

Technika Cyfrowa - sprawozdanie 1

Łukasz Jezapkowicz

15.03.2020

- 1 Bazując na dowolnych bramkach logicznych, proszę od podstaw zaprojektować, zbudować i przetestować układ sumujący (arytmetycznie) dowolne dwie liczby trzybitowe.

1.1 Półsumator oraz pełny sumator

W celu dodania dwóch liczb trzybitowych musimy najpierw uporać się z problemem przeniesienia. W systemie binarnym, gdy dodajemy bit 1 z bitem 1 wynikiem nie jest 2 lecz 0 a przeniesienie wynosi 1. Dodając pierwszy bit pierwszej liczby z pierwszym bitem drugiej liczby nie musimy się tym martwić, lecz przy kolejnych bitach przeniesienie odgrywa kluczową rolę. Przykładowe dodawanie dwóch liczb binarnych czterobitowych widoczne jest poniżej:

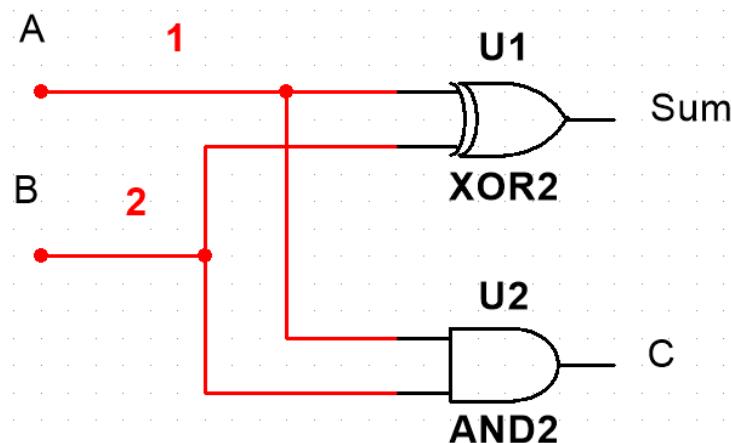
$$\begin{array}{r}
 & 1 & 1 & 1 \\
 & 1 & 0 & 1 & 1 \\
 + & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 0 & 1 & 0 & 0
 \end{array}
 \quad \begin{array}{l}
 \text{c - przeniesienie} \\
 \text{A} \\
 \text{B}
 \end{array}$$

By uporać się z tym problemem wprowadźmy dwa terminy: półsumator oraz pełny sumator.

Półsumator to układ, który na wejściu bierze jedynie dwa bity a na wyjściu produkuje wynik oraz przeniesienie. Poniżej widoczna tabelka prawdy półsumatora:

A	B	Sum	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

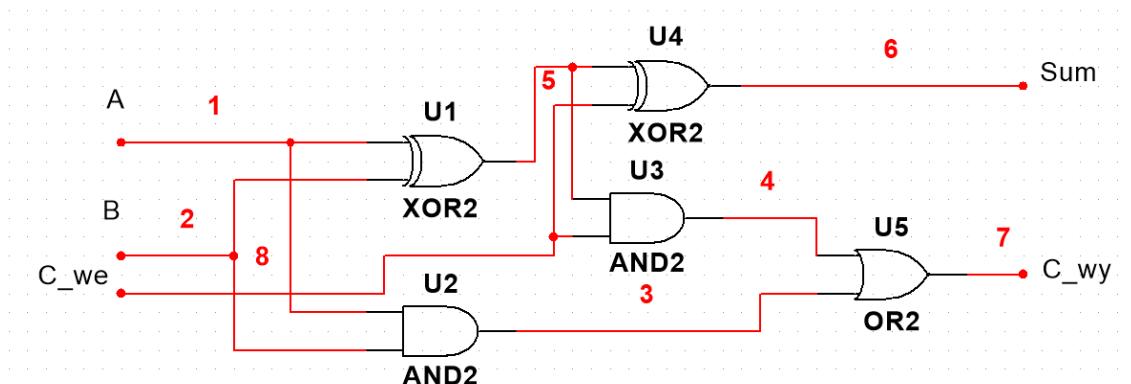
Jak widać *Sum* to *XOR* zaś *C* to *AND* a więc półsumator można łatwo przedstawić w układzie:



