Maciej Byczko Bartosz Matysiak	Prowadzący: dr inż. Jacek Mazurkiewicz	Numer ćwiczenia 4
PN 10:50 TP	Temat ćwiczenia: Układy wielobitowych wejść i wyjść	Ocena:
Grupa: B	Data wykonania: 8 Listopada 2021r.	

Spis treści

1 Zadanie 1

1.1 Polecenie

Detektor 2-znakowej sekwencji słów 8-bitowych: wejścia 2 znaków 8-bitowych, 1 wyjście 1-bitowe – sekwencja rozpoznana / sekwencja błędna. Źródło danych: początkowo ”guziki” przystawki, potem klawiatura PC via terminal.

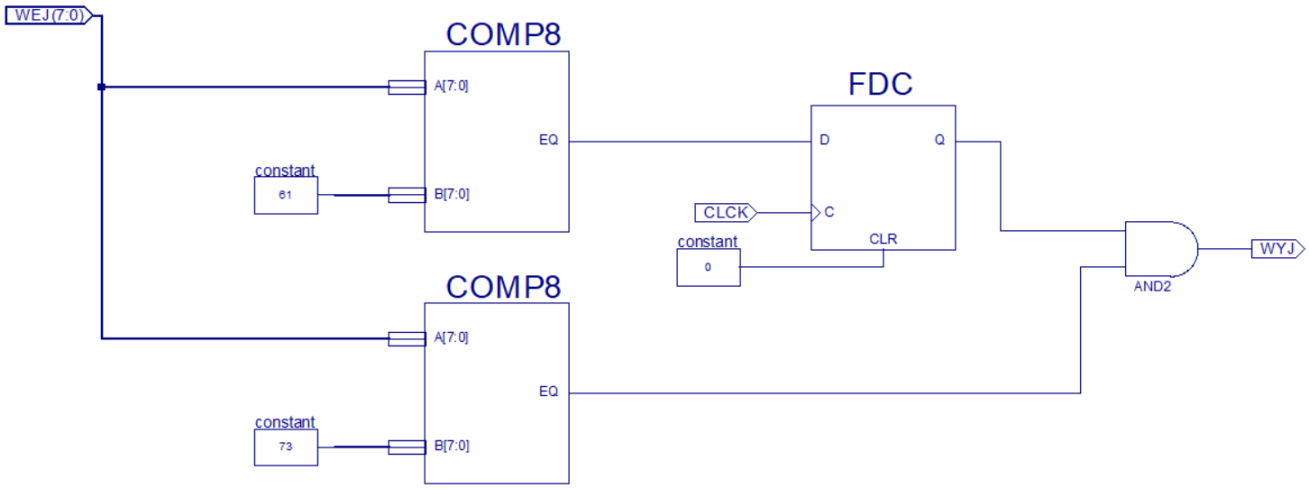
1.2 Rozwiązanie

Do rozwiązania problemu wymagane jest od nas podłączenie dwóch komparatorów 8-bitowych (COMP8), które po pobraniu wartości od użytkownika kolejno po sobie sprawdzają wprowadzone słowa.

Aby wymusić na użytkowniku wprowadzanie odpowiedniej kolejności wpisywania wartości zastosowaliśmy przerzutnik typu ”D” aby wytworzyć opóźnienie.

1.2.1 Schemat układu

Schemat dla wersji z przyciskami jako wejściem:



