Technika Cyfrowa

Sprawozdanie 1.

Bratek Natalia
Mrozowski Jakub
Stankiewicz Urszula
Walendzik Marcin

1. Polecenie

Bazując wyłącznie na bramkach NAND, zaprojektować, zbudować i przetestować układ kombinacyjny realizujący transkoder czterobitowej liczby naturalnej (wraz z zerem) na sześcio-bitową 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:

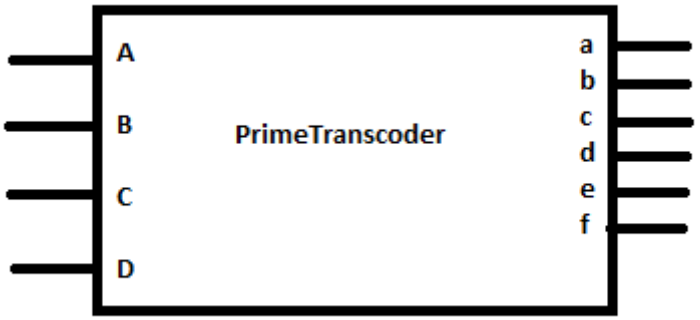
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53

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

Do minimalizacji potrzebnych funkcji należy wykorzystać tablice Karnaugh.

2. Opis rozwiązania

Układ opisany w treści zadania przyjmuje liczbę czterobitową i konwertuje ją na sześciobitową. Układ ten będzie więc miał cztery wejścia oraz sześć wyjść. Schematycznie można przedstawić go w następujący sposób:



Rys.1 Poglądowy schemat wejść i wyjść układu transkodera

Na rysunku 1 wejście "A" reprezentuje bit najstarszy, natomiast wejście "D" bit najmłodszy. Analogicznie wyjście "a" reprezentuje bit najstarszy, a wyjście "f" - najmłodszy.

W celu rozwiązania ćwiczenia stworzymy początkowo tablicę prawdy, która będzie przedstawiała sposób konwersji wejść transkodera na każde z jego wyjść. Posłużymy się kodami Grey'a, gdyż ułatwi nam to następny krok - tworzenie tablic Karnaugh. Dzięki ich zastosowaniu będziemy w stanie zminimalizować użyte przez nas funkcje logiczne konieczne do działania transkodera. Następnie przekształcimy otrzymane w ten sposób funkcje logiczne w taki sposób, by możliwa była ich implementacja na bramkach NAND. Na koniec zaimplementujemy otrzymane funkcje w programie Multisim oraz poddamy otrzymany w ten sposób układ testom: podepniemy go do generatora słów, analizatora stanów logicznych, komparatora oraz przerzutnika typu RS w celu detekcji potencjalnych błędów.

Dodatkowy układ, który zdecydowaliśmy się zaimplementować w celu wygodniejszego wyświetlania liczb w systemie dziesiętnym (pomocne w testowaniu układu - należy podpiąć wyświetlacze do wejść oraz wyjść), wygląda poglądowo w sposób następujący:

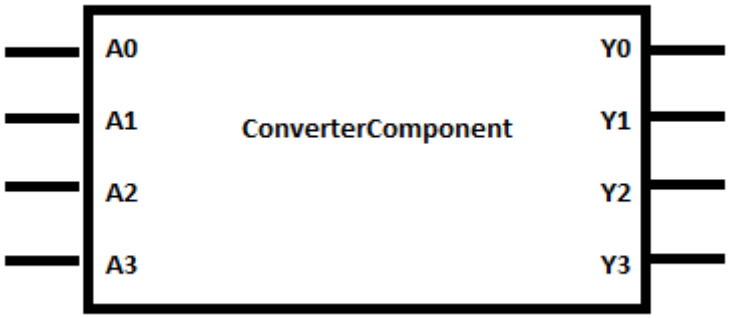
Rys.2 Poglądowy schemat układu konwertującego liczby binarne na dwucyfrowy kod BCD

Jest to układ, który pozwoli przekonwertować 6-bitową liczbę binarną na jej dwucyfrowy odpowiednik w kodzie BCD. Liczby zakodowane kodem BCD (ang. binary-coded decimal) można łatwo wyświetlić w systemie dziesiętnym, używając wyświetlaczy siedmiosegmentowych. Istnieją gotowe układy konwertujące liczby binarne na kod BCD, jednak nasz układ przyjmuje i zwraca liczby, które w systemie dziesiętnym są dwucyfrowe, a Multisim nie oferuje tego typu transkoderów.

Przedstawiony przez nas układ konwertujący zawiera wejścia B0, B1,..., B5, gdzie B0 reprezentuje bit najmłodszy, natomiast B5 - najstarszy. Wyjścia oznaczone są podobnie: P0 oznacza bit najmłodszy, a P5 najstarszy.

W celu wyświetlenia danej na wejściu liczby binarnej należy podpiąć do dwóch wyświetlaczy siedmiosegmentowych: wyjścia P0, P1, P2, P3 do wejść wyświetlacza odpowiadającego cyfrze dziesiętnej, a pozostałe do wyświetlacza cyfry jedności.

Warto też wspomnieć, że układ konwertujący na kod BCD stworzymy korzystając z trzech podukładów następującego typu:



Rys. 3 Poglądowy schemat jednego z komponentów stworzonego konwertera

3. Tabela prawdy

Tworzymy tabelę prawdy, wprowadzając do niej kolejne wartości wejść oraz oczekiwane dla nich wartości wyjść. Z boku tabeli dla przejrzystości umieściliśmy liczby wyrażone w sposób dziesiętny dla każdego zestawu wejść oraz wyjść. Korzystamy z kodów Grey'a, by później móc w prostszy sposób utworzyć tablice Karnaugh.

Wejście dziesiętnie	A	B	C	D	a	b	c	d	e	f	Wyjście dziesiętnie
0	0	0	0	0	0	0	0	0	1	0	2
1	0	0	0	1	0	0	0	0	1	1	3
3	0	0	1	1	0	0	0	1	1	1	7
2	0	0	1	0	0	0	0	1	0	1	5
6	0	1	1	0	0	1	0	0	0	1	17
7	0	1	1	1	0	1	0	0	1	1	19
5	0	1	0	1	0	0	1	1	0	1	13
4	0	1	0	0	0	0	1	0	1	1	11
12	1	1	0	0	1	0	1	0	1	1	43
13	1	1	0	1	1	0	1	0	0	1	41
15	1	1	1	1	1	1	0	1	0	1	53
14	1	1	1	0	1	0	1	1	1	1	47
10	1	0	1	0	0	1	1	1	1	1	31
11	1	0	1	1	1	0	0	1	0	1	37
9	1	0	0	1	0	1	1	1	0	1	29
8	1	0	0	0	0	1	0	1	1	1	23

Tabela 1. Tabela prawdy

4. Tablice Karnaugh

Dla każdego wyjścia a, b, c, d, e, f tworzymy tablice Karnaugh, przenosząc wartości wyjść z tabeli prawdy w odpowiednie komórki tabeli. Następnie łączymy w prostokąty komórki zawierające jedynki w taki sposób, by w każdym prostokącie była liczba jedynek będąca potęgą dwójki. Można też łączyć ze sobą skrajne górne komórki tablicy ze skrajnymi dolnymi oraz skrajne lewe ze skrajnymi prawymi (można je uznać za sąsiednie). W przedstawionych oznaczeniach każdy prostokąt ma inny kolor - jeśli kolor się powtarza, znaczy to, że łączymy dwa skrajne prostokąty ze sobą. Następnie znajdujemy wspólne wartości wejść A, B, C, D dla elementów danego prostokąta i na tej podstawie tworzymy funkcje logiczne. Poniżej prezentujemy tablice Karnaugh dla każdego wyjścia wraz z funkcją logiczną pozwalającą je poprawnie uzyskać. Każdą funkcję przekształcamy do postaci umożliwiającej zastosowanie bramek NAND.

a) Wyjście “a”

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	0

Tabela 2. Tablica Karnaugh: wyjście “a”

Funkcje logiczna:

$a = A * B + A * C * D$

Przekształcenie, by możliwe było zastosowanie bramek NAND:

$a = A * B + A * C * D = \overline{\overline{A * B + A * C * D}} = \overline{\overline{A * B} * \overline{A * C * D}}$

b) Wyjście “b”

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	1	0
10	1	1	0	1

Tabela 3. Tablica Karnaugh: wyjście “b”

Funkcje logiczna:

$$b = A * \overline{B} * \overline{C} + A * \overline{B} * \overline{D} + B * C * D + \overline{A} * B * C$$

Przekształcenie, by możliwe było zastosowanie bramek NAND:

$$b = A * \overline{B} * \overline{C} + A * \overline{B} * \overline{D} + B * C * D + \overline{A} * B * C =$$

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

$$= \overline{\overline{A * \overline{B} * \overline{C}} * \overline{A * \overline{B} * \overline{D}} * \overline{B * C * D} * \overline{\overline{A} * B * C}}$$

c) Wyjście “c”

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	0	1
10	0	1	0	1

Tabela 4. Tablica Karnaugh: wyjście “c”

Funkcja logiczna:

$$c = A * \overline{C} * D + B * \overline{C} + A * C * \overline{D} = \overline{\overline{A * \overline{C} * D + B * \overline{C} + A * C * \overline{D}}} =$$

$$= \overline{\overline{A * \overline{C} * D} * \overline{B * \overline{C}} * \overline{A * C * \overline{D}}}$$

d) Wyjście “d”

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	0	0	1	1
10	1	1	1	1

Tabela 5. Tablica Karnaugh: wyjście “d”

Funkcja logiczna:

$$d = A * \overline{B} + \overline{A} * B * \overline{C} * D + A * C + \overline{B} * C = \overline{\overline{A * \overline{B} + \overline{A} * B * \overline{C} * D + A * C + \overline{B} * C}} =$$

$$= \overline{\overline{A * \overline{B}} * \overline{\overline{A} * B * \overline{C} * D} * \overline{A * C} * \overline{\overline{B} * C}}$$

e) Wyjście “e”

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	0	1	0
11	1	0	0	1
10	1	0	0	1

Tabela 6. Tablica Karnaugh: wyjście “e”

Funkcja logiczna:

$$e = \overline{C} * \overline{D} + A * \overline{D} + \overline{A} * \overline{B} * D + \overline{A} * C * D =$$

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

$$= \overline{\overline{\overline{C} * \overline{D}} * \overline{A * \overline{D}} * \overline{\overline{A} * \overline{B} * D} * \overline{\overline{A} * C * D}}$$

f) Wyjście “f”

AB \ CD	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

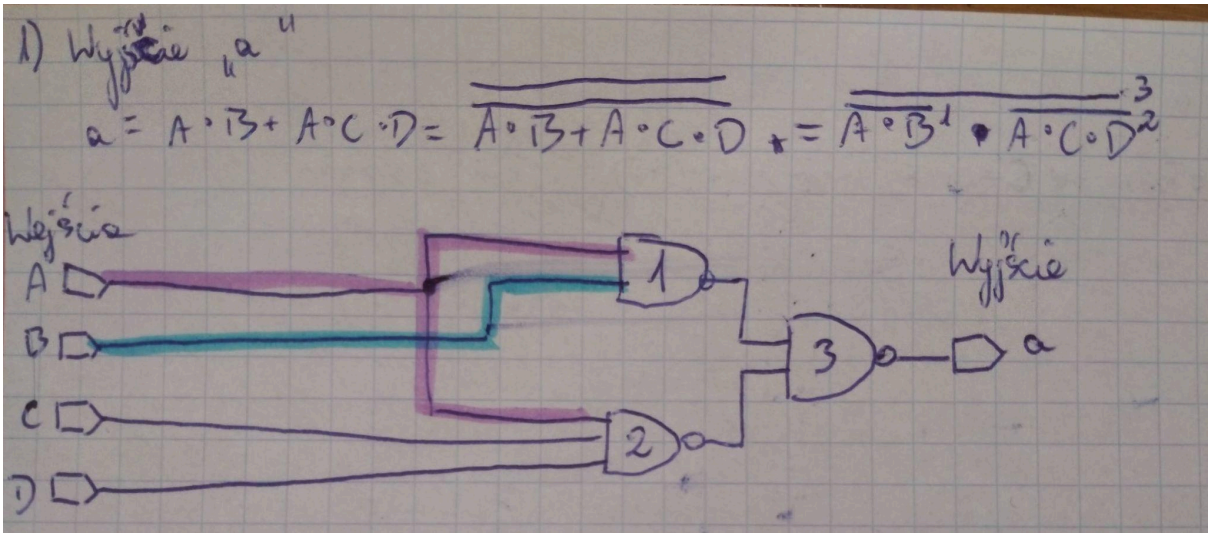
Tabela 7. Tablica Karnaugh: wyjście “f”

