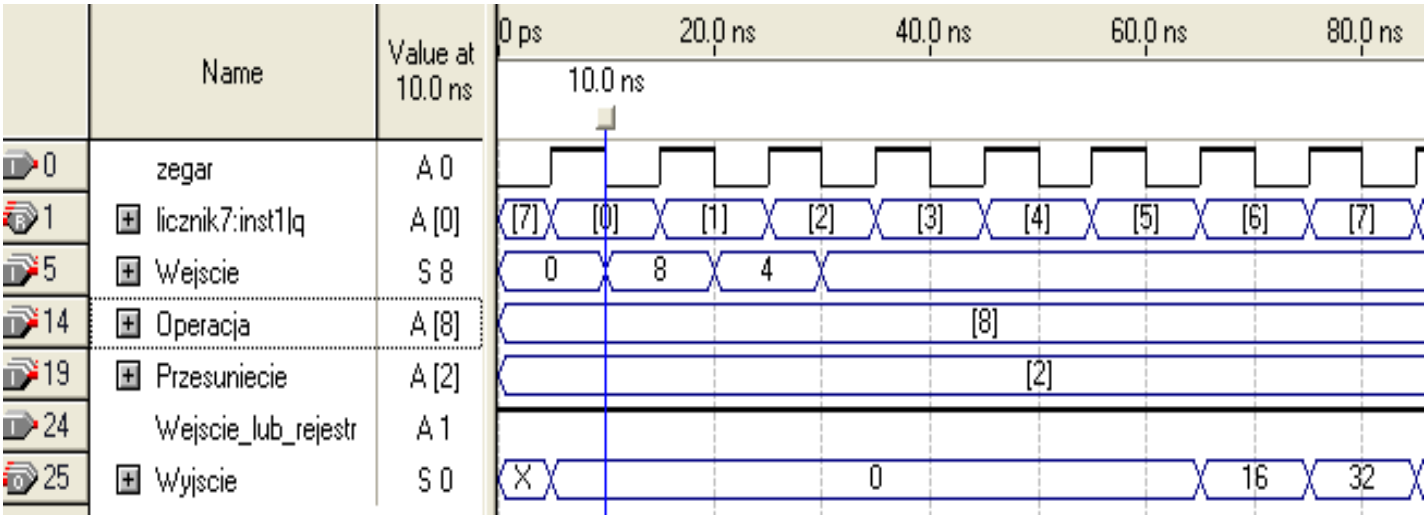
d) inkrementacja (inkrementowane jest I słowo: podane dla wartości licznika 0 i 1)



