POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Układy cyfrowe i systemy wbudowane

Laboratorium 5

Układy Kombinacyjne i Sekwencyjne w VHDL-u

Termin zajęć: Czwartek TP, 7:30

Autorzy:
Daria Jeżowska, 252731
Kacper Śleziak, 252703

Prowadzący zajęcia:
dr inż. Jacek Mazurkiewicz

Zadania do wykonania

1. Implementacja funkcji logicznej – każda grupa realizuje właściwą funkcję z Lab 2.
   G(w,x,y,z) = Π(0, 2, 3, 4, 6, 7, 9, 11, 12, 13, 15)
2. Implementacja układu translatora kodu z Lab 2.
   Każde zadanie realizujemy w VHDL-u na dwa sposoby: - poprzez zapis równań boolowskich, - metodą zapisu tablicowego.
3. Implementacja układu detektora sekwencji z Lab 3.
4. Implementacja układu licznika z Lab 3. Każde zadanie realizujemy w VHDL-u jako maszynę stanów.

Zadanie 1

Implementacja funkcji logicznej – G(w,x,y,z) = Π(0, 2, 3, 4, 6, 7, 9, 11, 12, 13, 15)

Tabela prawdy oraz minimalizacja siatką Karnaugh

	w	x	y	z	S
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

wx\yz	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	0	0	1
10	1	0	0	1

$$S = \overline{w}\overline{y}z + w\overline{x}\overline{z} + wy\bar{z}$$

Kod w języku VHDL

W poniższym kodzie wyjście S został zapisany za pomocą równań boolowskich, czyli zapisanie jako wzór na podstawie powyższego wzoru stworzonego na podstawie siatki Karnaugh. Natomiast S2 - metodą zapisu tablicowego, która jest na podstawie powyższej tabeli prawdy. Dodatkowo musieliśmy użyć biblioteki IEEE.NUMERIC_STD, aby kod działał poprawnie,

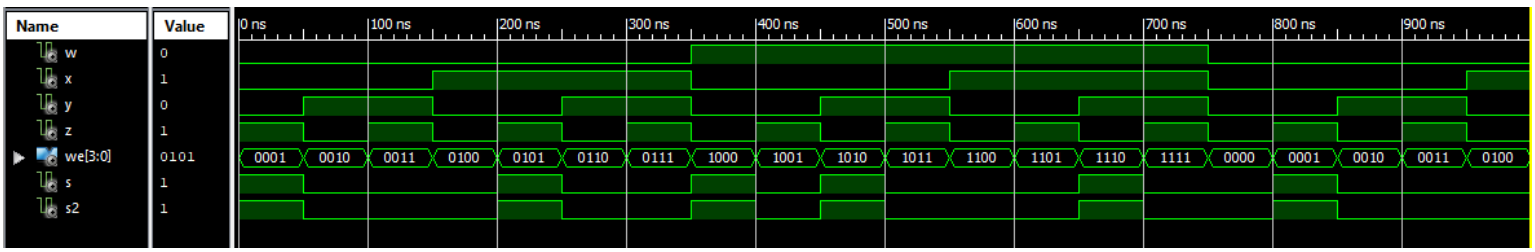
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity lab1 is
    Port ( S : out  STD_LOGIC;
           S2 : out  STD_LOGIC;
           W : in  STD_LOGIC;
           X : in  STD_LOGIC;
           Y : in  STD_LOGIC;
           Z : in  STD_LOGIC;
           We : in  STD_LOGIC_VECTOR (3 downto 0));
end lab1;

architecture Behavioral of lab1 is

begin

    S <= (not W and not Y and Z) or (W and not X and not Z) or (W and Y and not Z);

    with We select
        S2 <= '1' when "0001" | "0101" | "1000" | "1010" | "1110" ,
              '0' when others;

