Sprawozdanie 1

Fortuna Wojciech, Ramut Michał, Stylski Bartłomiej, Tendaj Konrad

16 Kwietnia 2024

1 Treść ćwiczenia 1

Bazując wyłącznie na bramkach NAND, **zaprojektować**, zbudować i przetestować układ kombinacyjny realizujący transkoder czterobitowej liczby naturalnej (wraz z zerem) na sześciobitową liczbę pierwszą.

Układ taki powinien zatem zamieniać kolejne liczby:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

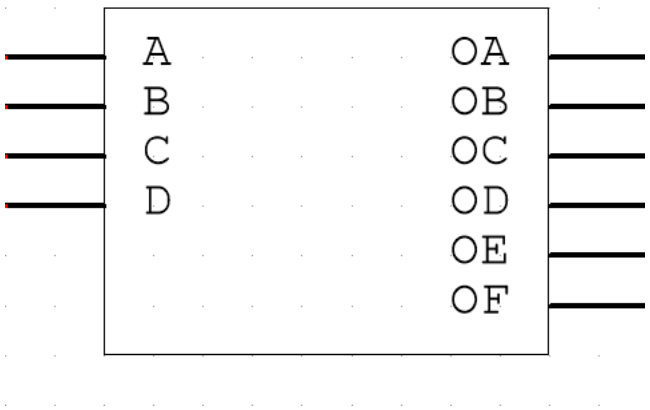
na odpowiednie kolejne liczby pierwsze:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53

Do przetestowania układu należy wykorzystać m.in.: wyświetlacze siedmiosegmentowe, generator słów i analizator stanów logicznych.

Do minimalizacji potrzebnych funkcji należy wykorzystać tablice Karnaugh.

2 Projekt transkodera z treści zadania

Nasz transkoder będzie składać się z 4 wejść i 6 wyjść:



Rysunek 1: Schemat transkodera

W tym celu skorzystamy z narzędzia podobwód (ang. “subcircuit”) w naszym programie Multisim.

Wejścia kolejno oznaczone A, B, C, D prezentują kolejne bity liczby wejściowej (od 0 do 15). Tak samo kolejne oznaczenia od OA, OB, OC, OD, OE, OF oznaczają kolejne bity liczby wyjściowej (od 2 do 53).

2.1 Tabela prawdy

Pierwszym etapem stworzenia takiego transkodera jest stworzenie tabeli prawdy. Tak prezentuje się tabela prawdy naszego układu:

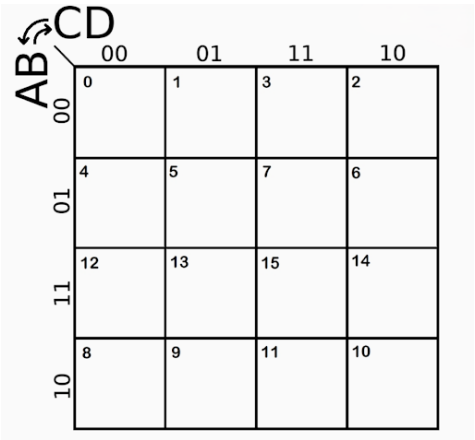
Liczba	A	B	C	D	Y1	Y2	Y3	Y4	Y5	Y6	Pierwsza
0	0	0	0	0	0	0	0	0	1	0	2
1	0	0	0	1	0	0	0	0	1	1	3
2	0	0	1	0	0	0	0	1	0	1	5
3	0	0	1	1	0	0	0	1	1	1	7
4	0	1	0	0	0	0	1	0	1	1	11
5	0	1	0	1	0	0	1	1	0	1	13
6	0	1	1	0	0	1	0	0	0	1	17
7	0	1	1	1	0	1	0	0	1	1	19
8	1	0	0	0	0	1	0	1	1	1	23
9	1	0	0	1	0	1	1	1	0	1	29
10	1	0	1	0	0	1	1	1	1	1	31
11	1	0	1	1	1	0	0	1	0	1	37
12	1	1	0	0	1	0	1	0	0	1	41
13	1	1	0	1	1	0	1	0	1	1	43
14	1	1	1	0	1	0	1	1	1	1	47
15	1	1	1	1	1	1	0	1	0	1	53

Tabela 1: Tabela prawdy

Naszym zadaniem jest znalezienie funkcji Y1, Y2, Y3, Y4, Y5 i Y6 przekształcających kolejne ciągi liczb w konkretne wartości z tabeli. Te funkcje przydadzą nam się później do stworzenia zestawu bramek logicznych do zaimplementowania tych funkcji w programie.

2.2 Tablice Karnaugh i wzory funkcji

Do znalezienia powyższych funkcji przydatne będą nam tutaj tablice Karnaugh. Taka tablica Karnaugh dla 4 zmiennych wygląda następująco:



Rysunek 2: Tablica Karnaugh

Jak widać w wierszach tablicy zapisujemy kolejne możliwe kombinacje wartości AB od 00, 01, 11 do 10. Tak samo robimy w przypadku kolumn tylko dla zmiennych CD.

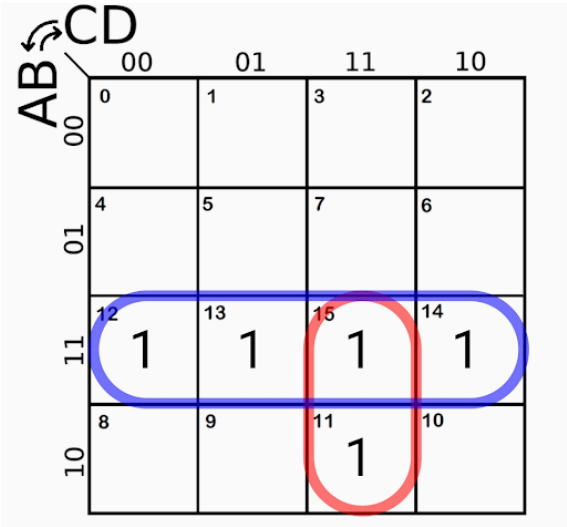
Gdy chcemy przepisać wartości z tabeli prawdy to po kolei patrzymy każdą wartość z zapisujemy w danym miejscu 0 lub 1. Np. mając liczbę binarną 1011, z której wynika wartość 1, wyciągamy dane, że $AB = 10, CD = 11$. Następnie odszukujemy takie miejsce w tablicy. W naszym przypadku będzie to pole 11. Wpisujemy w to pole wartość wyrażenia (czyli u nas 1).