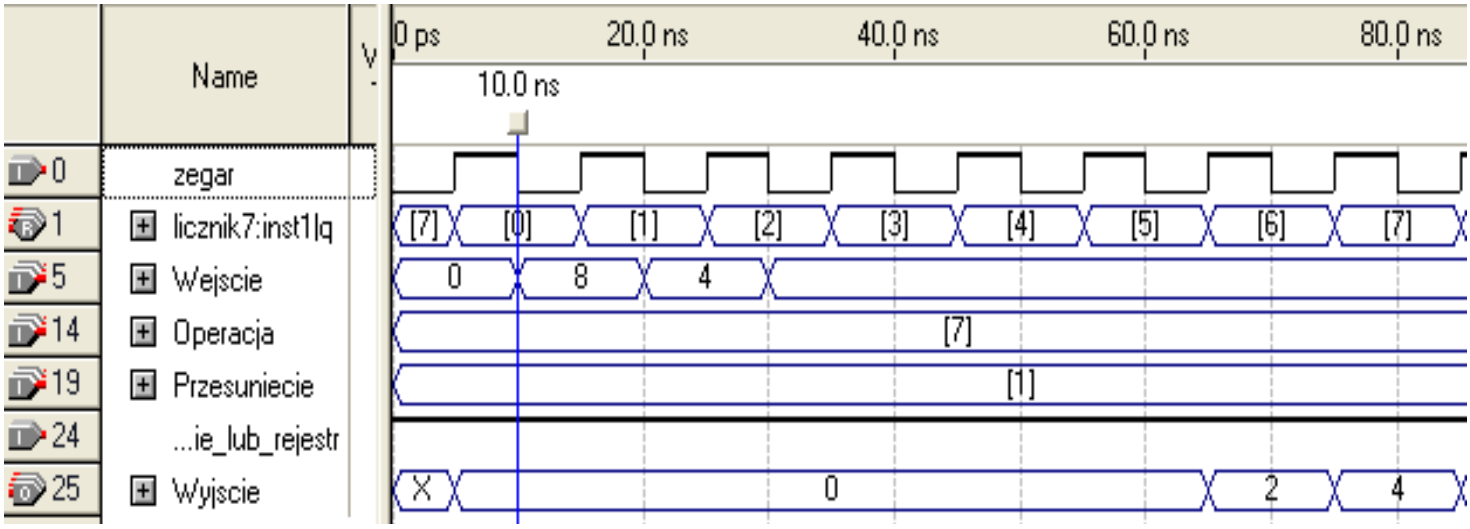
e) dekrementacja



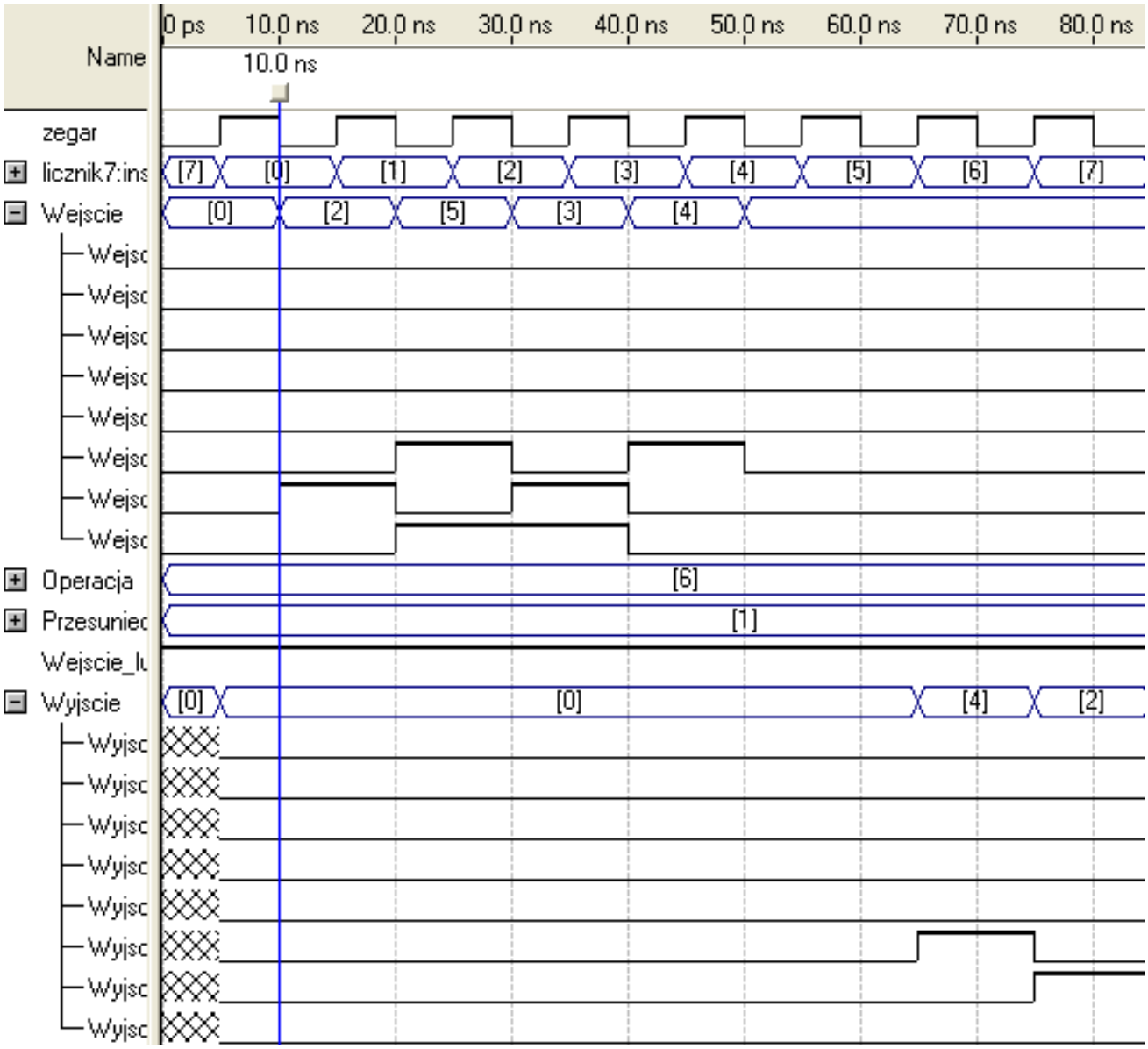
f) przesuniecie w lewo (przesuwamy o 2 miejsca tzn mnożymy przez 4)