end Behavioral;
```

Testbench i symulacja

Aby przetestować układ w symulacji musieliśmy stworzyć testbench. Sygnały wejściowe na start ustawione zostały na zero. Następnie w pętli for, działającej w zakresie od 1 do 16 ustawiamy wartość sygnałów. Wejście Z zmienia się najczęściej, więc zmienia się z każdym obrotem pętli. Natomiast pozostałe sygnały – co 2 (wejście Y), 4 (wejście X) i 8 (wejście W) taktów zegara. Natomiast dla wejścia zapisanego w formie wektora przypisujemy kolejne wartości iteratora i. Symulacja przebiegła poprawnie i jak możemy zobaczyć dla danych WXYZ – We – 1110 wyjścia S i S2 jest równe 1. Natomiast dla danych WXYZ – We – 0110 wyjścia S i S2 są równe 0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY zad1tb IS
END zad1tb;

ARCHITECTURE behavior OF zad1tb IS

    COMPONENT lab1
    PORT(
         S : OUT  std_logic;
         S2 : OUT  std_logic;
         W : IN  std_logic;
         X : IN  std_logic;
         Y : IN  std_logic;
         Z : IN  std_logic;
         We : IN  std_logic_vector(3 downto 0)
        );
    END COMPONENT;

   signal W : std_logic := '0';
   signal X : std_logic := '0';
   signal Y : std_logic := '0';
   signal Z : std_logic := '0';
   signal We : std_logic_vector(3 downto 0) := (others => '0');

   signal S : std_logic;
   signal S2 : std_logic;

   constant <clock>_period : time := 10 ns;
```

```
BEGIN

   uut: lab1 PORT MAP (
          S => S,
          S2 => S2,
          W => W,
          X => X,
          Y => Y,
          Z => Z,
          We => We
        );

   <clock>_process :process
   begin
   for i in 1 to 16 loop
      Z <= not Z;

      if (i mod 2 = 0) then
         Y <= not Y;
      end if;

      if (i mod 4 = 0) then
         X <= not X;
      end if;

      if (i mod 8 = 0) then
         W <= not W;
      end if;

      We <= STD_LOGIC_VECTOR(to_unsigned(i,4));
      wait for 50ns;

   end loop;

END;
```



