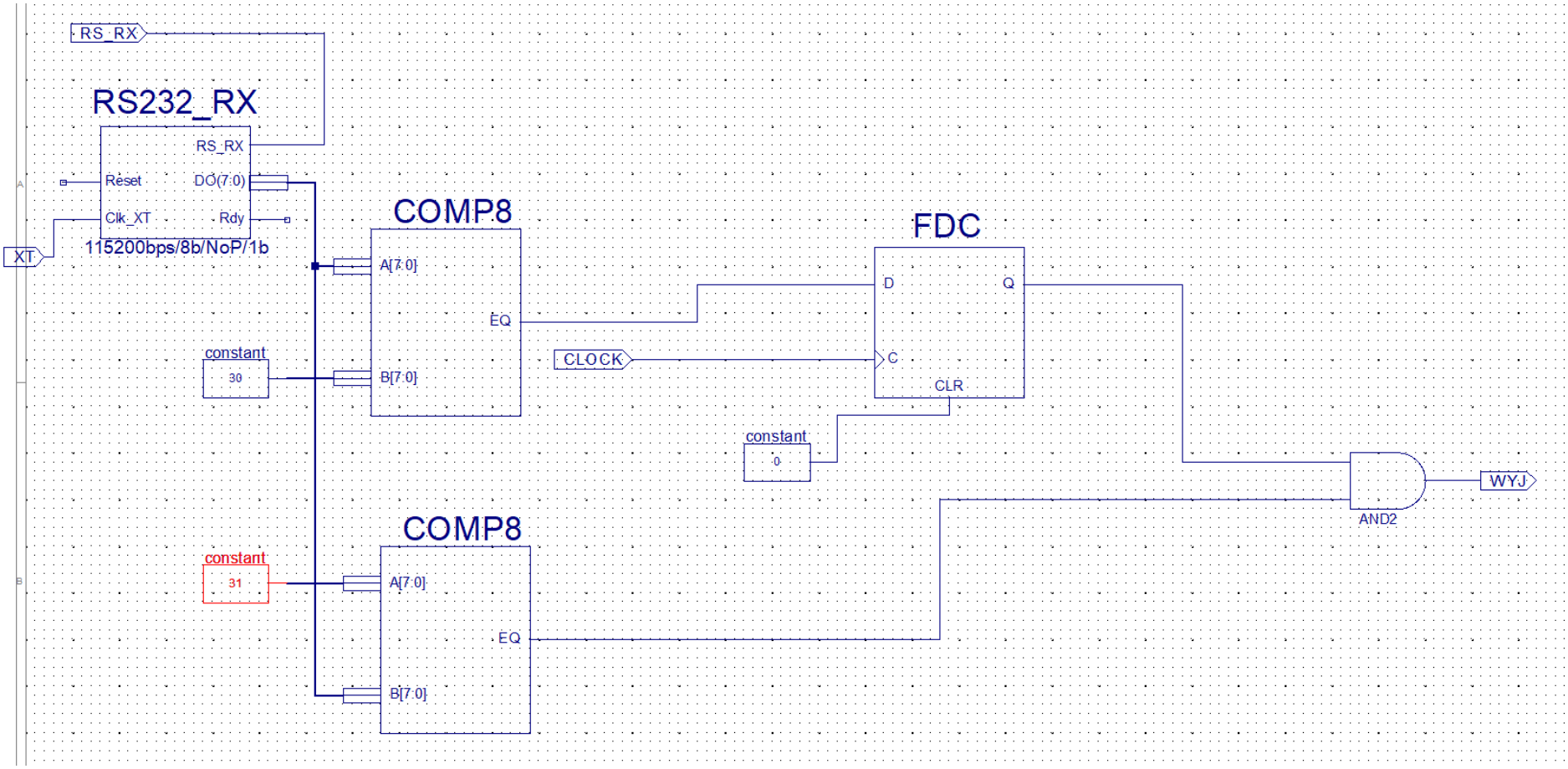
Termin zajęć Wtorek NP 7:30 – 11:00	Układy cyfrowe i systemy wbudowane	
Osoby wykonujące ćwiczenie: Jakub Suski 264028, Adam Czekalski 264488		**Grupa:** **D**
Tytuł ćwiczenia: Układy wielobitowych wejść i wyjść		**Laboratorium nr:** **4**
Data wykonania ćwiczenia	7.11.2023	**Ocena:**
Data oddania sprawozdania	21.11.2023	

Na zajęciach laboratoryjnych zaimplementowano układy, które posiadają wiele bitów na wejściu i wyjściu. W odróżnieniu od układów projektowanych na poprzednie zajęcia, tym razem nie wystarczyły schematyczne, sprawdzone metody syntezy – tworzenie tabeli przejść, siatki Karnaugh czy tworzenie z nich równań. Tradycyjna metoda syntezy w tych przypadkach okazałaby się zbyt czasochłonna, kosztowna implementacyjnie (ilość potrzebnych bramek/okablowania) i łatwo by było w niej o pomyłkę. Trzeba było pomyśleć w jaki sposób można dany układ zrealizować, korzystając z gotowych układów np. sumatora, dekodera czy komparatora. Środowisko Xilinx jest bogate w funktory – m.in. mnogą ilość odmian dostępnych układów, co zarówno otwiera szeroki zakres możliwości jak i wymaga szczególnej ostrożności oraz wiedzy na temat poszczególnych wejść przy ich wyborze. Oprócz standardowej implementacji na diodach płytki CPLD ZL-9572, użyto także wyświetlacz 7-segmentowy znajdujący się na płytce oraz klawiaturę podpiętą do portu RS232.