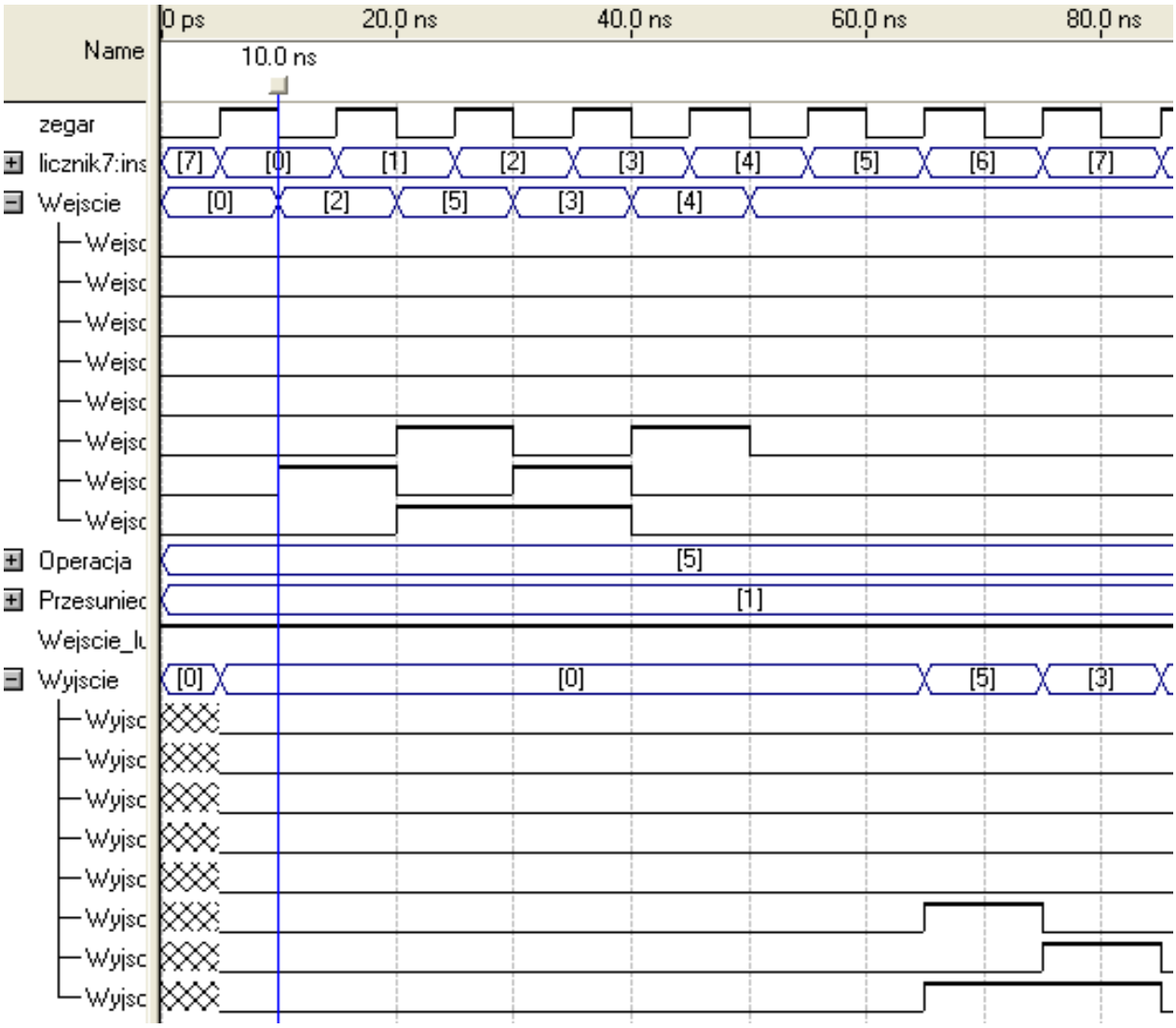
g) przesuniecie w prawo (przesuwamy o 1 miejsce tzn. dzielimy przez 2)