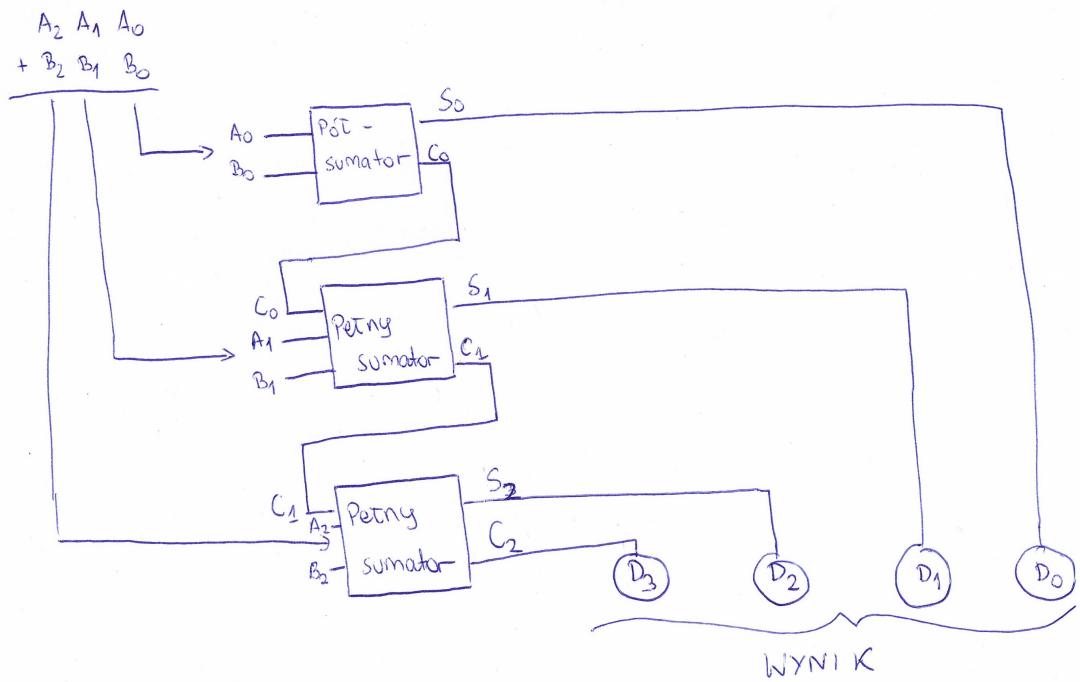
Pełny sumator to układ, który na wejściu oprócz dwóch bitów bierze również przeniesienie. Jego tabelka prawdy widoczna jest poniżej:

A	B	C_we	Sum	C_wy
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Układ pełnego sumatora jest nieco bardziej rozbudowany niż półsumatora. Warto zauważyć, że C_{wy} będzie równe 1 jeśli A i B będzie równe 1 lub jeśli jeden z bitów wejściowych oraz przeniesienie równe jest 1. Można zatem stwierdzić, że C_{wy} jest to bramka OR, do której wchodzi wynik AND z bitów wejściowych oraz AND z przeniesienia C_{we} oraz XOR'a bitów wejściowych natomiast Sum równy jest 1 jeżeli jeden z bitów wejściowych oraz przeniesienie równe jest 0 lub jeżeli bity wejściowe są sobie równe a przeniesienie równe jest 1. Można zatem stwierdzić, że Sum to bramka XOR, do której wchodzi wynik XOR'a bitów wejściowych oraz przeniesienie C_{we} . Poniżej widoczny układ pełnego sumatora:

