Układy cyfrowe i systemy wbudowane

SPRAWOZDANIE NR 3

Układy wielobitowych wejść i wyjść

PROWADZĄCY:

DR INŻ. JACEK MAZURKIEWICZ

AUTORZY:

MATEUSZ GRUSZKA 249448

BARTOSZ RUDNIK 248893

Grupa: C

TERMIN ZAJĘĆ:

WT 9:15 TP

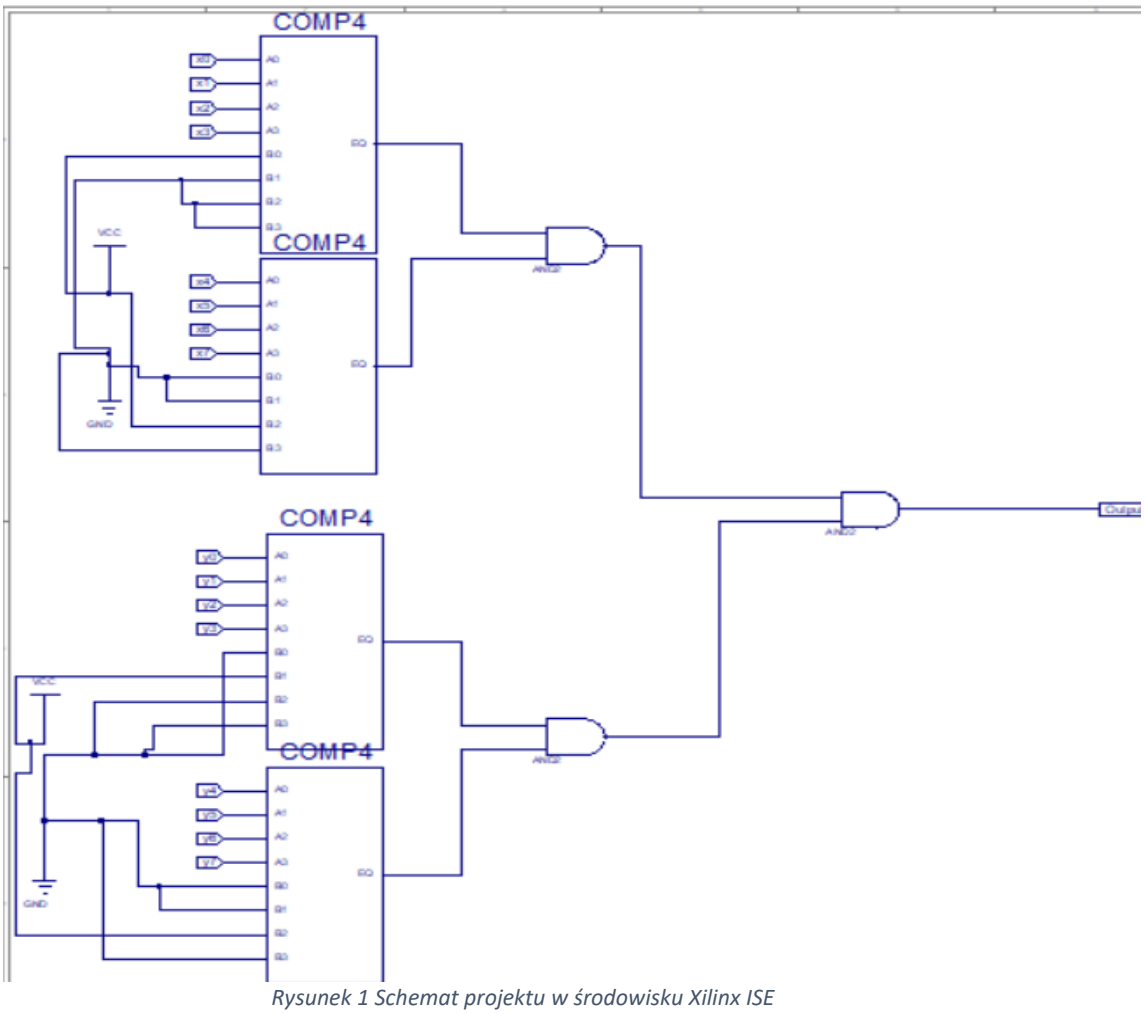
WROCŁAW, 24.11.2020 r.