1.2.2 Kod VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY scheme_scheme_sch_tb IS
7  END scheme_scheme_sch_tb;
8  ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS
9
10     COMPONENT scheme
11     PORT( WEJ   : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
12           WYJ  : OUT STD_LOGIC;
13           CLCK   : IN  STD_LOGIC);
14     END COMPONENT;
15
16     SIGNAL WEJ : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
17      SIGNAL WYJ : STD_LOGIC;
18      SIGNAL CLCK  : STD_LOGIC := '0';
19
20  BEGIN
21      UUT: scheme PORT MAP(
22       WEJ => WEJ,
23       WYJ => WYJ,
24       CLCK => CLCK
25      );
26
27    CLCK <= not CLCK after 50 ns;
28    WEJ <= B"0000_0000", B"0110_0001" after 200 ns, B"0010_0011" after
          300 ns, B"0110_0001" after 400 ns, B"0111_0011" after 500 ns;
29  END;
```

1.2.3 Symulacja

1.3 Fizyczna implementacja

1.3.1 Kod UCF

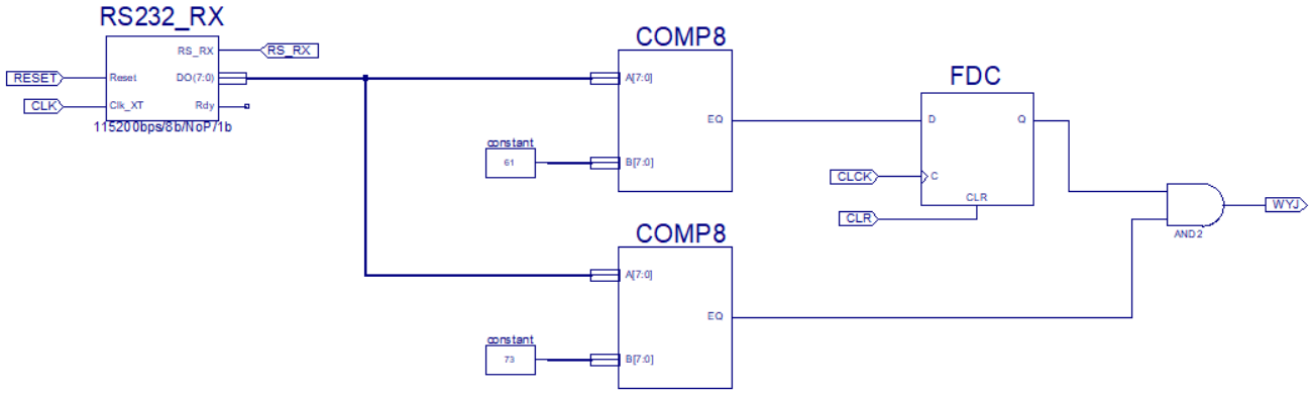
```
1  # Clocks
2  NET "CLCK" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;
3
4  # Keys
5  NET "WEJ(0)" LOC = "P42";
6  NET "WEJ(1)" LOC = "P40";
7  NET "WEJ(2)" LOC = "P43";
8  NET "WEJ(3)" LOC = "P38";
9  NET "WEJ(4)" LOC = "P37";
10 NET "WEJ(5)" LOC = "P36";  # shared with ROT_A
11 NET "WEJ(6)" LOC = "P24";  # shared with ROT_B
12 NET "WEJ(7)" LOC = "P39";  # GSR
13
14 # LEDS
15 NET "WYJ"  LOC = "P35";
```

1.4 Implementacja klawiatury

Aby wykorzystać klawiaturę komputera jako źródło danych via terminal to musieliśmy zaimportować moduł RS232_RX ze strony laboratorium.

Aby moduł został poprawnie zaimplementowany musieliśmy go umieścić na schemacie, wczytać nowe wejścia do pliku VHDL oraz przypisać te wejścia do odpowiednich elementów zestawu fizycznego za pomocą pliku UCF.

1.4.1 Schemat układu



1.4.2 Kod VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY scheme_scheme_sch_tb IS
7  END scheme_scheme_sch_tb;
8  ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS
9
10     COMPONENT scheme
11     PORT( WYJ   : OUT STD_LOGIC;
12           CLCK  : IN  STD_LOGIC;
13           CLR   : IN  STD_LOGIC;
14           CLK   : IN  STD_LOGIC;
15           RESET : IN  STD_LOGIC;
16           RS_RX : IN  STD_LOGIC);
17     END COMPONENT;
18
19     SIGNAL WYJ   : STD_LOGIC;
20     SIGNAL CLCK  : STD_LOGIC;
21     SIGNAL CLR   : STD_LOGIC;
22     SIGNAL CLK   : STD_LOGIC;
23     SIGNAL RESET : STD_LOGIC;
24     SIGNAL RS_RX : STD_LOGIC;
25
26 BEGIN
27     UUT: scheme PORT MAP(
28       WYJ => WYJ,
29       CLCK => CLCK,
30       CLR => CLR,
31       CLK => CLK,
32       RESET => RESET,
33       RS_RX => RS_RX
34     );
35 END;
```

1.4.3 Kod UCF

```
# Clocks
NET "CLCK" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;
NET "CLK" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "RESET" LOC = "P42";
NET "CLR" LOC = "P40";

# LEDS
NET "WYJ"  LOC = "P35";

# RS-232
NET "RS_RX" LOC = "P1";
```

Zadanie zostało wykonane bez większych komplikacji, z powodu sposobu implementacji poprawną kombinację (a,s) musimy wpisać odpowiednio szybko oraz w tym samym tempie co zegar podłączony do przerzutnika.

2 Zadanie 2

2.1 Polecenie

Układ arytmetyczny pracujący na dwóch argumentach 4-bitowych wyrażonych w kodzie Aikena i generujący stosowny wynik w tymże kodzie.

2.2 Rozwiązanie

Najłatwiejszym sposobem na wykonanie dodawania dwóch liczb w kodzie Aikena okazała się zamiana danych wejściowych na NKB. Cyfry 0-4 pokrywają się w obu kodach, więc musieliśmy zamienić jedynie cyfry 5-9. Zauważyliśmy że aby sprawnie wykonać tą zmianę to musieliśmy odjąć wartość 0110 (6) aby uzyskać wymaganą wartość binarną. Operandy już skonwertowane na NKB dodajemy przy pomocy zwykłego sumatora. Otrzymany wynik ponownie konwertujemy na kod Aikena, wykrywając także, czy należy on do dziedziny tego kodu, i sygnalizując to wyjściem przepełnienia. Potrzebne są zatem dwa typy podukładów: konwerter kodu Aikena na NKB, i konwerter odwrotny, z dodatkowym wyjściem przepełnienia.

Ustaliliśmy, że w momencie przepełnienia, czyli gdy wartość 9 zostanie przekroczona, zapala się lampka przepełnienia informująca, że wynik jest niewiarygodny (nieważny).

2.2.1 Tabele prawdy

Tabela 1: Konwerter kod Aikena - > NKB

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	-	-	-	-
0	1	1	0	-	-	-	-
0	1	1	1	-	-	-	-
1	0	0	0	-	-	-	-
1	0	0	1	-	-	-	-
1	0	1	0	-	-	-	-
1	0	1	1	0	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	0	1	1	1
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	1

Tabela 2: Konwerter NKB - > kod Aikena

S_4	S_3	S_2	S_1	S_0	OVL	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	1	0	1	1
0	0	1	1	0	0	1	1	0	0
0	0	1	1	1	0	1	1	0	1
0	1	0	0	0	0	1	1	1	0
0	1	0	0	1	0	1	1	1	1
0	1	0	1	0	1	-	-	-	-
0	1	0	1	1	1	-	-	-	-
0	1	1	0	0	1	-	-	-	-
0	1	1	0	1	1	-	-	-	-
0	1	1	1	0	1	-	-	-	-
0	1	1	1	1	1	-	-	-	-
1	0	0	0	0	1	-	-	-	-
1	0	0	0	1	1	-	-	-	-
1	0	0	1	0	1	-	-	-	-