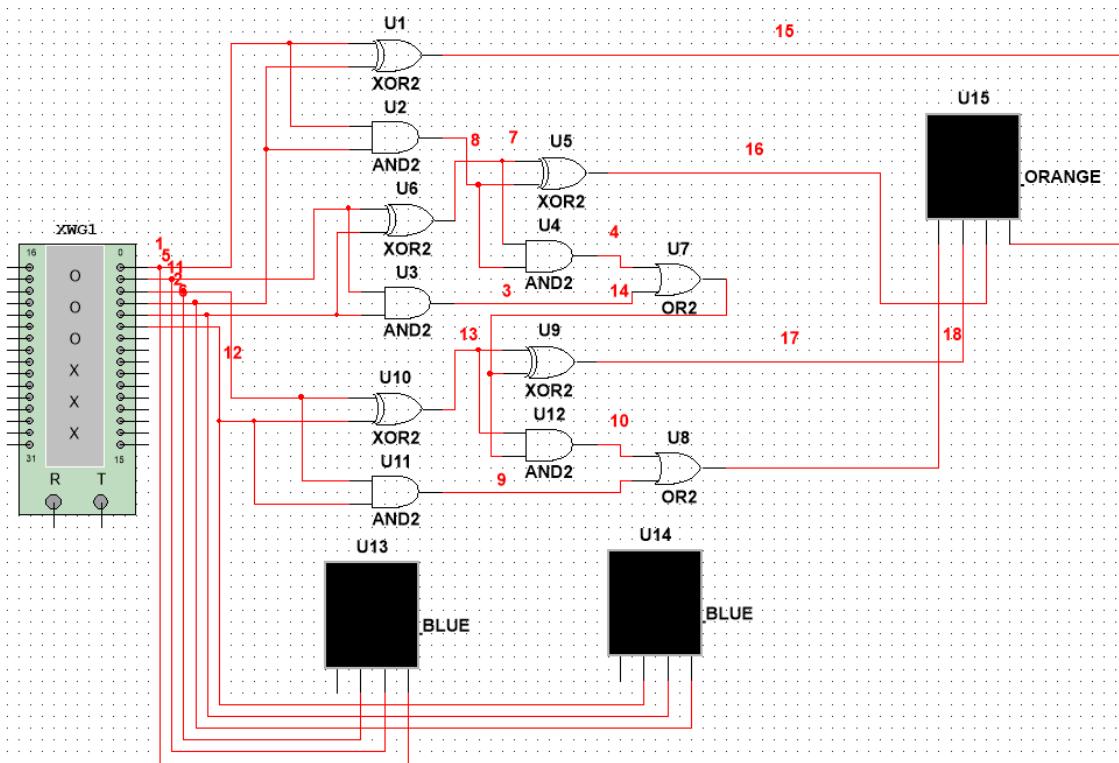


Wiedza na temat układów półsumatora oraz pełnego sumatora pozwala nam łatwo zaprojektować układ, który będzie dodawał dwie liczby trzybitowe oraz będzie uwzględniał przeniesienie podczas dodawania kolejnych bitów liczb. Przyjąłem, że najrozsądzniejszym wyborem jest przyjęcie możliwości pojawienia się czwartego bitu w liczbie jeżeli przeniesienie przy dodawaniu ostatnich bitów wyniesie 1. Układ składać będzie się z półsumatora (dla pierwszych bitów) oraz dwóch pełnych sumatorów. Szkic układu widoczny jest na zdjęciu poniżej:



1.2 Układ w Multisimie

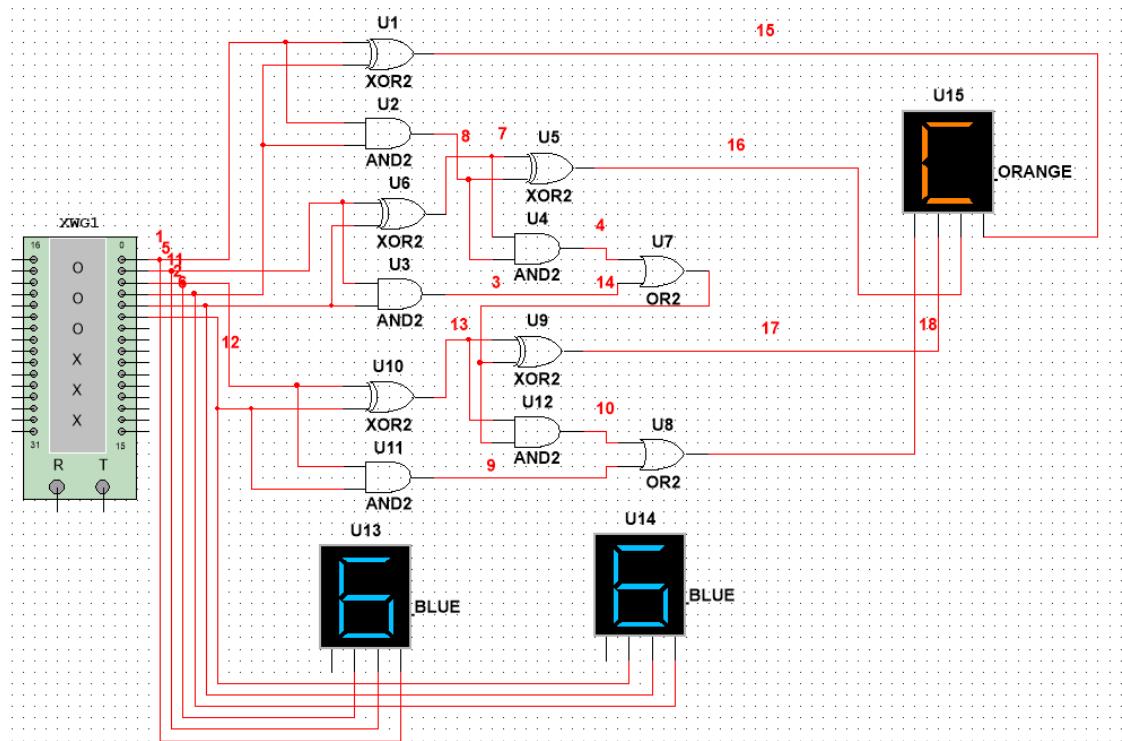
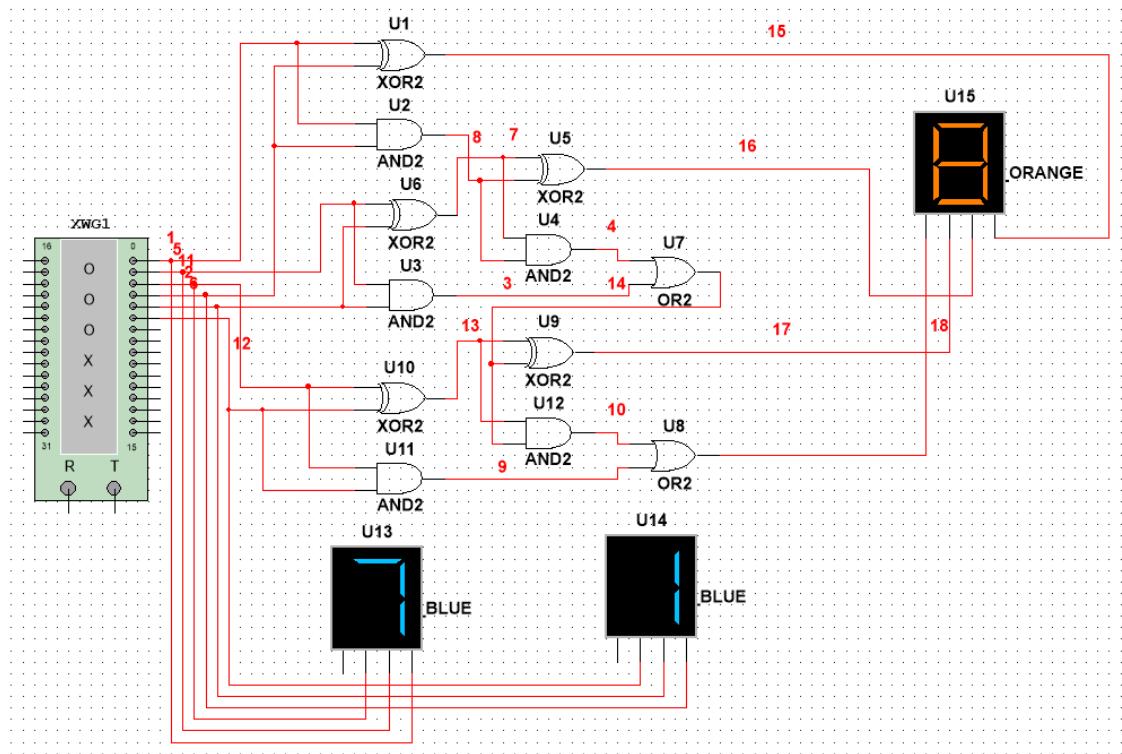
Na podstawie powyższego schematu nie jest trudnym zadaniem zbudowanie odpowiedniego układu w Multisimie. Poniżej zamieszczam układ zbudowany przeze mnie w Multisimie.