1. Cel ćwiczenia

Celem ćwiczenia było zrealizowanie środowisku Xilinx ISE czterech układów wielobitowych wejść i wyjść. Był to detektor 2-znakowej sekwencji słów 8-bitowych, 4-bitowy sumator pracujący w kodzie Graya, konwerter cyfry szesnastkowej zapisanej na czterech bitach na kod ASCII tej cyfry oraz komparator dwóch 4-bitowych cyfr.

2. Detektor 2-znakowej sekwencji słów 8-bitowych

Detektor na wejście przyjmuje 2 znaki 8-bitowe. Wyjście detektora jest jednobitowe i przedstawia komunikat sekwencja rozpoznana lub sekwencja nierozpoznana. Testowaną przez nas sekwencją znaków była sekwencja ‘AB’, gdzie ‘A’ = 0100 0001, a ‘B’ = 0100 0010. Koncepcja, którą przyjęliśmy podczas wykonywania tego układu zakłada użycie gotowych modeli komparatorów udostępnionych nam w bibliotekach środowiska Xilinx ISE. Korzystając ze wspomnianych komparatorów porównujemy pierwsze 8-bitów na wejściu z zapisem bitowym znaku ‘A’ i 8 kolejnych bitów na wejściu ze znakiem ‘B’. Następnie korzystając z bramki logicznej AND sprawdzamy czy w obu przypadkach bity na wejściu są takie same co bity porównywanych znaków. Jeśli bity są takie same oznacza to rozpoznanie pożądanej sekwencji.

