

VHDL, Altera, DE2 przykłady rozwiązań

Opracował: R.Walkowiak

16.12.2017

for generate

Instrukcja dla wektora obiektów

<etykieta_generate>:

```
for <loop_id> in <zakres> generate
```

```
-- wyrażenia równoległe
```

```
end generate;
```

Wyrażenie równoległe do wykorzystania w specyfikacji struktury/działania układu *architecture*. Parametr generate może być wykorzystany do indeksowania elementów wektora sygnałów wykorzystywanych w wyrażeniu równoległym lub portów konkretyzowanej jednostki. Etykiety opcjonalne, nie mogą zawierać indeksów.

przykład:

-- tworzenie sygnału (wektora) w oparciu o inne wektory

```
for i in WIDTH-1 downto 0 generate
```

```
s(i) <= a(i) xor b(i) xor c(i);
```

```
end generate;
```

entity wiele_bledow is port	-- linia 1
A: bit_vector(3 to 0);	-- linia 2
B: out std_logic_vector (3 downto 0);	-- linia 3
C: in bit_vector (0 to 5);)	-- linia 4
end wiele_bledow	-- linia 5
architecture nie_za_dobra of wiele_bledow	-- linia 6
begin	-- linia 7
Etykieta: process	-- linia 8
begin	-- linia 9
if c = x"F" then	-- linia 10
b<= a;	-- linia 11
else	-- linia 12
b<= '0101';	-- linia 13
end if	-- linia 14
end process;	-- linia 15
end nie_za-dobra	-- linia 16

Przykładowe błędy
składni i znaczenia
kodu VHDL

Poprawiony kod

```
library ieee;
use ieee.std_logic_1164.all;
entity bez_bledow is port(
    A: std_logic_vector(3 downto 0);
    B: out std_logic_vector (3 downto 0);
    C: in bit_vector (0 to 5));
end bez_bledow;

architecture lepsza of bez_bledow is
begin
    etykieta: process (c,a)
    begin
        if c = "001111" then
            b<= a;
        else
            b<= "0101";
        end if;
    end process;
end lepsza;
```

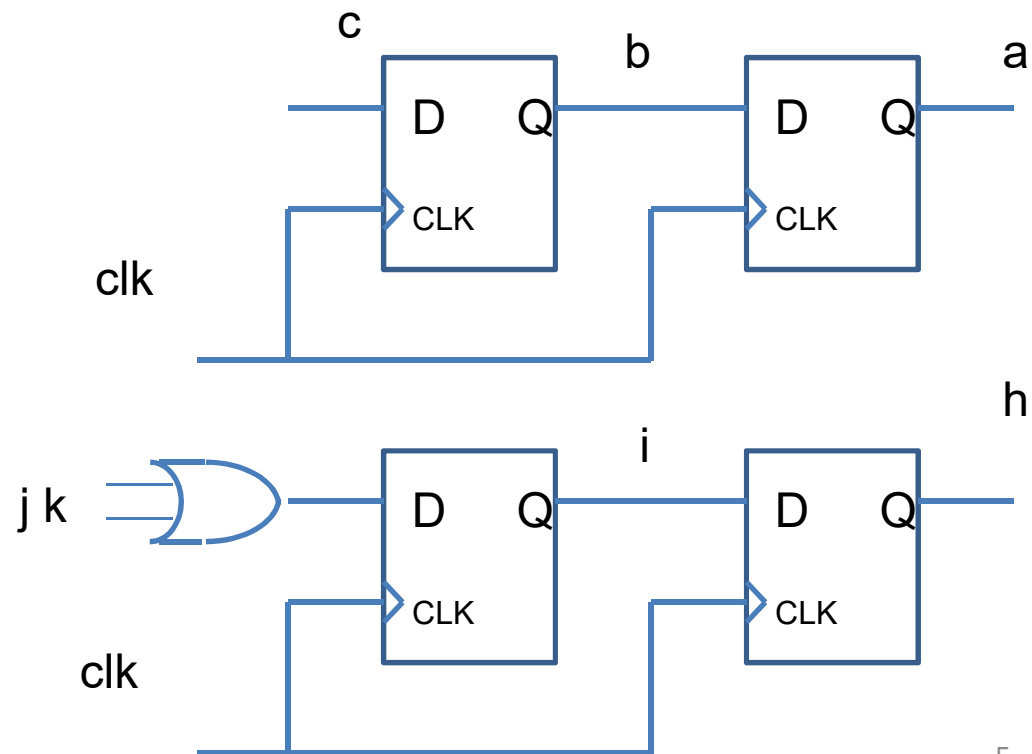
-- linia 1
-- linia 2
-- linia 3
-- linia 4
-- linia 5

-- linia 6
-- linia 7
-- linia 8
-- linia 9
-- linia 10
-- linia 11
-- linia 12
-- linia 13
-- linia 14
-- linia 15
-- linia 16

Jak to działa ?

Ile tu jest przerzutników?

```
seq:process (clk);  
begin  
    if clk'event and clk ='1'  
    then  
        b<=c;  
        a<=b;  
        h<=i;  
        i<=j or k;  
    end if;  
end process;
```



Znajdź różnicę 1

Działanie: $c \text{ clk} \Rightarrow b \text{ clk} \Rightarrow a$

```
seq:process (clk);  
begin  
    if clk'event and clk ='1' then  
        b<=c; -- sygnały  
        a<=b;  
    end if;  
end process;
```

Działanie $c \text{ clk} \Rightarrow a$

```
seq:process (clk);  
begin  
    If clk'event and clk ='1' then  
        b:=c; --zmienna i sygnał  
        a<=b;  
    end if;  
end process;
```

Znajdź różnicę 2

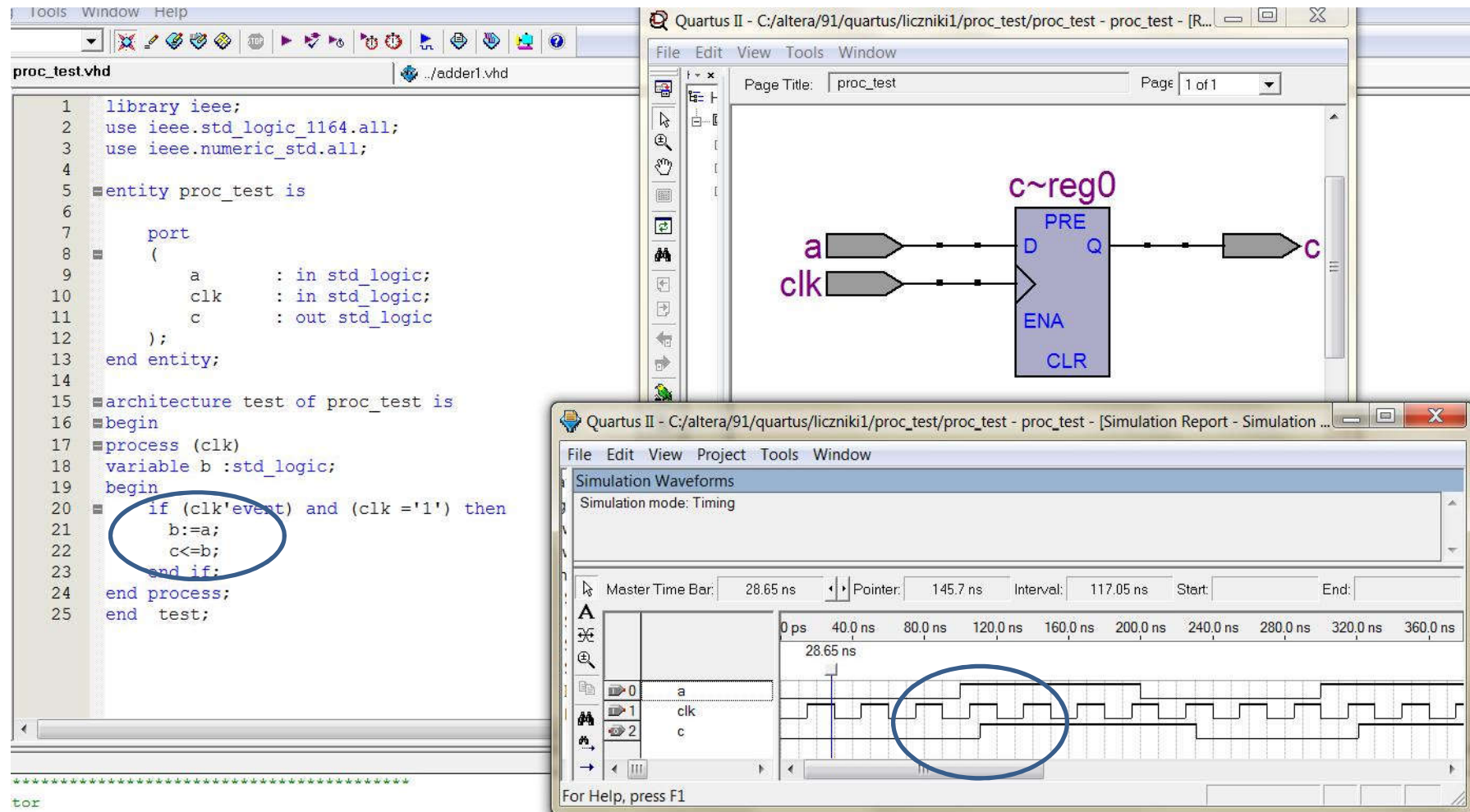
Działanie: $c \text{ clk} \Rightarrow a$

```
seq:process (clk);  
begin  
    If clk'event and clk ='1'  
        then  
            b:=c;  
            a<=b;  
        end if;  
end process;
```