Dla uproszczenia i nie zaciemniania tablicy nie zapisujemy wartości 0. Są u nas polami pustymi.

Następny etap jest nieco bardziej skomplikowany, bo łączymy w pary (lub więcej) nasze 1 i później odczytujemy jak wygląda ułożenie zmiennych w tych polach. Nie chcę się tutaj rozpisywać o tej metodzie, dlatego po więcej informacji odsyłam do wikipedii: link do strony.

Przechodząc do rozwiązania naszych funkcji (oczywiście symbol z linią nad nim oznacza 0 przy danym symbolu):

Tablica Karnaugh dla funkcji Y1:



Rysunek 3: Tablica dla funkcji Y1

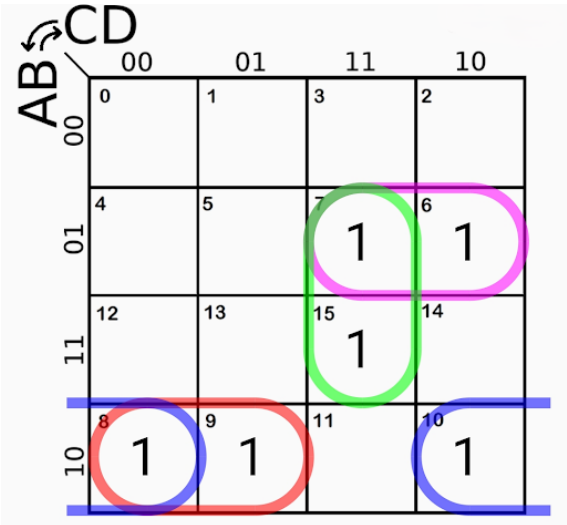
Możemy z tego wywnioskować, że nasza funkcji Y1 ma taki wzór:

$Y1 = ACD + AB$

ACD - ponieważ oba pola 11 i 15 leżą w miejscu, w którym A = 1, oraz w kolumnie w miejscu, gdzie CD = 11.

AB - ponieważ w miejscu, gdzie AB = 11

Tablica Karnaugh dla funkcji Y2:



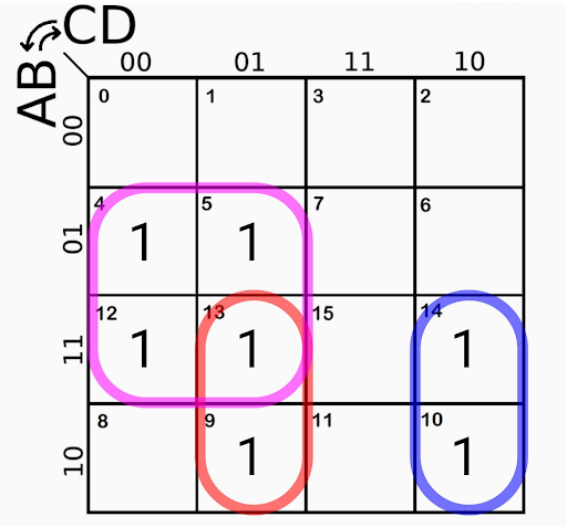
Rysunek 4: Tablica dla funkcji Y2

Wyciągamy informacje o Y2:

$Y2 = A\overline{BC} + A\overline{BD} + \overline{A}BC + BCD$

Jak widać możemy tutaj sklejać grupy również na granicach jak to jest zrobione w $A\overline{BD}$.

Tablica Karnaugh dla funkcji Y3:

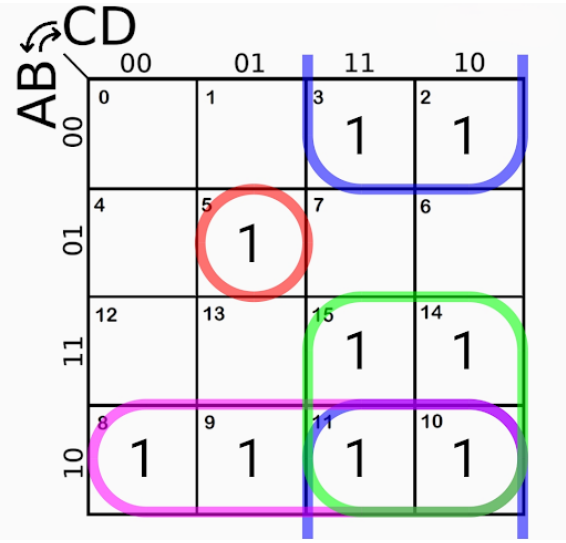


Naszym otrzymanym równaniem Y3 jest:

$Y3 = A\overline{C}D + AC\overline{D} + B\overline{C}$

Rysunek 5: Tablica dla funkcji Y3

Tablica Karnaugh dla funkcji Y4:

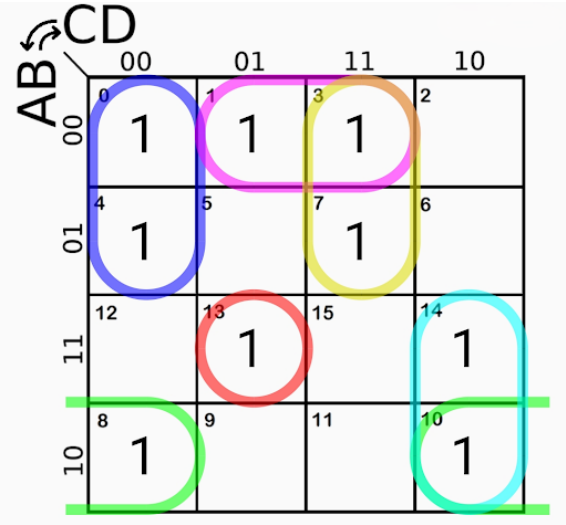


Naszym otrzymanym równaniem Y4 jest:

$Y4 = \overline{A}B\overline{C}D + \overline{B}C + A\overline{B} + AC$

Rysunek 6: Tablica dla funkcji Y4

Tablica Karnaugh dla funkcji Y5:

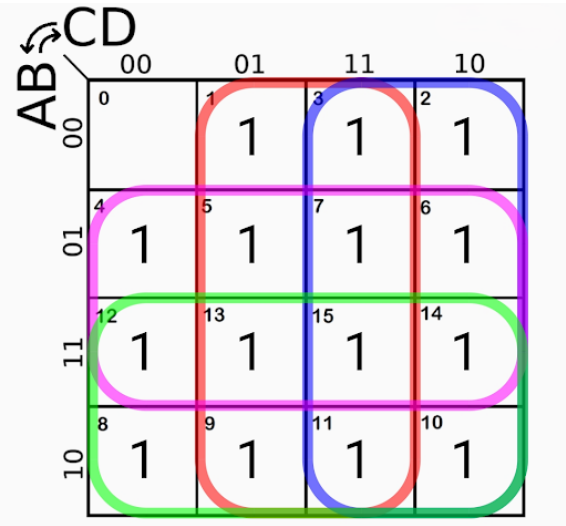


Funkcja Y5 jest najbardziej zaawansowana to implementacji:

$$Y5 = AB\overline{C}D + \overline{A}C\overline{D} + \overline{A}\overline{B}D + A\overline{B}\overline{D} + \overline{A}CD + AC\overline{D}$$

Rysunek 7: Tablica dla funkcji Y5

Tablica Karnaugh dla funkcji Y6:



Funkcja Y6 może i wygląda niepokojąco, ale jest bardzo prosta:

$$Y6 = D + C + B + A$$

Rysunek 8: Tablica dla funkcji Y6

2.3 Transformacja wzorów na NAND

Kolejnym krokiem do stworzenia naszego transkodera jest transformacja wzorów do bramek logicznych. Dodatkowym utrudnieniem w zadaniu jest wymóg stosowania tylko bramek NAND.

Bramkę NAND możemy zapisać również jako: $\overline{AB}$.

Do uproszczenia obwodów będziemy używać ten bramek NAND3, NAND4, czyli bramki NAND z kolejno 3, 4 wejściami.

Przekształcenie wzorów:

Ze wzorów de Morgana wiemy, że: $\overline{A} + \overline{B} = \overline{AB}$.
Możemy je również użyć na więcej niż 2 zmienne: $\overline{A} + \overline{B} + \overline{C} + \overline{D} = \overline{ABCD}$.

Wiemy też, że $A = \overline{\overline{A}}$

Przekształcamy wzór Y1:

$Y1 = ACD + AB = \overline{\overline{ACD}} + \overline{\overline{AB}} = \overline{\overline{ACD}\ \overline{AB}}$
Tworzy nam to układ 3 bramek NAND.

Przekształcamy wzór Y2:

$Y2 = A\overline{BC} + A\overline{BD} + \overline{A}BC + BCD = \overline{\overline{A\overline{BC}}} + \overline{\overline{A\overline{BD}}} + \overline{\overline{\overline{A}BC}} + \overline{\overline{BCD}} = \overline{\overline{A\overline{BC}}\ \overline{A\overline{BD}}\ \overline{\overline{A}BC}\ \overline{BCD}}$

Tutaj pojawia nam się problem z sygnałem przeciwnym np. $\overline{A}$ skoro możemy używać tylko bramek NAND.

Ale stworzenie bramki NOT z bramki NAND jest dosyć proste. Możemy wyprowadzić prosty wzór, że: $\overline{A} = \overline{AA}$. Znaczy to, żeby uzyskać sygnał przeciwny wystarczy podpiąć do bramki NAND ten sam sygnał 2 razy.

Przekształcamy wzór Y3:

$Y3 = A\overline{C}D + AC\overline{D} + B\overline{C} = \overline{\overline{A\overline{C}D}} + \overline{\overline{AC\overline{D}}} + \overline{\overline{B\overline{C}}} = \overline{\overline{A\overline{C}D}\ \overline{AC\overline{D}}\ \overline{B\overline{C}}}$

Jak widać wszędzie schemat przekształcenia jest podobny. Podwójnie zaprzeczamy w każdym elemencie alternatywy i łączymy zgodnie ze wzorem Morgana.

Przekształcamy wzór Y4:

$Y4 = \overline{A}B\overline{C}D + \overline{B}C + A\overline{B} + AC = \overline{\overline{\overline{A}B\overline{C}D}} + \overline{\overline{\overline{B}C}} + \overline{\overline{A\overline{B}}} + \overline{\overline{AC}} = \overline{\overline{\overline{A}B\overline{C}D}\ \overline{\overline{B}C}\ \overline{A\overline{B}}\ \overline{AC}}$

Przekształcamy wzór Y5:

$Y5 = AB\overline{C}D + \overline{AC}D + \overline{A}B\overline{D} + A\overline{B}\overline{D} + \overline{A}CD + AC\overline{D} = \overline{\overline{AB\overline{C}D}} + \overline{\overline{\overline{AC}D}} + \overline{\overline{\overline{A}B\overline{D}}} + \overline{\overline{A\overline{B}\overline{D}}} + \overline{\overline{\overline{A}CD}} + \overline{\overline{AC\overline{D}}} =$
$= \overline{\overline{AB\overline{C}D}\ \overline{\overline{AC}D}\ \overline{\overline{\overline{A}B\overline{D}}}\ \overline{\overline{A\overline{B}\overline{D}}}\ \overline{\overline{\overline{A}CD}}\ \overline{\overline{AC\overline{D}}}}$

Przekształcamy wzór Y6:

$Y6 = A + B + C + D = \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + \overline{\overline{D}} = \overline{\overline{A}\ \overline{B}\ \overline{C}\ \overline{D}}$

Gdzie pamiętamy, że negacja $\overline{A}$, to tak naprawdę $\overline{AA}$.