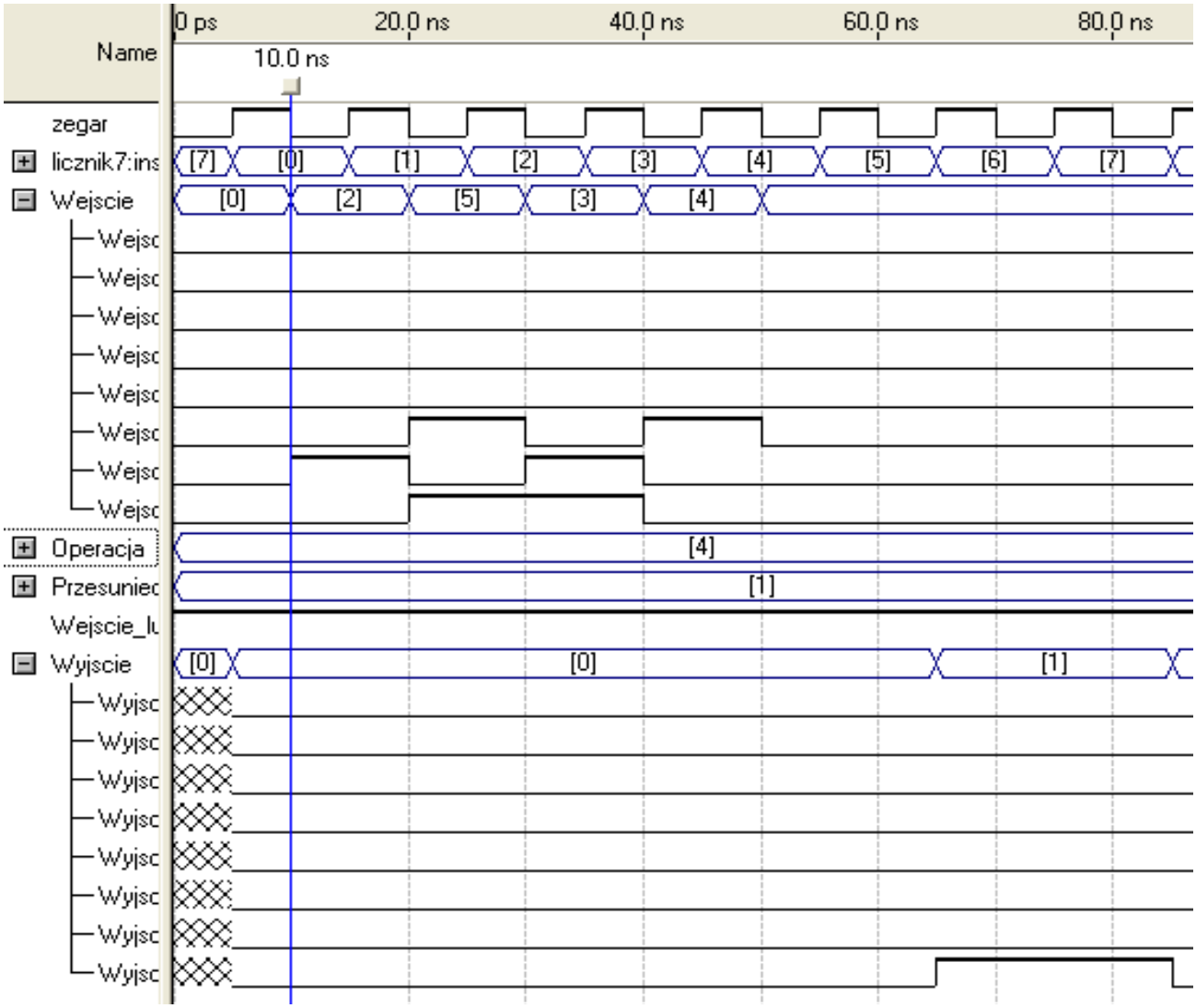
h) AND


Name
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
50.0 ns
60.0 ns
70.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
Wejscie
Operacja
Przesuniec
Wejscie_lu
Wyjscie
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[0]
[2]
[5]
[3]
[4]
[6]
[1]
[0]
[0]
[4]
[2]