Word Generator generuje wszystkie liczby binarne sześciobitowe (Up Counter $\rightarrow 64$)

1.3 Testowanie

Po przetestowaniu układu stwierdzam, iż jest on wykonany poprawnie. Poniżej pokazuje przykładowe wyniki dla wyniku będącego liczbą trzybitową jak i wyniku będącego liczbą czterobitową.



1.4 Wnioski

1.4.1 Czy układ działa poprawnie?

Przedstawione przeze mnie rozumowanie pozwoliło mi dojść do poprawnego wyniku końcowego, którym jest poprawnie zbudowany układ sumujący dowolne dwie liczby trzybitowe. Przedstawione powyżej zdjęcia układów pokazują przykładowe dodawanie pewnych par liczb. Wyniki są zgodne z założeniami więc stwierdzam, iż układ jest prawidłowy.

1.4.2 Co można było zrobić inaczej?

- zbudować układ nieuwzględniający nadmiarowego czwartego bitu.
- sprawdzić wyniki dla większej ilości danych.
- zautomatyzować testowanie poprzez dodanie gotowego sumatora liczb trzybitowych dla porównania wyników.
- dodać Logic Analyzer do układu.

1.4.3 Gdzie to można zastosować?

- sumatory liczb o innej liczbie bitów, wystarczy odpowiednio manipulować liczbą pełnych sumatorów.
- układy mnożące liczby z przeniesieniem.
- jednostka arytmetyczno-logiczna (ALU).
- sumatory posiadają zastosowanie w procesorach i kartach graficznych, gdzie pozwalają redukować złożoność układów.
- obliczanie adresów przez procesory.

2 Bazując na dowolnych bramkach logicznych, proszę od podstaw zaprojektować, zbudować i przetestować układ sprawdzający równość dwóch dowolnych liczb czterobitowych.