Działanie: $c \text{ clk} \Rightarrow b \text{ clk} \Rightarrow a$

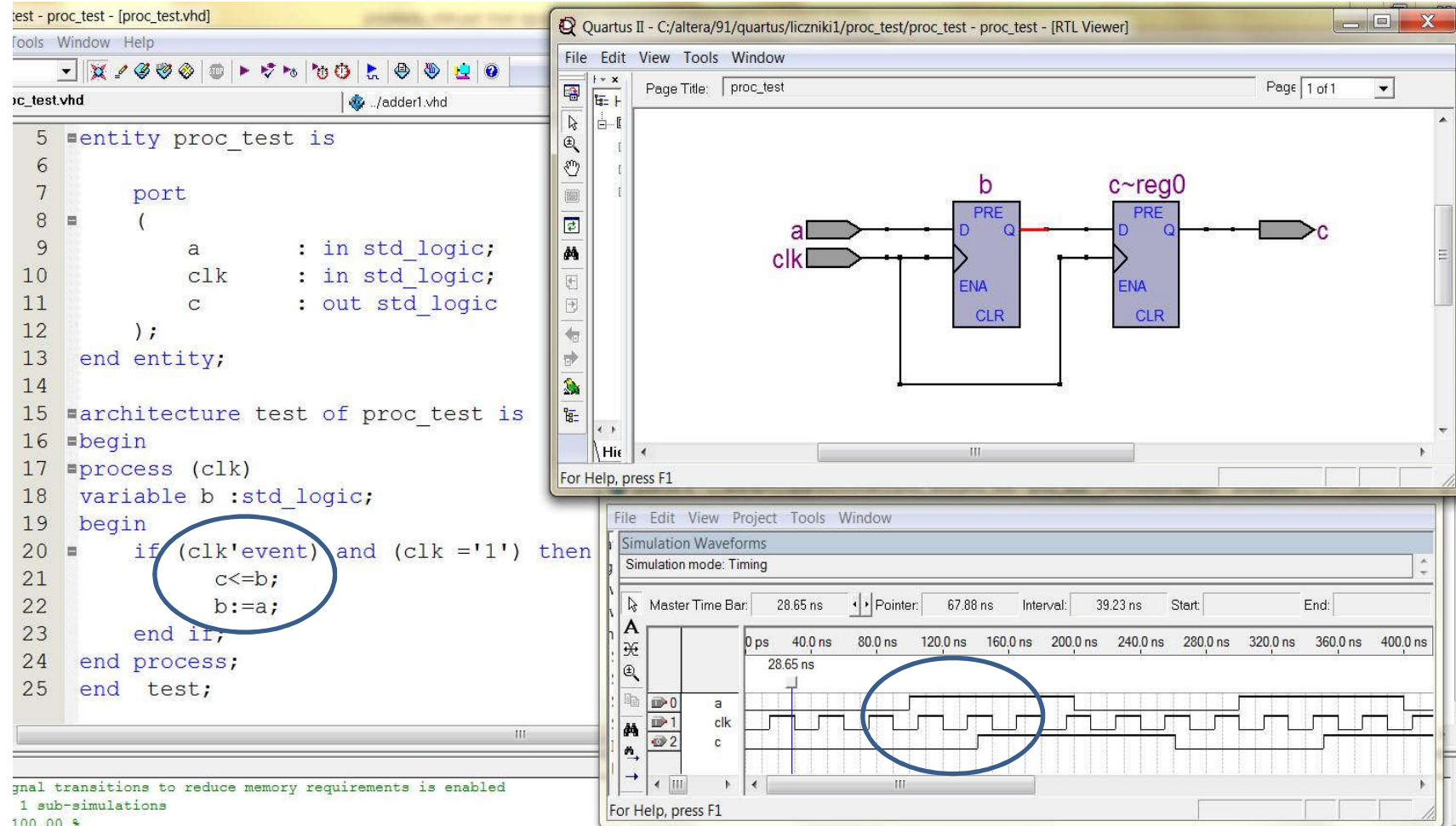
```
seq:process (clk);  
begin  
    If clk'event and clk ='1'  
        then  
            a<=b;  
            b:=c;  
        end if;  
end process;
```

Znajdź różnicę 2 – symulacja 1



Resultat symulacji i opcji **Technology map viewer – post fitting**

Znajdź różnicę 2 – symulacja 2



Resultat symulacji i opcji **Technology map viewer – post fitting**

Prosta realizacja sumatora

```
library ieee;
use ieee.std_logic_1164.all;
entity add_tab is
port(
  a, b : in std_logic_vector(7 downto 0);
  cout : out std_logic;
  suma : out std_logic_vector(7 downto 0)
);
end add_tab;

architecture arch of add_tab is
  signal tab_in : std_logic_vector(15 downto 0);
  signal tab_out : std_logic_vector(8 downto 0); -- 9 bitów wyniku
begin
  tab_in <= (a & b); -- konkatencja
  suma <= tab_out(7 downto 0);
  cout <= tab_out(8);

  PLA: process(tab_in)
  begin
    case tab_in is
      when "0000000000000000" => tab_out <= "000000000";
      when "0000000000000001" => tab_out <= "000000001";
      when "0000000000000010" => tab_out <= "000000010";
      when "0000000000000011" => tab_out <= "000000011";
      •••
      when "1111111111111110" => tab_out <= "111111101";
      when "1111111111111111" => tab_out <= "111111110";
      when others => tab_out <= „-----”; -- wartość dowolna
    end case;
  end process;
end;
```

Sumator jest logiką kombinacyjną, można zatem użyć najprostszego sposobu i opisać sumator **tablicą prawdy**

Zadanie: Zaprojektować sumator dwuargumentowy 8- bitowy

- Wejścia – A[7..0], B[7..0]
- Wyjścia – cout, suma[7..0]

Zalety: prosta koncepcja

Wady: Niestety nie jest to rozwiązanie do przyjęcia ze względu na rozmiar opisu:

- dla sumatora 8-bitowego tablica ma $2^{16} = 65\,536$ wierszy
- dla sumatora 16-bitowego tablica ma $2^{32} = 4\,294\,967\,296$ wierszy

Sumator **strukturalnie** – sumator z propagowanym przeniesieniem - rca

```
library ieee;
use ieee.std_logic_1164.all;
entity full_adder_1bit is
  port(
    a, b, cin : in std_logic;
    s, cout   : out std_logic
  );
end;
architecture arch of full_adder_1bit is
begin
  s   <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or
          ( b and cin);
end;
entity rca is
  generic( WIDTH : integer := 8 );
  port(
    a, b : in std_logic_vector(WIDTH-1 downto 0);
    cout : out std_logic;
    s    : out std_logic_vector(WIDTH-1 downto 0)
  );
end;
```

architecture arch of **rca** is

component full_adder_1bit is

```
  port(
    a, b, cin : in std_logic;
    s, cout   : out std_logic
  );
  end component;
  signal c : std_logic_vector(WIDTH downto 0);
  begin
    c(0) <= '0';
    cout <= c(WIDTH);
    Sum_wektor: for i in WIDTH-1 downto 0 generate
      full_adder_1bit
      port map(
        a    => a(i), b    => b(i),
        cin  => c(i), s    => s(i),
        cout => c(i+1)
      );
    end generate;
  end;
```

Konkretyzacja wektora sumatorów za pomocą generate.