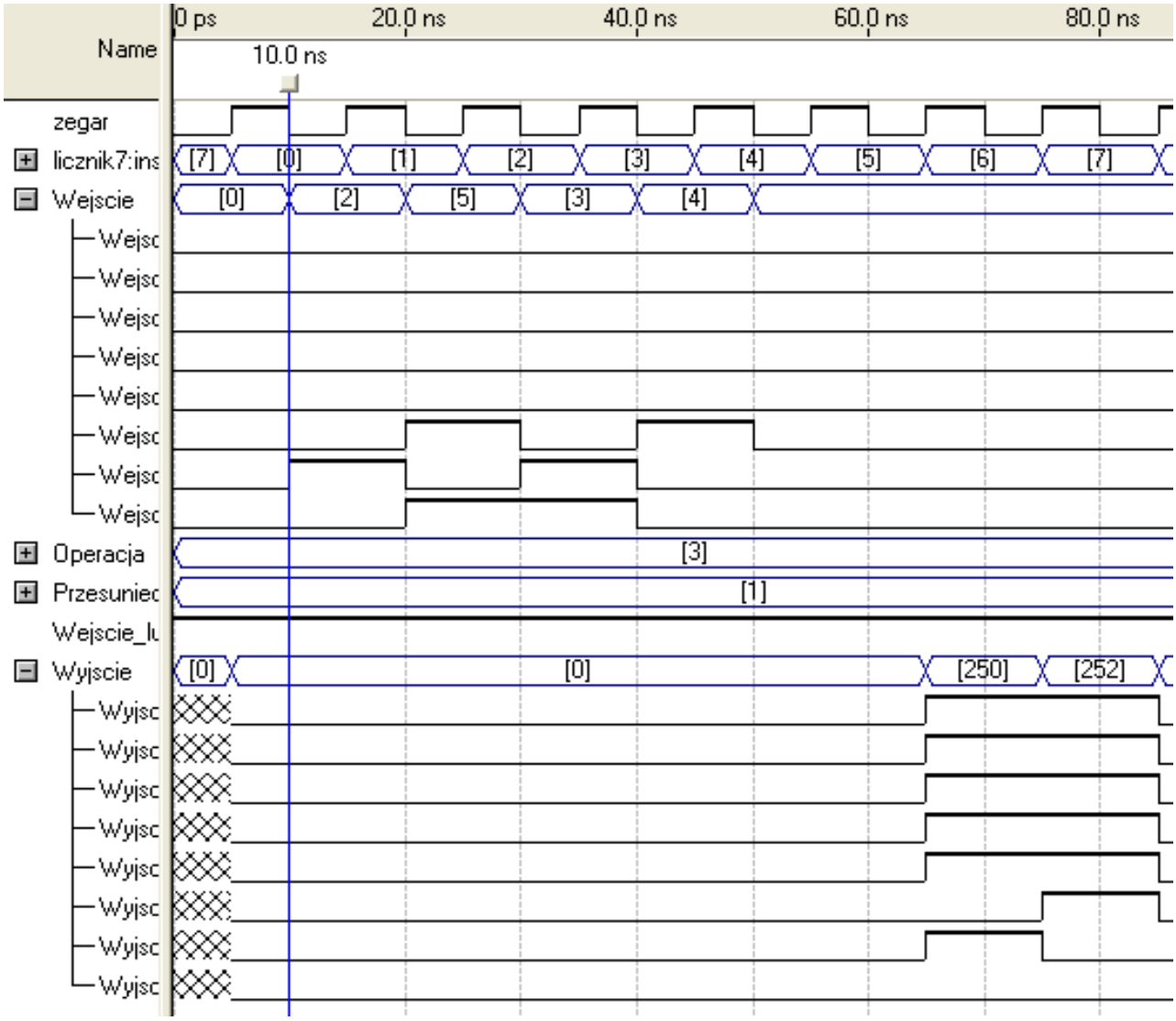
i) OR


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
Wejscie
Operacja
Przesuniec
Wejscie_lu
Wyjscie