Plik UCF i testowanie na płytce

Pierwsze cztery klawisze zostały przypisane do wektora We, a pozostałe cztery kolejno do W, X, Y oraz Z. Aby można było odczytać sygnały wyjściowe S2 przypisaliśmy dla pierwszej diody led, a S dla czwartej. Program po załadowaniu na płytkę działał zgodnie z założeniami. Mieliśmy jedynie problem, ponieważ klawisz przypisany do sygnału Z nie zawsze działał, jednakże ostatecznie udało nam się wszystko przetestować.

```
#------------------------------------------------------------
#  ZL-9572 CPLD board,  J.Sugier 2009
#------------------------------------------------------------

# Clocks
#NET "Clk_LF" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "We(0)" LOC = "P42";
NET "We(1)" LOC = "P40";
NET "We(2)" LOC = "P43";
NET "We(3)" LOC = "P38";
NET "W" LOC = "P37";
NET "X" LOC = "P36";  # shared with ROT_A
NET "Y" LOC = "P24";  # shared with ROT_B
NET "Z" LOC = "P39";  # GSR

# LEDS
NET "S2"  LOC = "P35";
#NET "LED<1>"  LOC = "P29";
#NET "LED<2>"  LOC = "P33";
#NET "LED<3>"  LOC = "P34";
NET "S"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

ZADANIE 2

Implementacja układu translatora kodu z Lab 2

Każde zadanie realizujemy w VHDL-u na dwa sposoby: - poprzez zapis równań boolowskich, - metodą zapisu tablicowego.

Tabela prawdy oraz minimalizacja siatkami Karnaugh

Dziesiętnie	NKB				Aikena (2421)			
	a	b	c	d	A	B	C	D
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1
10	1	0	1	0	-	-	-	-
11	1	0	1	1	-	-	-	-
12	1	1	0	0	-	-	-	-
13	1	1	0	1	-	-	-	-
14	1	1	1	0	-	-	-	-
15	1	1	1	1	-	-	-	-

A

ab\cd	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	-	-	-	-
10	1	1	-	-

$A = a + bd + bc = b(d+c) + a$

B

ab\cd	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	-	-	-	-
10	1	1	-	-

$B = b\bar{d} + a + bc = a + b(c + \bar{d})$

C

ab\cd	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	-	-	-	-
10	1	1	-	-

$C = a + b\bar{c}d + \bar{a}\bar{b}c$

D

ab\cd	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	-	-	-	-
10	0	1	-	-

$D = d$

Kod w języku VHDL

Aby zdefiniować wyjścia WyA, WyB, WyC, WyD wykorzystaliśmy funkcje boolowskie. Wyjścia WyA2, WyB2, WyC2 oraz WyD2 określiliśmy przy pomocy instrukcji with select, gdzie w zależności od konkretnej kombinacji zer i jedynek na wejściu nasze wyjście przyjmuje wartość 1. Aby uniknąć pisania wszystkich przypadków dla, których wyjście jest równe 0 użyliśmy instrukcji when others.

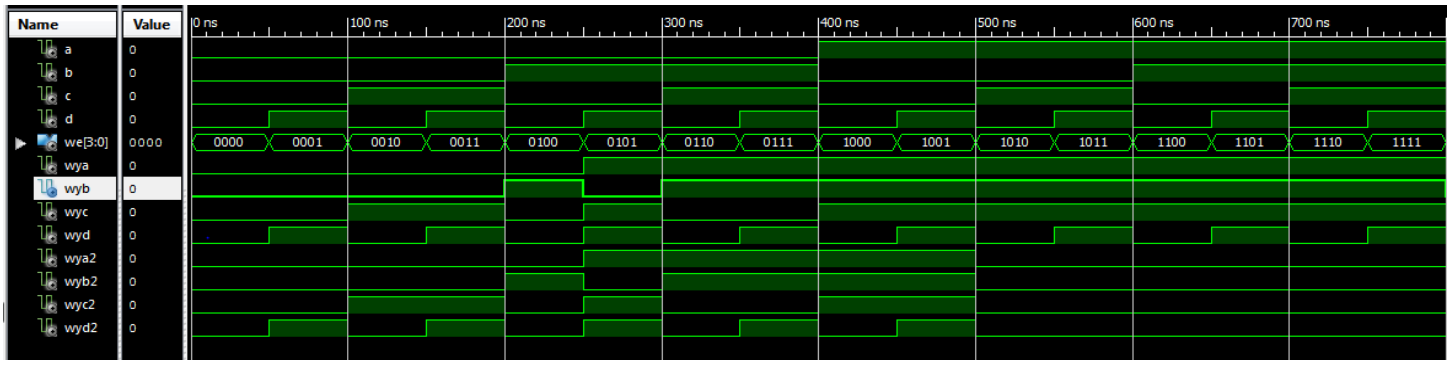
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity zad2 is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           We : in  STD_LOGIC_VECTOR (3 downto 0);
           WyA : out  STD_LOGIC;
           WyB : out  STD_LOGIC;
           WyC : out  STD_LOGIC;
           WyD : out  STD_LOGIC;
           WyA2 : out  STD_LOGIC;
           WyB2 : out  STD_LOGIC;
           WyC2 : out  STD_LOGIC;
           WyD2 : out  STD_LOGIC);
end zad2;

architecture Behavioral of zad2 is

begin

    WyA <= A or (B and D) or (B and C);
    WyB <= (B and not D) or A or (B and C);
    WyC <= A or (B and not C and D) or (not A and not B and C);
    WyD <= D;

    with We select
        WyA2 <= '1' when "0101" | "0110" | "0111" | "1000" | "1001" ,
                '0' when others;
    with We select
        WyB2 <= '1' when "0100" | "0110" | "0111" | "1000" | "1001"  ,
                '0' when others;
    with We select
        WyC2 <= '1' when "0010" | "0011" | "0101" | "1000" | "1001"  ,
                '0' when others;
    with We select
        WyD2 <= '1' when "0001" | "0011" | "0101" | "0111" | "1001"  ,
                '0' when others;

end Behavioral;
```