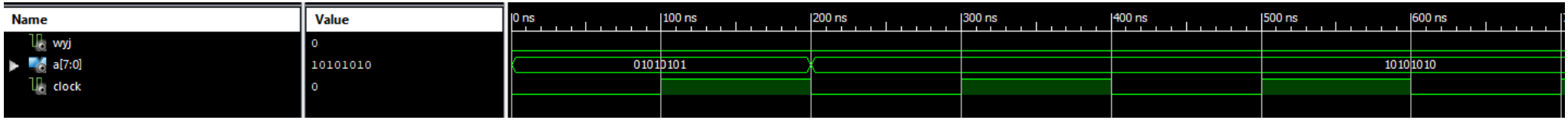
1. Detektor 2-znakowej sekwencji słów 8-bitowych: wejścia 2 znaków 8-bitowych, 1 wyjście 1-bitowe – sekwencja rozpoznana / sekwencja błędna

Zadanie zostało wykonanie empirycznie. Używamy dwóch komparatorów, jeśli najpierw na pierwszym została wykryta sekwencja, to stan o tym zostaje zapamiętany przez przerzutnik typu D. Następnie jeśli na drugim komparatorze również zostanie odczytana prawidłowa dla niego sekwencja, to dzięki bramce AND zobaczymy na wyjściu ‘1’, która będzie oznaczała sekwencję rozpoznaną. W przeciwnym wypadku będzie tam ciągłe ‘0’ - sekwencja błędna. Został również zastosowany konwerter sygnałów wejściowych z urządzenia peryferyjnego jakim była typowa klawiatura komputera “QWERTY”.



Rysunek 1-1 Schemat detektora 2-znakowej sekwencji słów 8-bitowych

Symulacja tego układu:



Rysunek 1-2 Symulacja

Plik VHDL:

```
15  LIBRARY ieee;
16  USE ieee.std_logic_1164.ALL;
17  USE ieee.numeric_std.ALL;
18  LIBRARY UNISIM;
19  USE UNISIM.Vcomponents.ALL;
20  ENTITY schl_schl_sch_tb IS
21  END schl_schl_sch_tb;
22  ARCHITECTURE behavioral OF schl_schl_sch_tb IS
23
24     COMPONENT schl
25     PORT( WYJ   :  OUT   STD_LOGIC;
26           CLOCK   :  IN STD_LOGIC;
27           A :  IN STD_LOGIC_VECTOR (7 DOWNTO 0));
28     END COMPONENT;
29
30     SIGNAL WYJ  :  STD_LOGIC;
31     SIGNAL CLOCK   :  STD_LOGIC := '0';
32     SIGNAL A :  STD_LOGIC_VECTOR (7 DOWNTO 0);
33
34  BEGIN
35
36     UUT: schl PORT MAP(
37        WYJ => WYJ,
38        CLOCK => CLOCK,
39        A => A
40     );
41
42  -- *** Test Bench - User Defined Section ***
43     A <= "01010101", "10101010" after 100 ns;
44     CLOCK <= not CLOCK after 100 ns;
45  -- *** End Test Bench - User Defined Section ***
46
47  END;
48
```

Rysunek 1-3 Fragment pliku VHDL z pobudzeniami testowymi

Plik UCF:

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

# Keys
#NET "A(0)" LOC = "P42";
#NET "A(1)" LOC = "P40";
#NET "A(2)" LOC = "P43";
#NET "A(3)" LOC = "P38";
#NET "A(4)" LOC = "P37";
#NET "A(5)" LOC = "P36";  # shared with ROT_A
#NET "A(6)" LOC = "P24";  # shared with ROT_B
#NET "A(7)" LOC = "P39";  # GSR

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

#NET "LED<8>"  LOC = "P13";   # shared with seg. B
#NET "LED<9>"  LOC = "P11";   # shared with seg. F
#NET "LED<10>" LOC = "P12";   # shared with seg. A
#NET "LED<11>" LOC = "P18";   # shared with seg. DP
#NET "LED<12>" LOC = "P22";   # shared with seg. C
#NET "LED<13>" LOC = "P20";   # shared with seg. G
#NET "LED<14>" LOC = "P19";   # shared with seg. D
#NET "LED<15>" LOC = "P14";   # shared with seg. E

# DISPL. 7-SEG
#NET "D7S_D<0>" LOC = "P8" | SLEW = "SLOW";
#NET "D7S_D<1>" LOC = "P6" | SLEW = "SLOW";
#NET "D7S_D<2>" LOC = "P4" | SLEW = "SLOW";
#NET "D7S_D<3>" LOC = "P9" | SLEW = "SLOW";
#NET "D7S_S<0>" LOC = "P12";  # Seg. A; shared with LED<10>
#NET "D7S_S<1>" LOC = "P13";  # Seg. B; shared with LED<8>
#NET "D7S_S<2>" LOC = "P22";  # Seg. C; shared with LED<12>
#NET "D7S_S<3>" LOC = "P19";  # Seg. D; shared with LED<14>
#NET "D7S_S<4>" LOC = "P14";  # Seg. E; shared with LED<15>
#NET "D7S_S<5>" LOC = "P11";  # Seg. F; shared with LED<9>
#NET "D7S_S<6>" LOC = "P20";  # Seg. G; shared with LED<13>
#NET "D7S_S<7>" LOC = "P18";  # Seg. DP;shared with LED<11>