2.4 System bramek logicznych

Finalnym etapem jest zbudowanie systemów logicznych do przekształconych wzorów:

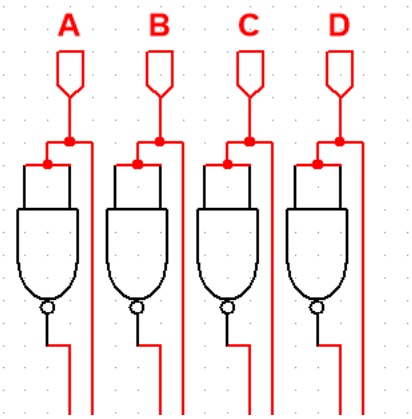
Oznaczenia:

- dla liczby 4 bitowej (sygnału wejściowego):
  - Bit odpowiadający za 2^3 -> A
  - Bit odpowiadający za 2^2 -> B
  - Bit odpowiadający za 2^1 -> C

  - Bit odpowiadający za 2^0 -> D

- dla liczby 6 bitowej (sygnału wyjściowego):
  - Bit odpowiadający za 2^5 -> Y1
  - Bit odpowiadający za 2^4 -> Y2
  - Bit odpowiadający za 2^3 -> Y3
  - Bit odpowiadający za 2^2 -> Y4
  - Bit odpowiadający za 2^1 -> Y5
  - Bit odpowiadający za 2^0 -> Y6

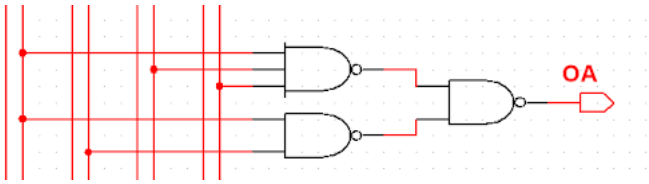
Do uproszczenia systemu bramek logicznych dziele sobie każdy z sygnałów na sygnał oryginalny i sygnał przeciwny do niego w ten sposób:



Rysunek 9: Sygnały wejściowe i sygnały do nich przeciwne

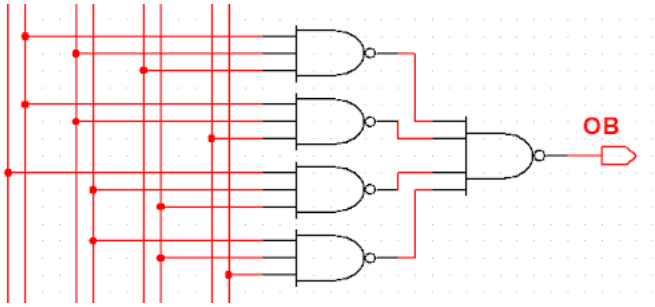
Budujemy zestawy bramek dla kolejnych funkcji:

Układ bramek dla $Y1 = \overline{\overline{ACD}\ \overline{AB}}$



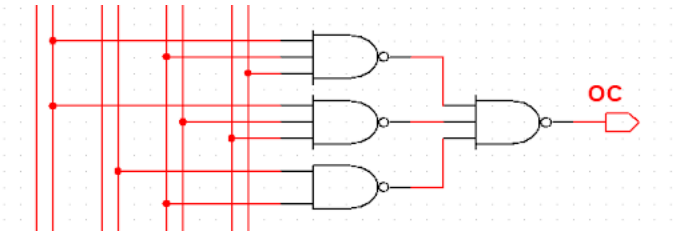
Rysunek 10: Układ bramek dla Y1

Układ bramek dla $Y2 = \overline{\overline{A\overline{BC}}\ \overline{A\overline{BD}}\ \overline{\overline{ABC}}\ \overline{BCD}}$



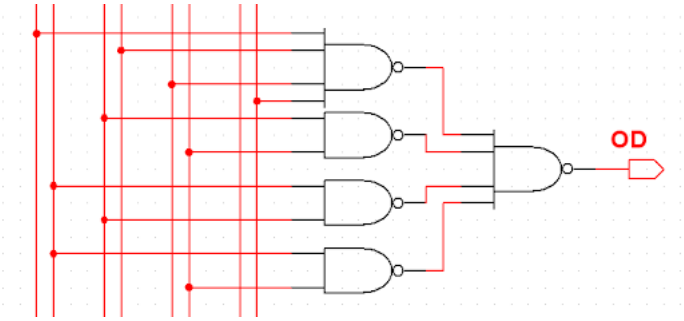
Rysunek 11: Układ bramek dla Y2

Układ bramek dla $Y3 = \overline{A\overline{C}DAC\overline{D}B\overline{C}}$



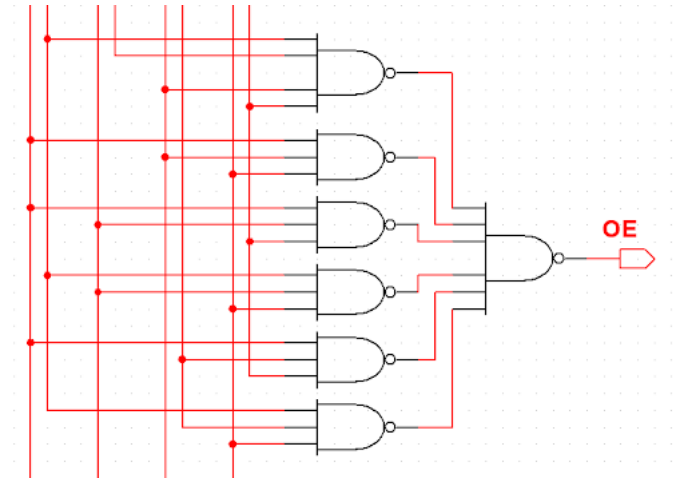
Rysunek 12: Układ bramek dla Y3

Układ bramek dla $Y4 = \overline{\overline{A}B\overline{C}D}\ \overline{\overline{B}\overline{C}}\ \overline{A\overline{B}}\ \overline{AC}$



Rysunek 13: Układ bramek dla Y4

Układ bramek dla $Y5 = \overline{\overline{AB\overline{C}D}}\ \overline{\overline{AC\overline{D}}}\ \overline{\overline{A\overline{B}D}}\ \overline{A\overline{B}\overline{D}}\ \overline{\overline{A}CD}\ \overline{AC\overline{D}}$