Wejściem układu są bity S(4:0), stanowiące wyjście sumatora czterobitowego (S(3:0) - bity cyfr wyjściowych sumatora, S_4 - bit Carry Out tegoż). Można zauważyć, że bit przepełnienia jest jedynką wtedy i tylko wtedy, gdy ustawiony jest bit S_4 lub gdy bity S(3:0) tworzą liczbę większą od 9 (w NKB); dzięki temu spostrzeżeniu (bezpośrednio potem oddanego na schemacie), możliwe stało się zredukowanie liczby argumentów funkcji logicznej z pięciu do czterech. Pozostałe bity wyjściowe również są funkcjami czterech bitów S(3:0), co umożliwia skonstruowanie przejrzystych siatek Karnaugh.

2.2.2 Siatki Karnaugh

Konwerter kod Aikena - > NKB:

A_3A_2 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	0	-	-	-
11	0	0	1	1
10	-	-	0	-

$Y_3 = A_2A_1$

A_3A_2 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	1	-	-	-
11	1	1	0	0
10	-	-	1	-

$Y_2 = A_3\overline{A_2} + A_2\overline{A_1}$

A_3A_2 \ A_1A_0	00	01	11	10
00	0	0	1	1
01	0	-	-	-
11	1	1	0	0
10	-	-	0	-

$Y_1 = A_3 \oplus A_1$

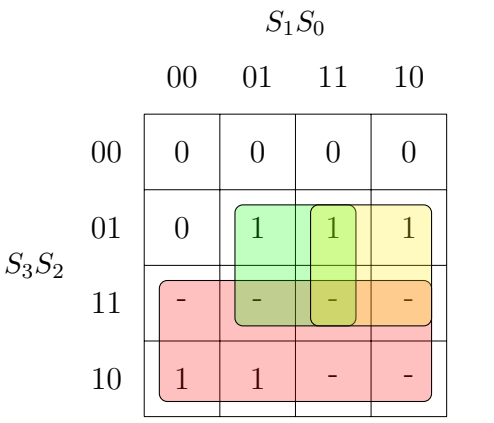
A_3A_2 \ A_1A_0	00	01	11	10
00	0	1	1	0
01	0	-	-	-
11	0	1	1	0
10	-	-	1	-

$Y_0 = A_0$

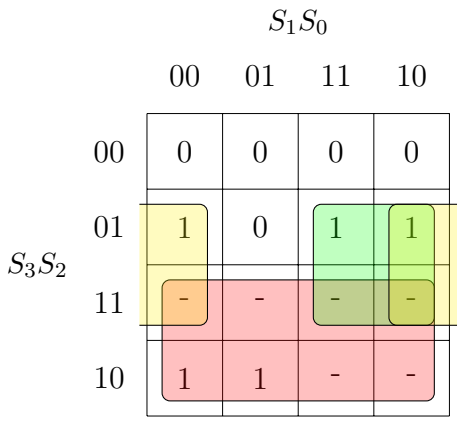
Konwerter NKB - > kod Aikena:

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

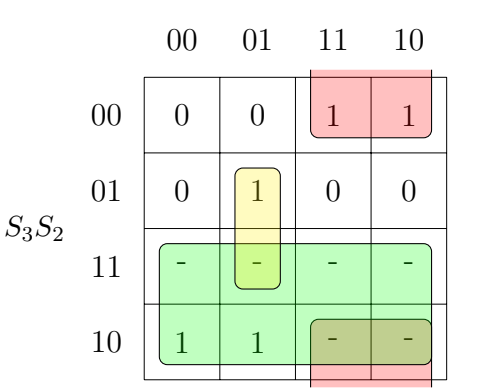
$OVL = S_3S_2 + S_3S_1 = S_3(S_2 + S_1)$