# Rotary encoder
#NET "ROT_A" LOC = "P36";  # shared with Key<5>
#NET "ROT_B" LOC = "P24";  # shared with Key<6>

# PS/2
#NET "PS2_Clk"   LOC = "P3";
#NET "PS2_Data"  LOC = "P2";

# RS-232
NET "RS_RX" LOC = "P1";
#NET "RS_TX" LOC = "P44";

```

Rysunek 1-4 Plik z rozszerzeniem .ucf

W pliku .ucf widoczne jest przypisanie wyjścia do diody odpowiadającej za pokazanie, czy sekwencja została rozpoznana oraz odkomentowanie wejścia dla klawiatury.

2. Sumator pracujący na dwóch argumentach 4-bitowych wyrażonych w kodzie BCD i generujący stosowny wynik w tym samym kodzie

Synteza:

n	Q3	Q2	Q1	Q0	C
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

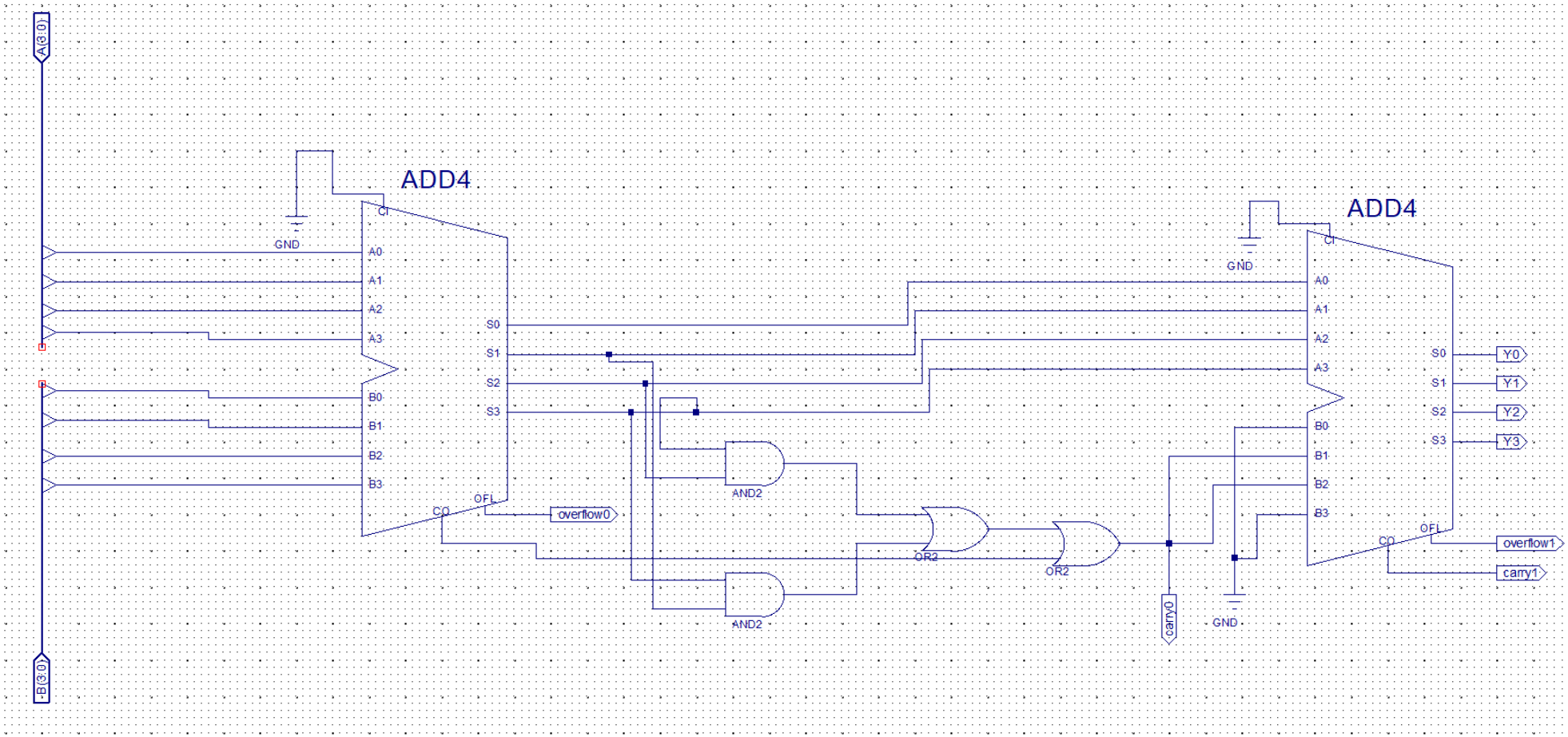
C

Q3Q2/Q1Q0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$C = Q_3Q_2 + Q_3Q_1$$

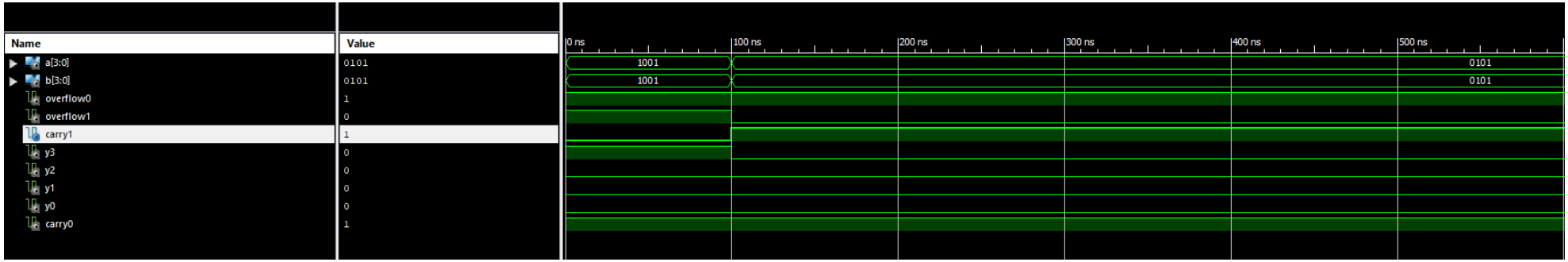
Dodajemy do tego jeszcze jeden sumator, gdzie dodajemy ‘0xx0’. ‘0’ są stałe, a ‘x’ zależy od wyjścia z przeniesienia z poprzedniego sumatora.

Schemat:



Rysunek 2-1 Schemat zadania 2.

Symulacja:



Rysunek 2-2 Symulacja

Plik VHDL:

```
15  LIBRARY ieee;
16  USE ieee.std_logic_1164.ALL;
17  USE ieee.numeric_std.ALL;
18  LIBRARY UNISIM;
19  USE UNISIM.Vcomponents.ALL;
20  ENTITY sch2_sch2_sch_tb IS
21  END sch2_sch2_sch_tb;
22  ARCHITECTURE behavioral OF sch2_sch2_sch_tb IS
23
24     COMPONENT sch2
25     PORT( A  :  IN STD_LOGIC_VECTOR (3 DOWNTO 0);
26           B :  IN STD_LOGIC_VECTOR (3 DOWNTO 0);
27           overflow0  :  OUT   STD_LOGIC;
28           overflow1  :  OUT   STD_LOGIC;
29           carry1  :  OUT   STD_LOGIC;
30           Y3   :  OUT   STD_LOGIC;
31           Y2   :  OUT   STD_LOGIC;
32           Y1   :  OUT   STD_LOGIC;
33           Y0   :  OUT   STD_LOGIC;
34           carry0  :  OUT   STD_LOGIC);
35     END COMPONENT;
36
37     SIGNAL A :  STD_LOGIC_VECTOR (3 DOWNTO 0);
38     SIGNAL B :  STD_LOGIC_VECTOR (3 DOWNTO 0);
39     SIGNAL overflow0  :  STD_LOGIC;
40     SIGNAL overflow1  :  STD_LOGIC;
41     SIGNAL carry1  :  STD_LOGIC;
42     SIGNAL Y3   :  STD_LOGIC;
43     SIGNAL Y2   :  STD_LOGIC;
44     SIGNAL Y1   :  STD_LOGIC;
45     SIGNAL Y0   :  STD_LOGIC;
46     SIGNAL carry0  :  STD_LOGIC;
47
48  BEGIN
49
50     UUT: sch2 PORT MAP(
51        A => A,
52        B => B,
53        overflow0 => overflow0,
54        overflow1 => overflow1,
55        carry1 => carry1,
56        Y3 => Y3,
57        Y2 => Y2,
58        Y1 => Y1,
59        Y0 => Y0,
60        carry0 => carry0
61     );
62
63  -- *** Test Bench - User Defined Section ***
64     A <= "1001", "0101" after 100 ns;
65     B <= "1001", "0101" after 100 ns;
66  -- *** End Test Bench - User Defined Section ***
67
68  END;
```

Rysunek 2-3 Fragment pliku VHDL z pobudzeniami testowymi

Plik UCF:

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

# LEDS
NET "Y0"  LOC = "P35";
NET "Y1"  LOC = "P29";
NET "Y2"  LOC = "P33";
NET "Y3"  LOC = "P34";
NET "carry0"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

Rysunek 2-4 Plik z rozszerzeniem .ucf

W pliku .ucf przypisane zostały ledy pokazujące wyjście w kodzie BCD oraz wszystkie klawisze podające wejście również w kodzie BCD.

3. Konwerter cyfry szesnastkowej zapisanej na czterech bitach od 0 do 9, A do F na kod ASCII tej cyfry – wyjście 8-bitowe

Liczba w systemie binarnym (wejście)	Cyfra w systemie szesnastkowym	Kod ASCII w systemie dziesiętnym	Kod ASCII w systemie binarnym (wyjście)
0000	0	48	0011 0000
0001	1	49	0011 0001
0010	2	50	0011 0010
0011	3	51	0011 0011
0100	4	52	0011 0100
0101	5	53	0011 0101