Testbench i symulacja

Aby przetestować układ w symulacji musieliśmy stworzyć testbench. Sygnały wejściowe na start ustawione zostały na zero. Następnie w pętli for, działającej w zakresie od 1 do 16 ustawiamy wartość sygnałów. Wejście D zmienia się najczęściej, więc zmienia się z każdym obrotem pętli. Natomiast pozostałe sygnały – co 2 (wejście C), 4 (wejście B) i 8 (wejście A) taktów zegara. Natomiast dla wejścia zapisanego w formie wektora przypisujemy kolejne wartości iteratora i. Na symulacji możemy zobaczyć wynik działania kodu – dla 0110 w NKB wynikiem jest 1100.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY zad2tb IS
END zad2tb;

ARCHITECTURE behavior OF zad2tb IS

    COMPONENT zad2
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         C : IN  std_logic;
         D : IN  std_logic;
         We : IN  std_logic_vector(3 downto 0);
         WyA : OUT  std_logic;
         WyB : OUT  std_logic;
         WyC : OUT  std_logic;
         WyD : OUT  std_logic;
         WyA2 : OUT  std_logic;
         WyB2 : OUT  std_logic;
         WyC2 : OUT  std_logic;
         WyD2 : OUT  std_logic
        );
    END COMPONENT;

   signal A : std_logic := '0';
   signal B : std_logic := '0';
   signal C : std_logic := '0';
   signal D : std_logic := '0';
   signal We : std_logic_vector(3 downto 0) := (others => '0');

   signal WyA : std_logic;
   signal WyB : std_logic;
   signal WyC : std_logic;
   signal WyD : std_logic;
   signal WyA2 : std_logic;
   signal WyB2 : std_logic;
   signal WyC2 : std_logic;
   signal WyD2 : std_logic;
```

```
BEGIN

   uut: zad2 PORT MAP (
          A => A,
          B => B,
          C => C,
          D => D,
          We => We,
          WyA => WyA,
          WyB => WyB,
          WyC => WyC,
          WyD => WyD,
          WyA2 => WyA2,
          WyB2 => WyB2,
          WyC2 => WyC2,
          WyD2 => WyD2
        );

   stim_proc: process
   begin
      D <= '0';
      C <= '0';
      B <= '0';
      A <= '0';
      wait for 50ns;

      for i in 1 to 16 loop
         D <= not D;
         if (i mod 2 = 0) then
            C <= not C;
         end if;
         if(i mod 4 = 0) then
            B <= not B;
         end if;
         if(i mod 8 = 0) then
            A <= not A;
         end if;
         We <= STD_LOGIC_VECTOR(to_unsigned(i,4));
         wait for 50 ns;
      end loop;
      wait;
   end process;

END;
```



Plik UCF

Pierwsze cztery klawisze zostały przypisane do wektora We. Jako wyjścia przypisaliśmy diody p35, p29, p33, p34.

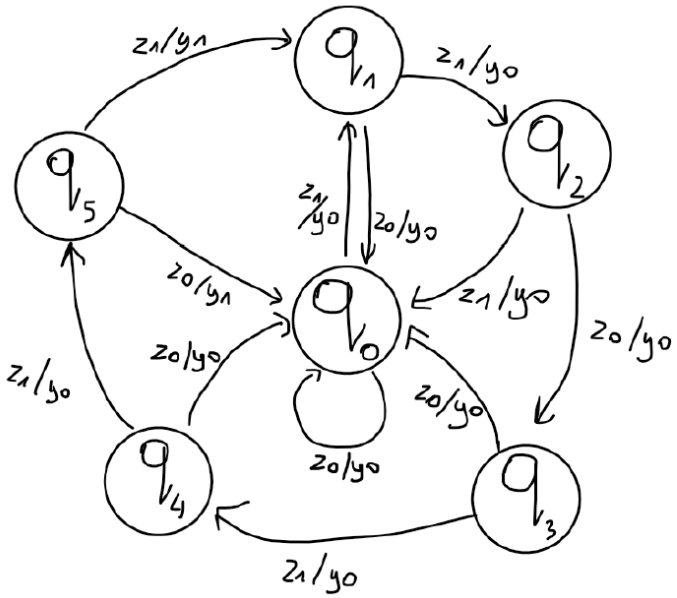
```
#------------------------------------------------------------
#  ZL-9572 CPLD board,  J.Sugier 2009
#------------------------------------------------------------