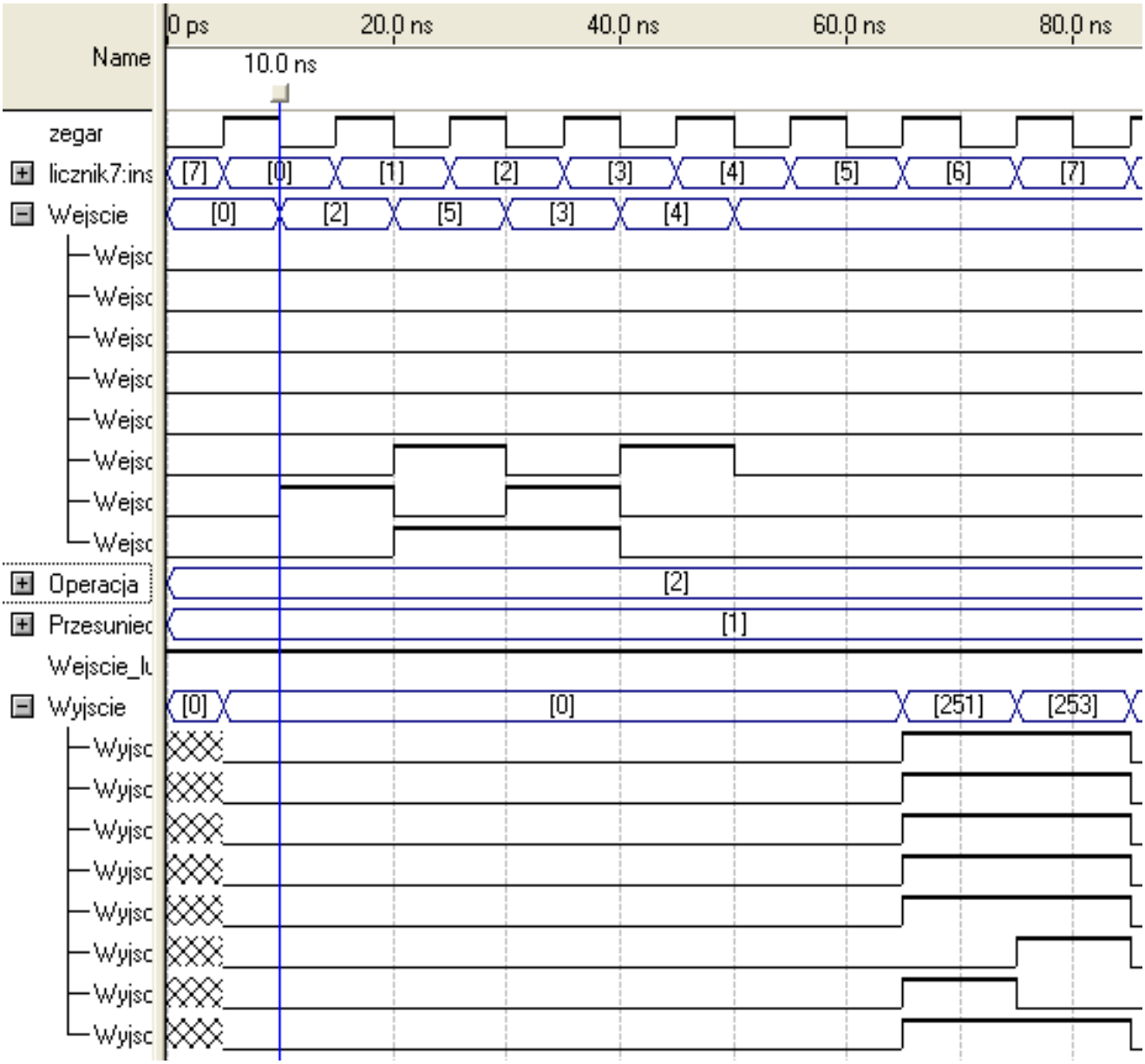
j) XOR


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
Wejscie
Operacja
Przesuniec
Wejscie_lu
Wyjscie
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[0]
[2]
[5]
[3]
[4]
[4]
[1]
[0]
[0]
[1]