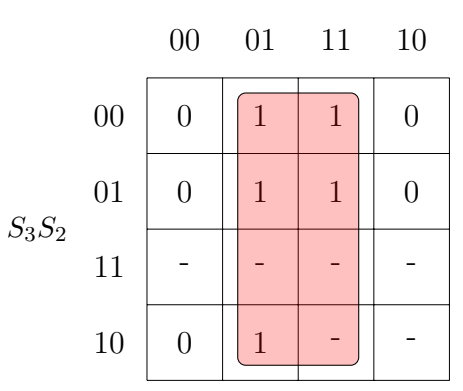
$$R_3 = S_3 + S_2S_1 + S_2S_0$$



$$R_2 = S_3 + S_2S_1 + S_2\overline{S_0}$$

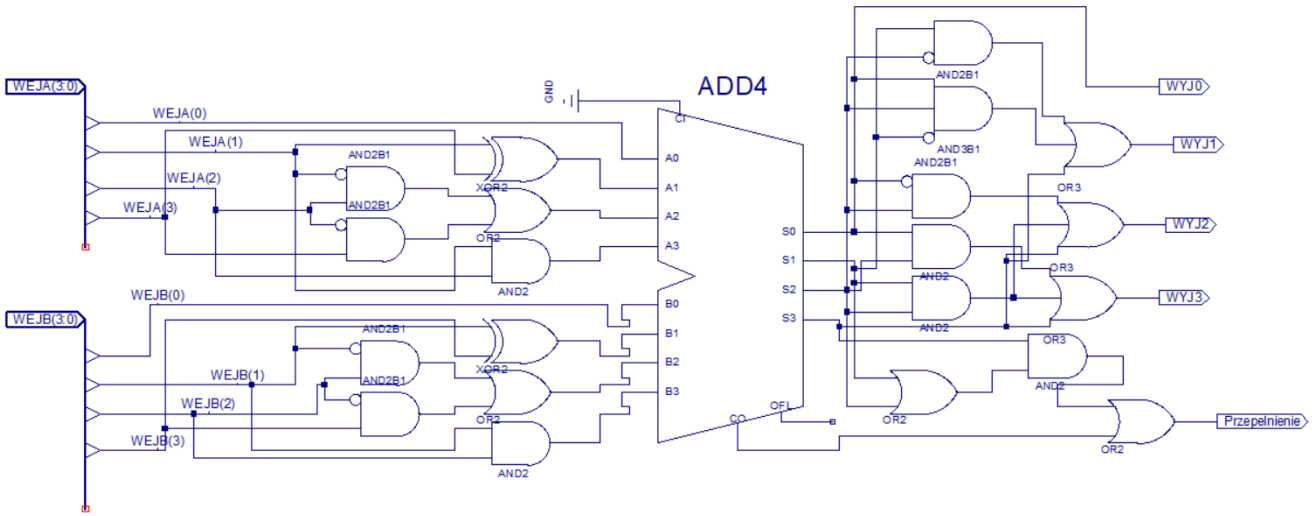


$$R_1 = S_3 + \overline{S_2}\ S_1 + S_2\overline{S_1}\ S_0$$



$$R_0 = S_0$$

2.2.3 Schemat układu

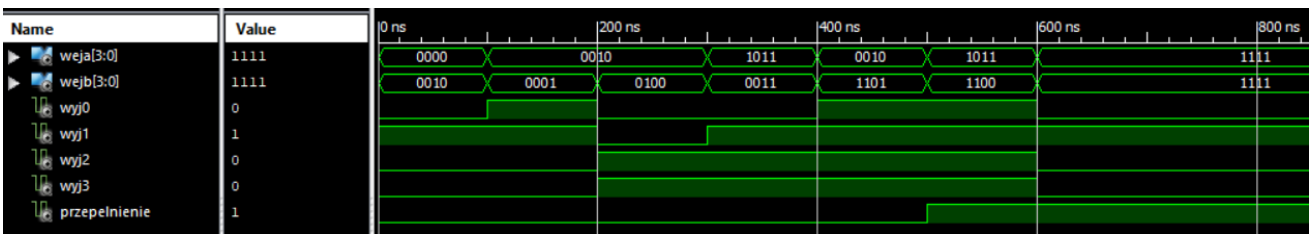


2.2.4 Kod VHDL

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.numeric_std.ALL;
4 LIBRARY UNISIM;
5 USE UNISIM.Vcomponents.ALL;
6 ENTITY aikenAdderScheme_aikenAdderScheme_sch_tb IS
```

```
7  END aikenAdderScheme_aikenAdderScheme_sch_tb;
8  ARCHITECTURE behavioral OF aikenAdderScheme_aikenAdderScheme_sch_tb
       IS
9
10     COMPONENT aikenAdderScheme
11     PORT( WYJ0 : OUT STD_LOGIC;
12           WEJA  : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
13           WEJB  : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
14           Przepelnienie : OUT STD_LOGIC;
15           WYJ1  : OUT STD_LOGIC;
16           WYJ2  : OUT STD_LOGIC;
17           WYJ3  : OUT STD_LOGIC);
18     END COMPONENT;
19
20     SIGNAL WYJ0  : STD_LOGIC;
21     SIGNAL WEJA  : STD_LOGIC_VECTOR (3 DOWNTO 0);
22     SIGNAL WEJB  : STD_LOGIC_VECTOR (3 DOWNTO 0);
23     SIGNAL Przepelnienie : STD_LOGIC;
24     SIGNAL WYJ1  : STD_LOGIC;
25     SIGNAL WYJ2  : STD_LOGIC;
26     SIGNAL WYJ3  : STD_LOGIC;
27
28 BEGIN
29
30     UUT: aikenAdderScheme PORT MAP(
31       WYJ0 => WYJ0,
32       WEJA => WEJA,
33       WEJB => WEJB,
34       Przepelnienie => Przepelnienie,
35       WYJ1 => WYJ1,
36       WYJ2 => WYJ2,
37       WYJ3 => WYJ3
38     );
39    WEJA <= "0000", "0010" after 100 ns, "0010" after 200 ns, "1011"
          after 300 ns, "0010" after 400 ns, "1011" after 500 ns, "1111"
          after 600 ns;
40    WEJB <= "0010", "0001" after 100 ns, "0100" after 200 ns, "0011"
          after 300 ns, "1101" after 400 ns, "1100" after 500 ns, "1111"
          after 600 ns;
41 END;
```

2.2.5 Symulacja



2.3 Fizyczna implementacja

2.3.1 Kod UCF

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

# LEDS
NET "WYJ0"   LOC = "P35";
NET "WYJ1"   LOC = "P29";
NET "WYJ2"   LOC = "P33";
NET "WYJ3"   LOC = "P34";
NET "Przepelnienie"   LOC = "P28";
```

W tej implementacji dodawanie następowało w czasie rzeczywistym, panel przycisków został podzielony na 2 sekcje dzięki czemu mogliśmy bezproblemowo wprowadzać 4-bitowe wartości.

3 Zadanie 3

3.1 Polecenie

Konwerter cyfry szesnastkowej zapisanej na czterech bitach od 0 do 9, A do F na kod ASCII tej cyfry – wyjście 8-bitowe. Prezentacja wyniku na diodach przystawki, potem na wyświetlaczu 7-segmentowym.