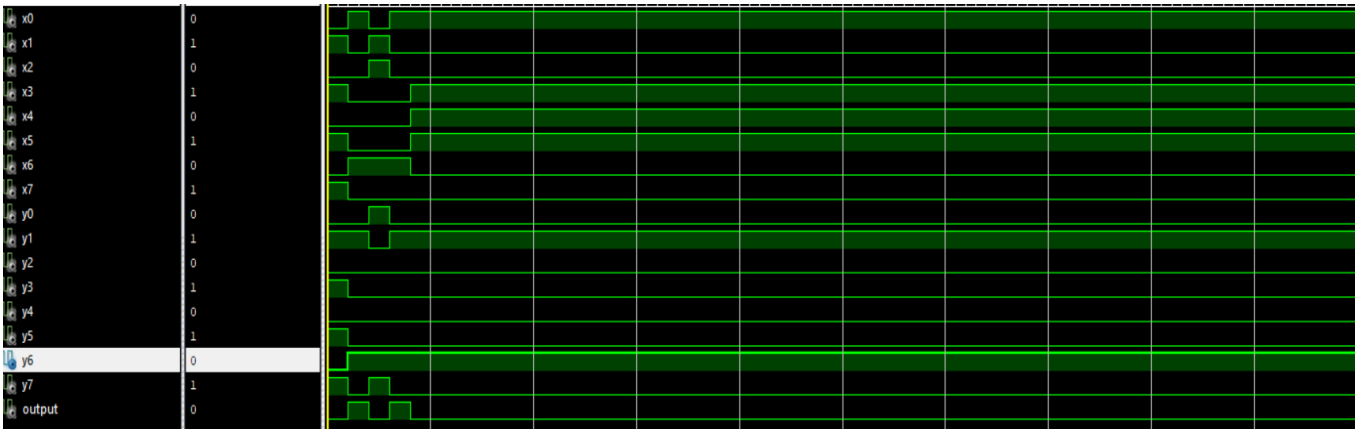


Rysunek 1 Schemat projektu w środowisku Xilinx ISE

Kombinacje sygnałów wejściowych zapisane w pliku testowym VHDL:

```
x0 <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
x1 <= '1', '0' after 20 ns, '1' after 40 ns, '0' after 60 ns;
x2 <= '0', '0' after 20 ns, '1' after 40 ns, '0' after 60 ns;
x3 <= '1', '0' after 20 ns, '1' after 80 ns;
x4 <= '0', '0' after 20 ns, '1' after 80 ns;
x5 <= '1', '0' after 20 ns, '1' after 80 ns;
x6 <= '0', '1' after 20 ns, '0' after 80 ns;
x7 <= '1', '0' after 20 ns;

y0 <= '0', '0' after 20 ns, '1' after 40 ns, '0' after 60 ns;
y1 <= '1', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
y2 <= '0', '0' after 20 ns;
y3 <= '1', '0' after 20 ns;
y4 <= '0', '0' after 20 ns;
y5 <= '1', '0' after 20 ns;
y6 <= '0', '1' after 20 ns;
y7 <= '1', '0' after 20 ns, '1' after 40 ns, '0' after 60 ns;
```



Rysunek 2 Wynik działania symulacji dla detektora 2-znakowej sekwencji

3. 4-bitowy sumator pracujący w kodzie Gray'a

Kod Gray'a jest dwójkowym, bez wagowym, nie pozycyjnym kodem, którego dwa kolejne słowa kodowe różnią się wyłącznie stanem jednego bitu. Kod Gray'a jest także kodem cyklicznym ponieważ jego pierwszy i ostatni wyraz również różnią się wyłącznie stanem jednego bita.

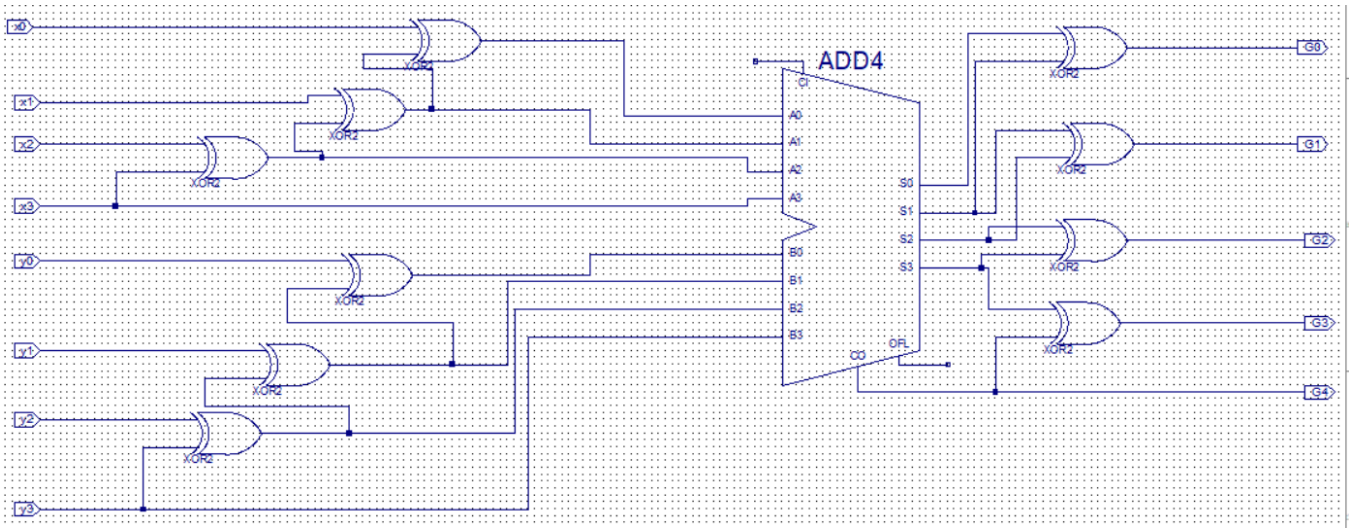
Rozwiązywanie tego zadania rozpoczęliśmy od przekształcenie wejścia podanego w kodzie Gray'a na kod naturalny binarny. Następnie wykorzystaliśmy gotowy model sumatora oferowany przez biblioteki dostępne w środowisku Xilinx ISE. W ostatnim kroku przekształciliśmy wynik otrzymany z sumatora z powrotem do kodu Gray'a.