Zastosowanie pakietu IEEE `numeric_std`

- Definiuje typ `signed` i typ `unsigned` jako `tablicę` elementów `std_logic`
- Dla `unsigned` tablica jest interpretowana w naturalnym kodzie binarnym
- Dla `signed` tablica jest interpretowana jako:
Wartości >0 znak moduł, wartości <0 znak U2
- Dla celów operacji na obiektach nowego typu przeciążono operatory: `abs`, `*`, `/`, `mod`, `rem`, `+`, `-` oraz operatory relacyjne.

Funkcje pakietu IEEE `numeric_std`:

- `shift_left(a, n)`, `shift_right(a, n)`, `rotate_left(a, n)`, `rotate_right(a, n)`
- `resize(a, n)` - zmiana rozmiaru
- `std_match(a, b)` - porównanie z uwzględnieniem stanu dowolnego '-',
wyjście typ `boolean`

Funkcje konwersji typów/rzutowanie – następna strona

Zastosowanie pakietu IEEE **numeric_std**

Funkcje konwersji typów/rzutowanie

Funkcja	Typ argumentu	Typ wyniku
std_logic_vector(a)	unsigned, signed	std_logic_vector
unsigned(a)	signed, std_logic_vector	unsigned
signed(a)	unsigned, std_logic_vector	signed
to_integer(a)	unsigned, signed	integer
to_unsigned(a, size)	natural	unsigned
to_signed(a, size)	unsigned integer	signed

Rady ogólne dla użytkowników VHDL

- Stosuj typy `std_logic` i `std_logic_vector` zamiast `bit` i `bit_vector`.
- Stosuj pakiet `numeric_std` i typy `unsigned` i `signed` dla realizacji operacji arytmetycznych.
- Stosuj malejący zakres tablic (`downto`) dla typów `unsigned`, `signed` i `std_logic_vector`.
- Stosuj nawiasów do sprecyzowania zamierzonej kolejności wykonania operacji.
- Nie używaj bezpośredniego przypisania `:=` dla nadania wartości początkowej sygnałom – przypisanie jest używane w symulacji lecz nie podlega syntezie w układzie docelowym.
- Stosuj **argumenty** o jednakowym rozmiarze dla operatorów relacyjnych – trudny do wykrycia błąd.

Sumator **behawioralnie**— operator sumowania/odejmowania wektorów

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;      -- pakiet pozwala na wykorzystanie unsigned/signed i
                                -- przeciążonych operatorów + i - do dodawania wektorów typu unsigned/signed
entity unsigned_adder_subtractor is
    generic(DATA_WIDTH : natural := 8);
    port  (a: in unsigned ((DATA_WIDTH-1) downto 0);
           b: in unsigned ((DATA_WIDTH-1) downto 0);
           add_sub : in std_logic;  result  : out unsigned ((DATA_WIDTH-1) downto 0) );
end entity;
architecture rtl of unsigned_adder_subtractor is
begin
    process(a,b,add_sub)
    begin
        -- dodawaj jeśli "add_sub" jest 1, w przeciwnym przypadku odejmij
        if (add_sub = '1') then
            result <= a + b;
        else result <= a - b;
        end if;
    end process;
end rtl;
```

Sumator – operator sumowania/odejmowania wektorów, std_logic i konwersja typów

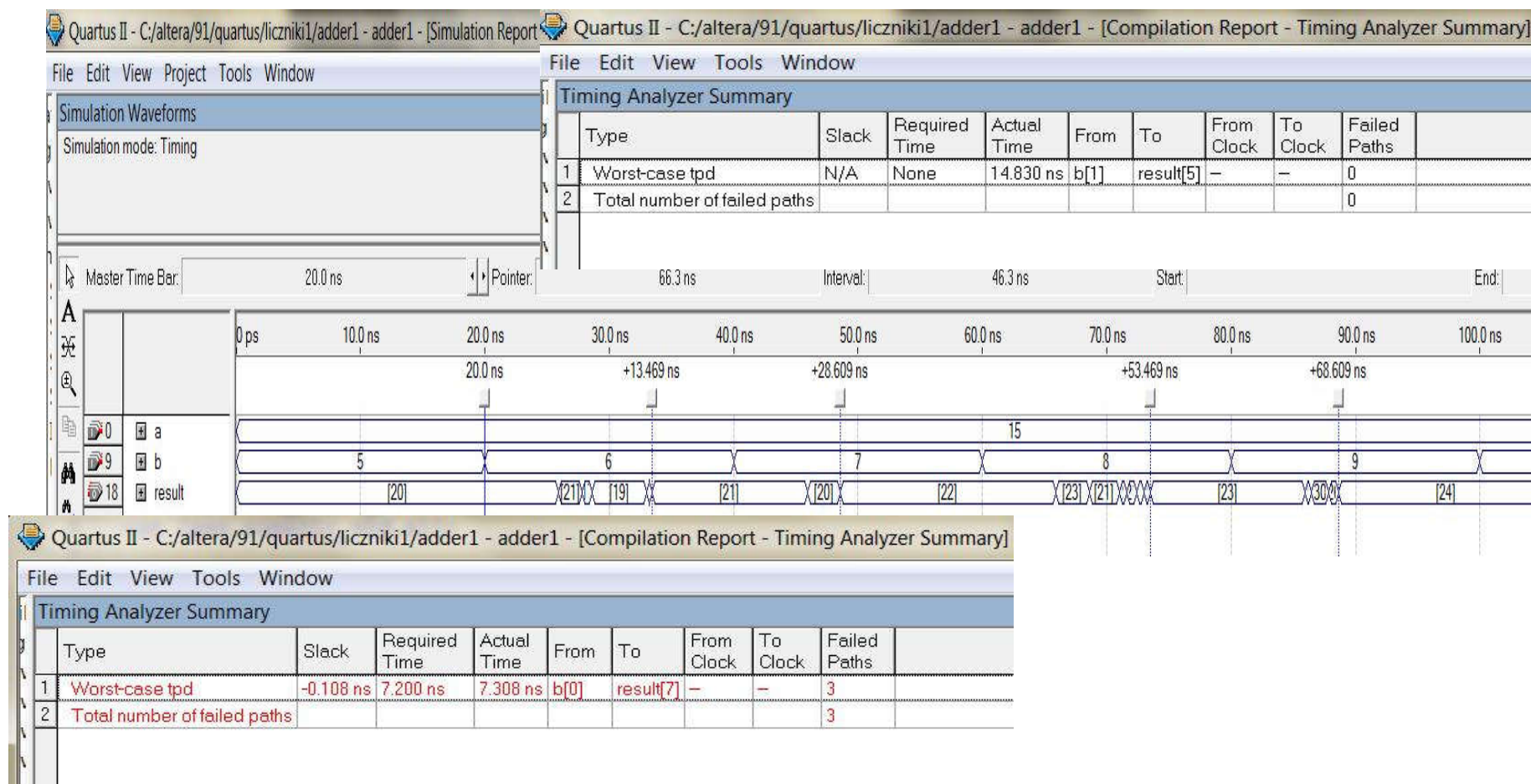
```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;           -- zastosowanie pakietu pozwala na wykorzystanie przeciążonych
                                     -- operatorów +/- do dodawania wektorów typu unsigned /signed

entity unsigned_adder_subtractor is
    generic(DATA_WIDTH : natural := 8);
    port    (a: in std_logic_vector ((DATA_WIDTH-1) downto 0);
             b: in std_logic_vector ((DATA_WIDTH-1) downto 0);
             add_sub : in std_logic;
             result   : out std_logic_vector ((DATA_WIDTH-1) downto 0) );
end entity;
architecture rtl of unsigned_adder_subtractor is
begin
    process(a,b,add_sub)
    begin
        if (add_sub = '1') then
            result <= std_logic_vector( unsigned(a) + unsigned(b) );
        else
            --zastosowanie funkcji konwersji typów dla skorzystania z operatorów +-
            result <= std_logic_vector(unsigned(a) - unsigned(b));
        end if;
    end process;
end rtl;
```

Sumator (użycie IEEE.std_logic_unsigned.all) – operator sumowania/odejmowania wektorów dla std_logic