2.1 Równość dwóch bitów

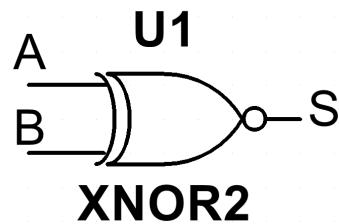
W celu sprawdzenia równości dwóch liczb czterobitowych musimy na początku znaleźć sposób na porównanie dwóch bitów i ocenienia ich równości. Nasz problem można opisać ukazaną poniżej tabelą prawdy:

A	B	S
0	0	1
0	1	0
1	0	0
1	1	1

Nie trudno zauważyc, że nasza tabela jest negacją $XOR'a$, którego tabelę widać poniżej:

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Wniosek z powyższych tabelek jest taki, że równość dwóch bitów możemy łatwo sprawdzić przy pomocy zanegowanego $XOR'a$.

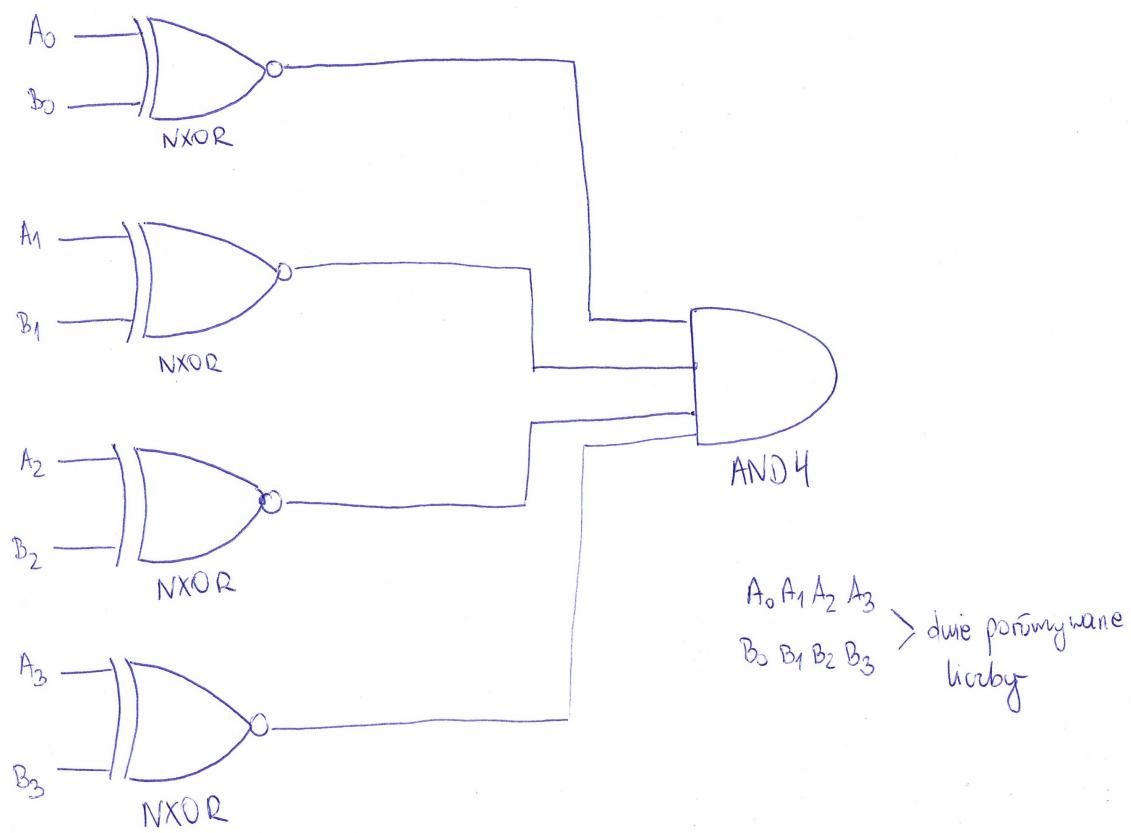


2.2 Równość dwóch liczb czterobitowych

Wiedząc jak sprawdzić równość dwóch dowolnych bitów możemy łatwo przejść do operacji porównania dwóch liczb czterobitowych. Naszym planem jest porównanie każdego z 4 bitów liczby pierwszej z odpowiednim bitem liczby drugiej. Tak więc będziemy porównywać pierwszy bit liczby pierwszej z pierwszym bitem liczby drugiej, drugi bit liczby pierwszej z drugim bitem liczby drugiej... . Oczywistym faktem jest, że nasze dwie liczby będą równe jeżeli każdy ich bit jest równy co równoważne jest zwróceniem 1 w każdym z porównań.