Funkcja logiczna:

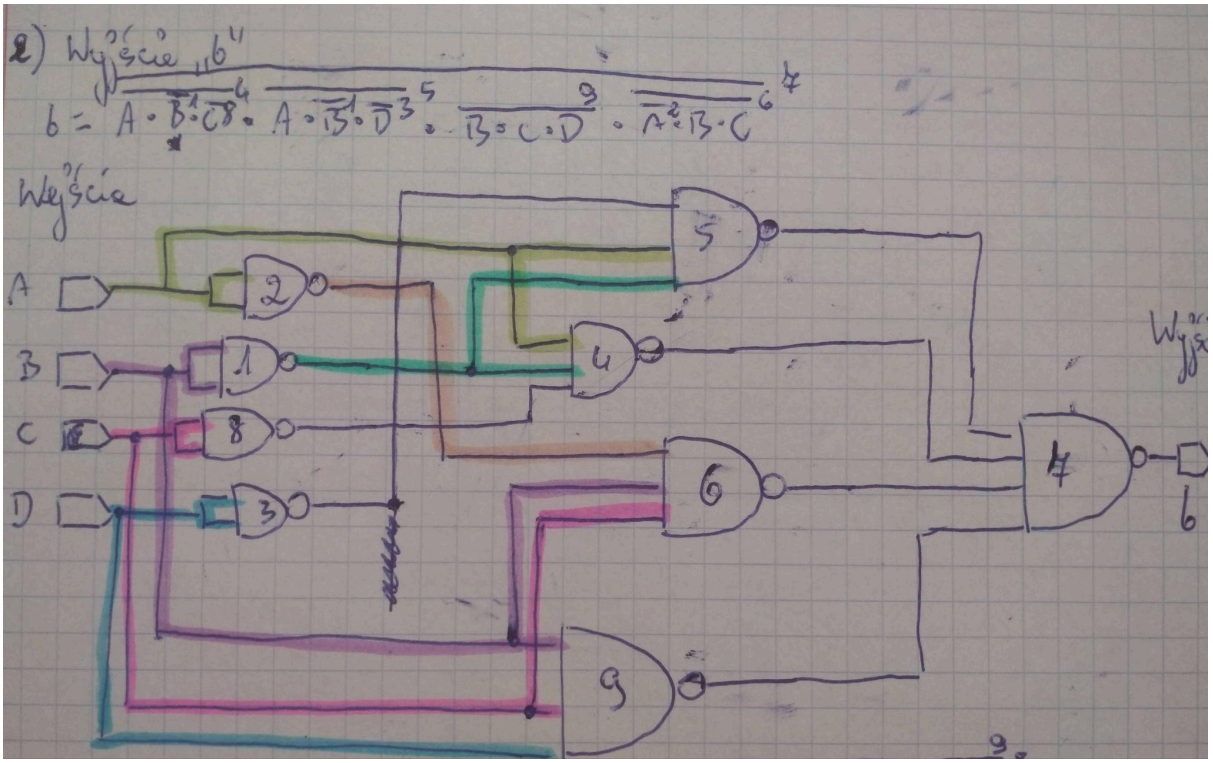
$$f = A + B + C + D = \overline{\overline{A + B + C + D}} = \overline{\overline{A} * \overline{B} * \overline{C} * \overline{D}}$$

5. Schemat układu

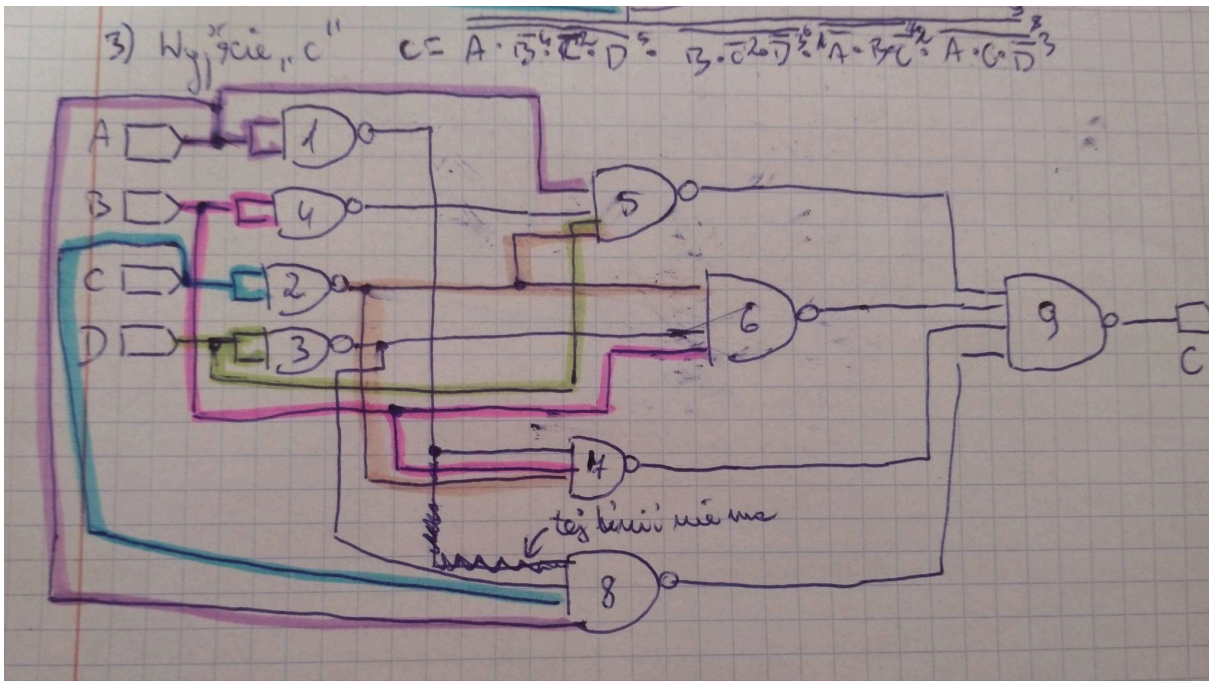
Poniżej przedstawiamy schematy układu dla poszczególnych wejść (podobnie jak w programie Multisim, kropka oznacza połączenie przewodów, a jej brak - przecięcie. Dla ułatwienia w trzech pierwszych układach zaznaczyliśmy przewody kolorami):



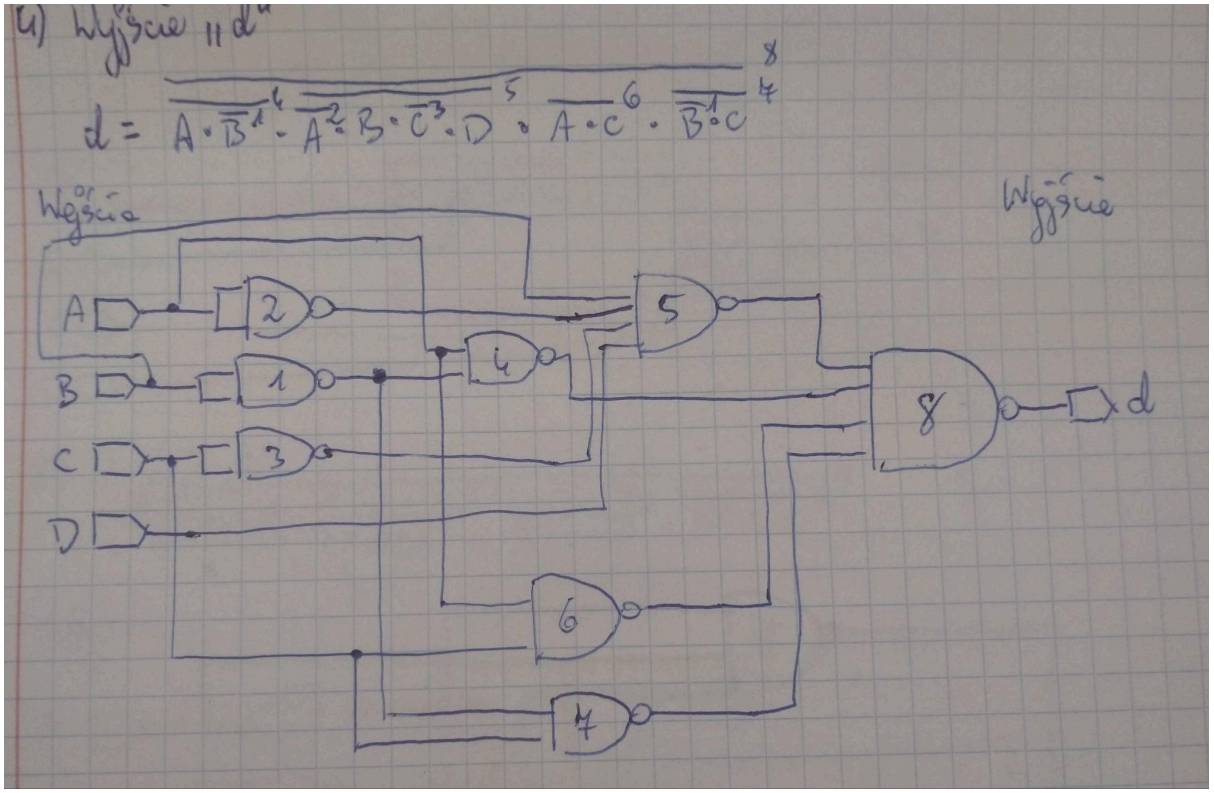
Rys. 4 Schemat układu dla wyjścia “a”



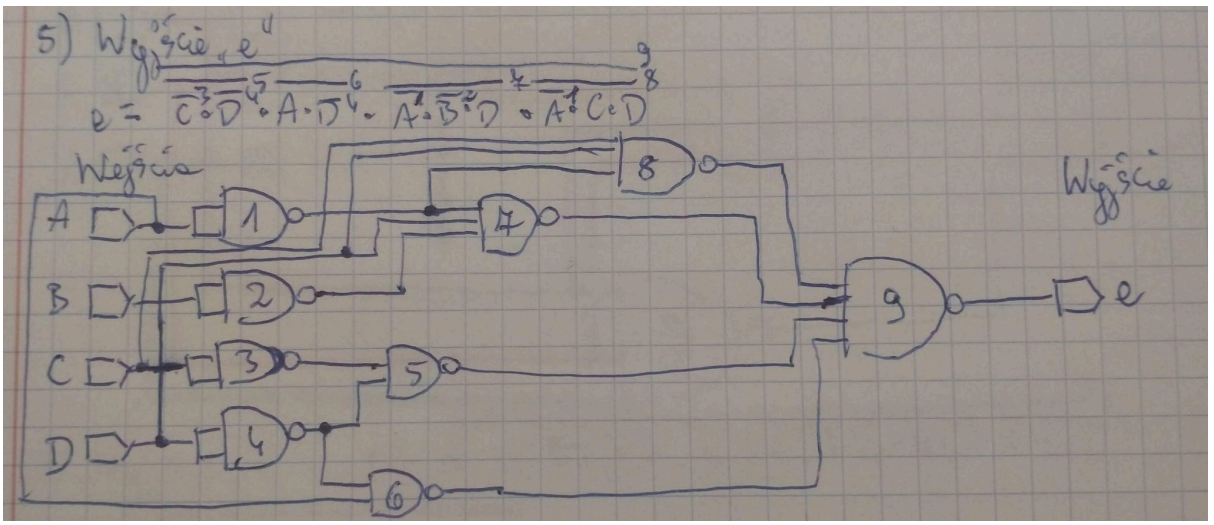
Rys. 5 Schemat układu dla wyjścia “b”