```
library ieee; use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;      -- zastosowanie pakietu pozwala na wykorzystanie przeciążonych
-- operatorów +/-/ * i innych do dodawania wektorów typu std_logic_vector i innych typów
entity unsigned_adder_subtractor is
    generic(DATA_WIDTH : natural := 8);
    port    (a: in std_logic_vector ((DATA_WIDTH-1) downto 0);
             b: in std_logic_vector ((DATA_WIDTH-1) downto 0);
             add_sub : in std_logic;
             result   : out std_logic_vector ((DATA_WIDTH-1) downto 0) );
end entity;
architecture rtl of unsigned_adder_subtractor is
begin
    process(a,b,add_sub)
    begin
        if (add_sub = '1') then
            result <= a+b ;
        else
            result <= a-b ;
        end if;
    end process;
end rtl;
```

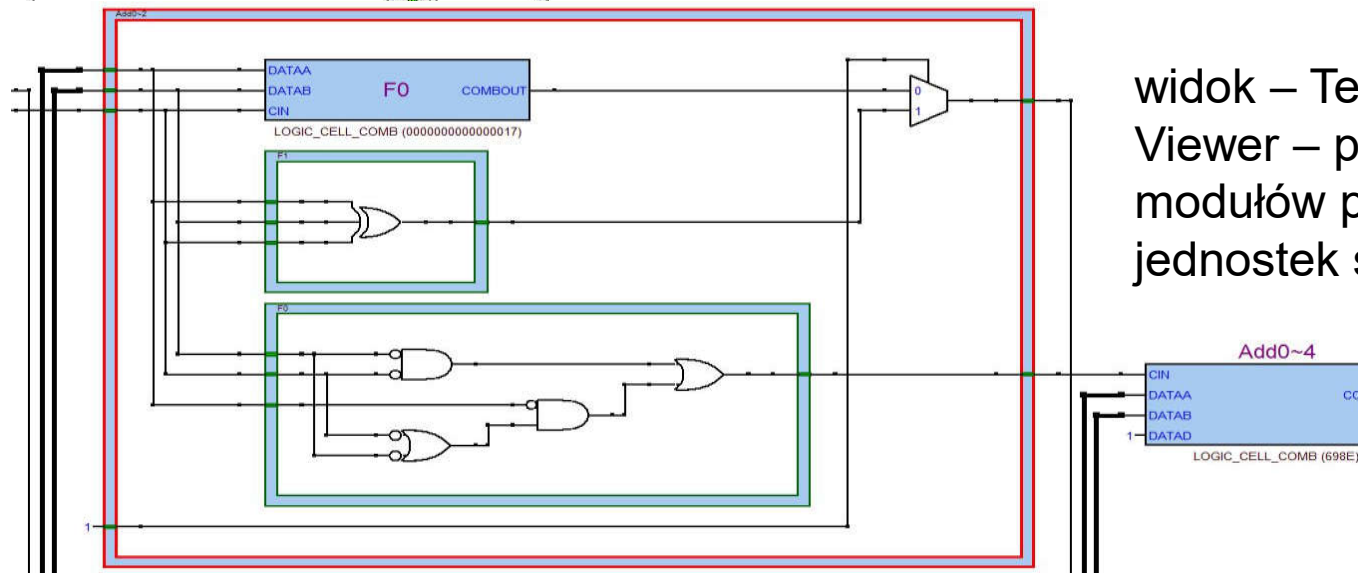
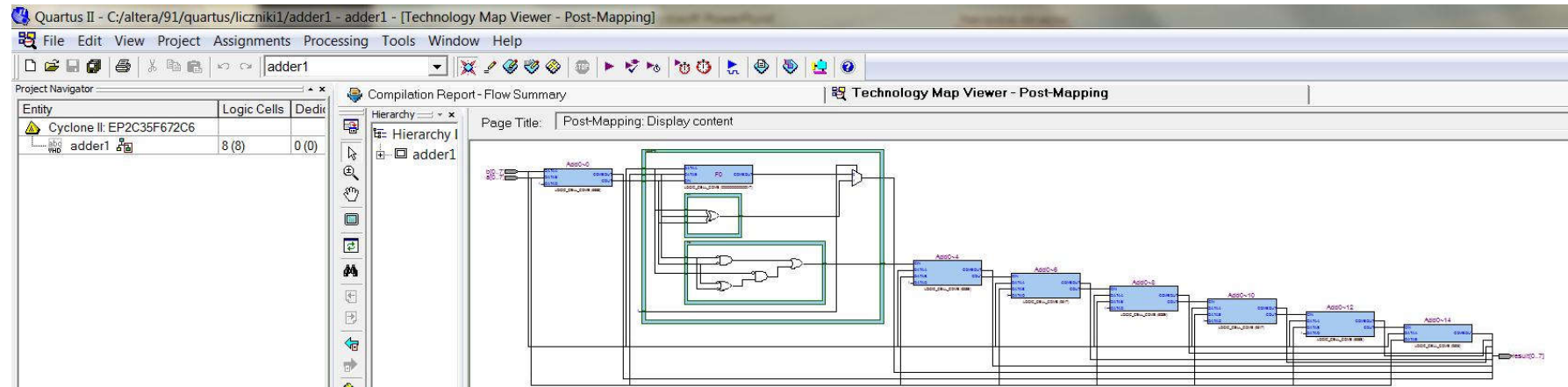
Sumator przebiegi czasowe operator sumowania/odejmowania wektorów



CZASY - Opcja View report w Classic timing analysis
tpd – czas propagacji układu kombinacyjnego – sumator 8
bitowy, rozwiązanie pierwotne i optymalizacja (14,8 ns, 7,3 ns)

Sumator realizacja

operator sumowania/odejmowania wektorów



widok – TechnologyMap Viewer – po przydzieleniu modułów projektu do jednostek sprzętu (mapping)

```

library ieee;
use ieee.std_logic_1164.all;

entity cla_adder is
  generic( WIDTH: integer := 8);
  port(
    a, b : in std_logic_vector(WIDTH-1 downto 0);
    cout : out std_logic;
    s : out std_logic_vector(WIDTH-1 downto 0)
  );
end;

```

architecture arch of cla_adder is

```

  signal c : std_logic_vector(WIDTH-1 downto 0);
  signal p : std_logic_vector(WIDTH-1 downto 0);
  signal g : std_logic_vector(WIDTH-1 downto 0);

```

CLA adder
sumator z przeniesieniem
równoległym

```

begin
  c(0) <= '0';
  cout <= g(WIDTH-1) or (p(WIDTH-1)
                        and c(WIDTH-1));

  p_g_structure:
    for i in WIDTH-1 downto 0 generate
      p(i) <= a(i) xor b(i);
      g(i) <= a(i) and b(i);
    end generate;

```

```

  carry_structure:
    for i in WIDTH-1 downto 1 generate
      c(i) <= g(i-1) or (p(i-1) and c(i-1));
    end generate;

```