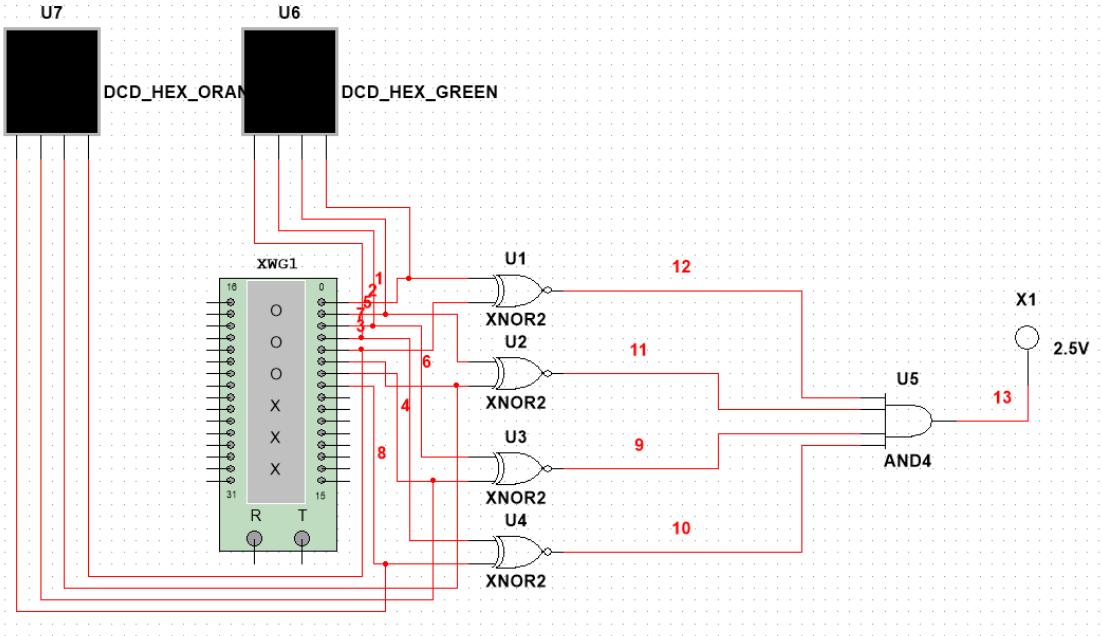
Input A	Input B	Input C	Input D	Output Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Jak widać nasza tabela jest rozwinięciem tabeli AND na cztery bity. Wystarczy więc złączyć wyniki naszych czterech porównań i przesłać je na wejście AND dla 4 bitów. Taki zabieg wystarcza by móc skutecznie porównać równość dwóch liczb czterobitowych. Poniżej widoczne jest zdjęcie naszkicowanego układu:



2.3 Układ w Multisimie

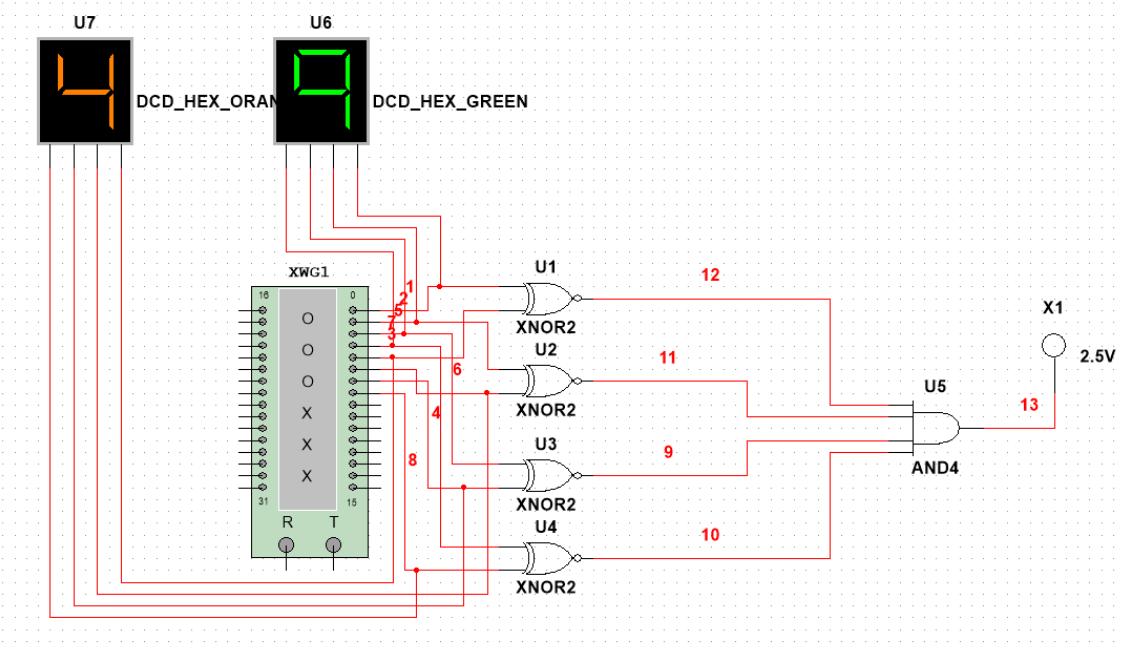
Kolejnym krokiem jest przelanie naszego pomysłu na odpowiedni układ w Multisimie w celu przetestowania poprawności jego działania. Poniżej widoczny zbudowany przeze mnie układ:

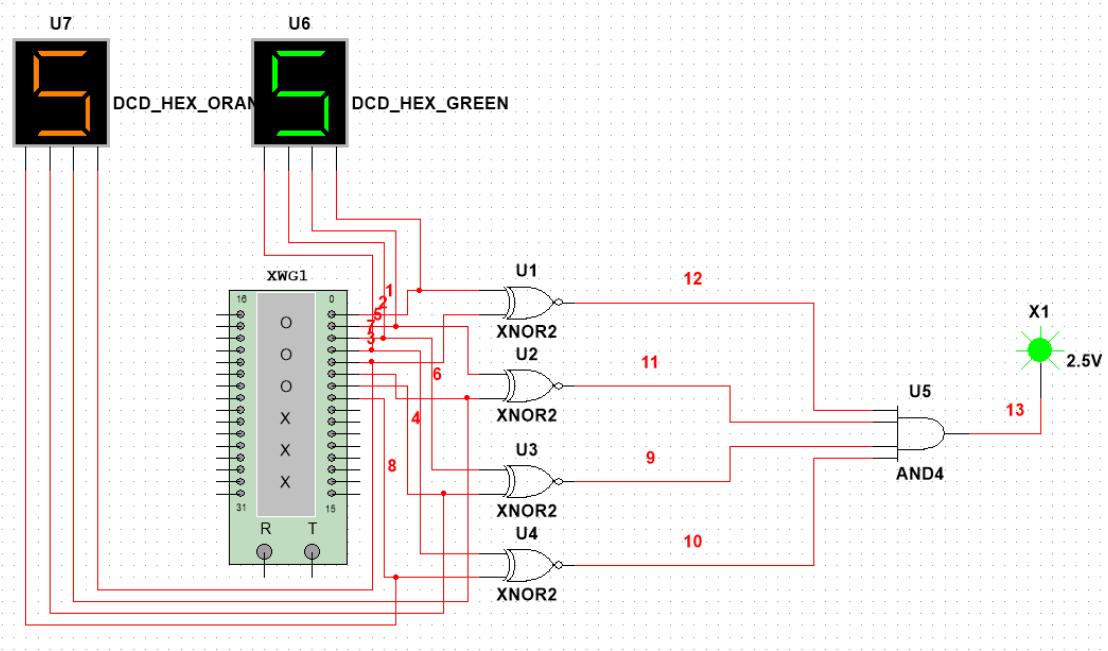


Word Generator generuje wszystkie liczby binarne ośmiobitowe (Up Counter \rightarrow 256)

2.4 Testowanie

Po przetestowaniu układu stwierdzam, iż jest on wykonany poprawnie. Poniżej pokazuje przykładowe wyniki dla liczb różnych oraz równych:





2.5 Wnioski

2.5.1 Czy układ działa poprawnie?

Przedstawione przeze mnie rozumowanie pozwoliło mi dojść do poprawnego wyniku końcowego, którym jest poprawnie zbudowany układ sprawdzający równość dwóch liczb czterobitowych. Przedstawione powyżej zdjęcia układów pokazują kilka przykładów porównywania dwóch liczb czterobitowych. Na podstawie przeprowadzonych testów stwierdzam, iż układ jest poprawny.

2.5.2 Co można było zrobić inaczej?

- rozbudowanie układu w celu porównania (większa/mniejsza) liczb zamiast samego sprawdzania równości.
- sprawdzić wyniki dla większej ilości danych.
- zautomatyzować testowanie poprzez dodanie gotowego układu porównującego dwie liczby czterobitowe dla porównania wyników.
- dodać Logic Analyzer do układu.