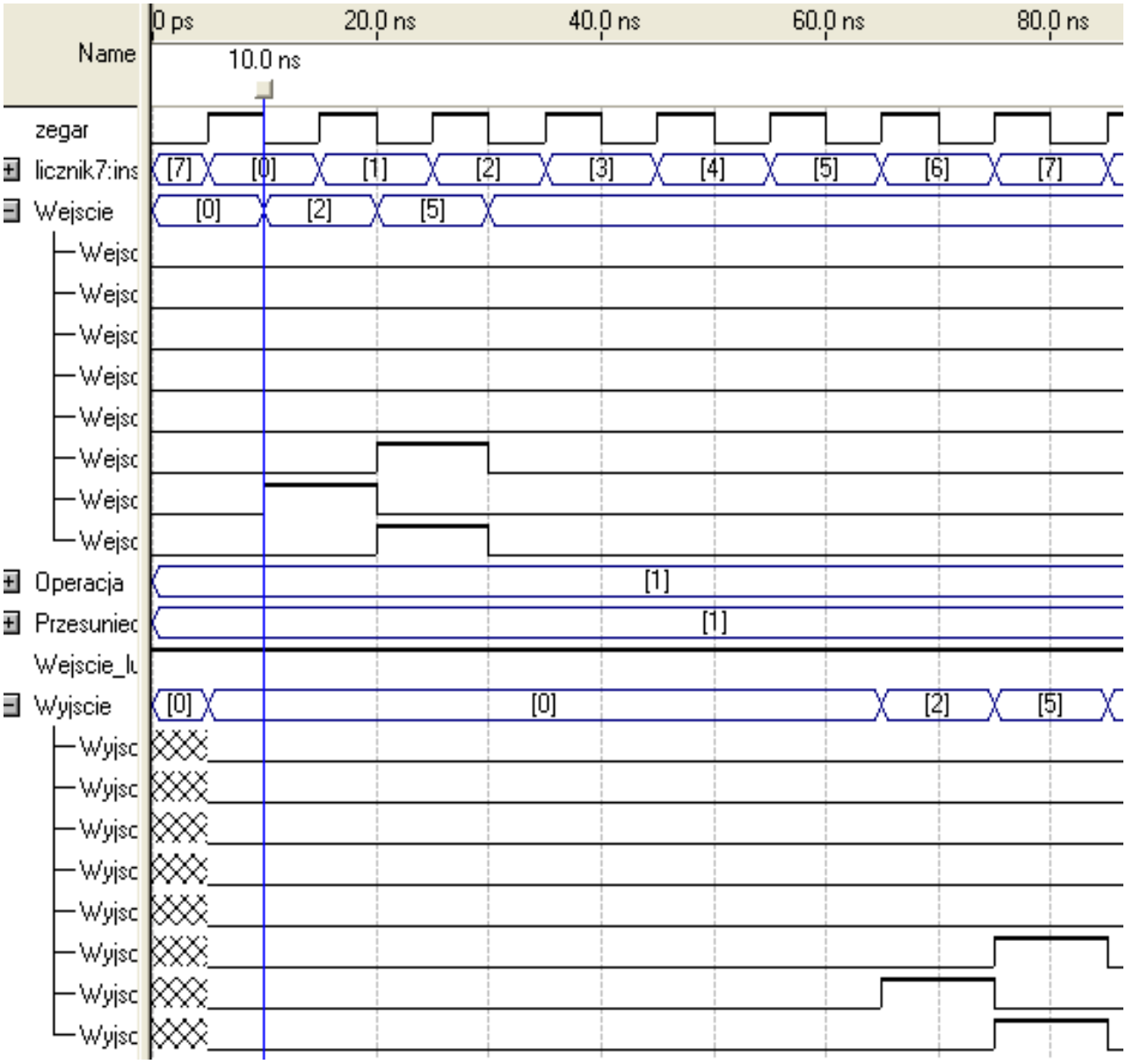
k) NOR


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
Wejscie
[0]
[2]
[5]
[3]
[4]
Wejsc
Operacja
[3]
Przesuniec
[1]
Wejscie_lu
Wyjscie
[0]
[0]
[250]
[252]
Wyjsc