Przekształcania z kodu Gray'a na kod naturalny binarny wykonaliśmy według algorytmu:

1. Najstarszy bit kodu Gray'a kopiujemy do najstarszego bitu kodu naturalnego binarnego.
2. Pozostałe bity obliczamy ze wzoru $B_i = G_i$ XOR B_{i+1}, gdzie B_i jest i-tym bitem wyrazu naturalnego binarnego, a G_i jest i-tym bitem wyrazu kodu Graya.

Przekształcenie z kodu naturalnego binarnego na kod Gray'a wykonaliśmy według algorytmu:

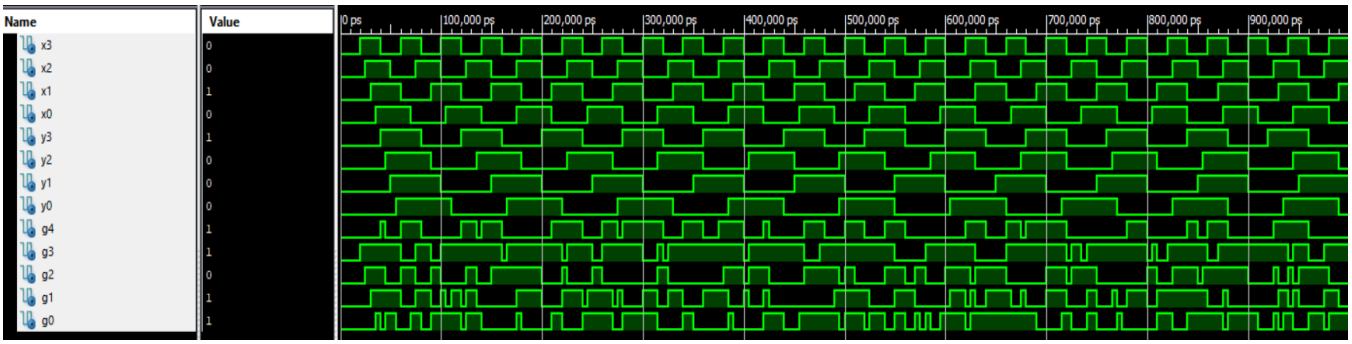
1. Najstarszy bit kodu naturalnego binarnego kopiujemy do najstarszego bitu wyrazu kodu Gray'a.
2. Pozostałe bity obliczamy ze wzoru Gi = Bi XOR Bi+1, gdzie Gi jest i-tym bitem wyrazu kodu Gray'a, a Bi jest i-tym bitem wyrazu kodu naturalnego binarnego.



Rysunek 3 Schemat projektu w środowisku Xilinx ISE

Kombinacje sygnałów wejściowych zapisane w pliku testowym VHDL:

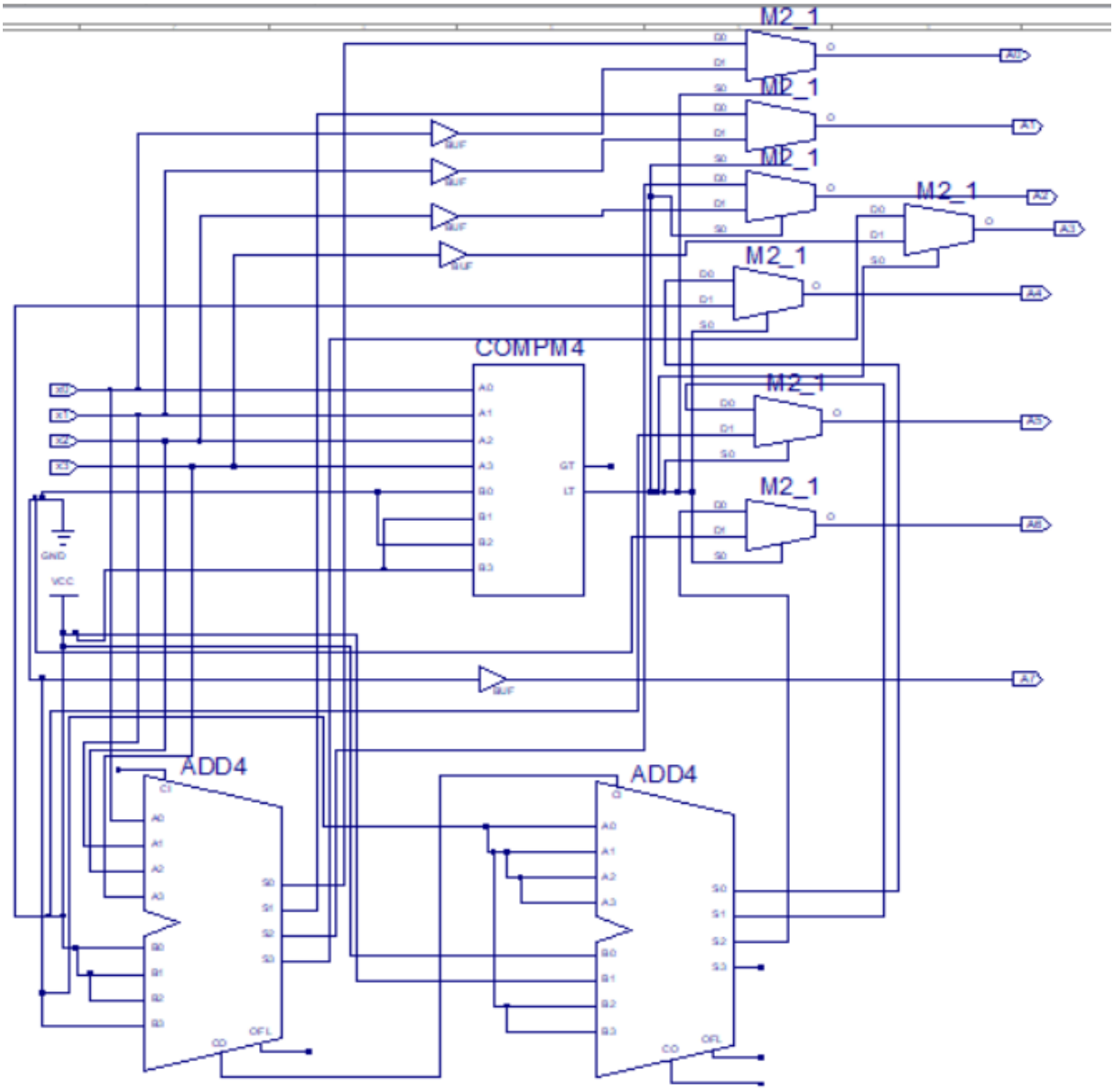
```
x3 <= not x3 after 20 ns;
 x2 <= not x2 after 25 ns;
 x1 <= not x1 after 30 ns;
 x0 <= not x0 after 35 ns;
 y3 <= not y3 after 40 ns;
 y2 <= not y2 after 45 ns;
 y1 <= not y1 after 50 ns;
 y0 <= not y0 after 55 ns;
```