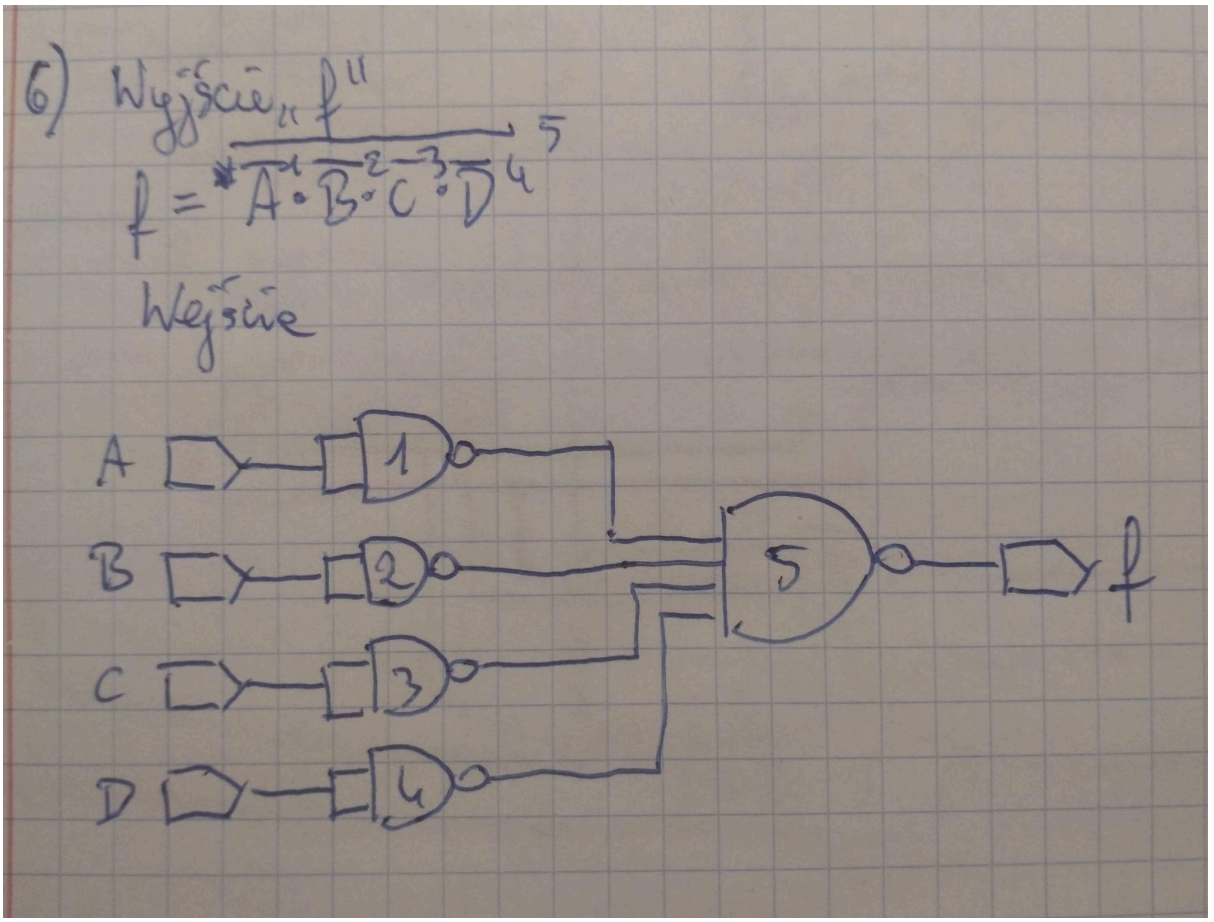
Rys. 6 Schemat układu dla wyjścia “c”



Rys. 7 Schemat układu dla wyjścia “d”



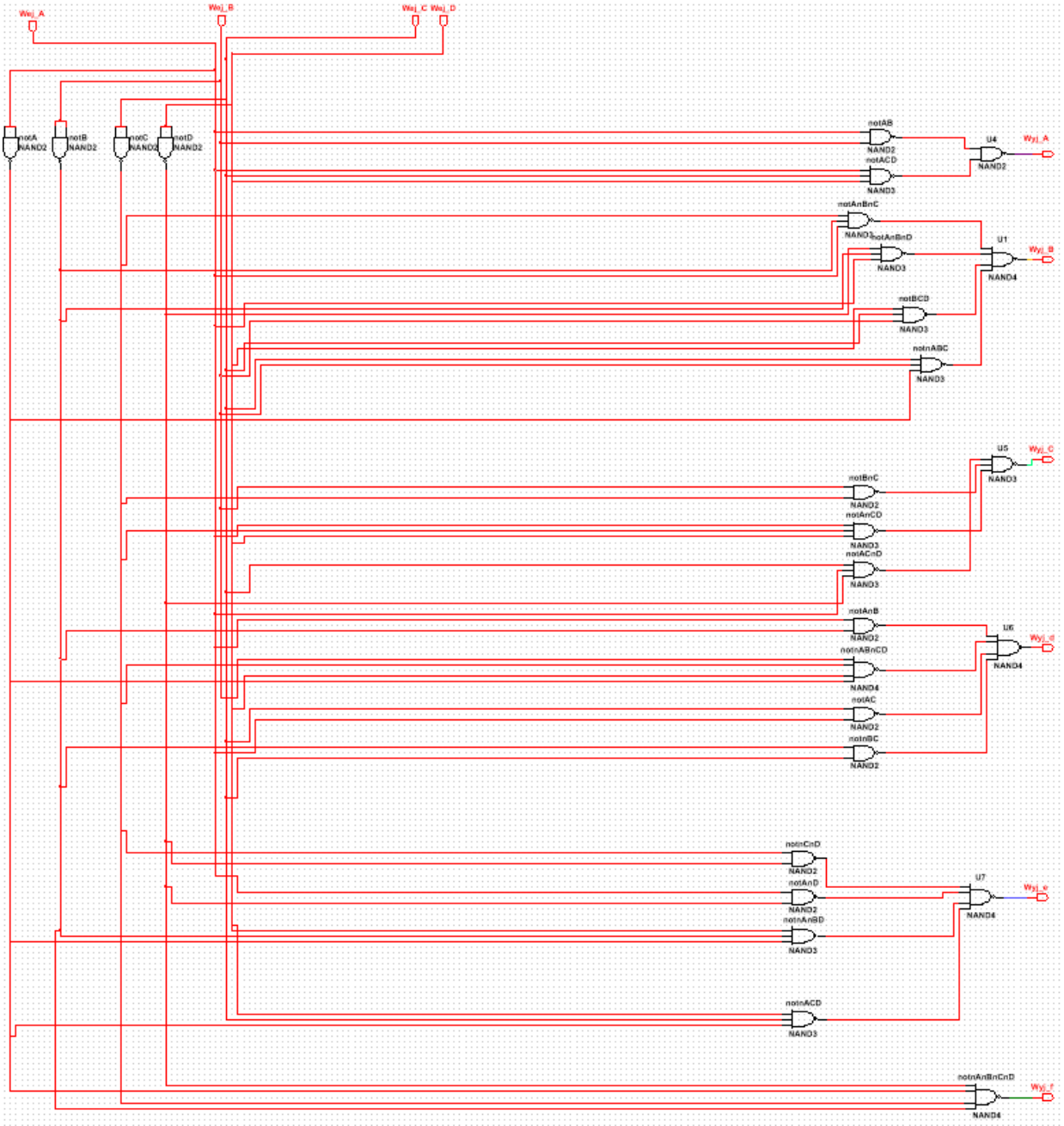
Rys. 8 Schemat układu dla wyjścia “e”



Rys. 9 Schemat układu dla wyjścia “f”

6. Implementacja w programie Multisim

Następnie zaimplementowaliśmy układ transkodujący w programie Multisim. Układ zawiera wyłącznie bramki NAND:

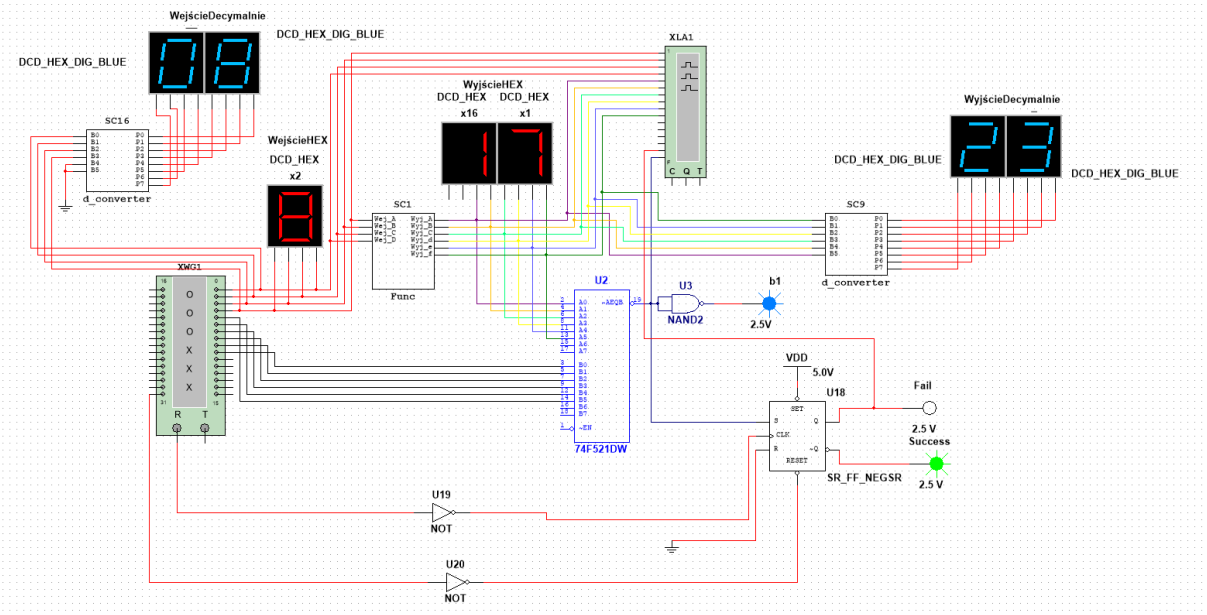


Rys. 10 Implementacja układu transkodującego w programie Multisim

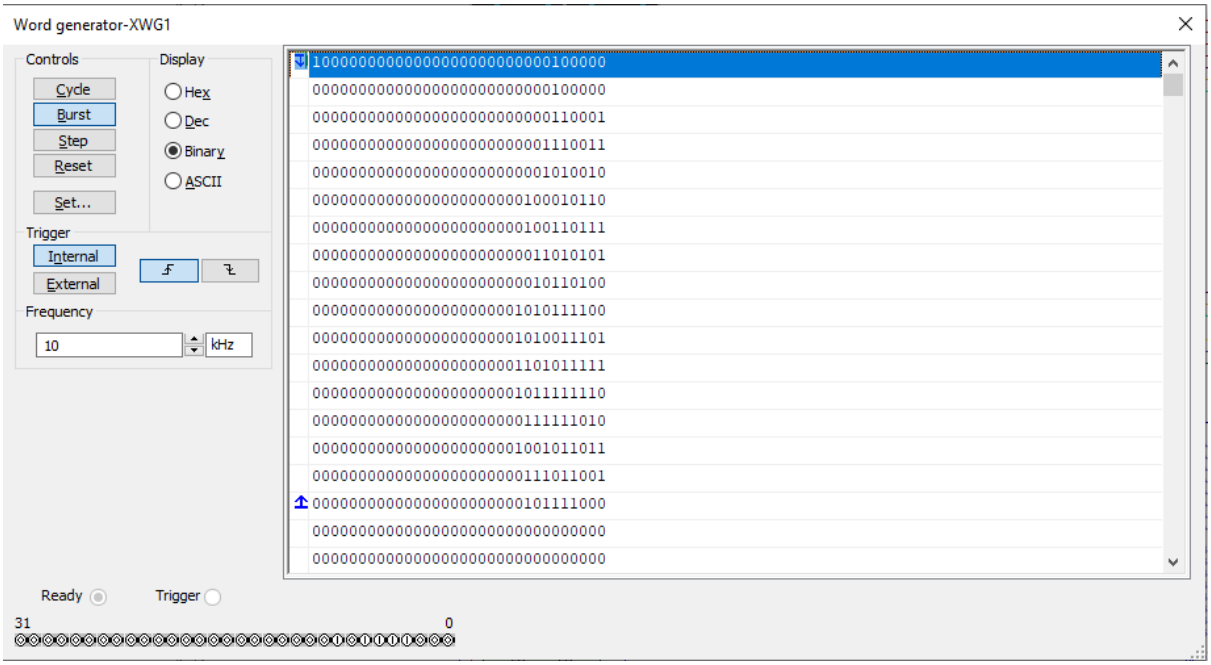
7. Układ testujący

W celu przetestowania transkodera stworzyliśmy układ testujący.
Układ zawiera:

- Generator słów, w którym na czterech najmłodszych bitach zakodowane są wszystkie możliwe wejścia (0-15), a na kolejnych sześciu najmłodszych poprawne wyjścia. Dodatkowa pierwsza wartość zakodowana w generatorze ma najstarszy bit ustawiony na “1” logiczną, która podpięta jest do wejścia “RESET” przerzutnika. Umożliwi to resetowanie stanu przerzutnika na początku każdego cyklu testów.
- Analizator stanów logicznych, do którego podpinamy wyjścia odpowiadające czterem najmłodszym bitom z generatora słów oraz sześć wyjść transkodera, wyjście z komparatora oraz wyjście główne (Q) z przerzutnika. Pozwoli nam to lepiej śledzić poprawność wyników.
- Komparator - podpinamy do niego wyjścia z układu oraz poprawne wyjścia zakodowane w generatorze słów. Na wyjściu podpinamy przewód do obu wejść bramki NAND, by uzyskać sygnał zaprzeczony. Do wyjścia bramki podpinamy diodę, która będzie świecić jeśli dane wyjście transkodera jest poprawne.
- Wyświetlacze segmentowe czerwone - wyświetlają wejście oraz wyjście transkodera w systemie heksadecymalnym
- Wyświetlacze segmentowe niebieskie - wyświetlają wejście oraz wyjście transkodera w systemie dziesiętnym
- Przerzutnik - pozwala na szybkie przetestowanie układu. Jeżeli którekolwiek z wyjść okaże się niepoprawne, na wyjściu Q otrzymamy “1” logiczną i zapali się czerwona dioda. W przeciwnym razie będzie się świecić zielona (podłączona do ~Q). W celu synchronizacji czasowej i eliminacji błędów wynikających z opóźnień przesyłanego sygnału, do wejścia CLK transkodera podpinamy wyjście R generatora słów. Generator prześle sygnał o swojej gotowości w odpowiednim momencie i dopiero wtedy przerzutnik sprawdzi, czy podpięty do wejścia S sygnał z komparatora jest niski czy wysoki - przy wysokim zapali się dioda czerwona.

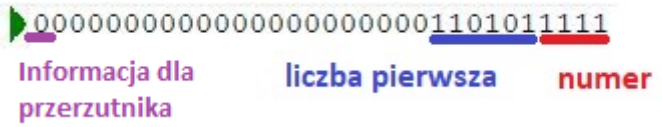


Rys. 11. Schemat układu testującego w programie Multisim



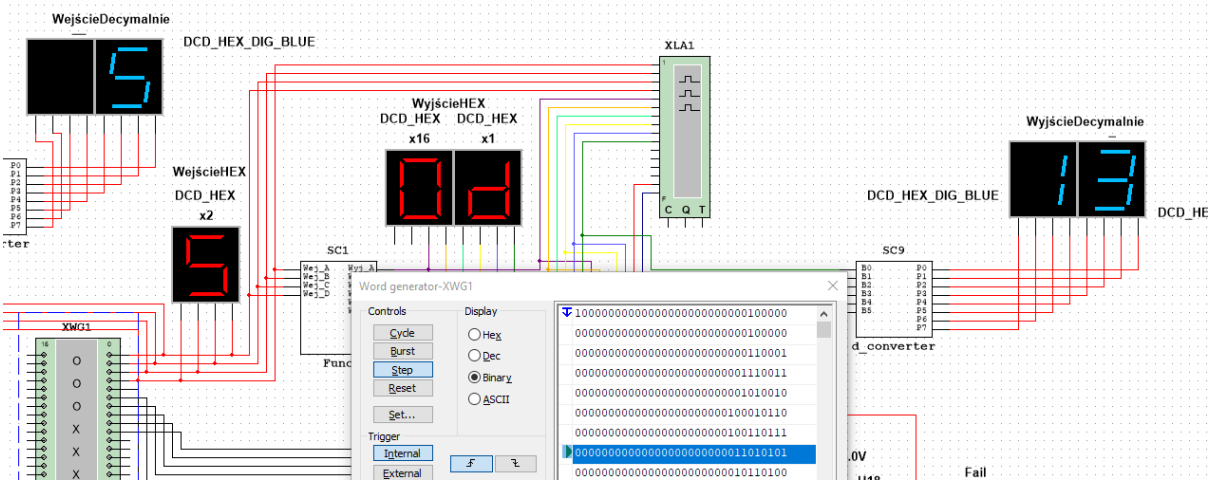
Rys. 12. Odczyt sygnału z generatora słów - pierwszy sygnał resetuje przerzutnik

Ostatnie od prawej strony 4 bity posłużyły do podania liczby w systemie binarnym przed zakodowaniem jej do postaci 6 - bitowej jako odpowiedniej liczby pierwszej. Najstarszy bit odpowiada informacji o resecie dla przerzutnika (reset, kiedy bit jest “1” logiczną).



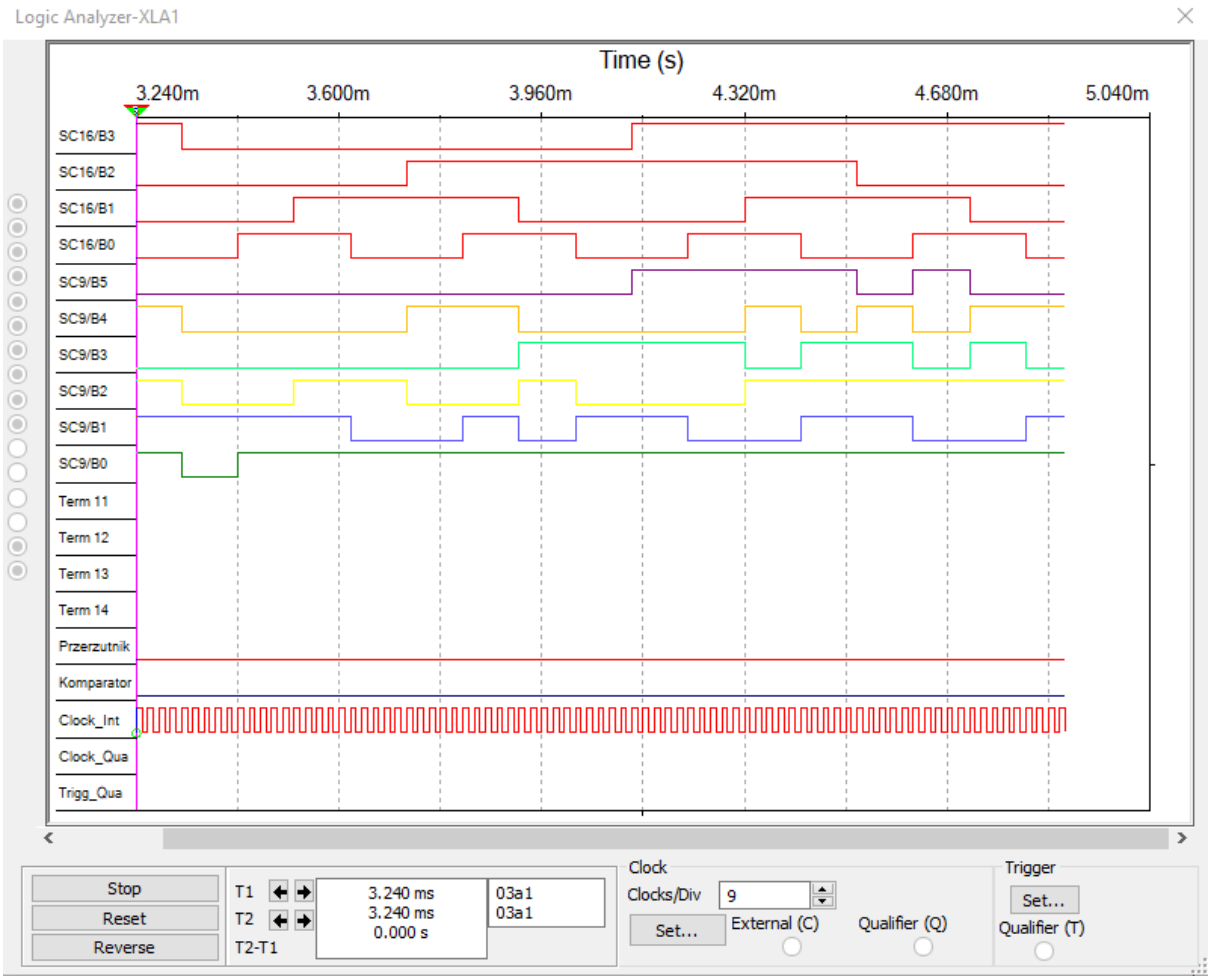
Rysunek 13. Zapis sygnału w generatorze słów

Każdy bit wyjściowy z układu transkodującego jest podłączony do odpowiedniego segmentu wyświetlacza siedmiosegmentowego. Pozwala to na weryfikację poprawności działania układu.



Rys. 13. Wyświetlanie wejścia oraz wyjścia transkodera

Możemy się teraz przyjrzeć wykresowi wygenerowanemu przez analizator stanów logicznych (Rys. 14). Pierwsze cztery czerwone linie odpowiadają kolejnym 4 bitom sygnału wejścia transkodera, a następne 6 kolorowych sygnałom jego wyjścia. Poniżej widać sygnał z przerzutnika (czerwony), z komparatora (granatowy) oraz takty zegara. Sygnały z komparatora oraz przerzutnika pozostają niskie dla wszystkich możliwych wejść, co stanowi dowód na poprawność zaprojektowanego transkodera.



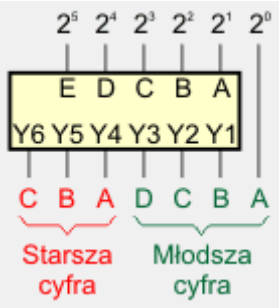
Rys. 14. Wykres wygenerowany przez analizator stanów logicznych

8. Wyświetlanie w systemie dziesiętnym

Ciekawym problemem, który napotkaliśmy, było wyświetlanie wyników na wyświetlaczach siedmiosegmentowych. Co prawda istnieją wyświetlacze, które same w sobie zawierają odpowiedni konwerter, który pozwala wyświetlić podpiętą liczbę binarną w systemie szesnastkowym (które zdecydowaliśmy się także użyć w celu weryfikacji poprawności działania wyświetlania decymalnego), jednak program Multisim nie oferuje żadnych konwerterów ani specjalnych wyświetlaczy dla systemu dziesiętnego. Do problemu próbowaliśmy podejść na dwa sposoby - za pierwszym razem musieliśmy popełnić błąd, gdyż wyświetlacz nie działał w oczekiwany sposób. Drugie podejście okazało się być sukcesem.