```

  sum_structure:
    for i in WIDTH-1 downto 0 generate
      s(i) <= a(i) xor b(i) xor c(i);
    end generate;
end;

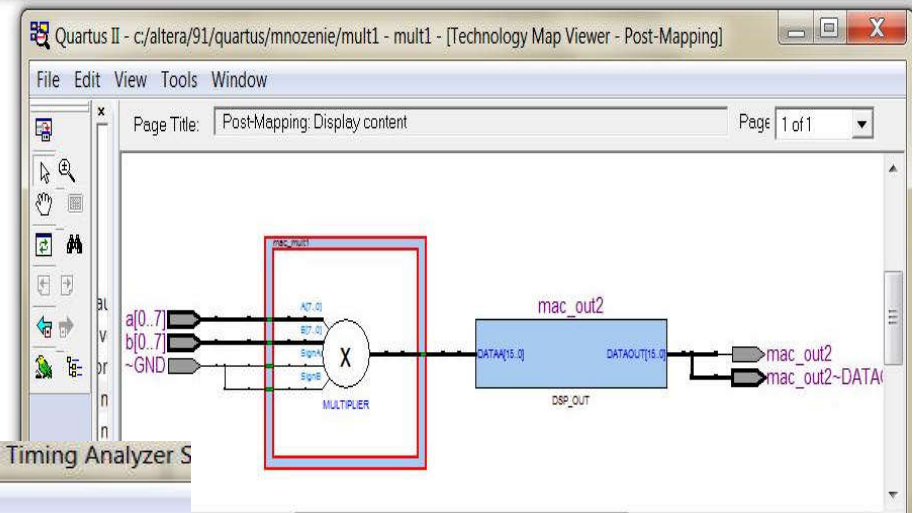
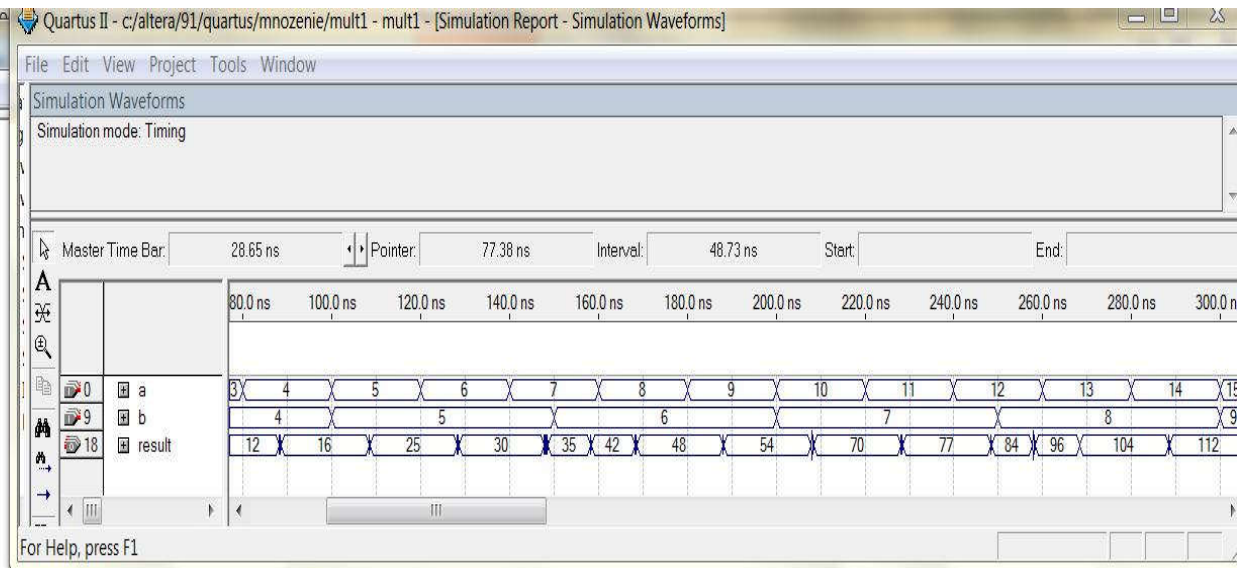
```

realizacja za pomocą wielu poziomów
bramek (Quartus,FPGA) nie pozwala na
skrócenie czasu przetwarzania w stosunku
do układu rca (sumatora z propagowanym
przeniesieniem)

```

1  -- Quartus II VHDL Template
2  -- Unsigned Multiply
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7
8  entity unsigned_multiply is
9
10     generic
11     (
12         DATA_WIDTH : natural := 8
13     );
14
15     port
16     (
17         a      : in unsigned ((DATA_WIDTH-1) downto 0);
18         b      : in unsigned ((DATA_WIDTH-1) downto 0);
19         result  : out unsigned ((2*DATA_WIDTH-1) downto 0)
20     );
21
22 end entity;
23
24 architecture rtl of unsigned_multiply is
25 begin
26
27     result <= a * b;
28
29 end rtl;
30

```



Quartus II - c:/altera/91/quartus/mnozenie/mult1 - mult1 - [Compilation Report - Timing Analyzer Summary]

Timing Analyzer Summary									
Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths	
1 Worst-case tpd	-1.762 ns	7.000 ns	8.762 ns	a[3]	result[4]	-	-	200	
2 Total number of failed paths								200	

Mnożenie przy wykorzystaniu wbudowanych w Cyclone II jednostek mnożącej – wysoka prędkość przetwarzania (8,7ns)

Sterowanie wyświetlaczem 7-segmentowym

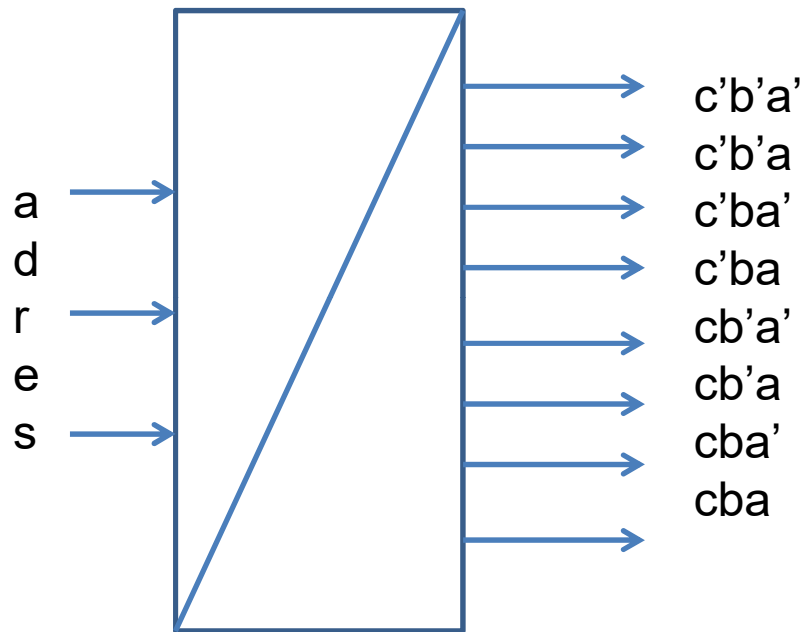
```
library ieee;
use ieee.std_logic_1164.all;

entity seven_seg_controler is
port(  x : in std_logic_vector (3 downto 0);
      a, b, c, d, e, f, g : out std_logic);
end seven_seg_controler;
```

```
architecture example of seven_seg_controler is
signal output : std_logic_vector (6 downto 0);
begin
    with x select
    output <= "1111110" when "0000",
              "0110000" when "0001",
              "1101101" when "0010",
              "1111001" when "0011",
              "0110011" when "0100",
              "1011011" when "0101",
              "1011111" when "0110",
              "1110000" when "0111",
              "1111111" when "1000",
              "1111011" when "1001",
              "-----" when others;

    (a, b, c, d, e, f, g) <= output;
    -- uwaga na poziom aktywny wyświetlacza
end example;
```

Dekoder



entity decoder is