Rysunek 14: Układ bramek dla Y5

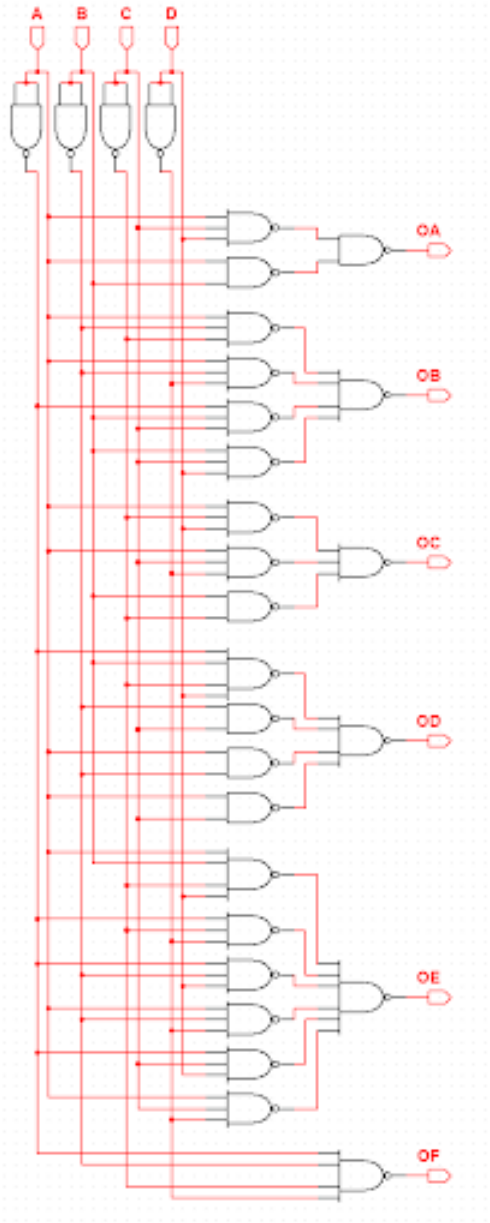
Układ bramek dla $Y6 = \overline{\overline{A}\ \overline{B}\ \overline{C}\ \overline{D}}$

Rysunek 15: Układ bramek dla Y6

2.5 Finalny układ logiczny

Łącząc wszystkie powyższy podukłady otrzymujemy jeden układ logiczny, które jest naszym transkoderem 4-bitowej liczby naturalnej na 6-bitową liczbę pierwszą.

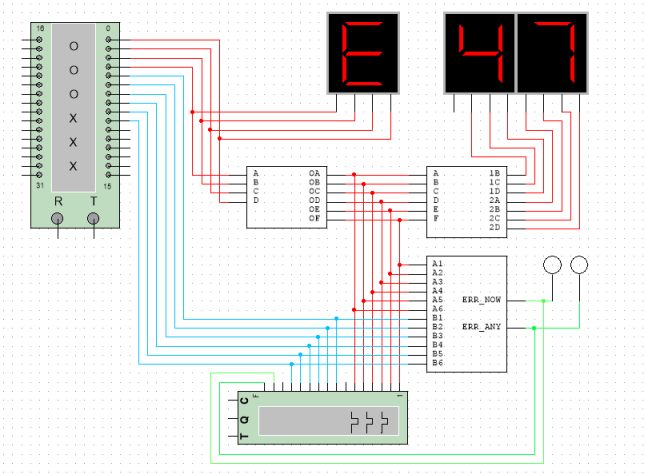
Tak prezentuje się nasz układ w całości:



Rysunek 16: Układ bramek w całości

3 Układ testowy

Aby przetestować utworzony układ, zaprojektowaliśmy system testowania otrzymanych wyników. Wykorzystaliśmy do tego generator słów i analizator stanów logicznych oraz prosty podukład zbudowany za pomocą bramek XOR do porównywania oczekiwanych wyników z rzeczywistym.



Rysunek 17: Schemat całego układu wraz z układem testującym

Układ zawiera:

1. Generator słów (XWG1)
2. Transkoder 4-bitowej liczby naturalnej na 6-bitową liczbę pierwszą
3. Transkoder do wyświetlania liczby dwucyfrowej
4. Wyświetlacze siedmiosegmentowe (3 sztuki)
5. Analizator stanów logicznych
6. Tester równości sygnałów z diodami

3.1 Generator słów

W generatorze słów utworzyliśmy sekwencję szesnastu 10-bitowych ciągów.

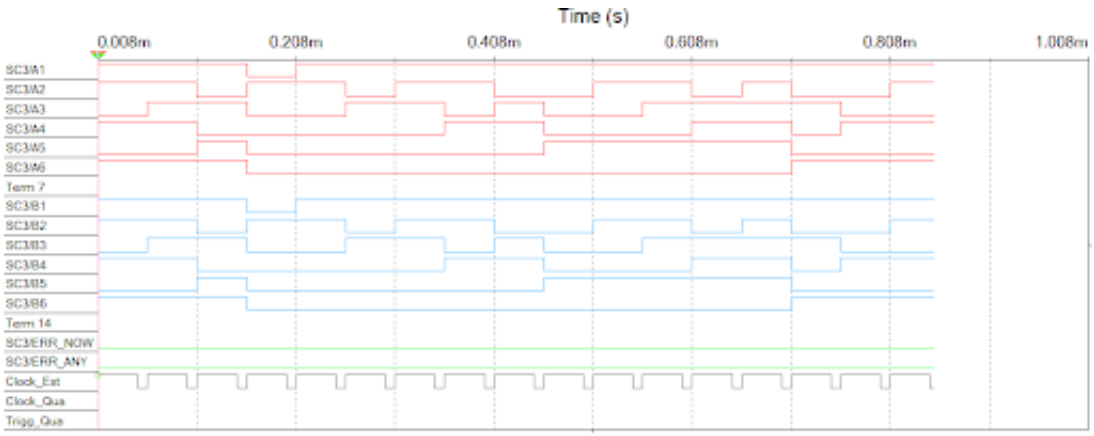
```
0000000000000000000000000100000
0000000000000000000000000110001
0000000000000000000000001010010
0000000000000000000000001110011
0000000000000000000000010110100
0000000000000000000000011010101
0000000000000000000000100010110
0000000000000000000000100110111
0000000000000000000000101111000
0000000000000000000000111011001
0000000000000000000000111111010
0000000000000000000001001011011
0000000000000000000001010011100
0000000000000000000001010111101
0000000000000000000001011111110
0000000000000000000001101011111
```

Rysunek 18: Ciągi bitowe w generatorze słów

Pierwsze 4 bity, licząc od prawej strony, reprezentują sekwencję liczb od 0 do 15. Natomiast na pozostałych 6 bitach zapisane są pierwsze 16 liczb pierwszych. Wyjścia odpowiedzialne za dane wejściowe odpowiednio podłączyliśmy do układu.