8.1. Podejście pierwsze: implementacja układu 74185

Na stronie https://eduinf.waw.pl/inf/prg/010_uc/74185.php zaprezentowany został istniejący układ 74185, który sześciobitową liczbę binarną konwertuje na kod BCD pozwalający wyświetlić liczbę dwucyfrową w systemie dziesiętnym. Poniżej prezentujemy poglądowy schemat układu (obrazek z powyższej strony internetowej):



Rys. 15 Schemat układu 74185

Na podanej stronie internetowej znajduje się również tablica prawdy, na podstawie której stworzyliśmy tablice Karnaugh (skorzystaliśmy w tym celu ze strony https://charlie-coleman.com/) oraz odpowiednie funkcje logiczne pozwalające nam zbudować układ.

wartość dwójkowa	Wejścia						Wyjścia							
	E	D	C	B	A	$\overline{G}$	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1
0-1	0	0	0	0	0	0	1	1	0	0	0	0	0	0
2-3	0	0	0	0	1	0	1	1	0	0	0	0	0	1
4-5	0	0	0	1	0	0	1	1	0	0	0	0	1	0
6-7	0	0	0	1	1	0	1	1	0	0	0	0	1	1
8-9	0	0	1	0	0	0	1	1	0	0	0	1	0	0
10-11	0	0	1	0	1	0	1	1	0	0	1	0	0	0
12-13	0	0	1	1	0	0	1	1	0	0	1	0	0	1
14-15	0	0	1	1	1	0	1	1	0	0	1	0	1	0
16-17	0	1	0	0	0	0	1	1	0	0	1	0	1	1
18-19	0	1	0	0	1	0	1	1	0	0	1	1	0	0
20-21	0	1	0	1	0	0	1	1	0	1	0	0	0	0
22-23	0	1	0	1	1	0	1	1	0	1	0	0	0	1
24-25	0	1	1	0	0	0	1	1	0	1	0	0	1	0
26-27	0	1	1	0	1	0	1	1	0	1	0	0	1	1
28-29	0	1	1	1	0	0	1	1	0	1	0	1	0	0
30-31	0	1	1	1	1	0	1	1	0	1	1	0	0	0
32-33	1	0	0	0	0	0	1	1	0	1	1	0	0	1
34-35	1	0	0	0	1	0	1	1	0	1	1	0	1	0
36-37	1	0	0	1	0	0	1	1	0	1	1	0	1	1
38-39	1	0	0	1	1	0	1	1	0	1	1	1	0	0
40-41	1	0	1	0	0	0	1	1	1	0	0	0	0	0
42-43	1	0	1	0	1	0	1	1	1	0	0	0	0	1
44-45	1	0	1	1	0	0	1	1	1	0	0	0	1	0
46-47	1	0	1	1	1	0	1	1	1	0	0	0	1	1
48-49	1	1	0	0	0	0	1	1	1	0	0	1	0	0
50-51	1	1	0	0	1	0	1	1	1	0	1	0	0	0
52-53	1	1	0	1	0	0	1	1	1	0	1	0	0	1
54-55	1	1	0	1	1	0	1	1	1	0	1	0	1	0
56-57	1	1	1	0	0	0	1	1	1	0	1	0	1	1
58-59	1	1	1	0	1	0	1	1	1	0	1	1	0	0
60-61	1	1	1	1	0	0	1	1	1	1	0	0	0	0
62-63	1	1	1	1	1	0	1	1	1	1	0	0	0	1
dowolne	X	X	X	X	X	1	1	1	1	1	1	1	1	1

Rys. 16 Tabela prawdy układu 74185

Rys. 17 Tablica Karnaugh: wyjście Y7 oraz Y8

Funkcja logiczna:

Y7=Y8=1

Rys. 18 Tablica Karnaugh: wyjście Y6

Funkcja logiczna:

$$Y6 = e * c + e * d = \overline{\overline{e * c} * \overline{e * d}}$$

Rys. 19 Tablica Karnaugh: wyjście Y5

Funkcja logiczna:

$$Y5 = c * b * d + \overline{e} * d * c + \overline{e} * d * b + e * \overline{d} * \overline{c} =$$

$$= \overline{\overline{c * b * d} * \overline{\overline{e} * d * c} * \overline{\overline{e} * d * b} * \overline{e * \overline{d} * \overline{c}}}$$

Rys. 20 Tablica Karnaugh: wyjście Y4

Funkcja logiczna:

$$Y4 = e * \overline{d} * \overline{c} + e * \overline{c} * b + d * \overline{c} * \overline{b} * a + \overline{e} * d * \overline{c} * \overline{b} + \overline{e} * \overline{d} * c * b + \overline{e} * \overline{d} * c * a +$$

$$+ \overline{e} * c * b * a + e * d * c * \overline{b} =$$

$$= \overline{\overline{e * \overline{d} * \overline{c}} * \overline{e * \overline{c} * b} * \overline{d * \overline{c} * \overline{b} * a} * \overline{\overline{e} * d * \overline{c} * \overline{b}} * \overline{\overline{e} * \overline{d} * c * b} * \overline{\overline{e} * \overline{d} * c * a} *$$

$$* \overline{\overline{e} * c * b * a} * \overline{e * d * c * \overline{b}}$$

Rys. 21 Tablica Karnaugh: wyjście Y3

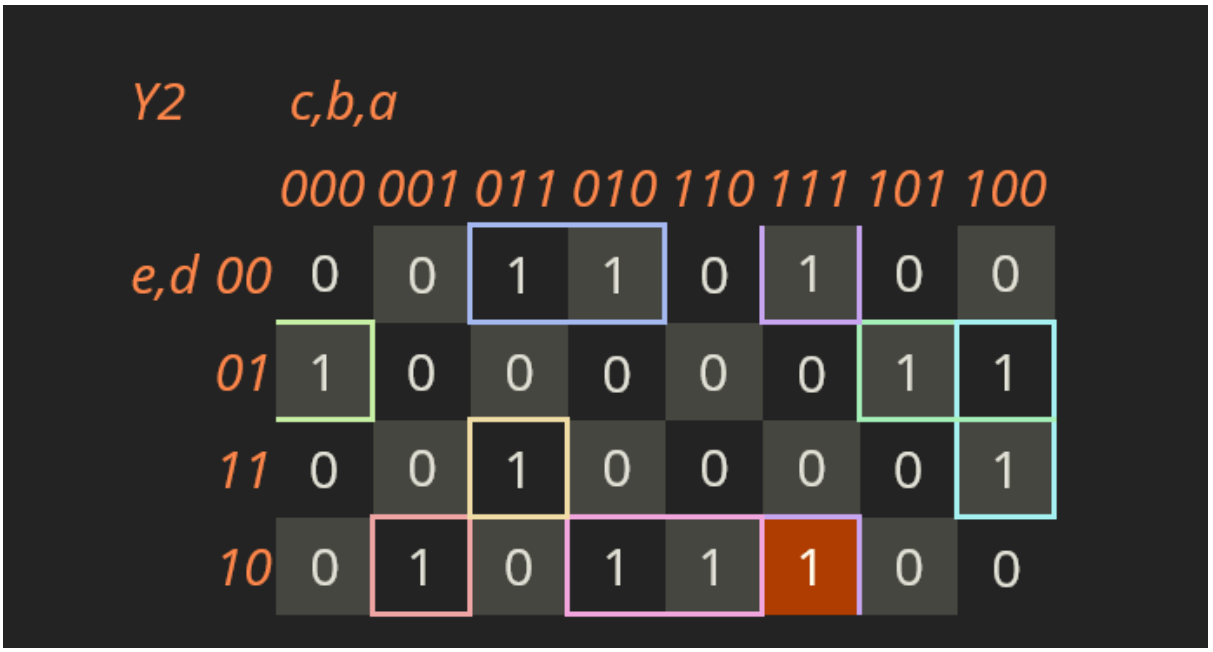
Funkcja logiczna:

$$Y3 = \overline{e} * \overline{d} * c * \overline{b} * \overline{a} + \overline{e} * d * \overline{c} * \overline{b} * a + \overline{e} * d * c * b * \overline{a} + e * d * \overline{c} * \overline{b} * \overline{a} +$$

$$+ e * d * c * \overline{b} * a + e * \overline{d} * \overline{c} * b * a =$$

$$= \overline{\overline{e} * \overline{d} * c * \overline{b} * \overline{a} * \overline{\overline{e} * d * \overline{c} * \overline{b} * a} * \overline{\overline{e} * d * c * b * \overline{a} * \overline{e} * d * \overline{c} * \overline{b} * \overline{a}} +$$

$$* \overline{\overline{e * d * c * \overline{b} * a} * \overline{e * \overline{d} * \overline{c} * b * a}}$$



Rys. 22 Tablica Karnaugh: wyjście Y2

Funkcja logiczna:

$$Y2 = \overline{e} * \overline{d} * \overline{c} * b + \overline{e} * \overline{d} * c * b * a + \overline{e} * d * \overline{b} * \overline{a} + \overline{e} * d * c * \overline{b} + e * d * \overline{c} * b * a +$$

$$+ e * \overline{d} * \overline{c} * \overline{b} * a + e * \overline{d} * b * \overline{a} + \overline{d} * c * b * a =$$

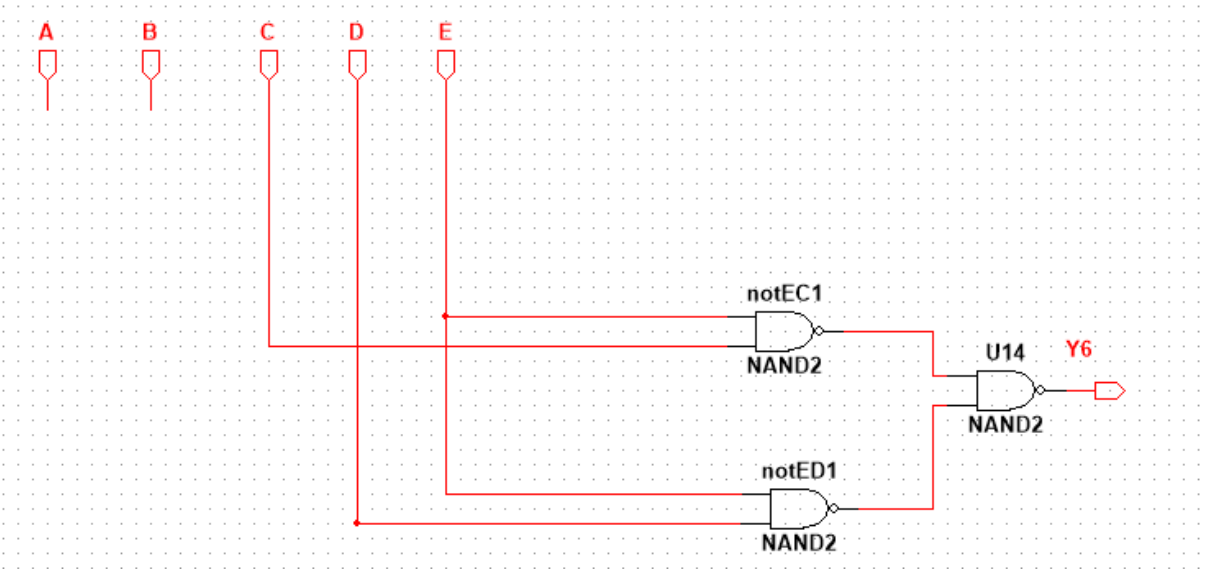
$$= \overline{\overline{e} * \overline{d} * \overline{c} * b} * \overline{\overline{e} * \overline{d} * c * b * a} * \overline{\overline{e} * d * \overline{b} * \overline{a}} * \overline{\overline{e} * d * c * \overline{b} * e * d * \overline{c} * b * a} *$$
$$* \overline{e * \overline{d} * \overline{c} * \overline{b} * a} * \overline{e * \overline{d} * b * \overline{a}} * \overline{\overline{d} * c * b * a}$$