```
port(
    adr  : in std_logic_vector (1 downto 0);
    out0, out1, out2, out3 : out std_logic
);

end decoder;
```

architecture data_flow of decoder is

```
signal output : std_logic_vector (3 downto 0);
```

```
begin
```

```
with adr select
```

```
output <= "0001" when "00",
```

```
        "0010" when "01",
```

```
        "0100" when "10",
```

```
        "1000" when "11",
```

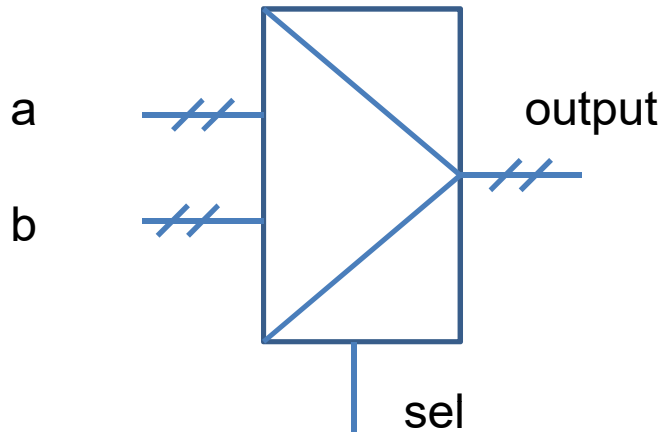
```
        "----" when others;
```

```
(out3, out2, out1, out0) <= output;
```

```
end data_flow;
```

Multiplekser-

uciążliwy opis tablicą prawdy



```
library ieee;
use ieee.std_logic_1164.all;
entity mux_2x2 is
  port(
    a, b : in std_logic_vector (1 downto 0);
    sel  : in std_logic;
    output : out std_logic_vector (1 downto 0)
  );
end mux_2x3;
```

architecture **wiele_linii_kodu** of **mux_2x2** is
 signal i : std_logic_vector (4 downto 0);

begin

i <= sel & a & b;

with i select

output <= "00" when "00000",

"01" when "00100",

"10" when "01001",

"11" when "01101",

"01" when "10001",

"--" when others;

end **wiele_linii_kodu**;

-- 32 linie kodu dla pary wektorów 2 bitowych

Multiplexer - poprawiony (?)

```
architecture malo_linii of mux_2x3 is
    signal i : std_logic_vector (4 downto 0);
begin
    i <= sel & a & b;
    with i select
        output <=  "00" when "000--",
                   "01" when "001--",
                   „10" when "010--",
                   „11" when "011--",
                   "00" when "1--00",
                   "01" when "1--01",
                   „10" when "1--10",
                   „11" when "1--11",
                   "--" when others;
end malo_linii ;
```

Kompilacja

Warning (10325): VHDL Choice warning at
multip.vhd(14): ignored choice containing
meta-value ""000--""

Warning (10325): VHDL Choice warning at
multip.vhd(15): ignored choice containing
meta-value ""001--""

Wartość – *dowolna* nie występuje w algebrze
Boola, lecz może (poprawna składniowo) być
wykorzystana przy definicji projektu.
Występująca na wyjściach pozwala na
optymalizację – wartość ta może być dowolna.
Na wejściu w rzeczywistości nie występuje i
dlatego nie zostanie uwzględniona przez
program CAD.

Efektem realizacji jest układ z dowolnymi
wartościami na wyjściu, np. stałe zera.

Równoważne z :

with i select

output <= "--" when others;

Multiplekser - efektywnie

```
library ieee;
use ieee.std_logic_1164.all;
entity mux_2x2 is
  port(
    a, b : in std_logic_vector (1 downto 0);
    sel : in std_logic;
    output :
      out std_logic_vector (1 downto 0)
  );
end mux_2x2;
```

```
architecture data_flow1 of mux_2x2 is
begin
  with sel select
    o <=  a      when '0',
          b      when others;
end data_flow1;
-- lub wariant 2
architecture data_flow2 of mux_2x2 is
begin
  o <=  a when (sel = '0') else b;
end data_flow2;
```

ALU – jednostka arytmetyczno - logiczna

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- +- dla signed/unsigned
entity myALU is
    port(
        code    : in std_logic_vector (1 downto 0);
        arg1, arg2 : in std_logic_vector (7 downto 0);
        result    : out std_logic_vector (7 downto 0);
    end myALU;
```

```
architecture data_flow of myALU is
    signal sum, diff : std_logic_vector (7 downto 0);
```

```
begin
```

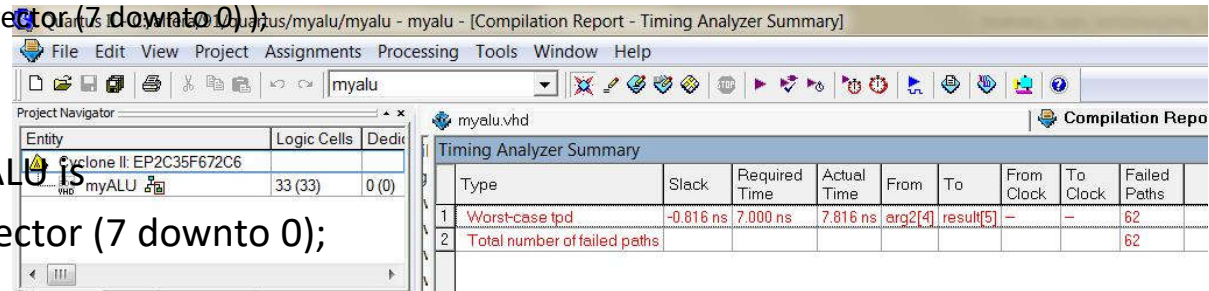
```
    sum <= std_logic_vector(signed(arg1) + signed(arg2));
```

```
    diff <= std_logic_vector(signed(arg1) - signed(arg2));
```

```
    with code select
```

```
        result <= sum          when "00",
                diff          when "01",
        arg1 and arg2          when "10",
        arg1 or arg2           when others;
```

```
end data_flow;
```



Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1 Worst-case tpd	-0.816 ns	7.000 ns	7.816 ns	arg2[4]	result[5]	—	—	62
2 Total number of failed paths								62

code	działanie
00	dodawanie
01	odejmowanie
10	iloczyn logiczny bitów
11	suma logiczna bitów

Rejestr uniwersalny

- Synchroniczny
- Zerowanie asynchroniczne
- Wpis równoległy (wejście danych)
- Pamiętanie
- Przesunięcie w prawo (najstarszy bit - wejście szeregowe)
- Przesunięcie w lewo (najmłodszy bit - wejście szeregowe)
- Wejście sterowania trybem

Rejestr uniwersalny

Jedno podstawienie q_next oraz sterowane wartością sygnału trybu – bez procesu w opisie działania odzwierciedlenie sposobu działania przerzutnika