3.2 Rozwiązanie

W celu dokonania konwersji, układ sprawdza czy prowadzona cyfra posiada wartość mniejszą niż 10. W celu uzyskania kodu ASCII w takim wypadku należy dodać 48 do wprowadzonej wartości. W przeciwnym razie należy dodać 55. Schemat działa na podobnej zasadzie co sumator z zadania 2 - wynik konwersji jest podawany na poszczególne bity sumatora, tak aby utworzył on potrzebną wartość. Do tego zadania wykorzystaliśmy 2 sumatory 4-bitowe.

3.2.1 Tabela prawdy

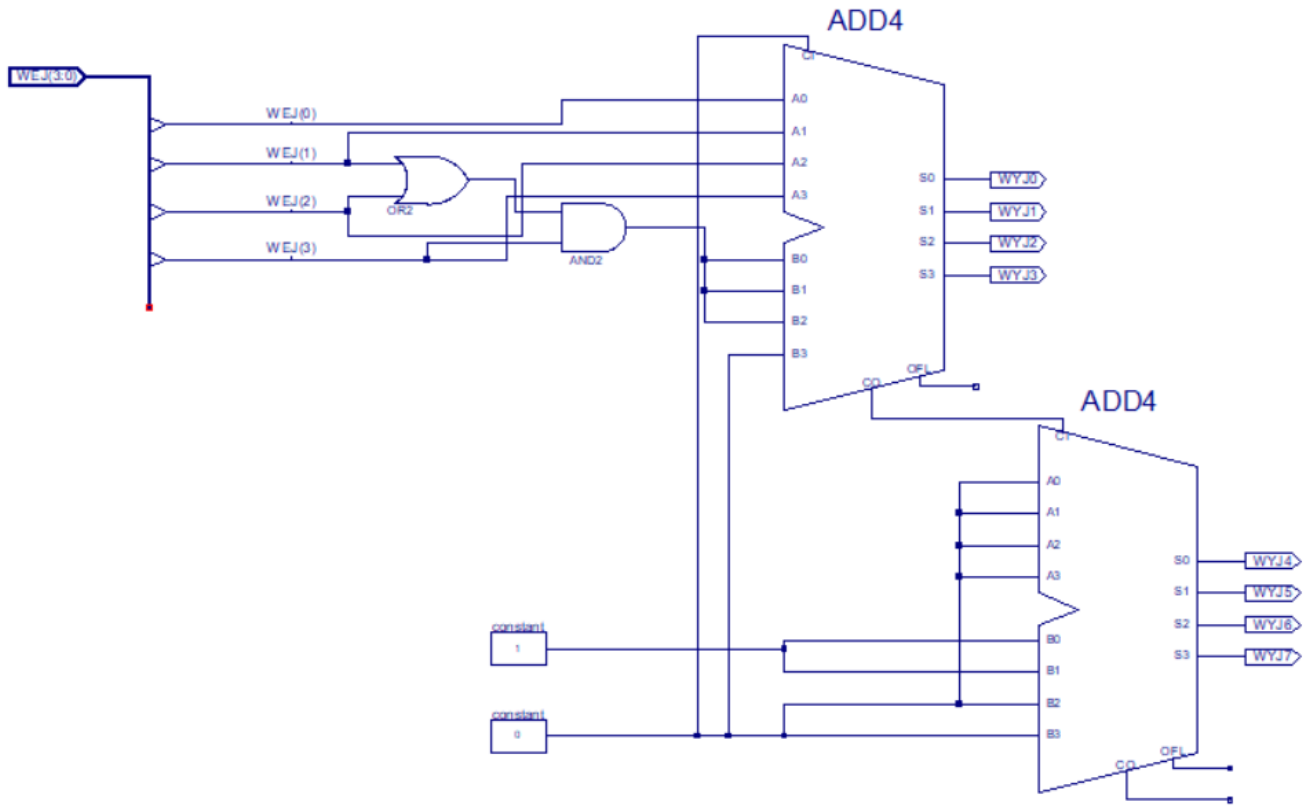
H_3	H_2	H_1	H_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

3.2.2 Siatka Karnaugh

H_3H_2 \ H_1H_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$Y = H_3H_2 + H_3H_1 = H_3(H_2 + H_1)$$

3.2.3 Schemat układu

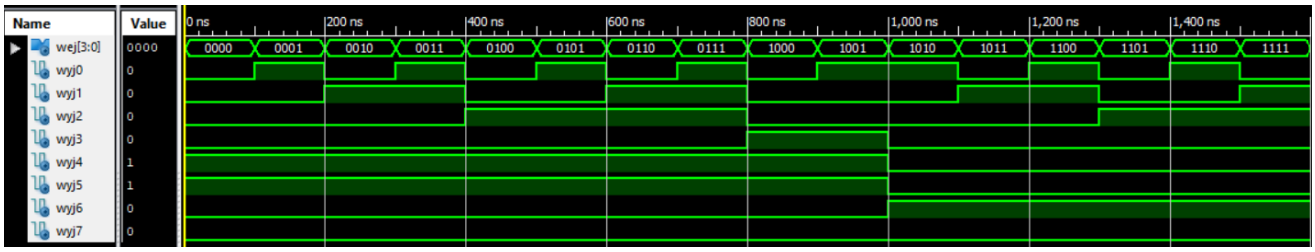


3.2.4 Kod VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY scheme_scheme_sch_tb IS
7  END scheme_scheme_sch_tb;
8  ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS
9
10     COMPONENT scheme
11     PORT( WEJ   : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
12           WYJ0  : OUT STD_LOGIC;
13           WYJ1  : OUT STD_LOGIC;
14           WYJ2  : OUT STD_LOGIC;
15           WYJ3  : OUT STD_LOGIC;
16           WYJ4  : OUT STD_LOGIC;
17           WYJ5  : OUT STD_LOGIC;
18           WYJ6  : OUT STD_LOGIC;
19           WYJ7  : OUT STD_LOGIC);
20     END COMPONENT;
21
22     SIGNAL WEJ : STD_LOGIC_VECTOR (3 DOWNTO 0);
23     SIGNAL WYJ0  : STD_LOGIC;
24     SIGNAL WYJ1  : STD_LOGIC;
25     SIGNAL WYJ2  : STD_LOGIC;
```