# Clocks
#NET "Clk_LF" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "We(0)" LOC = "P42";
NET "We(1)" LOC = "P40";
NET "We(2)" LOC = "P43";
NET "We(3)" LOC = "P38";
#NET "We(0)" LOC = "P37";
#NET "We(0)" LOC = "P36";  # shared with ROT_A
#NET "We(0)" LOC = "P24";  # shared with ROT_B
#NET "We(0)" LOC = "P39";  # GSR

# LEDS
NET "WyD"  LOC = "P35";
NET "WyC"  LOC = "P29";
NET "WyB"  LOC = "P33";
NET "WyA"  LOC = "P34";
#NET "LED<4>"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

Zadanie 3

Implementacja układu detektora sekwencji z Lab 3

Detektor sekwencji **11011** na podstawie automatu Mealy'ego

Automat Mealy'ego oraz tabela przejść



Alfabet wejściowy:
z_1 - na wejściu dostarczamy 1
z_0 - na wejściu dostarczamy 0

Alfabet wyjściowy:
y_1 - na wyjściu otrzymujemy 1, co oznacza, że sekwencja została wykryta
y_0 - na wyjściu otrzymujemy 0, co oznacza, że sekwencja nie została wykryta

Stany wewnętrzne
q_0 - stan początkowy
q_1 - stan do którego przechodzimy po wykryciu pierwszej w sekwencji jedynki
q_2 - stan do którego przechodzimy po wykryciu drugiej w sekwencji jedynki
q_3 - stan do którego przechodzimy po wykryciu pierwszego w sekwencji zera
q_4 - stan do którego przechodzimy po wykryciu trzeciej w sekwencji jedynki
q_5 - stan do którego przechodzimy po wykryciu ostatniej w sekwencji jedynki

Q(t)				Q(t+1)		
Z	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	0	1	0	0	0
1	0	0	1	0	1	0
0	0	1	0	0	1	1
1	0	1	0	0	0	0
0	0	1	1	0	0	0
1	0	1	1	1	0	0
0	1	0	0	0	0	0
1	1	0	0	1	0	1
0	1	0	1	0	0	0
1	1	0	1	0	0	1

Z	Q2	Q1	Q0	y_i	Y
0	0	0	0	y_0	0
1	0	0	0	y_0	0
0	0	0	1	y_0	0
1	0	0	1	y_0	0
0	0	1	0	y_0	0
1	0	1	0	y_0	0
0	0	1	1	y_0	0
1	0	1	1	y_0	0
0	1	0	0	y_0	0
1	1	0	0	y_0	0
0	1	0	1	y_1	1
1	1	0	1	y_1	1