3.2 Analizator stanów logicznych

Do sprawdzania poprawności układu obserwowaliśmy stany, które analizator otrzymywał. Częstotliwość zegara w analizatorze stanów logicznych jest dwukrotnie wyższa niż w generatorze słów. Czerwonym kolorem oznaczony jest sygnał otrzymany z transkodera. Kolorem niebieskim oznaczone są bity liczby docelowej. Kolorem zielonym prezentowane są kolejno czy błąd występuje teraz, czy jakikolwiek błąd wystąpił. Test zakończył się pomyślnie dla każdej kombinacji wejściowej.

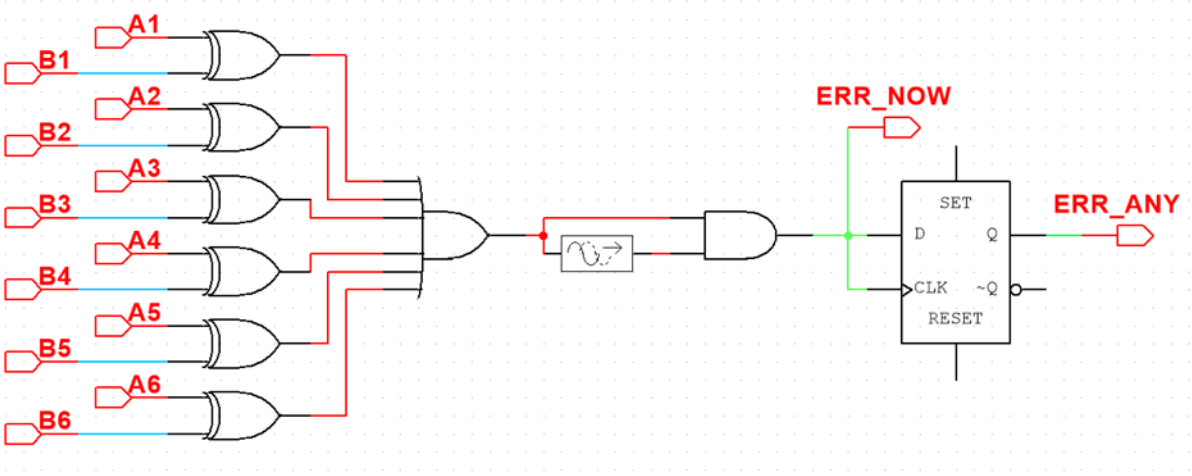


Rysunek 19: Test poprawności

3.3 Podukład do porównywania sygnałów

Do porównania sygnałów użyliśmy bramki XOR, które sprawdzają każdy bit liczb osobno. Na koniec układ został podłączony do przerzutnika typu D, który obsługuje zapalanie diod sygnalizujących o wystąpieniu błędu w danym momencie (ERR_NOW) oraz o zaistnieniu jakiegokolwiek błędu zamiany liczb (ERR_ANY).

Układ przedstawiony na środku porównuje sygnał z tym samym sygnałem opóźnionym o pół cyklu zegara generatora. Zapobiega to występowaniu błędu wynikającego z opóźnienia bramek logicznych naszego transkodera. Bez niego układ wyłapuje błąd, który nie występuje.



Rysunek 20: Podukład do porównywania sygnałów

D	Q(Current)	Q(Next)
0	0	0
0	1	0
1	0	1
1	1	1

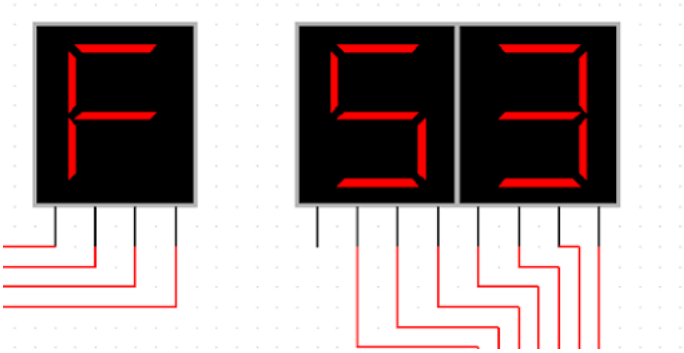
Tabela 2: Tabela przejść dla przeżutnika typu D

4 Wyświetlacze siedmiosegmentowe

W układzie występują 3 wyświetlacze:

- Pojedynczy wyświetlacz pokazuje liczbę w systemie szesnastkowym, który wyświetla cyfrę reprezentującą sygnał wejściowy. Jest on podłączony 4 przewodami, w efekcie ma możliwość wyświetlania wszystkich cyfr od 0 do F (czyli 15).
- Pozostałe dwa wyświetlacze pokazują liczbę dwucyfrową w systemie dziesiętnym, który wyświetla 2 cyfry reprezentującą sygnał wyjściowy.
  - Wyświetlacz odpowiadający za wyświetlanie cyfr jedności jest podłączony 4 przewodami, w efekcie ma możliwość wyświetlania wszystkich cyfr od 0 do F (czyli 15), jednakże nigdy nie otrzyma sygnału na wyświetlanie cyfr wykraczających poza system dziesiętny.
  - Z kolei wyświetlacz odpowiadający za wyświetlanie cyfr jedności jest podłączony 3 przewodami, w efekcie ma możliwość wyświetlania wszystkich cyfr od 0 do 7. Nie ma potrzeby, by wyświetlał on większe cyfr, ponieważ największa liczba, jaką ten program powinien wyświetlać to 53, czyli żadna z cyfr dziesiętnych do wyświetlenia nie będzie większa niż 5.

Każdy z wyświetlaczy jest narzędziem typu “hex_display”. Sygnał wejściowy przed dotarciem do wyświetlaczy prezentujących liczbę pierwszą jest nieco opóźniony w stosunku do sygnału wejściowego, co wynika z konieczności przeprowadzenia sygnału przez bramki logiczne.