Rys. 23 Tablica Karnaugh: wyjście Y1

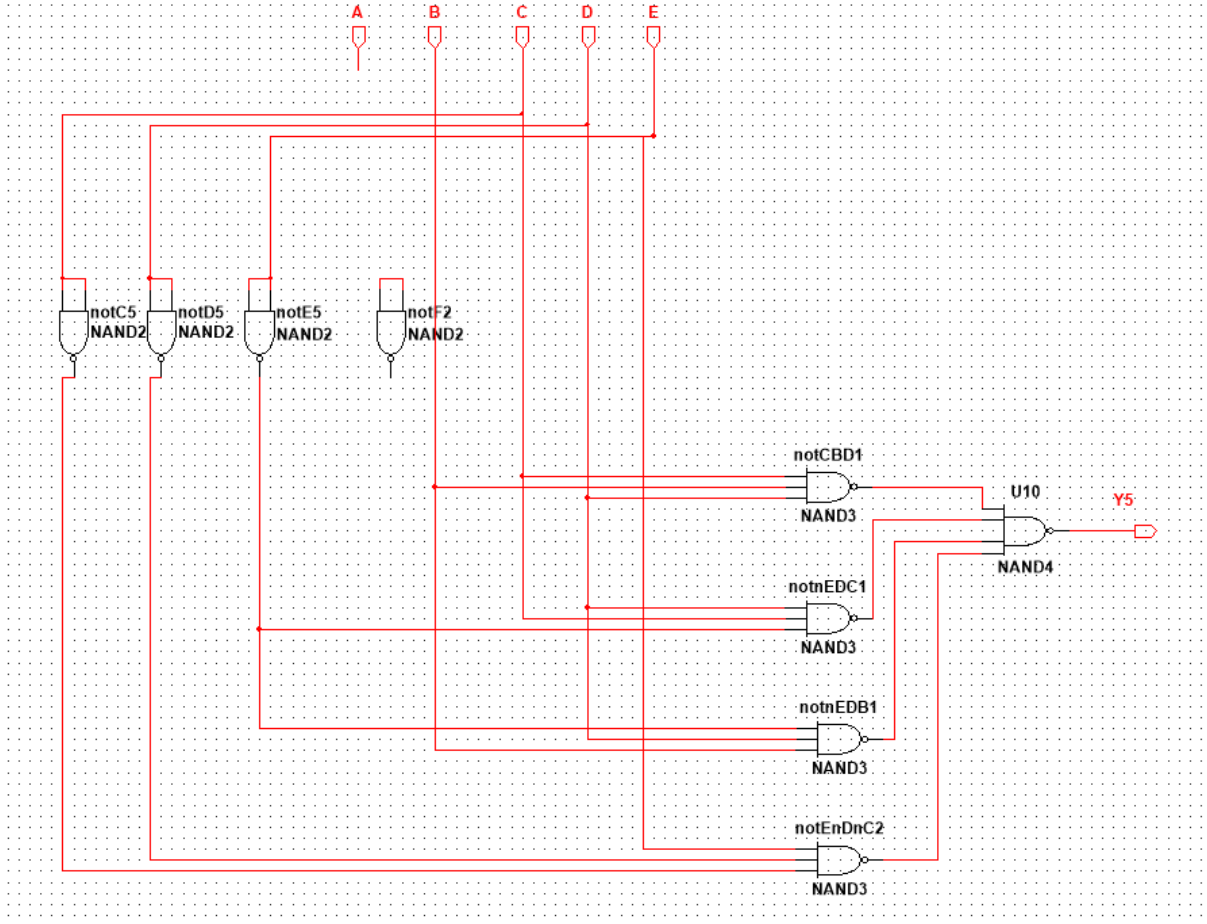
Funkcja logiczna:

$$Y1 = \overline{e} * \overline{d} * \overline{c} * a + \overline{e} * \overline{d} * c * b * \overline{a} + \overline{e} * d * \overline{c} * \overline{b} * \overline{a} + \overline{e} * \overline{c} * b * a + \overline{e} * d * c * \overline{b} * a +$$
$$+ e * \overline{c} * b * \overline{a} + e * c * b * a + e * \overline{d} * c * a + e * d * c * \overline{b} * \overline{a} + e * \overline{d} * \overline{c} * \overline{b} * \overline{a} =$$
$$= \overline{\overline{e} * \overline{d} * \overline{c} * a} * \overline{\overline{e} * \overline{d} * c * b * \overline{a}} * \overline{\overline{e} * d * \overline{c} * \overline{b} * \overline{a}} * \overline{\overline{e} * \overline{c} * b * a} * \overline{\overline{e} * d * c * \overline{b} * a} *$$
$$* \overline{e * \overline{c} * b * \overline{a}} * \overline{e * c * b * a} * \overline{e * \overline{d} * c * a} * \overline{e * d * c * \overline{b} * \overline{a}} * \overline{e * \overline{d} * \overline{c} * \overline{b} * \overline{a}}$$

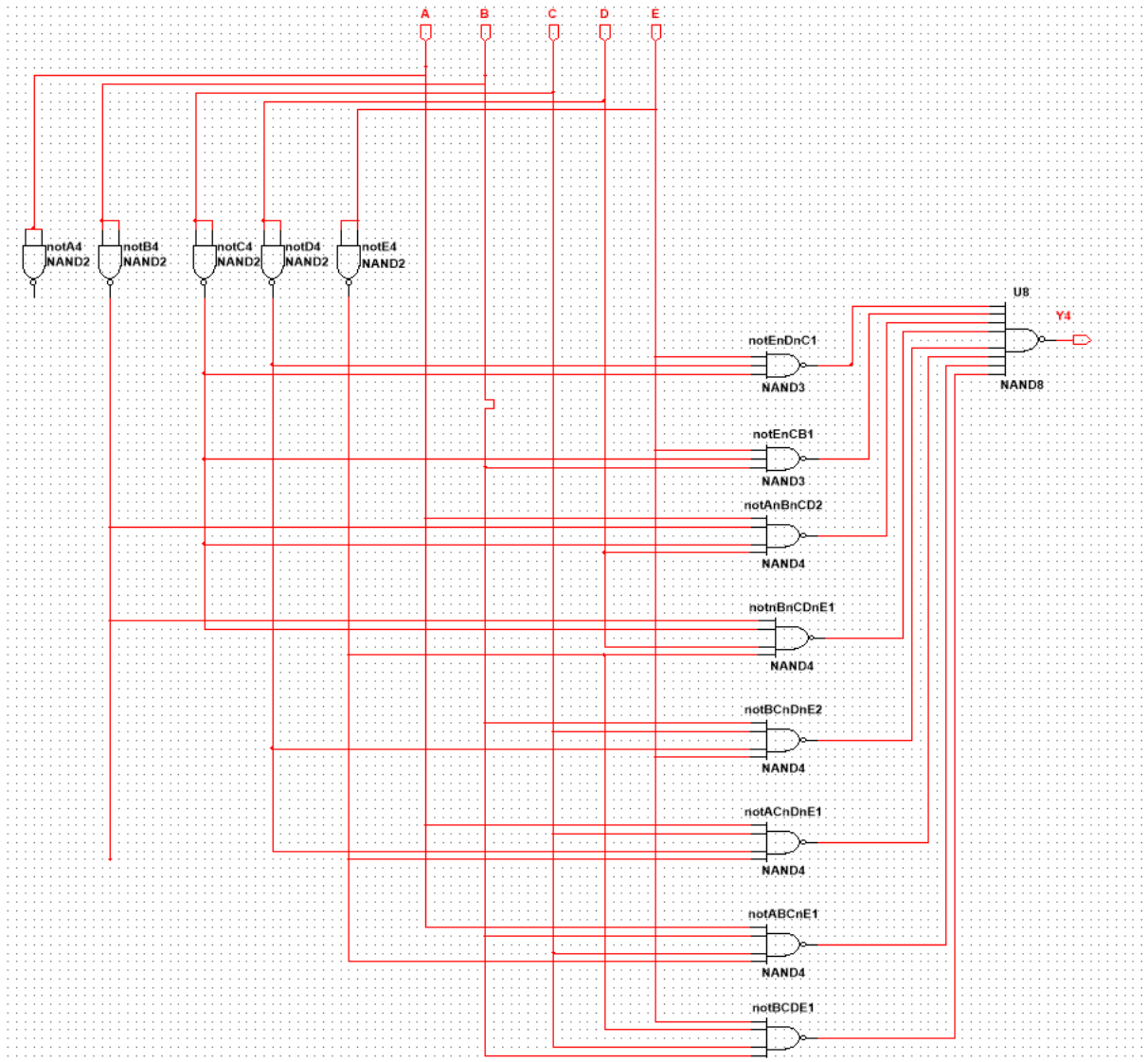
Następnie dokonaliśmy implementacji w programie Mulstisim:



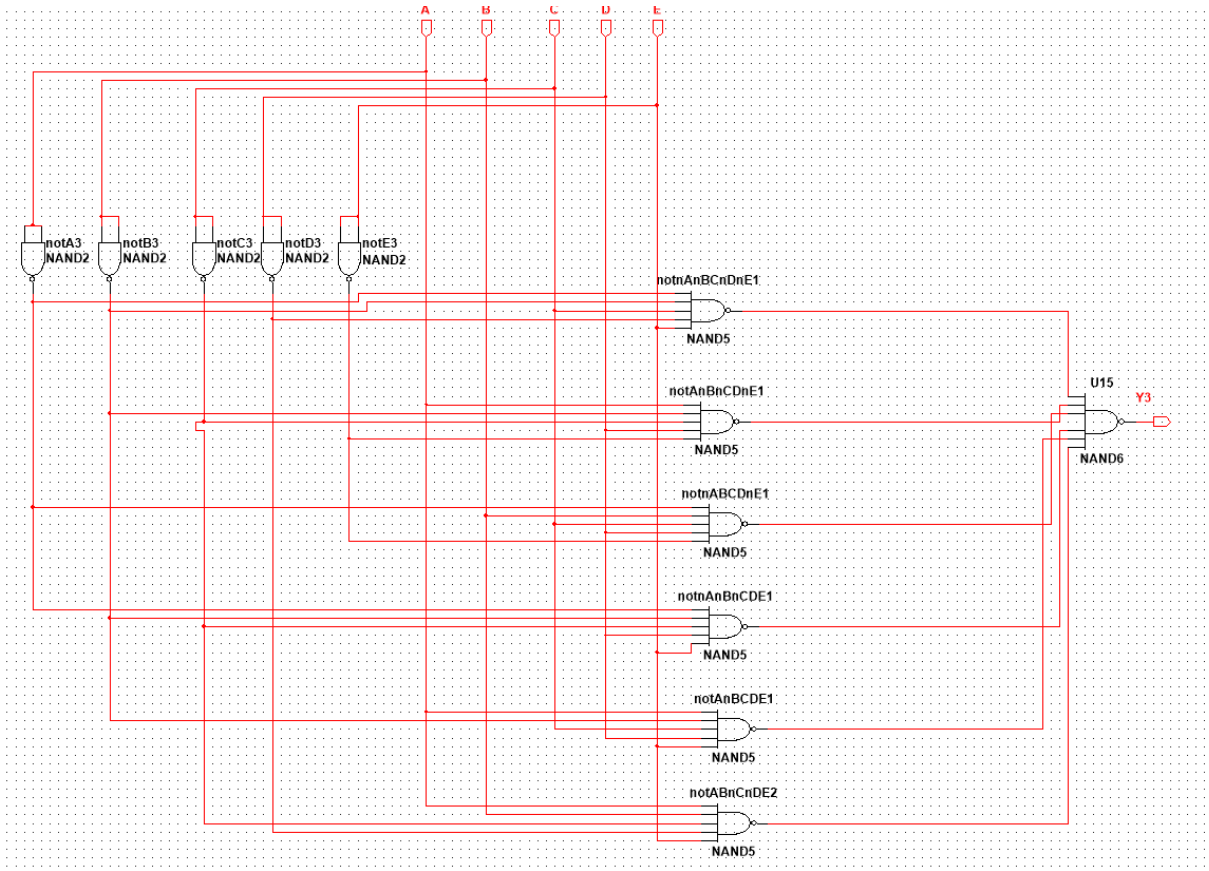
Rys. 24 Schemat układu 74185: wyjście Y6