Kod w języku VHDL

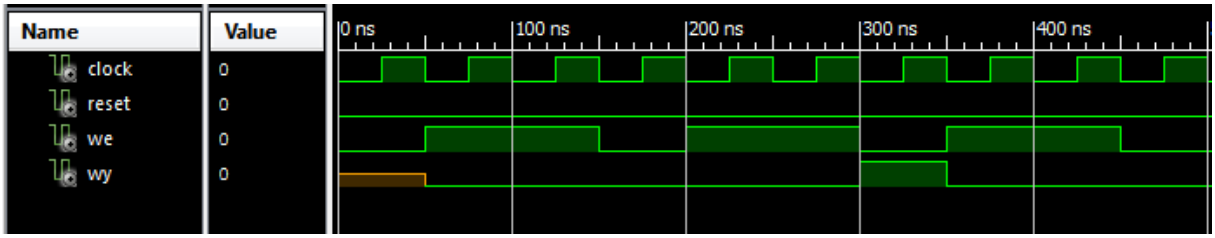
Do napisania tego kodu użyliśmy instrukcji case-when. W zależności od aktualnego stanu oraz wejścia wybierany jest stan kolejny. Przykładowo – dla stanu A i wejścia ‘1’ następnym stanem jest stan B, z kolei na wejścia ‘0’ stan A pozostaje niezmienny.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity zad4 is
    Port ( Wy : out  STD_LOGIC;
           Clock : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           We : in  STD_LOGIC);
end zad4;

architecture Behavioral of zad4 is
    type state_type is (A, B, C, D, E, F);
    signal state, next_state : state_type;
begin
    process1 : process(Clock)
    begin
        if rising_edge(Clock) then
            if Reset = '1' then
                state <= A;
            else
                state <= next_state;
            end if;
        end if;
    end process process1;

    process2 : process(Clock)
    begin
        case state is
            when A =>
                if We = '1' then
                    next_state <= B;
                    wy <= '0';
                else
                    next_state <= A;
                end if;
            when B =>
                if We = '1' then
                    next_state <= C;
                    wy <= '0';
                else
                    next_state <= A;
                end if;
            when C =>
                if We = '0' then
                    next_state <= D;
                    wy <= '0';
                else
                    next_state <= A;
                end if;
            when D =>
                if We = '1' then
                    next_state <= E;
                    wy <= '0';
                else
                    next_state <= A;
                end if;
            when E =>
                if We = '1' then
                    next_state <= F;
                    wy <= '0';
                else
                    next_state <= A;
                end if;
            when F =>
                if We = '1' then
                    next_state <= B;
                    wy <= '1';
                else
                    wy <= '1';
                    next_state <= A;
                end if;
        end case;
    end process process2;

end Behavioral;
```

Testbench i symulacja

Aby wykonać symulację musieliśmy stworzyć testbench. Poza ustaleniem wartości początkowych ustawiony jest także zegar, który zmienia swoją wartość co 25ns.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY zad4tb IS
END zad4tb;

ARCHITECTURE behavior OF zad4tb IS

    COMPONENT zad4
    PORT(
         Wy : OUT  std_logic;
         Clock : IN  std_logic;
         Reset : IN  std_logic;
         We : IN  std_logic
        );
    END COMPONENT;

   signal Clock : std_logic := '0';
   signal Reset : std_logic := '0';
   signal We : std_logic := '0';

   signal Wy : std_logic:='0';

   constant Clock_period : time := 25 ns;
BEGIN

   uut: zad4 PORT MAP (
          Wy => Wy,
          Clock => Clock,
          Reset => Reset,
          We => We
        );

  Reset <= '0';
  We <= '0', '1' after 50ns, '0' after 150ns, '1' after 200ns, '0' after 350ns, '1' after 400ns, '0' after 450ns;
  Clock <= not Clock after 25ns;

END;
```