0110	6	54	0011 0110
0111	7	55	0011 0111
1000	8	56	0011 1000
1001	9	57	0011 1001
1010	A	65	0100 0001
1011	B	66	0100 0010
1100	C	67	0100 0011

1101	D	68	0100 0100
1110	E	69	0100 0101
1111	F	70	0100 0110

Jak można zauważyć w powyższej tabeli, kod ASCII cyfry od 0 do 9 otrzymujemy zmieniając tylko 4 najmłodsze bity o 1. 4 najstarsze pozostają bez zmian („0011”). Podobnie z kodem ASCII cyfry od A do F – 4 najstarsze bity zawsze wyglądają tak: „0100”, jedynie 4 młodsze ulegają zmianie o 1. Co więcej, kod ASCII cyfry 9 i litery A różni się od siebie o 7. Korzystając z zauważonych zależności konstruujemy układ.

Na wejściu podajemy 4-bitową liczbę, która jest odpowiednikiem cyfry szesnastkowej. W konstrukcji układu skorzystano z 2 sumatorów 4-bitowych. Do wejść bitów sumatora znajdującego się na górze z oznaczeniami B podpięte są bity wprowadzanej liczby na wejściu. Natomiast przed wejściami z oznaczeniami A znajdują się bramki logiczne, które wykrywają czy na wejściu układu podano liczbę z zakresu [10;15]:

	$Q_3Q_2Q_1Q_0$	Y
0	0000	0
1	0001	0
2	0010	0
3	0011	0
4	0100	0
5	0101	0
6	0110	0
7	0111	0
8	1000	0
9	1001	0
10	1010	1
11	1011	1
12	1100	1
13	1101	1
14	1110	1
15	1111	1

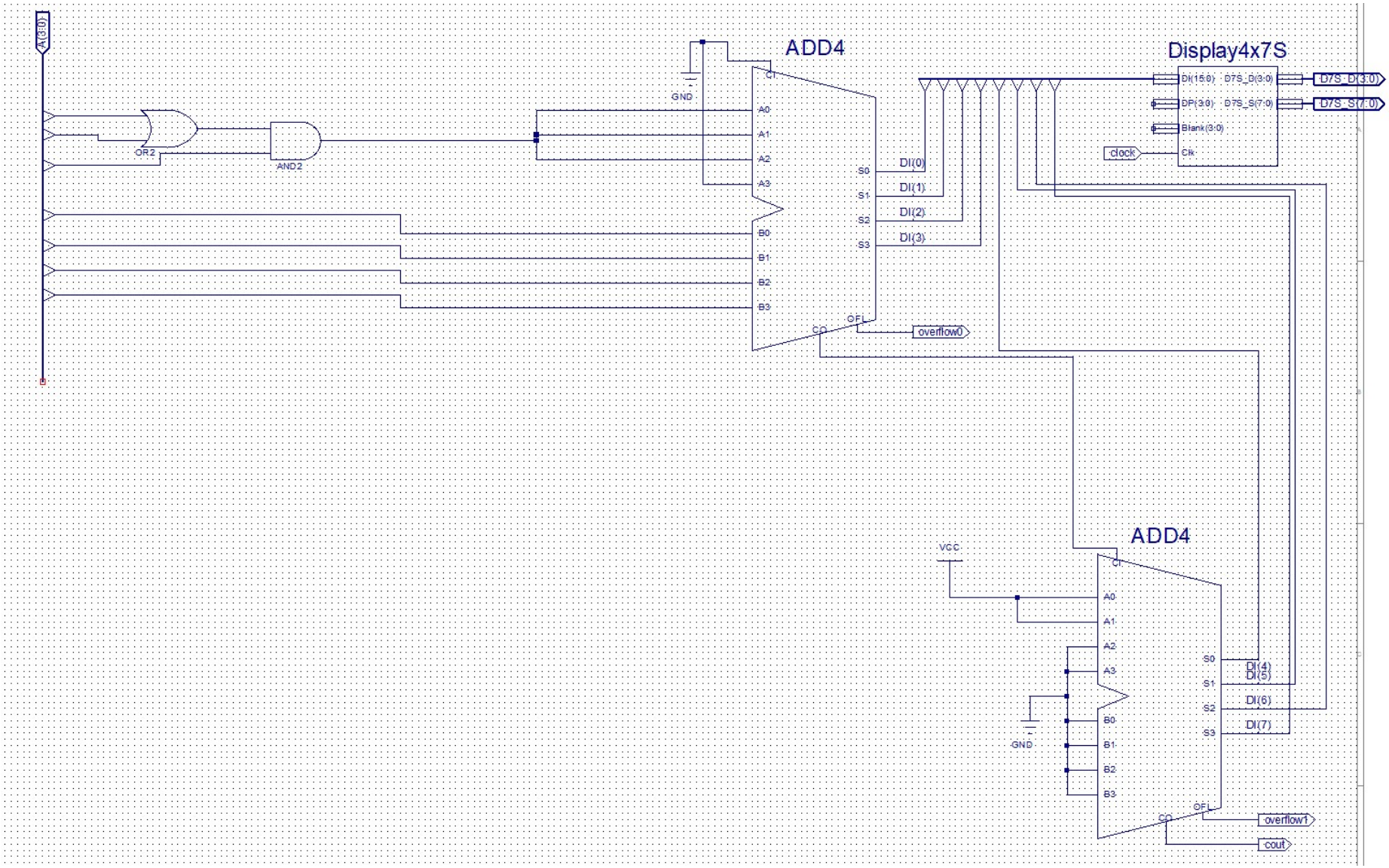
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Q_3Q_2 \ Q_1Q_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

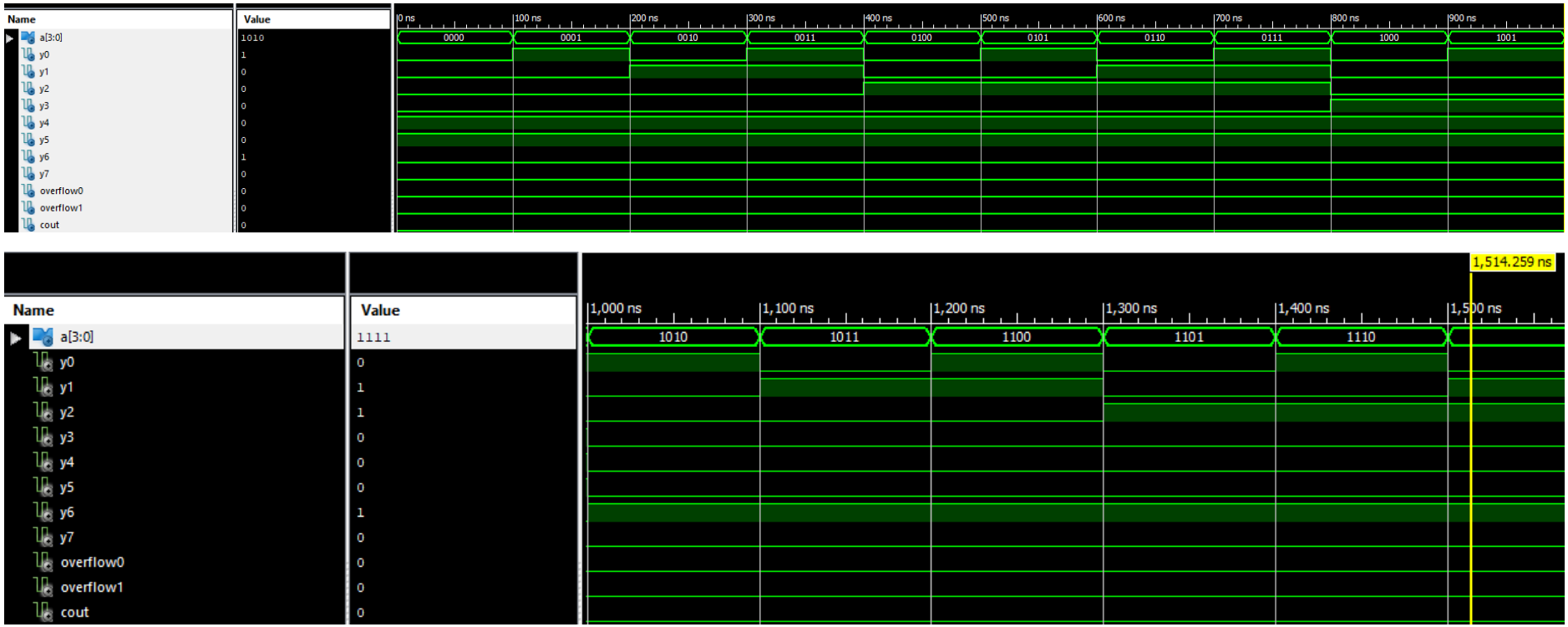
$$Y = Q_3Q_2 + Q_3Q_1 = Q_3(Q_1 + Q_2)$$

Jeśli tak, to na wejścia z oznaczeniami A podajemy liczbę 7 („0111”) w celu dodania jej do liczby podanej na wejściu układu. Jeśli nie to do liczby podanej na wejściu dodajemy 0 („0000”). Na wyjściu sumatora znajdującego się na górze otrzymujemy 4 najmłodsze bity kodu ASCII. Natomiast sumator znajdujący się na dole otrzymuje na wejściu liczbę A=”0011” oraz B=”0000”. Jeśli na wyjściu pożyczki CO sumatora znajdującego się na górze pojawi się sygnał „1”, a będzie się pojawiał w momencie podania na wejściu układu liczby z zakresu [10;15], wtedy na wyjściu dolnego sumatora otrzymamy liczbę „0100”. Na wyjściu dolnego sumatora otrzymujemy 4 najstarsze bity kodu ASCII.

Po sprawdzeniu poprawności działania układu na płytce na diodach, w układzie podpięto wyświetlacz 7-segmentowy. Na wyświetlaczu pojawiał się kod ASCII w systemie szesnastkowym, w zależności od wprowadzonej liczby na wejściu układu za pomocą guzików.



Rysunek 3-1 Układ konwertera cyfry szesnastkowej na kod ASCII



Rysunek 3-2 Symulacja