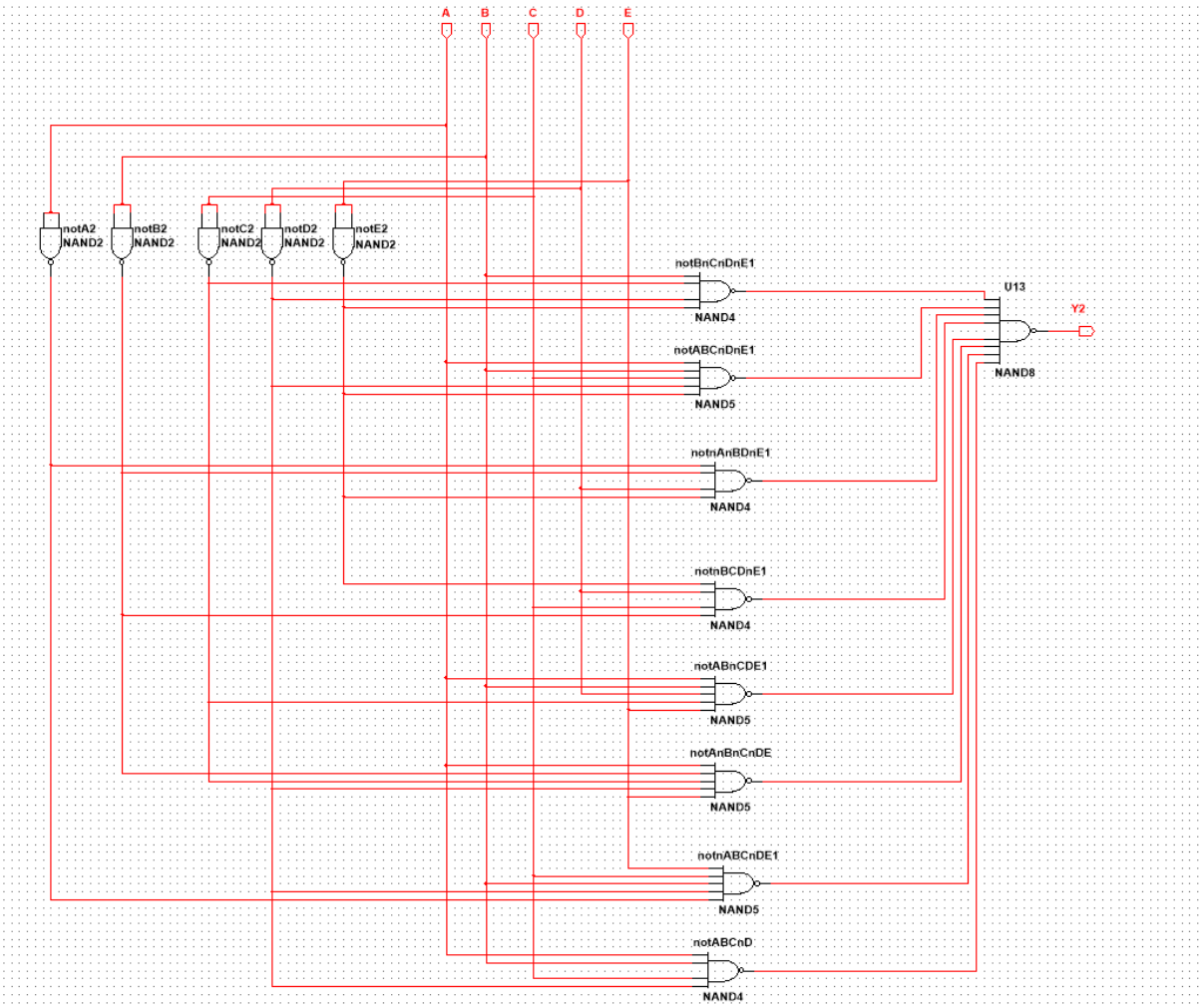
Rys. 25 Schemat układu 74185: wyjście Y5



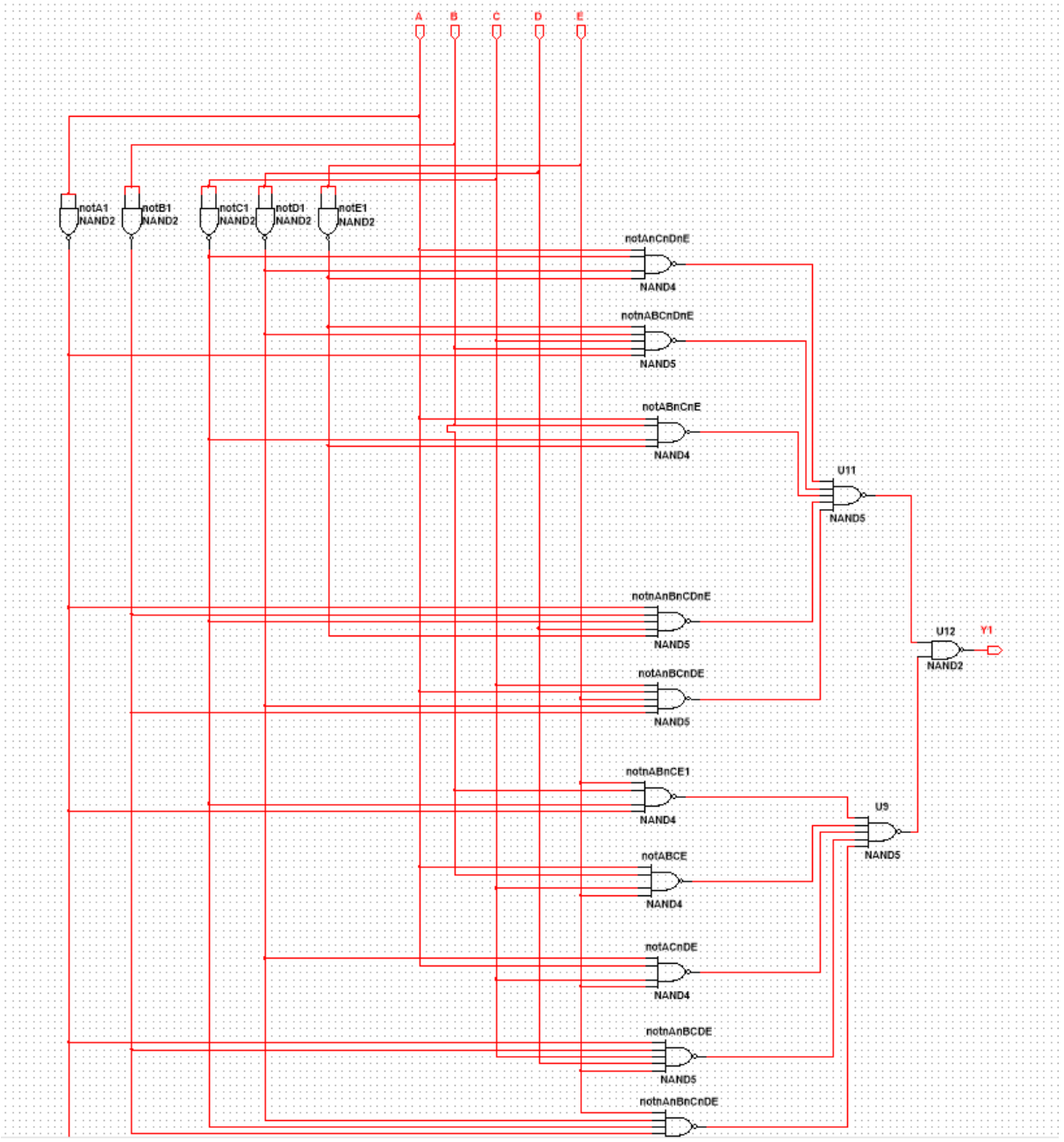
Rys. 26 Schemat układu 74185: wyjście Y4



Rys. 27 Schemat układu 74185: wyjście Y3



Rys. 28 Schemat układu 74185: wyjście Y2

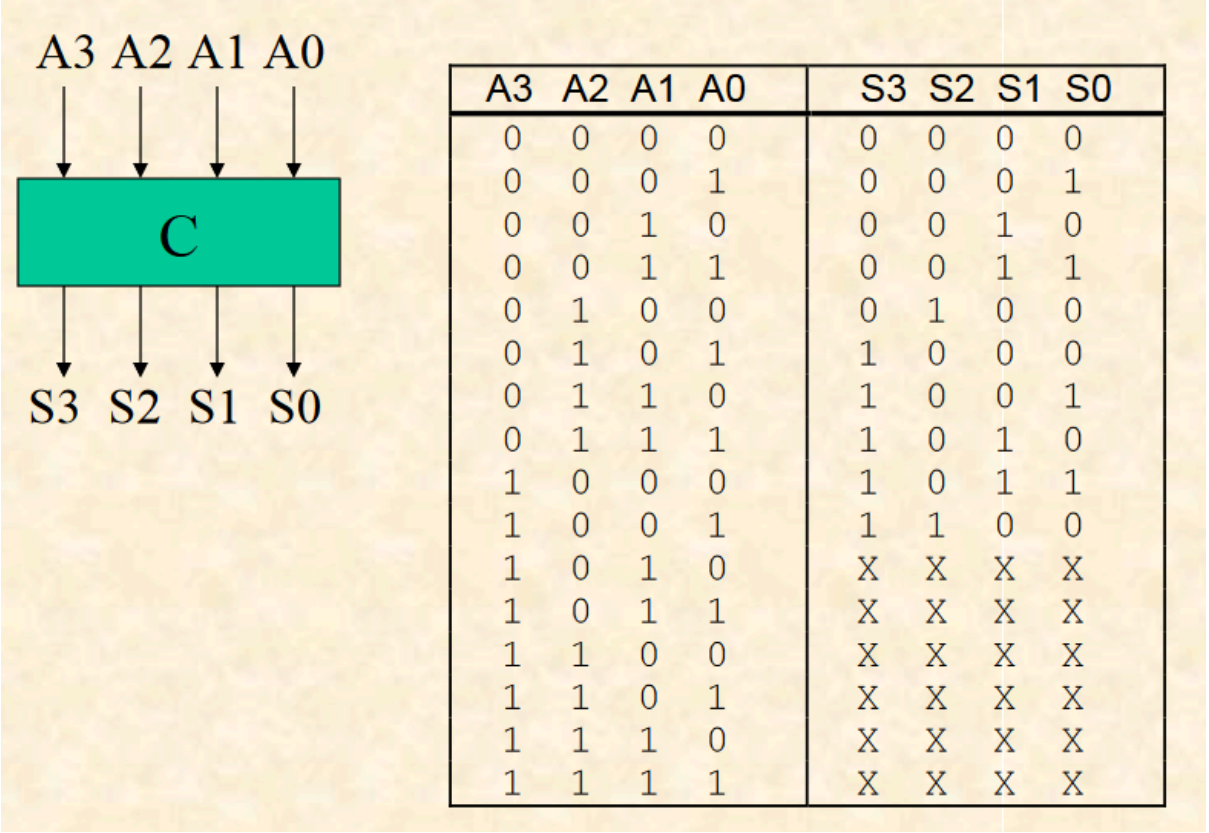


Rys. 29 Schemat układu 74185: wyjście Y1

Po podpięciu do generatora słów okazało się jednak, że układ nie działa. Oznacza to, że gdzieś musieliśmy popełnić błąd. Mogło się to stać podczas przekształcania funkcji logicznych na bramki NAND lub też podczas implementacji. Pomylić się było nietrudno zważając na dużą liczbę wielowejściowych bramek w każdym podukładzie.

8.2. Podejście drugie: metoda Add-Shift-3

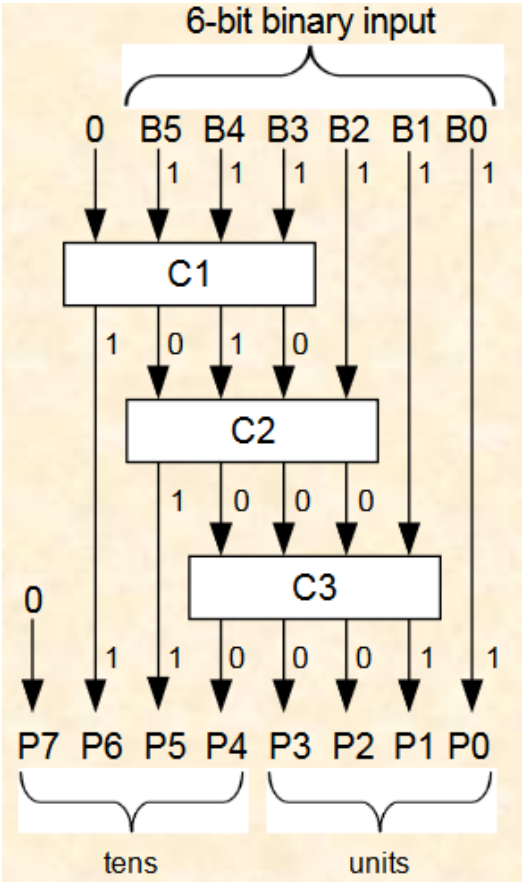
W drugim podejściu skorzystaliśmy ze sposobu przedstawionego w prezentacji: https://www.mikrocontroller.net/attachment/19220/D3.1_Binary2BCD.ppt.pdf. Należało zaimplementować komponent dany następującą tablicą prawdy (Rys. 30, 31 pochodzą z powyższej prezentacji):



A3	A2	A1	A0	S3	S2	S1	S0
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	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Rys. 30 Schemat komponentu konwertera decymalnego

Następnie połączyliśmy trzy tego typu komponenty w jeden układ:



Rys. 31 Połączenie komponentów w jeden układ

Tym razem stworzenie tablic Karnaugh, funkcji oraz zaimplementowanie komponentów okazało się o wiele łatwiejsze niż w podejściu pierwszym ze względu na to, że komponent posiada tylko cztery wejścia i cztery wyjścia i trudniej było się pomylić. Poniżej prezentujemy tablice Karnaugh dla poszczególnych wyjść komponentu:

A3A2 \ A1A0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

Tabela 8. Tablica Karnaugh: wyjście S3

Funkcja logiczna:

$S3 = A2 * A0 + A2 * A1 + A3$

A3A2 \ A1A0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

Tabela 9. Tablica Karnaugh: wyjście S2

Funkcja logiczna:

$$S2 = A2 * \overline{A1} * \overline{A0} + A3 * A0$$

A3A2 \ A1A0	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	X	X	X	X
10	1	0	X	X

Tabela 10. Tablica Karnaugh: wyjście S1

Funkcja logiczna:

$$S1 = A1 * A0 + A3 * \overline{A0} + \overline{A2} * A3$$

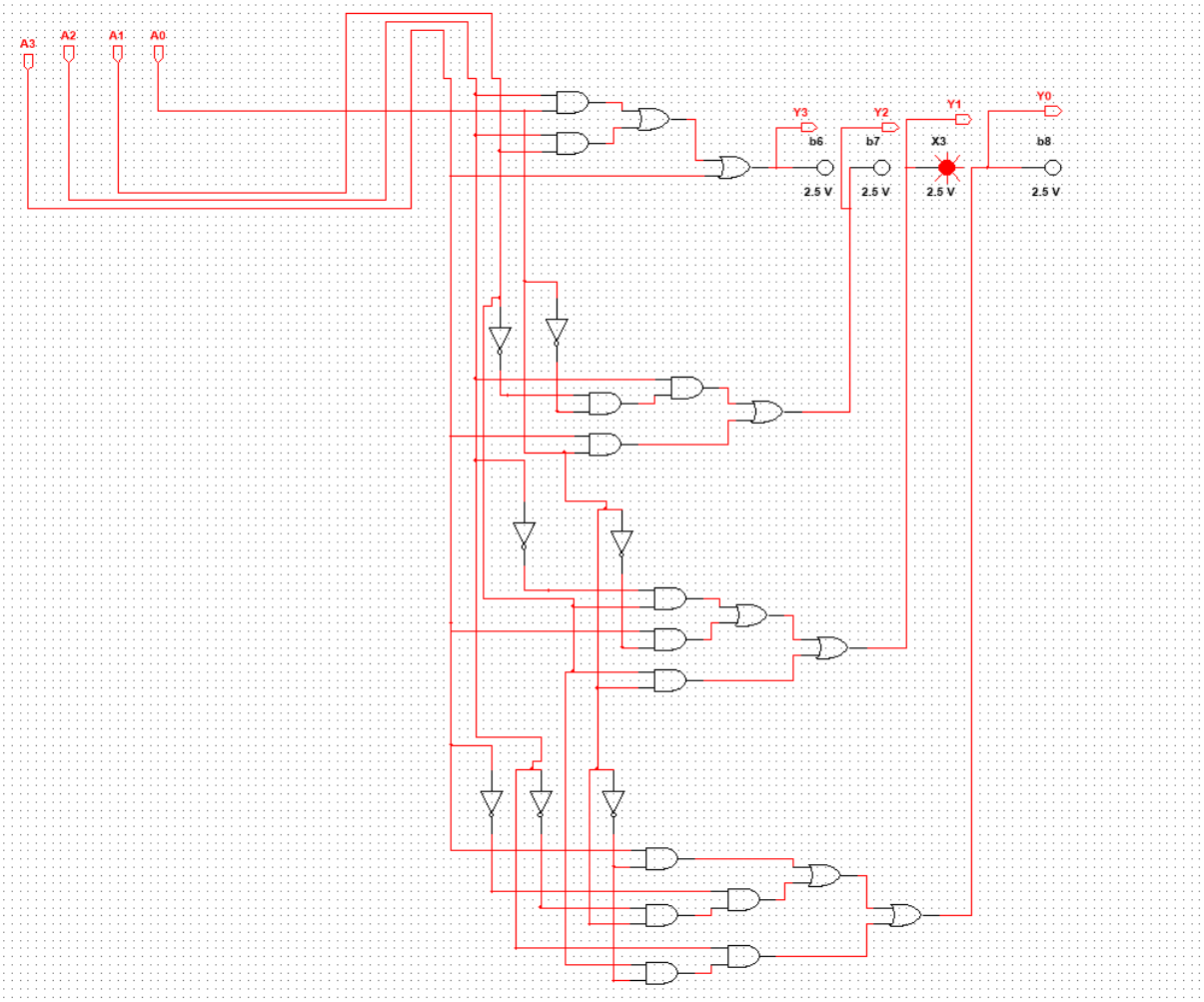
A3A2 \ A1A0	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

Tabela 11. Tablica Karnaugh: wyjście S0

Funkcja logiczna:

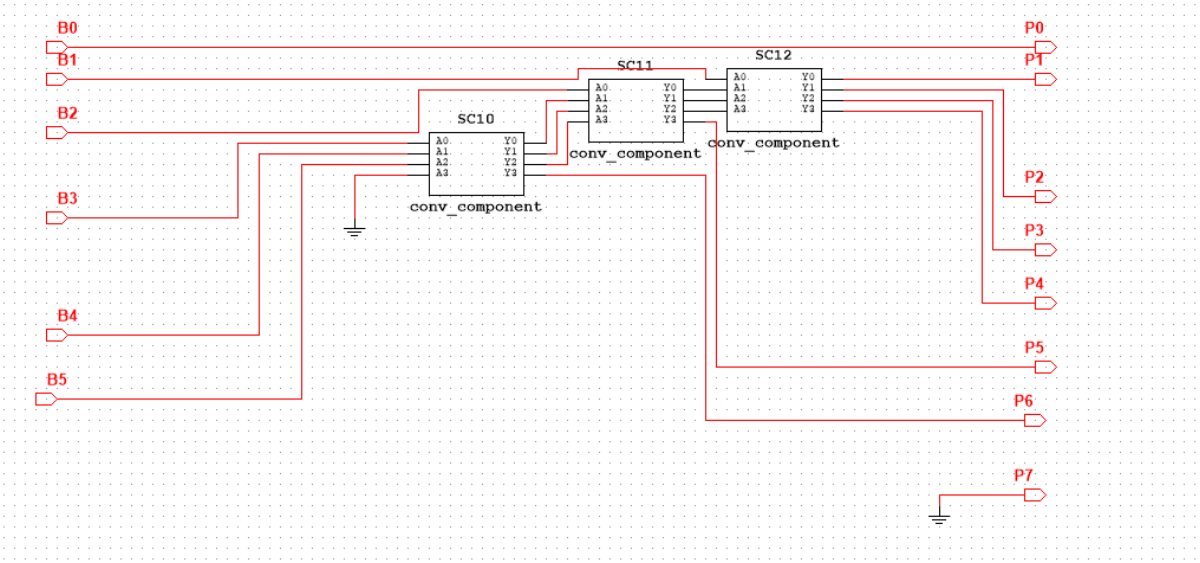
$$S0 = \overline{A3} * \overline{A2} * A0 + A2 * A1 * \overline{A0} + A3 * \overline{A0}$$

Następnie zaimplementowaliśmy funkcje logiczne w programie Multisim. W celu usprawnienia implementacji skorzystaliśmy z narzędzia “Logic converter”, które wygenerowało odpowiednie implementacje podanych wyżej funkcji logicznych dla każdego wyjścia. Otrzymaliśmy poniższy układ:



Rys. 32 Komponent C układu konwertującego - implementacja w programie Multisim

Później połączyliśmy trzy komponenty C w jeden układ (jak na Rys. 31):



Rys. 33 Schemat układu konwertującego sześciobitową liczbę binarną na kod BCD umożliwiający wyświetlenie liczby w systemie dziesiętnym

Tak przygotowany układ podpięliśmy do układu testującego transkoder liczb pierwszych w celu wyświetlenia wejść oraz wyjść transkodera.

Tym razem udało się osiągnąć sukces - liczby wyświetlane w systemie decymalnym (wyświetlacze niebieskie na Rys. 11) są równe tym wyświetlanym w systemie heksadecymalnym (wyświetlacze czerwone na Rys. 11)

8.2.1 Wyjaśnienie działania układu

Układ korzysta z algorytmu Add-Shift-3. Binarną liczbę wejściową poddajemy przekształceniom. W każdym kroku bity grupujemy po 4 (4 najmłodsze odpowiadają cyfrze jedności, 4 kolejne cyfrze dziesiętnej itd.). Algorytm składa się z następujących kroków:

1) Liczbę przesuwamy bitowo w lewo
2) Jeśli wykonaliśmy już 8 przesunięć, to kolejne grupy bitów reprezentują odpowiednie cyfry w systemie dziesiętnym
3) Jeżeli którakolwiek z grup BCD ma wartość większą lub równą 5, to dodajemy liczbę 3 do tej grupy

Algorytm ilustruje poniższy obrazek (źródło: Binary-to-BCD Converter):

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Rys. 34 Przykład działania algorytmu Add-Shift-3

9. Wnioski i możliwe zastosowania

Wnioski z projektowania i testowania układu transkodera czterobitowej liczby naturalnej na sześcio-bitową liczbę pierwszą, bazującego na bramkach NAND:

- Testowanie za pomocą generatora słów i analizatora stanów logicznych znacząco ułatwia weryfikację poprawności działania układu - pozwala zaobserwować ewentualne błędy
- Zastosowanie przerzutnika pozwala szybko i automatycznie przetestować układ - w przypadku błędu na którymkolwiek wyjściu zapali się czerwona lampka i nie zgaśnie mimo poprawnych wyjść późniejszych - pozwoli nam to nie przeoczyć żadnego błędu
- Należy synchronizować zegar przerzutnika, by otrzymać prawidłowe dane o poprawności działania transkodera
- Proces minimalizacji za pomocą tablic Karnaugh pozwala na zredukowanie złożoności układu
- Zastosowanie bramek NAND pozwoliło zredukować liczbę użytych tranzystorów
- Wyświetlanie liczb w systemie dziesiętnym nie jest operacją trywialną. Istniejące transkodery ciężko zaimplementować samemu ze względu na stopień skomplikowania tablic Karnaugh oraz funkcji logicznych. Ciekawym rozwiązaniem jest zastosowanie podukładu bazującego na algorytmie Add-Shift-3

Zastosowania:

- W aplikacjach kryptograficznych, transkodery mogą być używane do generowania sekwencji liczb pierwszych, które są kluczowym elementem wielu algorytmów szyfrowania (np. algorytm RSA). Algorytmy szyfrujące korzystające z liczb pierwszych są bezpieczne ze względu na brak podzielności przez żadną inną liczbę oprócz 1.

- W symulacjach komputerowych, grach logicznych, które wymagają generowania liczb pierwszych, taki układ mógłby być wykorzystywany do tworzenia sekwencji lub do weryfikacji rozwiązań bazujących na liczbach pierwszych.

- W systemach zabezpieczeń, takich jak zamki szyfrowe czy systemy autentykacji, może być potrzebne szybkie generowanie lub weryfikowanie sekwencji opartych na liczbach pierwszych. Przydatne jest to np. dla kart kredytowych oraz w bankach

- W telekomunikacji liczby pierwsze są używane w celu detekcji błędó oraz automatycznego poprawienia utraconych przy przesyle danych

- Generacja liczb pseudolosowych w komputerze również odbywa się na bazie liczb pierwszych

- W komputerach kwantowych liczby pierwsze są wykorzystywane do szybkiej faktoryzacji dużych liczb

- Transkoder mógłby się również przydać podczas kompresji danych, która przy wykorzystaniu liczb pierwszych jest efektywniejsza

Źródła

https://unacademy.com/content/ssc/study-material/mathematics/prime-numbers-in-real-life-situations/
https://buki.org.pl/blogs/tajemnica-liczb-pierwszych/
https://www.mikrocontroller.net/attachment/19220/D3.1_Binary2BCD.ppt.pdf
https://eduinf.waw.pl/inf/prg/010_uc/74185.php