Plik UCF

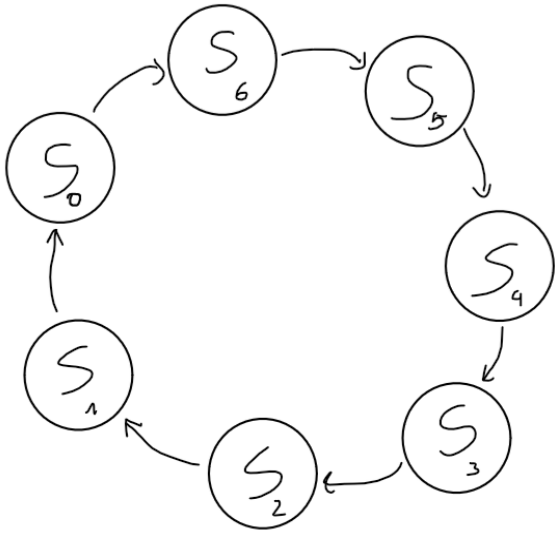
Jako wejścia przypisaliśmy 2 pierwsze przyciski, a jako wyjście led „P35”

```
#------------------------------------------------------------
#  ZL-9572 CPLD board,  J.Sugier 2009
#------------------------------------------------------------

# Clocks
NET "Clock" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "We" LOC = "P42";
NET "Reset" LOC = "P40";
#NET "We(2)" LOC = "P43";
#NET "We(3)" LOC = "P38";
#NET "Key<4>" LOC = "P37";
#NET "Key<5>" LOC = "P36";  # shared with ROT_A
#NET "Key<6>" LOC = "P24";  # shared with ROT_B
#NET "Key<7>" LOC = "P39";  # GSR

# LEDS
NET "Wy"  LOC = "P35";
#NET "WyC2"  LOC = "P29";
#NET "WyB2"  LOC = "P33";
#NET "WyA2"  LOC = "P34";
#NET "LED<4>"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

Zadanie 4

Licznik synchroniczny o zadanych parametrach funkcjonalnych:
synchroniczny, mod 7, negatywny, kod Aikena

Graf i tabela przejść



Q(t)				Q(t+1)			
Q3	**Q2**	**Q1**	**Q0**	**Q3**	**Q2**	**Q1**	**Q0**
6	1	1	0	0	1	0	1
5	1	0	1	1	0	1	0
4	0	1	0	0	0	0	1
3	0	0	1	1	0	0	1
2	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
0	0	0	0	0	1	1	0

Kod w języku VHDL

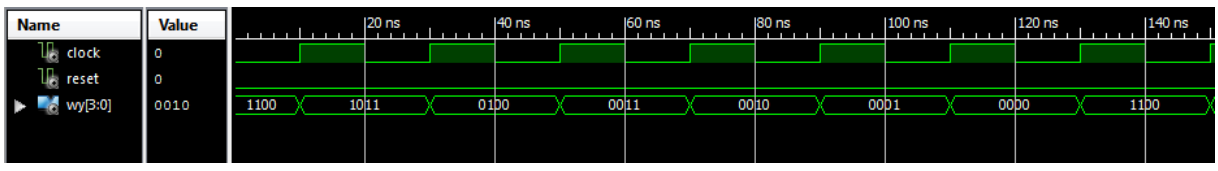
Do napisania tego kodu użyliśmy instrukcji case-when. W zależności od aktualnego stanu wybierany jest stan kolejny. Przykładowo – dla stanu A, stanem następnym będzie stan B. Następnie za pomocą aktualnego stanu na wyjściu wyświetla się – dla stanu A na wyjściu pojawi się ***1100***, a w przypadku stanu np. G, będzie to ***0000***. Dodatkowo, aby zabezpieczyć nasz dodaliśmy linijkę „XXXX" when others, co zapobiega ewentualnym dziwnym zachowaniom. Na początku kolejnego przebiegu procesu nasz aktualny stan staje się stanem następnym (widać to w process1). Jedynym wyjątkiem jest sytuacja, gdy Reset będzie jedynką. Wtedy resetujemy nasz licznik i stanem aktualnym będzie A.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity zad3 is
    Port ( Wy : out  STD_LOGIC_VECTOR (3 downto 0);
           Clock : in  STD_LOGIC;
           Reset : in  STD_LOGIC);
end zad3;

architecture Behavioral of zad3 is
    type state_type is (A,B,C,D,E,F,G);
    signal state, next_state : state_type;

begin
    process1 : process(Clock)
    begin
        if rising_edge(Clock) then
            if Reset = '1' then
                state <= A;
            else
                state <= next_state;
            end if;
        end if;
    end process process1;

    process2 : process(state)
    begin
        next_state <= state; -- by default

        case state is
            when A =>
                next_state <= B;
            when B =>
                next_state <= C;
            when C =>
                next_state <= D;
            when D =>
                next_state <= E;
            when E =>
                next_state <= F;
            when F =>
                next_state <= G;
            when G =>
                next_state <= A;
        end case;
    end process process2;

    with state select
        Wy <= "1100" when A,
              "1011" when B,
              "0100" when C,
              "0011" when D,
              "0010" when E,
              "0001" when F,
              "0000" when G,
              "XXXX" when others;
end Behavioral;
```