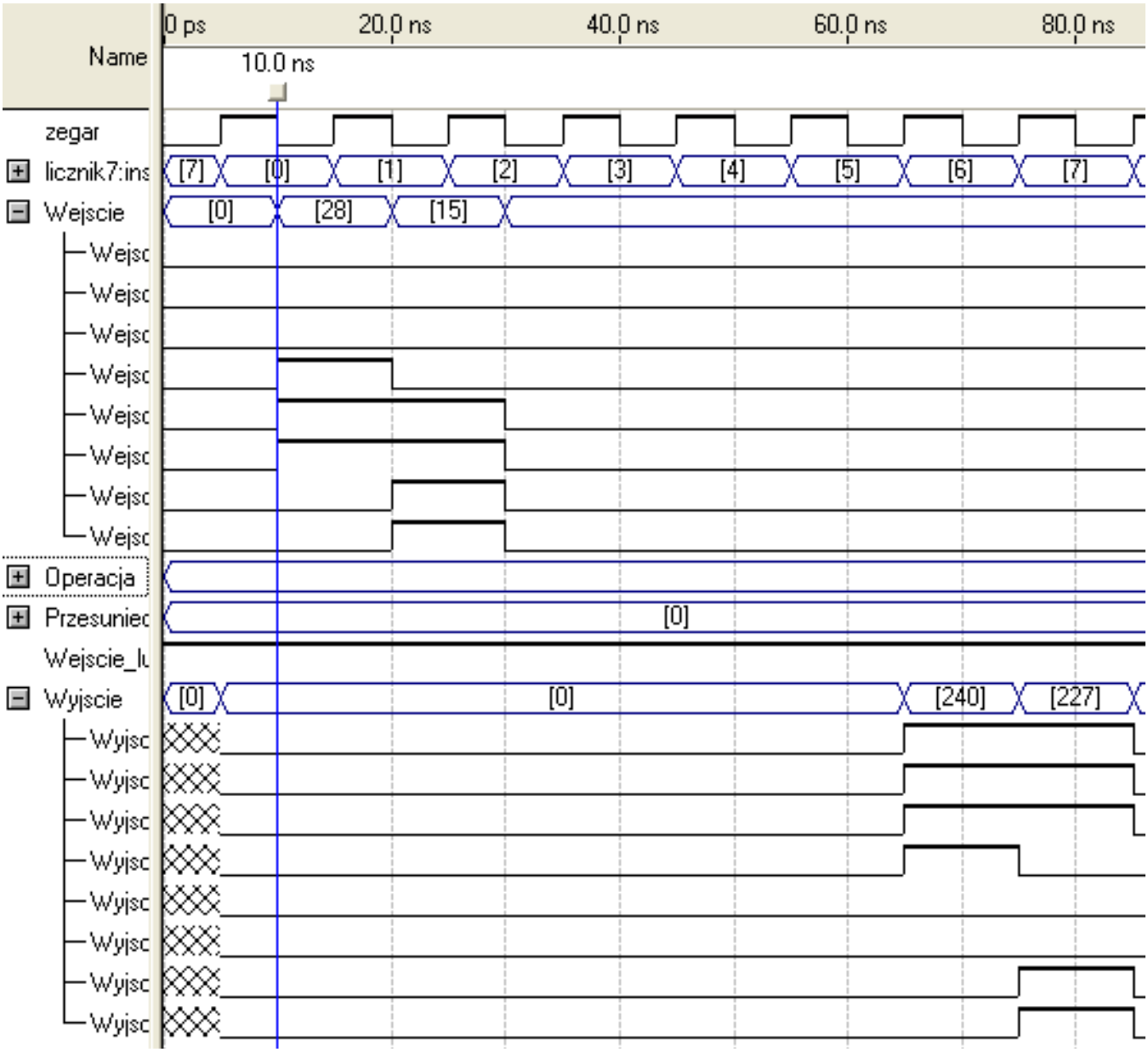
l) NAND


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
Wejscie
Wejsc
Operacja
Przesuniec
Wejscie_lu
Wyjscie
Wyjsc
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[251]
[253]

ł) SWAP


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
Wejscie
Operacja
Przesuniec
Wejscie_lu
Wyjscie
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[0]
[2]
[5]
[1]
[1]
[0]
[0]
[2]
[5]

m) NOT


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
10.0 ns
zegar
licznik7:ins
[7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
Wejscie
[0]
[28]
[15]
Wejsc
Operacja
Przesuniec
[0]
Wejscie_lu
Wyjscie
[0]
[0]
[240]
[227]
Wyjsc

n) Przykład z wykorzystaniem zawartości rejestru. W pierwszym okresie zliczania licznika dodajemy dwie liczby, w drugim okresie od wyniku dodawania (zawartość rejestru) odejmujemy liczbę wprowadzoną na wejście arytmometru.