Rysunek 4 Wynik działania symulacji dla sumatora działającego w kodzie Graya

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

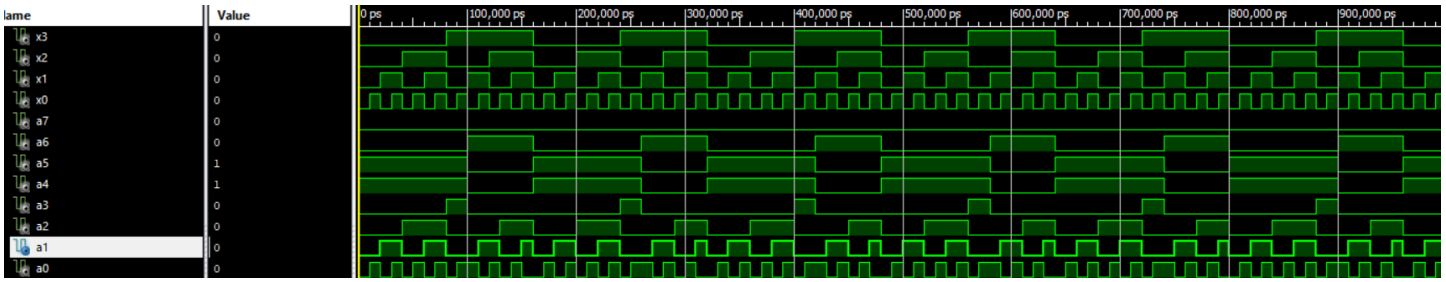
W celu rozwiązania tego zadania podzieliliśmy je na dwa mniejsze podzadanie. Pierwszym podzadaniem było obsłużenie cyfr od 0 do 9. Aby uzyskać kod ASCII cyfry z zakresu [0,9] należy dodać do niej wartość $(48)_{10} = (0011\ 0000)_2$. Ponieważ wszystkie cyfry z zakresu [0,9] mają postać $(0000\ xxxx)_2$, dodawanie nie jest konieczne, a wystarczające do uzyskania oczekiwanego rezultatu jest przepisanie bitów wejściowych i czterech najstarszych bitów liczby $(48)_{10}$. Drugie podzadanie polega na dodaniu do bitów wejściowych wartości $(55)_{10}$. Tym razem konieczne jest faktyczne wykonanie dodawania, w tym celu wykorzystaliśmy dostępny w bibliotece Xilinx ISE moduł sumatora. Do połączenia dwóch podzadań w jeden schemat użyliśmy komparatora oraz multiplekserów. W zależności od sygnału wyjściowego komparatora multipleksery realizowały albo wariant dla cyfr od 0 do 9 albo wariant dla cyfr A do F.



Rysunek 5 Schemat w środowisku Xilinx ISE

Kombinacje sygnałów wejściowych zapisane w pliku testowym VHDL:

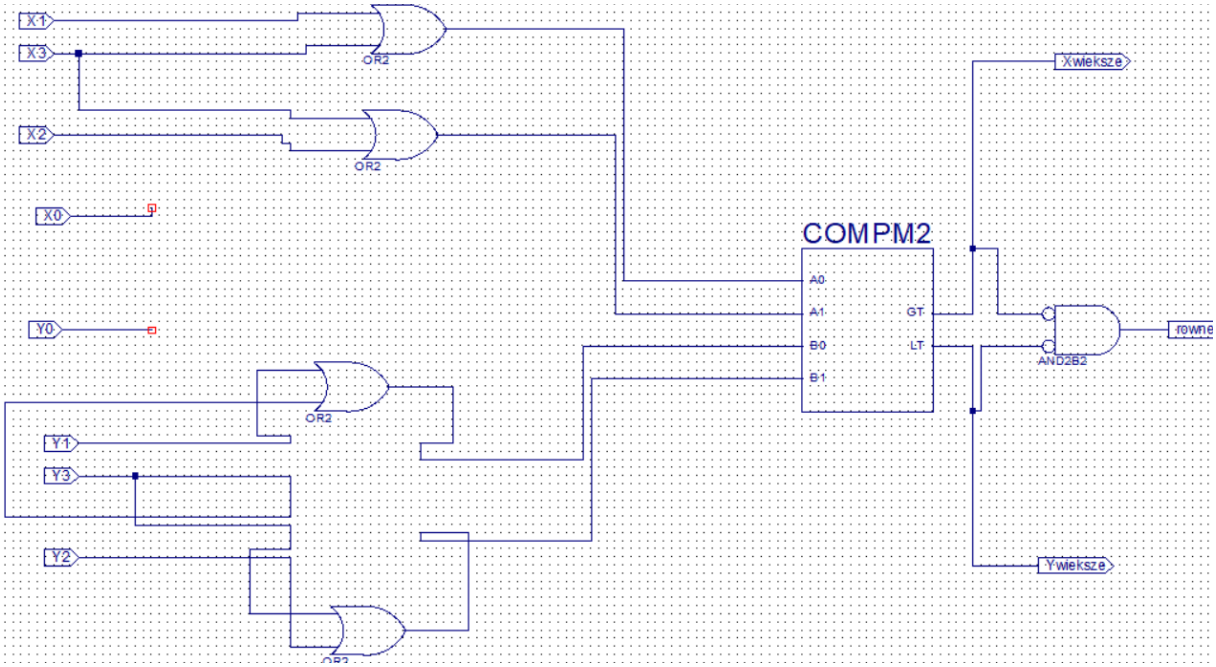
```
x0 <= not x0 after 10 ns;
 x1 <= not x1 after 20 ns;
 x2 <= not x2 after 40 ns;
 x3 <= not x3 after 80 ns;
```