```
26      SIGNAL WYJ3   : STD_LOGIC;
27      SIGNAL WYJ4   : STD_LOGIC;
28      SIGNAL WYJ5   : STD_LOGIC;
29      SIGNAL WYJ6   : STD_LOGIC;
30      SIGNAL WYJ7   : STD_LOGIC;
31
32  BEGIN
33      UUT: scheme PORT MAP(
34       WEJ => WEJ,
35       WYJ0 => WYJ0,
36       WYJ1 => WYJ1,
37       WYJ2 => WYJ2,
38       WYJ3 => WYJ3,
39       WYJ4 => WYJ4,
40       WYJ5 => WYJ5,
41       WYJ6 => WYJ6,
42       WYJ7 => WYJ7
43      );
44     WEJ <= "0000", "0001" after 100 ns, "0010" after 200 ns, "0011"
           after 300 ns, "0100" after 400 ns, "0101" after 500 ns, "0110"
           after 600 ns, "0111" after 700 ns, "1000" after 800 ns, "1001"
           after 900 ns, "1010" after 1000 ns, "1011" after 1100 ns, "1100"
            after 1200 ns, "1101" after 1300 ns, "1110" after 1400 ns, "
           1111" after 1500 ns;
45  END;
```

3.2.5 Symulacja



3.3 Fizyczna implementacja

3.3.1 Kod UCF

```
1  # Keys
2  NET "WEJ(0)" LOC = "P42";
3  NET "WEJ(1)" LOC = "P40";
4  NET "WEJ(2)" LOC = "P43";
5  NET "WEJ(3)" LOC = "P38";
6
7  # LEDS
8  NET "WYJ0"   LOC = "P35";
9  NET "WYJ1"   LOC = "P29";
10 NET "WYJ2"   LOC = "P33";
11 NET "WYJ3"   LOC = "P34";
12 NET "WYJ4"   LOC = "P28";
```

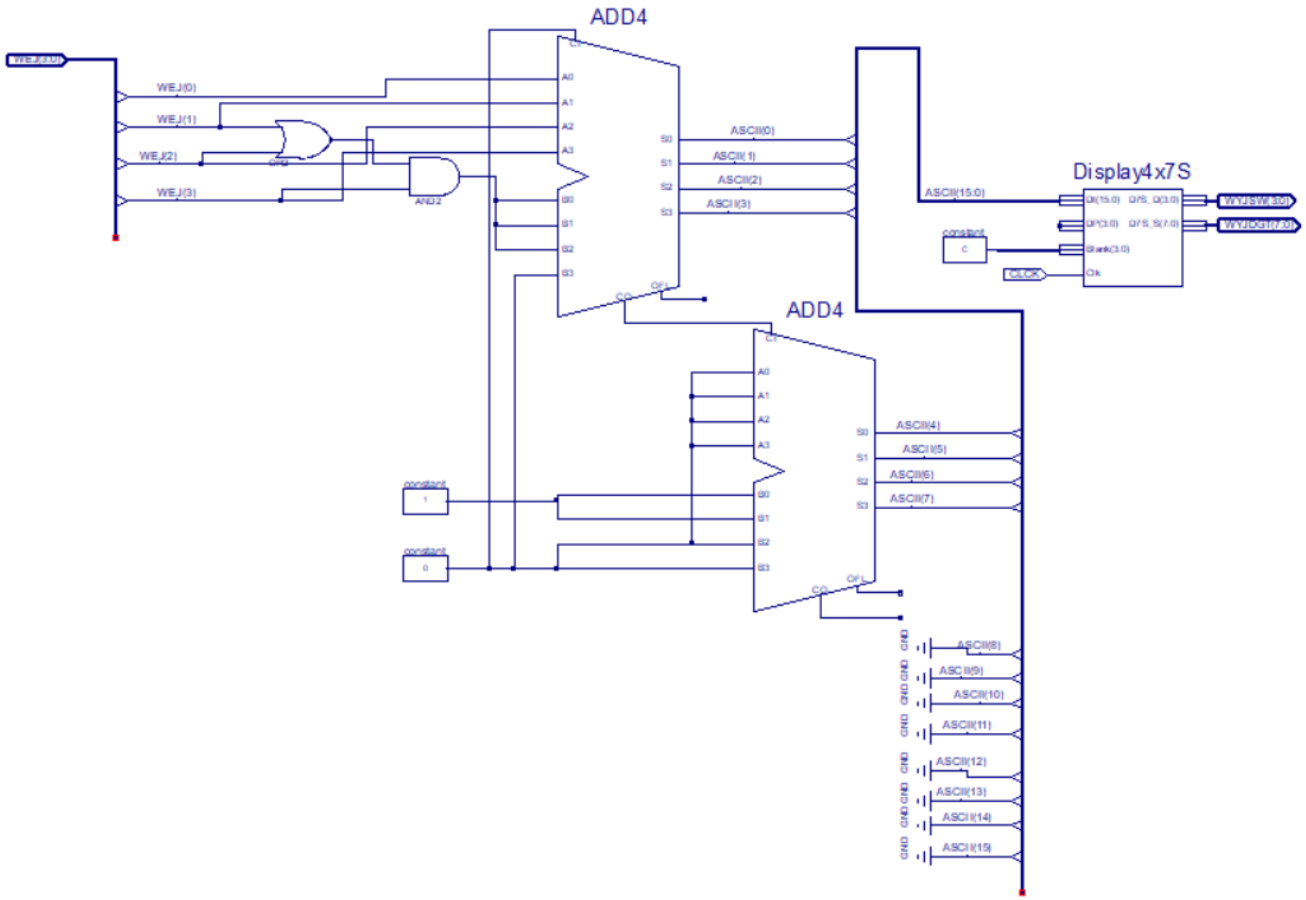
```
NET "WYJ5"   LOC = "P27";
NET "WYJ6"   LOC = "P26";
NET "WYJ7"   LOC = "P25";
```

3.4 Implementacja wyświetlacz

Aby wykorzystać wyświetlacz do zaprezentowania wynikowego kodu dziesiętnego ASCII to musieliśmy zaimportować moduł Display4x7S ze strony laboratorium.

Aby moduł został poprawnie zaimplementowany musieliśmy go umieścić na schemacie, wczytać nowe wejścia do pliku VHDL oraz przypisać te wejścia do odpowiednich elementów zestawu fizycznego. Moduł też wymaga od nas podłączenia do zegara wysokiej częstotliwości aby poprawnie wyświetlał wartości na ekranie zestawu.

3.4.1 Schemat układu



3.4.2 Kod VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY scheme_scheme_sch_tb IS
END scheme_scheme_sch_tb;
ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS

   COMPONENT scheme
```

```
    PORT( WEJ   : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
          WYJSW : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
          WYJDGT  : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
          CLCK  : IN  STD_LOGIC);
    END COMPONENT;

    SIGNAL WEJ : STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL WYJSW : STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL WYJDGT  : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL CLCK  : STD_LOGIC;

BEGIN
    UUT: scheme PORT MAP(
     WEJ => WEJ,
     WYJSW => WYJSW,
     WYJDGT => WYJDGT,
     CLCK => CLCK
    );
END;
```