Rysunek 21: Wyświetlacze

Sygnał reprezentujący liczbę pierwszej jest reprezentowany w układzie w postaci 6 bitów, żeby możliwe było wyświetlenie takiej liczby na wyświetlaczu w postaci 2 cyfr w systemie dziesiętnym konieczna jest konwersja sygnału. Dlatego też konieczne było stworzenie układu, który:

- Na wejście przyjmuje 6 przewodów, reprezentujących liczbę 6-bitową
- Na wyjście wyprowadza 7 przewodów, odpowiadających za 2-cyfrową liczbę dziesiętną, z których 4 przewody odpowiadają za 4 bity liczby jedności, a pozostałe 3 odpowiadają za 3 bity liczby dziesiętnej

Własnoręczne stworzenie 7 tablic Karnaugh na 6 zmiennych i znalezienie odpowiadającym im 7 funkcji byłoby zbyt czasochłonne dlatego wykorzystaliśmy narzędzie na stronie: http://www.32x8.com/var6.html.

Natomiast do wygenerowania układu bramek logicznych ze wzorów wykorzystaliśmy generowanie układu z bramek NAND z narzędzia "Logic converter" w Multisimie.
Oznaczenia dla sygnałów wchodzących każdy reprezentujący poszczególny bit:

- Bit odpowiadający za 2^5 -> A
- Bit odpowiadający za 2^4 -> B
- Bit odpowiadający za 2^3 -> C
- Bit odpowiadający za 2^2 -> D
- Bit odpowiadający za 2^1 -> E
- Bit odpowiadający za 2^0 -> F

Poniżej przedstawienie kolejnych układów bramek logicznych dla każdego z poszczególnych bitów sygnału wychodzącego wraz z ich zapisem logicznym (wszystkie bramki są typu NAND, ale zapis logiczny jest zapisany inaczej, ponieważ zapisanie go tylko przy pomocy operatorów NAND wymagałoby zapisu za bardzo skomplikowanego i nieczytelnego):

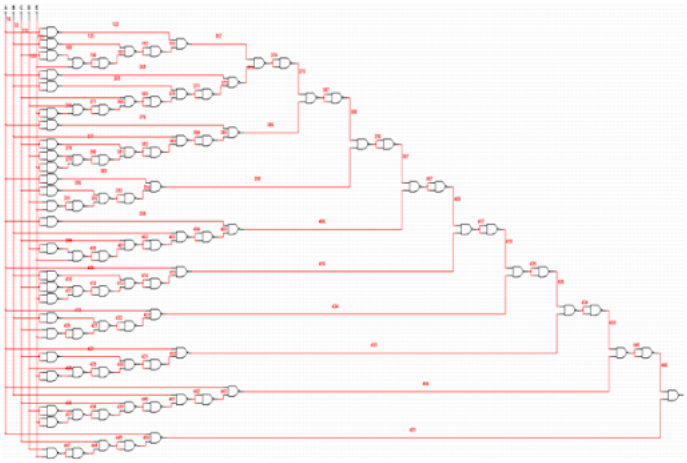
1. **Cyfra jedności:**
   - **Bit odpowiadający za 2^0:**

     Zapis logiczny: $Y = F$

Rysunek 22: Cyfra jedności - bit odpowiadający za 2^0

   - **Bit odpowiadający za 2^1:**

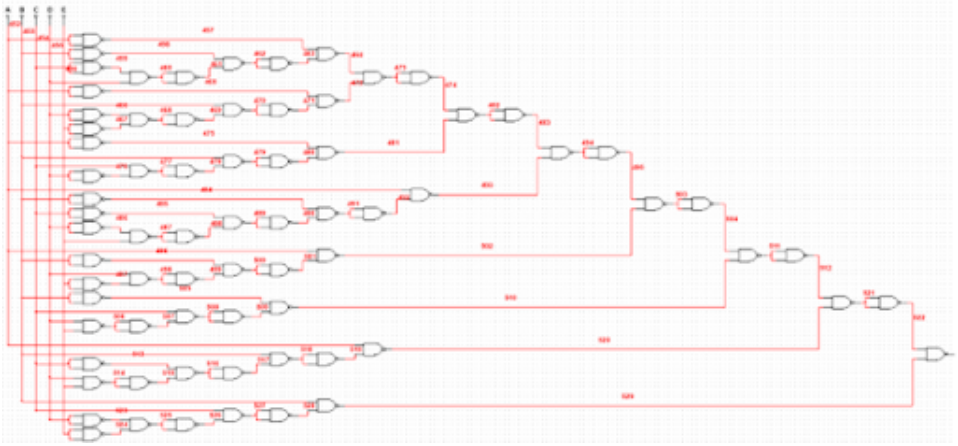
     Zapis logiczny: $Y = \overline{A}\ \overline{B}\ \overline{C}\ E + \overline{A}\ \overline{C}\ D\ E + A\ \overline{B}\ \overline{C}\ \overline{E} + A\ \overline{C}\ D\ \overline{E} + A\ \overline{B}\ C\ E + A\ C\ D\ E + \overline{A}\ \overline{B}\ C\ D\ \overline{E} + \overline{A}\ B\ \overline{C}\ \overline{D}\ \overline{E} + \overline{A}\ B\ C\ \overline{D}\ E + A\ B\ C\ \overline{D}\ \overline{E}$