Rysunek 6 Wynik działania symulacji dla konwertera cyfry szesnastkowej

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

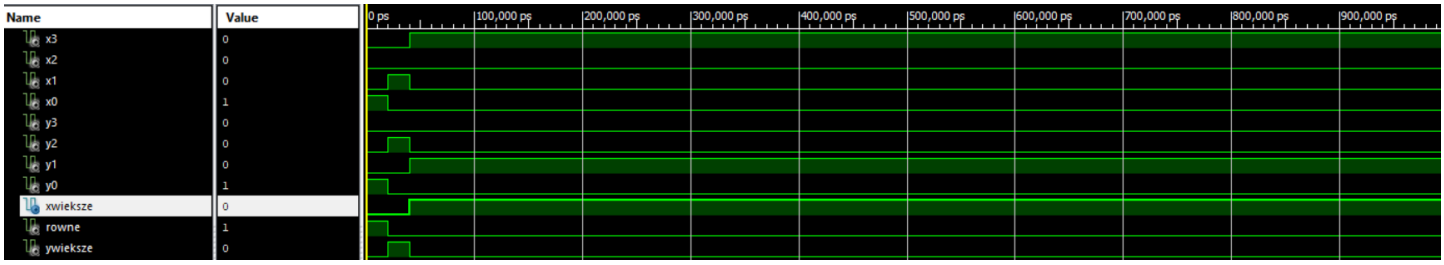
W celu rozwiązania tego zadania wykonaliśmy konwersje z kodu 2 z 4 do kodu naturalnego binarnego. Następnym podjętym przez nas krokiem było użycie komparatora w celu porównania wejściowych bitów. Użyty przez nas model komparatora posiada dwa wyjścia. Wyjście GT przyjmuje stan aktywny gdy wejścia A są większe niż wejścia B. Wyjście LT przyjmuje stan aktywny gdy wejścia B są większe niż wejścia A. W celu obsłużenia przypadku równy zastosowaliśmy bramkę AND z dwoma zanegowanymi wejściami. Bramka ta zwróci na wyjściu sygnał aktywny jedynie w przypadku gdy oba wyjścia komparatora będą miały stan niski.



Rysunek 7 Schemat w środowisku Xilinx ISE

Kombinacje sygnałów wejściowych zapisane w pliku testowym VHDL:

```
X0 <= '1', '0' after 20 ns, '0' after 40 ns;
X1 <= '0', '1' after 20 ns, '0' after 40 ns;
X2 <= '0', '0' after 20 ns, '0' after 40 ns;
X3 <= '0', '0' after 20 ns, '1' after 40 ns;
Y0 <= '1', '0' after 20 ns, '0' after 40 ns;
Y1 <= '0', '0' after 20 ns, '1' after 40 ns;
Y2 <= '0', '1' after 20 ns, '0' after 40 ns;
Y3 <= '0', '0' after 20 ns, '0' after 40 ns;
```



Rysunek 8 Wynik działania symulacji dla komparatora działającego w kodzie 2 z 4

6. Wnioski

Zrealizowanie powyższych ćwiczeń pozwoliło nam na lepsze zrozumienie i zapoznanie się z układami wielobitowych wejść i wyjść. Układy te w porównaniu do realizowanych na poprzednich zajęciach układów kombinacyjnych i układów sekwencyjnych nie mają gotowego algorytmu gwarantującego uzyskanie rozwiązania. Podobnie jak na przy realizacji poprzednich ćwiczeń bardzo przydatna okazała się wiedza uzyskana na III semestrze podczas realizacji kursu Logika Układów Cyfrowych. Kolejną cenną umiejętnością, którą uzyskaliśmy podczas realizacji układów wielobitowych wejść i wyjść jest umiejętność korzystania z elementów dostępnych w bibliotekach Xilinx ISE. Odpowiednie wykorzystanie tych elementów pozwoliło na sprawne wykonanie wszystkich przedstawionych zadań.