3.4.3 Kod UCF

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

# Keys
NET "WEJ(0)" LOC = "P42";
NET "WEJ(1)" LOC = "P40";
NET "WEJ(2)" LOC = "P43";
NET "WEJ(3)" LOC = "P38";

# DISPL. 7-SEG
NET "WYJSW(0)" LOC = "P8" | SLEW = "SLOW";
NET "WYJSW(1)" LOC = "P6" | SLEW = "SLOW";
NET "WYJSW(2)" LOC = "P4" | SLEW = "SLOW";
NET "WYJSW(3)" LOC = "P9" | SLEW = "SLOW";
NET "WYJDGT(0)" LOC = "P12";  # Seg. A; shared with LED<10>
NET "WYJDGT(1)" LOC = "P13";  # Seg. B; shared with LED<8>
NET "WYJDGT(2)" LOC = "P22";  # Seg. C; shared with LED<12>
NET "WYJDGT(3)" LOC = "P19";  # Seg. D; shared with LED<14>
NET "WYJDGT(4)" LOC = "P14";  # Seg. E; shared with LED<15>
NET "WYJDGT(5)" LOC = "P11";  # Seg. F; shared with LED<9>
NET "WYJDGT(6)" LOC = "P20";  # Seg. G; shared with LED<13>
#NET "WYJDGT(7)" LOC = "P18"; # Seg. DP;shared with LED<11>
```

Na początku mieliśmy zagwozdkę co mamy zrobić z bitami wejściowymi których jest 16 a my generujemy tylko 8. Uporaliśmy się z problemem poprzez wypełnienie pozostałych bitów zerami.

Efekt wyświetlenia wartości dziesiętnej kodu ASCII na ekranie zestawu był bardzo satysfakcjonujący.

4 Zadanie 4

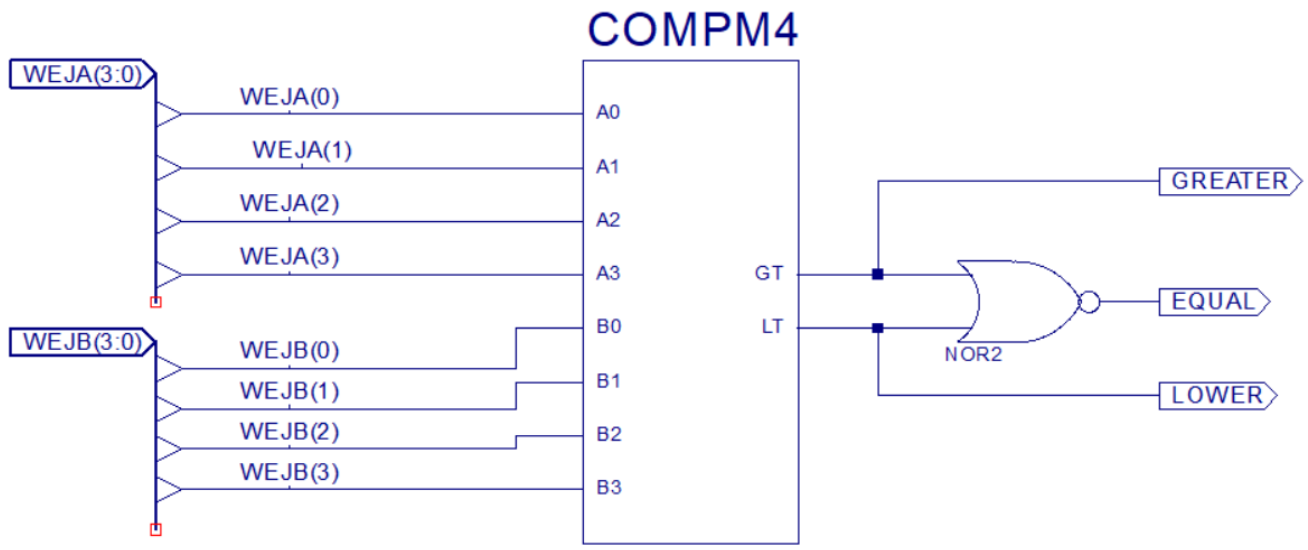
4.1 Polecenie

Komparator dwóch 4-bitowych cyfr: 2 wejścia po 4 bity, 3 wyjścia 1-bitowe: mniejszy, większy, równy pracujący w kodzie Aikena.

4.2 Rozwiązanie

Na początku zadania zauważyliśmy że porównywanie wartości w kodzie Aikena wygląda identycznie jak w kodzie NKB. Dzięki czemu udało nam się skonstruować nieskomplikowany schemat wraz z 3 wyjściami. Do wykonania zadania wykorzystaliśmy wbudowany komparator w środowisku.

4.2.1 Schemat układu

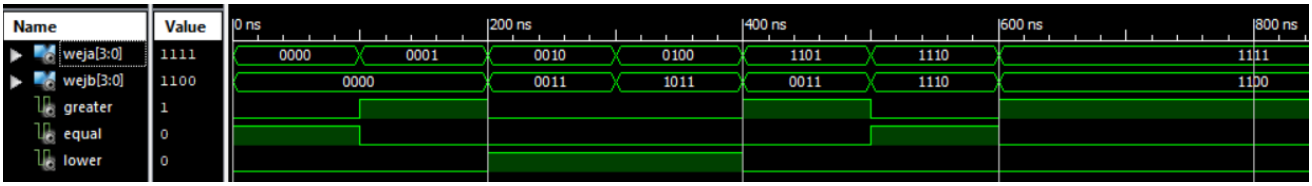


4.2.2 Kod VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY scheme_scheme_sch_tb IS
7  END scheme_scheme_sch_tb;
8  ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS
9
10    COMPONENT scheme
11    PORT( WEJA : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
12          WEJB  : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
13          WIEKSZE : OUT STD_LOGIC;
14          MNIEJSZE  : OUT STD_LOGIC;
15          ROWNE : OUT STD_LOGIC);
16    END COMPONENT;
17
18    SIGNAL WEJA  : STD_LOGIC_VECTOR (3 DOWNTO 0);