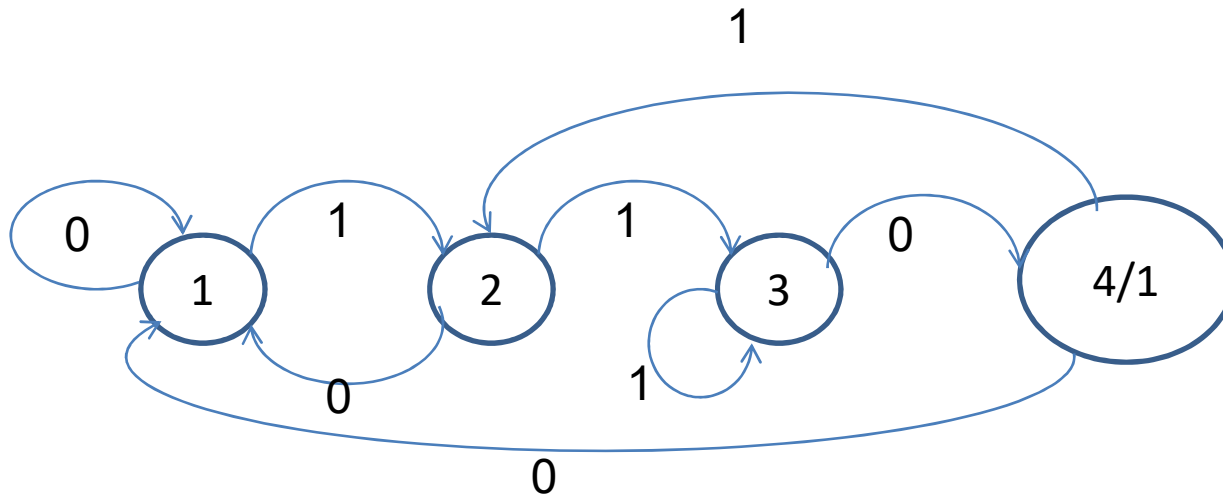
```
library ieee;
use ieee.std_logic_1164.all;
entity universal_reg is
port(
    clk : in std_logic;
    rst : in std_logic;
    mode : in std_logic_vector (1 downto 0);
    d : in std_logic_vector (7 downto 0);
    q : out std_logic_vector (7 downto 0)
);
end universal_reg;
architecture behavior of universal_reg is
constant LOAD : std_logic_vector (1 downto 0) := "11";
constant SHIFT_R : std_logic_vector (1 downto 0) := "10";
constant SHIFT_L : std_logic_vector (1 downto 0) := "01";
constant HOLD : std_logic_vector (1 downto 0) := "00";

signal q_reg, q_next : std_logic_vector (7 downto 0);
-- Dla udostępnienia wyjścia – sygnały niewidoczne na zewnątrz
```

```
Begin -- trzy bloki II
process (clk, rst)
begin
    if (rst = '1') then
        q_reg <= (others => '0');
    elsif rising_edge(clk) then
        -- przepisanie zboczem
        q_reg <= q_next;
    end if;
end process; -- równolegle z
with mode select -- aktualna nowa wartość
    q_next <= d when LOAD,
        d(7) & q_reg(7 downto 1) when
            SHIFT_R,
        q_reg(6 downto 0) & d(0) when
            SHIFT_L,
        q_reg when HOLD,
        q_reg when others;
q <= q_reg; -- q sygnał w trybie out
end behavior;
```

Automat wykrywający ciąg 110 realizacja wielosegmentowa (A.Moora)

- Niezależne moduły:
 - Analiza wejścia i ustalenie **stanu kolejnego** automatu
 - Określenie stanu wyjścia na podstawie stanu bieżącego
 - Przejście do stanu kolejnego



Automat wykrywający ciąg 110

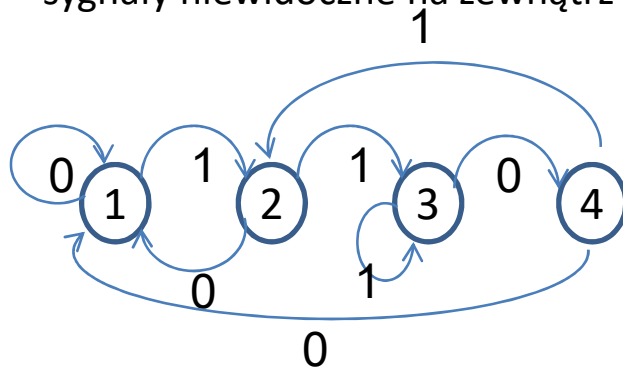
realizacja wielosegmentowa

(A.Moora)

```

library ieee;
use ieee.std_logic_1164.all;
entity sequence_search is
port(
    clk : in std_logic;
    a_rst : in std_logic; -- priorytet
    s_rst: in std_logic;
    stream : in std_logic;
    found : out std_logic
);
end sequence_search;
architecture behaviour of sequence_search is
    type STATE_TYPE is (s1, s2, s3,s4)
    signal state_reg, state_next : STATE_TYPE;
    -- sygnały niewidoczne na zewnątrz

```



```

begin
    -- wyznaczenie bieżącego stanu
    process (clk, a_rst)
    begin
        if (a_rst = '1') then -- reset asynchroniczny
            state_reg <= s1;
        elsif rising_edge(clk) then
            state_reg <= state_next;
        -- na podstawie stanu „przyszłego”
        end if;
    end process;

```

```
process (state_reg, stream) -- wyznaczenie nowego --  
    stanu  
    -- możliwość wyznaczenia wyjść zależnych od  
    -- stanu i wejść (Mealy'ego), brak zegara na liście cz.
```

```
begin
```

```
    if s_rst = '1' then -- reset synchroniczny
```

```
        state_next <= s1;
```

```
    else
```

```
        case state_reg is
```

```
            when s1 =>
```

```
                if (stream = '1') then
```

```
                    state_next <= s2;
```

```
                else
```

```
                    state_next <= s1;
```

```
                end if;
```

```
            when s2 =>
```

```
                if (stream = '1') then
```

```
                    state_next <= s3;
```

```
                else
```

```
                    state_next <= s1;
```

```
                end if;
```

Automat wykrywający ciąg 110 cd

```
        when s3 =>
```

```
            if (stream = '1') then
```

```
                state_next <= s3;
```

```
            else
```

```
                state_next <= s4;
```

```
            end if;
```

```
        when s4 =>
```

```
            if (stream = '1') then
```

```
                state_next <= s2;
```

```
            else
```

```
                state_next <= s0;
```

```
            end if;
```

```
        end case;
```

```
    endif;
```

```
end process;
```

Automat wykrywający ciąg 110 cd

```
process (state_reg) -- wyznaczenie  
                        -- wyjść
```

```
begin  
    case state_reg is  
        when s4 =>  
            found <= '1';  
        when others  
            found <= '0';  
    end process;  
end behaviour;
```

Kodowanie stanów:

Możliwe bezpośrednie przypisanie kodowania stanom automatu poprzez podanie wartości kodowej i wykorzystanie typu 'std_logic_vector'

Np. :

```
constant s1 : std_logic_vector (2 downto 0) := "01";  
constant s2 : std_logic_vector (2 downto 0) := "11";  
constant s3 : std_logic_vector (2 downto 0) := „10”;  
constant s4 : std_logic_vector (2 downto 0) := „00”;  
signal state_reg, state_next : std_logic_vector (2  
    downto 0);
```