Testbench i symulacja

Aby wykonać symulację musieliśmy stworzyć testbench. Poza ustaleniem wartości początkowych ustawiony jest także zegar, który zmienia swoją wartość co 10ns. Symulacja zadziałała poprawnie. Jak widać niżej stany przechodzą kolejno tak jak na tabeli przejść wyżej.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY zad3_tb IS
END zad3_tb;

ARCHITECTURE behavior OF zad3_tb IS

    COMPONENT zad3
    PORT(
         Wy : OUT  std_logic_vector(3 downto 0);
         Clock : IN  std_logic;
         Reset : IN  std_logic
        );
    END COMPONENT;

   signal Clock : std_logic := '0';
   signal Reset : std_logic := '0';

   signal Wy : std_logic_vector(3 downto 0);

   constant Clock_period : time := 10 ns;

BEGIN

   uut: zad3 PORT MAP (
          Wy => Wy,
          Clock => Clock,
          Reset => Reset
        );

   Clock <= not Clock after 10ns;

END;
```



Plik UCF i testowanie na płytce

Jedynym klawiszem aktywnym jest Reset. Na diody led 0-4 dodaliśmy wyjście zapisane w postaci wektora Wy. Program działał poprawnie na płytce, stany przechodziły zgodnie z tabelą przejść, a po wciśnięciu resetu wracamy do stanu początkowego.

```
#------------------------------------------------------------
#  ZL-9572 CPLD board,  J.Sugier 2009
#------------------------------------------------------------

# Clocks
NET "Clock" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "Reset" LOC = "P42";
#NET "Reset" LOC = "P40";
#NET "We(2)" LOC = "P43";
#NET "We(3)" LOC = "P38";
#NET "Key<4>" LOC = "P37";
#NET "Key<5>" LOC = "P36";  # shared with ROT_A
#NET "Key<6>" LOC = "P24";  # shared with ROT_B
#NET "Key<7>" LOC = "P39";  # GSR

# LEDS
NET "Wy(0)"  LOC = "P35";
NET "Wy(1)"  LOC = "P29";
NET "Wy(2)"  LOC = "P33";
NET "Wy(3)"  LOC = "P34";
#NET "LED<4>"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

Wnioski

Zadanie laboratoryjne numer 5 nie sprawiło nam większych problemów, do wykonania zadań mieliśmy przygotowane wszystkie obliczenia oraz opisaną część teoretyczną w poprzednich sprawozdaniach. Trudność tego laboratorium sprowadziła się jedynie do dokładniejszego przestudiowania języka VHDL z którym na dobrą sprawę mieliśmy do czynienia wyłącznie gdy generował się automatycznie przy zaprojektowanych przez nas układach. Wykonanie zadań przy pomocy samego języka VHDL jest zdecydowanie czytelniejsze dla programisty i nie wymaga dokładnego studiowania układu aby zrozumieć treść programu gdybyśmy chcieli wrócić do niego po dłuższym czasie. Przy samej implementacji nie musieliśmy się martwić, że jakaś bramka zostało błędnie podłączona, a debugowanie polegało wyłącznie na przeglądaniu kodu.