Rysunek 23: Cyfra jedności - bit odpowiadający za 2^1

- **Bit odpowiadający za 2^2:**

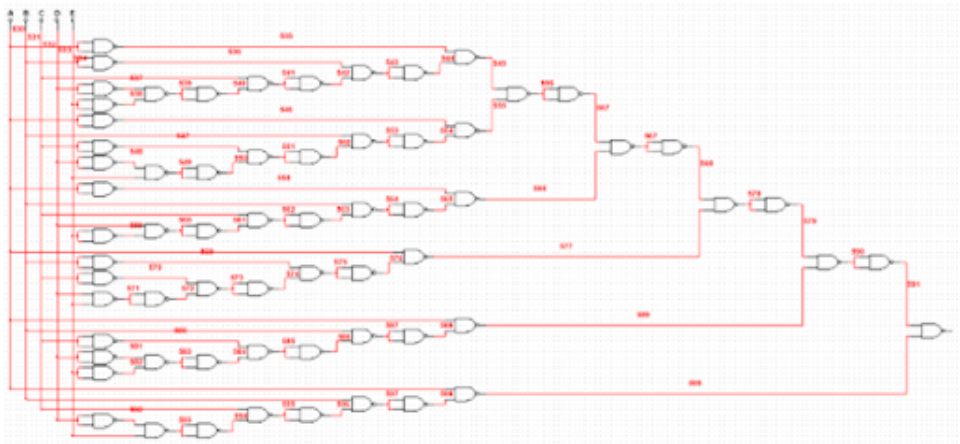
  Zapis logiczny: $Y = \overline{A}\ \overline{B}\ \overline{C}\ D + \overline{B}\ C\ D\ E + \overline{A}\ B\ \overline{D}\ \overline{E} + \overline{A}\ BC\overline{D} + B\ C\ \overline{D}\ \overline{E} + A\ \overline{B}\ D\ \overline{E} + A\ \overline{B}\ \overline{C}\ \overline{D}\ E + A\ B\ \overline{C}\ D\ E$



Rysunek 24: Cyfra jedności - bit odpowiadający za 2^2

- **Bit odpowiadający za 2^3:**

  Zapis logiczny: $Y = \overline{A}\ \overline{B}\ C\ \overline{D}\ \overline{E} + \overline{A}\ B\ \overline{C}\ \overline{D}\ E + \overline{A}\ B\ C\ D\ \overline{E} + A\ \overline{B}\ \overline{C}\ D\ E + A\ B\ \overline{C}\ \overline{D}\ \overline{E} + A\ B\ C\ D$

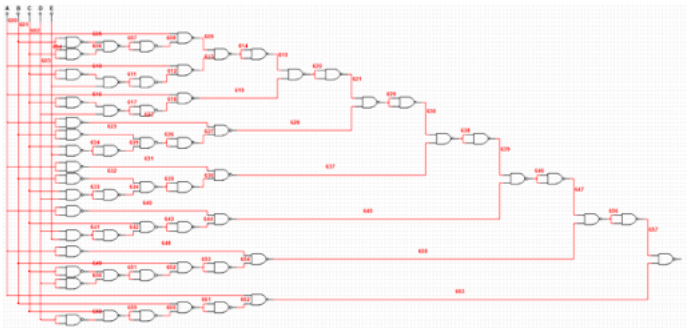


Rysunek 25: Cyfra jedności - bit odpowiadający za 2^3

2. **Cyfra dziesiętna:**

- **Bit odpowiadający za 2^0:**

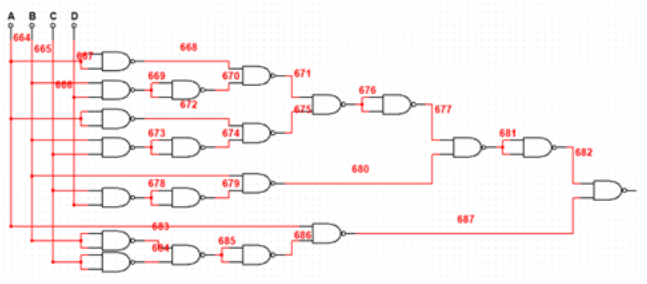
  Zapis logiczny: $Y = A\ \overline{B}\ \overline{C} + A\ \overline{C}\ E + A\ \overline{C}\ D + \overline{A}\ \overline{B}\ C\ E + \overline{A}\ \overline{B}\ C\ D + \overline{A}\ C\ D\ E + \overline{A}\ B\ \overline{C}\ \overline{D} + A\ B\ C\ \overline{D}$



Rysunek 26: Cyfra dziesiętna - bit odpowiadający za 2^0

- **Bit odpowiadający za 2^1:**

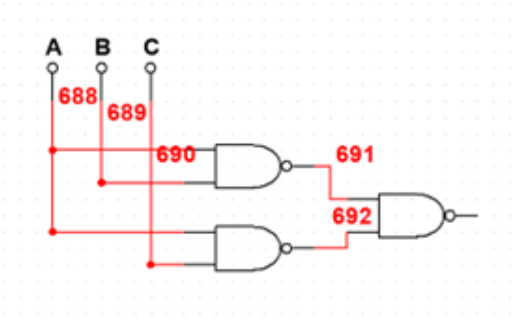
  Zapis logiczny: $Y = \overline{A}\ B\ D + \overline{A}\ B\ C + B\ C\ D + A\overline{B}\ \overline{C}$



Rysunek 27: Cyfra dziesiętna - bit odpowiadający za 2^1

- **Bit odpowiadający za 2^2:**

  Zapis logiczny: $Y = A\ C + A\ B$



Rysunek 28: Cyfra dziesiętna - bit odpowiadający za 2^2

5 Wnioski

Nasz układ został zaprojektowany prawidłowo, ponieważ przeszedł wszystkie testy zawierające każdą możliwość wykorzystania układu.

Co można by było zrobić lepiej:

- Przewody w transkoderach mogłyby zostać pomalowane na różne kolory, aby móc łatwiej zobaczyć co jest podłączone do czego
- Transkoder przekształcający 6-bitowy sygnał binarny mógłby być wykonany troszkę lepiej, bo został generowany przez urządzenia "logic converter"(a one nie są najefektywniejsze), ponieważ nie wykorzystuje on bramek więcej niż dwuwejściowych

Co można by było zrobić gorzej:

- Układ działa, a mógłby nie działać
- Układ, poza transkoderem do wyświetlacza jest dosyć kompaktowy
- Układ ładnie się prezentuje

Przykładowe zastosowania transkoderów, w ogóle:

- Transkodery są wykorzystywane do konwersji danych z jednego formatu do drugiego (np. z MPEG-2 na H.264).
- Kompresja danych
- Podobny jak wykorzystany w zadaniu transkoder liczby bitowej do wyświetlaczy siedmiosegmentowych

Przykładowe zastosowania naszego transkodera 4-bit na 6-bitowe liczby pierwsze:

- Kryptografia - układ mógłby być wykorzystany do generowania lub przetwarzania kluczy kryptograficznych opartych na liczbach pierwszych.
- Losowe generowanie liczby pierwszej.
- Nie ma żadnych konkretnych zastosowań.