19    SIGNAL WEJB  : STD_LOGIC_VECTOR (3 DOWNTO 0);
```

```
20     SIGNAL WIEKSZE : STD_LOGIC;
21     SIGNAL MNIEJSZE  : STD_LOGIC;
22     SIGNAL ROWNE : STD_LOGIC;
23
24 BEGIN
25     UUT: scheme PORT MAP(
26       WEJA => WEJA,
27       WEJB => WEJB,
28       WIEKSZE => WIEKSZE,
29       MNIEJSZE => MNIEJSZE,
30       ROWNE => ROWNE
31      );
32
33   WEJA <= "0000", "0001" after 100 ns, "0010" after 200 ns, "0100"
        after 300 ns, "1101" after 400 ns, "1110" after 500 ns, "1111"
        after 600 ns;
34   WEJB <= "0000", "0000" after 100 ns, "0011" after 200 ns, "1011"
        after 300 ns, "0011" after 400 ns, "1110" after 500 ns, "1100"
        after 600 ns;
35 END;
```

4.2.3 Symulacja



4.3 Fizyczna implementacja

4.3.1 Kod UCF

```
1  # Keys
2  NET "WEJA(0)" LOC = "P42";
3  NET "WEJA(1)" LOC = "P40";
4  NET "WEJA(2)" LOC = "P43";
5  NET "WEJA(3)" LOC = "P38";
6  NET "WEJB(0)" LOC = "P37";
7  NET "WEJB(1)" LOC = "P36";  # shared with ROT_A
8  NET "WEJB(2)" LOC = "P24";  # shared with ROT_B
9  NET "WEJB(3)" LOC = "P39";  # GSR
10
11 # LEDS
12 NET "MNIEJSZE"  LOC = "P35";
13 NET "ROWNE"  LOC = "P29";
14 NET "WIEKSZE"  LOC = "P33";
```

Panel przycisków podzieliliśmy na 2 części tak samo jak w zadaniu nr. 2 aby móc bezproblemowo wprowadzać wartości. Gdy wartość w lewej sekcji była mniejsza niż prawa, zapalała się prawa dioda sygnalizująca, że wartość po prawej stronie jest większa i vice versa, gdy wartości były równe to świeciła się dioda środkowa informująca o równości wartości.

5 Wnioski

Przy wykonywaniu zadań niezbędne okazały się inteligentne i proste rozwiązania problemów jak zauważenie w zadaniu 4, że porównywanie w kodzie Aikena jest identyczne z kodem KNB.

Małą trudność sprawiły nam moduły implementujące działanie wyświetlacza LCD oraz wprowadzanie wejścia za pomocą klawiatury przez terminal, lecz po przeczytaniu instrukcji i wykonaniu opisanych kroków dobrnęliśmy do rozwiązania problemu.

Najtrudniejszym zadaniem okazało się zadanie 2 gdyż wymagało to od nas dokonania kilku założeń ze względu na specyfikację kodu: Ignorowanie wyjścia gdy będzie przepełnienie oraz odejmowanie/dodawanie wartości 6 gdy zakres przekroczy wartość 4.