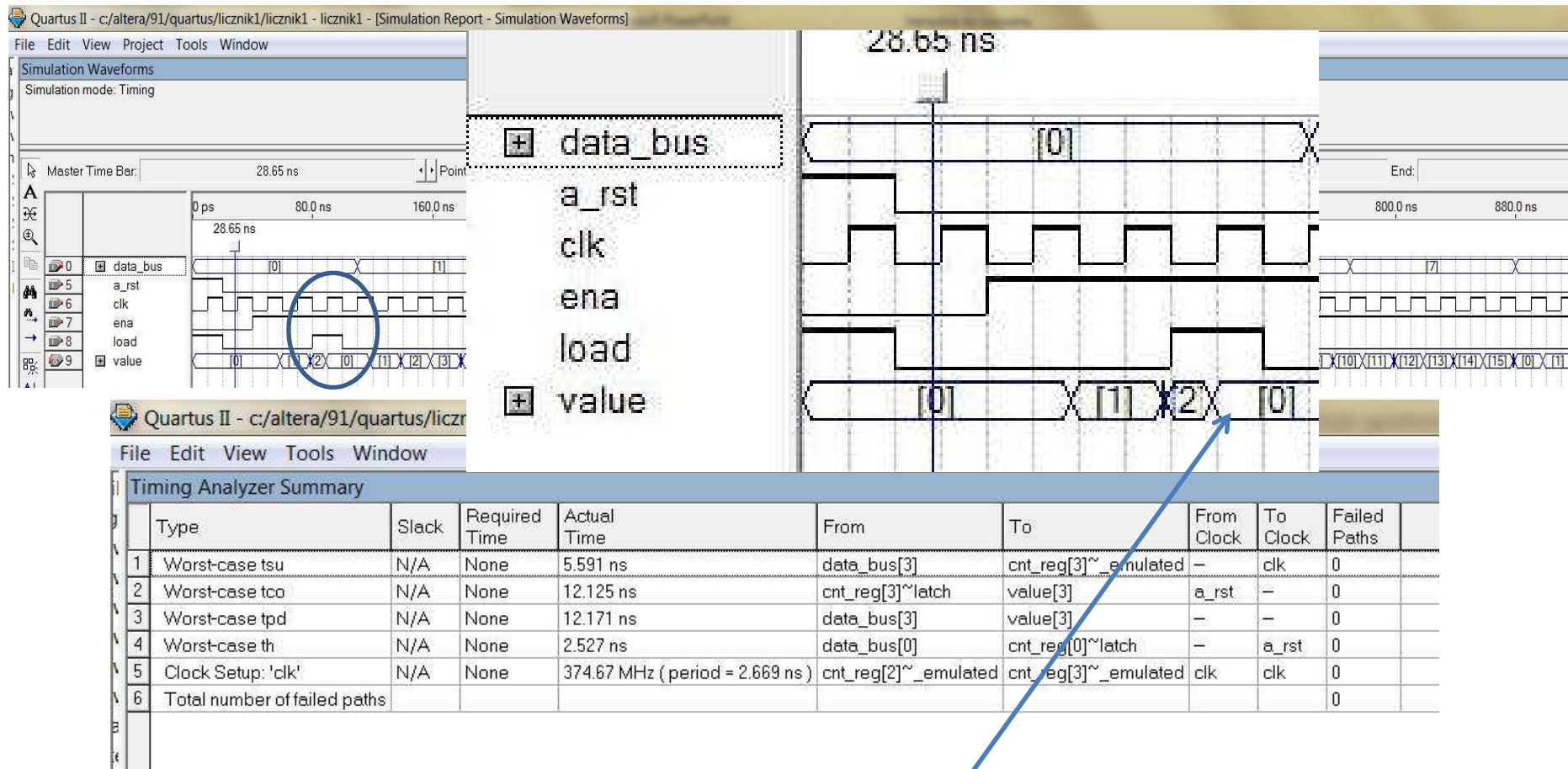
Licznik NKB

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity cnt_4b is
    port( clk      : in std_logic;
          a_rst    : in std_logic;
          s_load   : in std_logic;
          ena      : in std_logic;
          data_bus  : in std_logic_vector(3 downto 0);
          value     : out std_logic_vector(3 downto 0)
    );
end cnt_4b;

-- zliczanie realizowane jest z wykorzystaniem
sumatora (+ dla unsigned)
```

```
architecture behaviour of cnt_4b is
    signal cnt_reg : unsigned(3 downto 0);
begin
    process (clk, a_rst)
    begin
        if (a_rst = '1') then
            cnt_reg <= (others => '0');
        elsif (rising_edge(clk)) then
            if (s_load = '1') then
                cnt_reg <= unsigned(data_bus);
            elsif (ena = '1') then
                cnt_reg <= cnt_reg + 1;
            end if;
        end if;
    end process;
    value <= std_logic_vector(cnt_reg);
    -- tutaj generacja wyjścia stanu pełnego licznika (if
    -- proces)
end behaviour ;
```

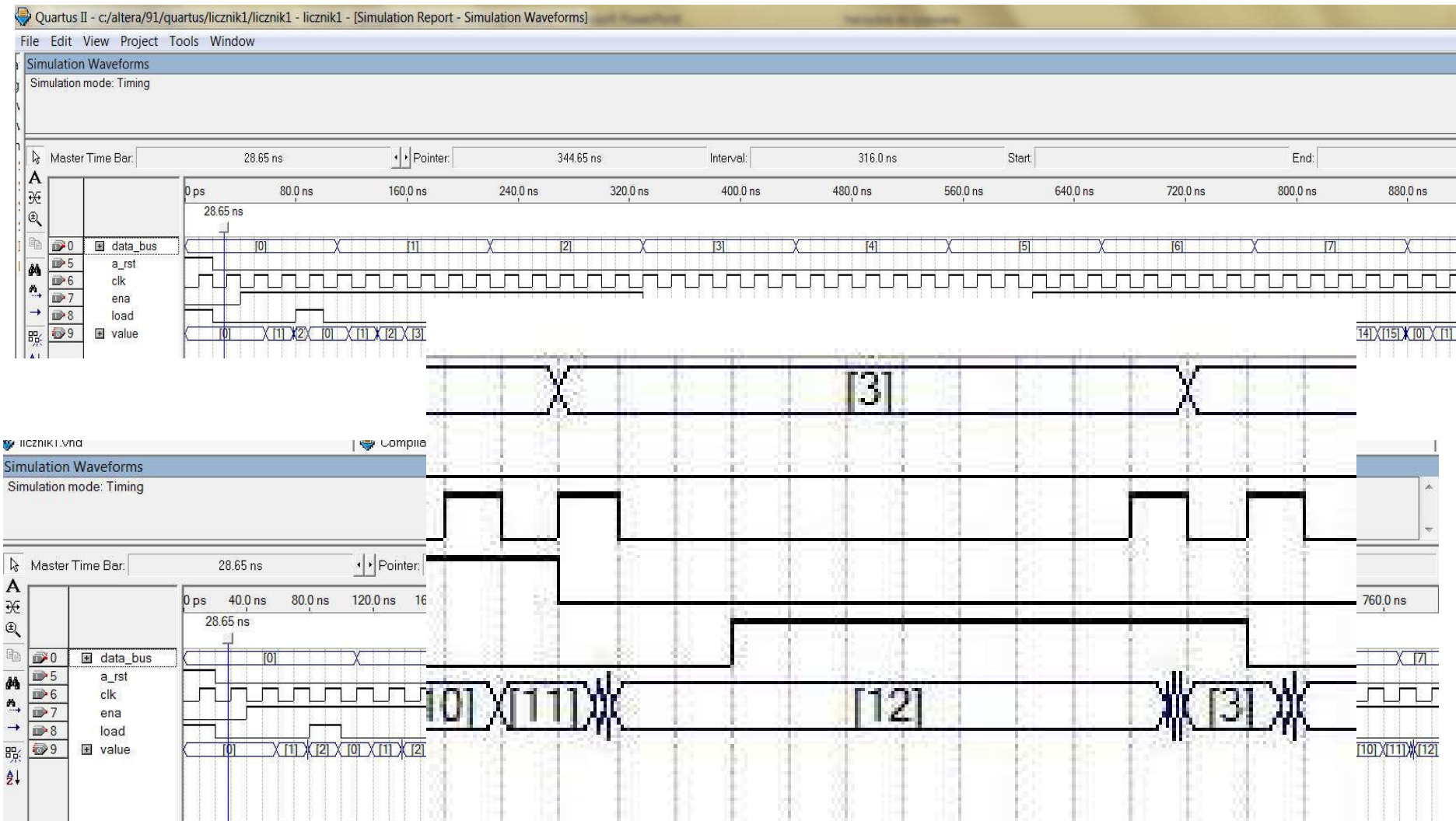
Symulacja licznika mod 16



Czy to jest licznik z poprzedniej strony ?

Nie – ładowanie tutaj jest chyba asynchroniczne.

Licznik z asynchronicznym resetem i synchronicznym wpisem



Zasady realizacji logiki sekwencyjnej w VHDL

- Wszystkie **rejestry** powinny być taktowane tym samym sygnałem zegarowym (**synchronicznie**)
- Należy oddzielać elementy pamięciowe i opisywać je w oddzielnych segmentach.
- Należy **unikać** jednosegmentowego stylu opisu automatu.
- Elementy pamięciowe powinny być opisywane w prosty sposób tak, aby narzędzia syntezy wykorzystywały elementy biblioteczne do ich realizacji.
- Reset **asynchroniczny** powinien być wykorzystywany jedynie do inicjalizacji systemu.
- Do zerowania/ustawiania rejestrów w czasie pracy należy używać sygnału **synchronicznego**.