```
24      COMPONENT sch3
25      PORT( A   :   IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
26            Y0    :   OUT STD_LOGIC;
27            Y1    :   OUT STD_LOGIC;
28            Y2    :   OUT STD_LOGIC;
29            Y3    :   OUT STD_LOGIC;
30            Y4    :   OUT STD_LOGIC;
31            Y5    :   OUT STD_LOGIC;
32            Y6    :   OUT STD_LOGIC;
33            Y7    :   OUT STD_LOGIC;
34            overflow0 :   OUT STD_LOGIC;
35            overflow1 :   OUT STD_LOGIC;
36            cout  :   OUT STD_LOGIC);
37      END COMPONENT;
38
39      SIGNAL A :   STD_LOGIC_VECTOR (3 DOWNTO 0);
40      SIGNAL Y0    :   STD_LOGIC;
41      SIGNAL Y1    :   STD_LOGIC;
42      SIGNAL Y2    :   STD_LOGIC;
43      SIGNAL Y3    :   STD_LOGIC;
44      SIGNAL Y4    :   STD_LOGIC;
45      SIGNAL Y5    :   STD_LOGIC;
46      SIGNAL Y6    :   STD_LOGIC;
47      SIGNAL Y7    :   STD_LOGIC;
48      SIGNAL overflow0 :   STD_LOGIC;
49      SIGNAL overflow1 :   STD_LOGIC;
50      SIGNAL cout  :   STD_LOGIC;
51
52  BEGIN
53
54      UUT: sch3 PORT MAP(
55           A => A,
56           Y0 => Y0,
57           Y1 => Y1,
58           Y2 => Y2,
59           Y3 => Y3,
60           Y4 => Y4,
61           Y5 => Y5,
62           Y6 => Y6,
63           Y7 => Y7,
64           overflow0 => overflow0,
65           overflow1 => overflow1,
66           cout => cout
67      );
68
69  -- *** Test Bench - User Defined Section ***
70      A<="0000", "0001" after 100 ns, "0010" after 200 ns, "0011" after 300 ns,"0100" after 400 ns,
71      "0101" after 500 ns, "0110" after 600 ns, "0111" after 700 ns, "1000" after 800 ns,
72      "1001" after 900 ns, "1010" after 1000 ns, "1011" after 1100 ns, "1100" after 1200 ns,
73      "1101" after 1300 ns, "1110" after 1400 ns, "1111" after 1500 ns;
74  -- *** End Test Bench - User Defined Section ***
75
76  END;
```

Rysunek 3-3 Fragment pliku VHDL z pobudzeniami testowymi

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

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

# LEDS
#NET "Y0"  LOC = "P35";
#NET "Y1"  LOC = "P29";
#NET "Y2"  LOC = "P33";
#NET "Y3"  LOC = "P34";
#NET "Y4"  LOC = "P28";
#NET "Y5"  LOC = "P27";
#NET "Y6"  LOC = "P26";
#NET "Y7"  LOC = "P25";

#NET "LED<8>"  LOC = "P13";   # shared with seg. B
#NET "LED<9>"  LOC = "P11";   # shared with seg. F
#NET "LED<10>" LOC = "P12";   # shared with seg. A
#NET "LED<11>" LOC = "P18";   # shared with seg. DP
#NET "LED<12>" LOC = "P22";   # shared with seg. C
#NET "LED<13>" LOC = "P20";   # shared with seg. G
#NET "LED<14>" LOC = "P19";   # shared with seg. D
#NET "LED<15>" LOC = "P14";   # shared with seg. E

# DISPL. 7-SEG
NET "D7S_D<0>" LOC = "P8" | SLEW = "SLOW";
NET "D7S_D<1>" LOC = "P6" | SLEW = "SLOW";
NET "D7S_D<2>" LOC = "P4" | SLEW = "SLOW";
NET "D7S_D<3>" LOC = "P9" | SLEW = "SLOW";
NET "D7S_S<0>" LOC = "P12";   # Seg. A; shared with LED<10>
NET "D7S_S<1>" LOC = "P13";   # Seg. B; shared with LED<8>
NET "D7S_S<2>" LOC = "P22";   # Seg. C; shared with LED<12>
NET "D7S_S<3>" LOC = "P19";   # Seg. D; shared with LED<14>
NET "D7S_S<4>" LOC = "P14";   # Seg. E; shared with LED<15>
NET "D7S_S<5>" LOC = "P11";   # Seg. F; shared with LED<9>
NET "D7S_S<6>" LOC = "P20";   # Seg. G; shared with LED<13>
NET "D7S_S<7>" LOC = "P18";   # Seg. DP;shared with LED<11>

```

Rysunek 3-4 Plik z rozszerzeniem .ucf

W pliku z rozszerzeniem *.ucf* w celu użycia wyświetlacza odkomentowano przypisane wejścia do wyprowadzeń pod sekcją „DISPL. 7-SEG” oraz przypisano wejście zegarowe „clock”. Przyciski K7-K4 pozwalają na wprowadzenie liczby na wejście układu.

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

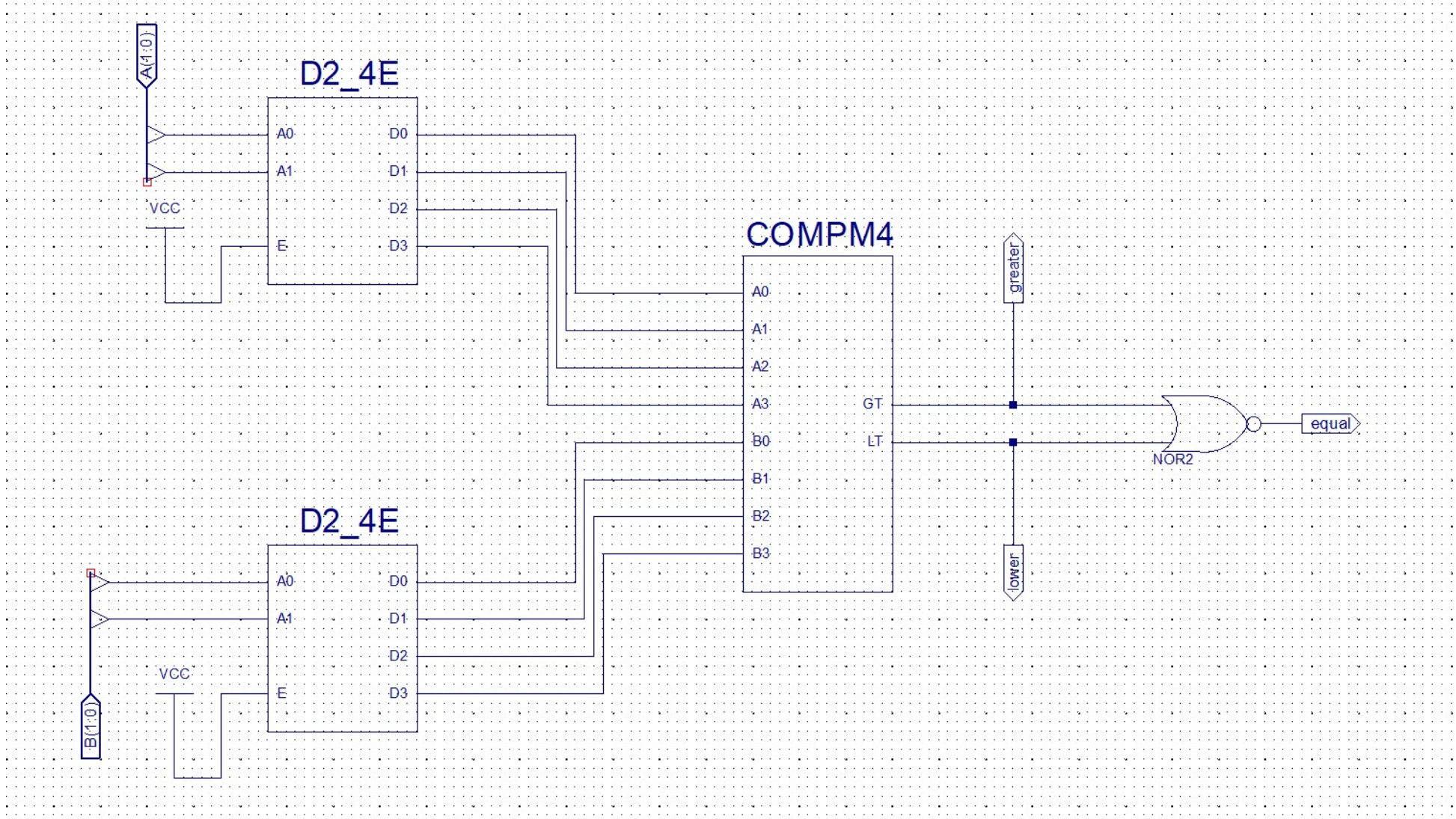
Kod 1 z n to kod, w którym słowa binarne o długości n bitów zawierają zawsze tylko i wyłącznie 1 bit o wartości „1”. Pozostałe bity mają wartość „0”. Kod 1 z 4 prezentuje się następująco:

n		Kod 1 z 4
0	00	0001
1	01	0010
2	10	0100
3	11	1000

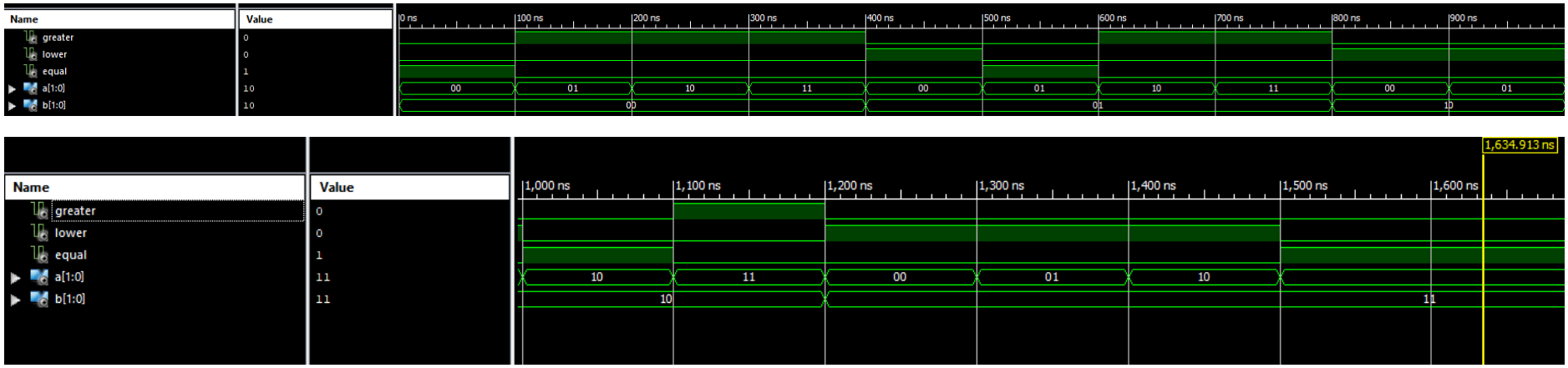
W układzie wykorzystano 2 dekodery 4-bitowe. Dekoder zamienia kod binarny podany na wejściu na kod 1 z n. Następnie obie reprezentacje kodu 1 z 4 są poprowadzone do 4-bitowego komparatora. W tym układzie nie były wymagane żadne szczególne modyfikacje, ponieważ porównywanie liczb w kodzie 1 z 4 wygląda identycznie jak w kodzie NKB:

A	B	A>B	A=B	A<B
0001	0001	0	1	0
0001	0010	0	0	1
0001	0100	0	0	1
0001	1000	0	0	1
0010	0001	1	0	0
0010	0010	0	1	0
0010	0100	0	0	1
0010	1000	0	0	1
0100	0001	1	0	0
0100	0010	1	0	0
0100	0100	0	1	0
0100	1000	0	0	1
1000	0001	1	0	0
1000	0010	1	0	0
1000	0100	1	0	0
1000	1000	0	1	0

Na wyjściu komparatora otrzymujemy flagę „GT”, która przyjmie wartość „1”, gdy A>B oraz flagę „LT”, która przyjmie wartość „1” gdy A<B. Dodatkowo, na wyjściu układu zastosowano bramkę NOR, w celu otrzymania flagi, która przyjmie wartość „1” gdy A=B. Stanie się to w chwili, gdy GT=LT=0, czyli liczba nie będzie ani większa ani mniejsza.



Rysunek 4-1 Układ komparatora 2 liczb 4-bitowych w kodzie 1 z 4



Rysunek 4-2 Symulacja

```
15  LIBRARY ieee;
16  USE ieee.std_logic_1164.ALL;
17  USE ieee.numeric_std.ALL;
18  LIBRARY UNISIM;
19  USE UNISIM.Vcomponents.ALL;
20  ENTITY sch4_sch4_sch_tb IS
21  END sch4_sch4_sch_tb;
22  ARCHITECTURE behavioral OF sch4_sch4_sch_tb IS
23
24     COMPONENT sch4
25     PORT( greater    :   OUT STD_LOGIC;
26           lower :   OUT STD_LOGIC;
27           equal :   OUT STD_LOGIC;
28           A :   IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
29           B :   IN  STD_LOGIC_VECTOR (1 DOWNTO 0));
30     END COMPONENT;
31
32     SIGNAL greater    :   STD_LOGIC;
33     SIGNAL lower :   STD_LOGIC;
34     SIGNAL equal :   STD_LOGIC;
35     SIGNAL A :   STD_LOGIC_VECTOR (1 DOWNTO 0);
36     SIGNAL B :   STD_LOGIC_VECTOR (1 DOWNTO 0);
37
38  BEGIN
39
40     UUT: sch4 PORT MAP(
41          greater => greater,
42          lower => lower,
43          equal => equal,
44          A => A,
45          B => B
46     );
47
48  -- *** Test Bench - User Defined Section ***
49     A <= "00", "01" after 100 ns, "10" after 200 ns, "11" after 300 ns, "00" after 400 ns,
50     "01" after 500 ns, "10" after 600 ns, "11" after 700 ns, "00" after 800 ns,
51     "01" after 900 ns, "10" after 1000 ns, "11" after 1100 ns, "00" after 1200 ns,
52     "01" after 1300 ns, "10" after 1400 ns, "11" after 1500 ns;
53      B <= "00", "01" after 400 ns, "10" after 800 ns, "11" after 1200 ns;
54  -- *** End Test Bench - User Defined Section ***
55
56  END;
```

Rysunek 4-3 Fragment pliku VHDL z pobudzeniami testowymi

```
5   # Keys
6   NET "A(0)" LOC = "P42";
7   NET "A(1)" LOC = "P40";
8   NET "B(0)" LOC = "P43";
9   NET "B(1)" LOC = "P38";
10  #NET "Key<4>" LOC = "P37";
11  #NET "Key<5>" LOC = "P36";  # shared with ROT_A
12  #NET "Key<6>" LOC = "P24";  # shared with ROT_B
13  #NET "Key<7>" LOC = "P39";  # GSR
14
15  # LEDS
16  NET "greater"  LOC = "P35";
17  NET "lower"  LOC = "P29";
18  NET "equal"  LOC = "P33";
19  #NET "LED<3>"  LOC = "P34";
```

Rysunek 4-4 Fragment pliku z rozszerzeniem .ucf

Przyciski K1-K0 pozwalają wprowadzić liczbę A, natomiast K3-K2 liczbę B w kodzie NKB do układu. Dioda nr 0 przestaje świecić gdy A>B, dioda nr 1 przestaje świecić gdy A<B a dioda nr 2 przestaje świecić, gdy A=B.

5. Wnioski

Wszystkie układy działały poprawnie.