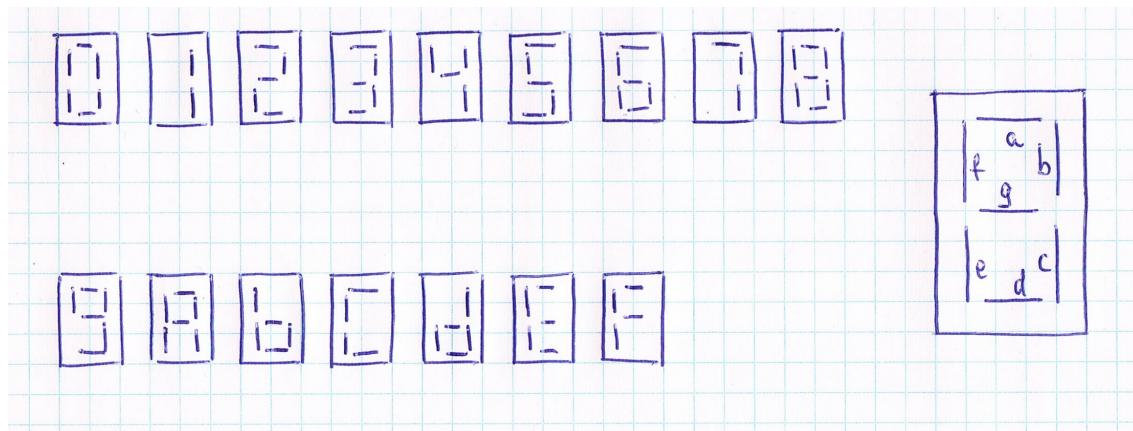
2.5.3 Gdzie to można zastosować?

- w układach porównujących liczby o innej ilości bitów poprzez manipulacje ilością bramek XNOR
- komparatory używane są w procesorach oraz mikrokontrolerach.
- w układach kontrolujących wartości pewnych danych takich jak temperatura, lokalizacja itp.
- weryfikacja haseł.

- 3 Korzystając z tablic Karnaugh'a, zaprojektować transkoder czterobitowych cyfr szesnastkowych (od 0 do F) na pojedynczy wyświetlacz siedmiosegmentowy. Następnie proszę zbudować i przetestować układ w programie Multisim. Proszę maksymalnie zminimalizować ilość użytych bramek.**

3.1 Po co tablice Karnaugh'a?

Pierwszym krokiem w celu zaprojektowania transkodera czterobitowych cyfr szesnastkowych na pojedynczy wyświetlacz siedmiosegmentowy jest ustalenie wyglądu każdej cyfry oraz przypisanie jakiejś nazwy do każdego segmentu wyświetlacza (ja nazwałem małymi literami alfabetu). Przyjęta przeze mnie symbolika widoczna jest na zdjęciu poniżej:



Dla tak przyjętej symboliki "Tabela prawdy" przyjmuje następującą postać:

(H) D

	Bin	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1
(A)10	1010	1	1	1	0	1	1	1
(B)11	1011	0	0	1	1	1	1	1
(C)12	1100	1	0	0	1	1	1	0
(D)13	1101	0	1	1	1	1	0	1
(E)14	1110	1	0	0	1	1	1	1
(F)15	1111	1	0	0	0	1	1	1

Znalezienie odpowiedniej funkcji logicznej pozwalającej na opisanie wyżej pokazanej tabeli na pierwszy rzut oka jest orzechem naprawdę twardym do zgryzienia. Zadanie staje się o wiele prostsze, gdy do pomocy użyjemy tablic Karnaugh'a.

3.2 Tablice Karnaugh'a

Metoda Karnaugh'a to sposób minimalizacji funkcji logicznych. Polega ona na zapisaniu zmiennych do specjalnej mapy Karnaugh'a, gdzie część zmiennych przypisana jest do wierszy, a część do kolumn. Dzięki numerowaniu przy pomocy kodu Graya można łatwo grupować sąsiednie pola różniące się tylko jedną zmienną. Grupować można logiczne 1 lub logiczne 0. Pierwszy sposób daje nam wynik jako sumę produktów, drugi zaś jako produkt sum. W celu minimalizacji ilości bramek logicznych wykonałem obydwa sposoby grupowania, dzięki czemu mogłem wybrać wynik potrzebujący mniejszą ilość bramek logicznych. Poniżej widoczne są wszystkie obliczenia:

		Segment a	cd		
		00	01	11	10
ab	00	1	0	1 1	1
	01	0	1	1 1	1
	11	1	0	1 1	1
	10	1	1	0	1

$f(a,b,c,d) = \bar{a}bd + bc + \bar{a}\bar{b}\bar{c} + \bar{a}c + \bar{a}\bar{d} + \bar{b}\bar{d}$

		Segment a	cd		
		00	01	11	10
ab	00	1	0	1	1
	01	0	1	1	1
	11	1	0	1	1
	10	1	1	0	1

$f(a,b,c,d) = (\bar{a}+b+\bar{c}+\bar{d})(\bar{a}+\bar{b}+c+\bar{d})$

$(a+\bar{b}+c+d)(a+b+c+\bar{d})$

Segment b

		cd			
		00	01	11	10
ab	00	1	1	1	1
	01	1	0	1	0
	11	0	1	0	0
	10	1	1	0	1

$$f(a,b,c,d) = \bar{a}cd + \bar{b}\bar{d} + \bar{a}\bar{c}\bar{d} + \bar{a}\bar{b} + a\bar{c}\bar{d}$$

Segment b

		cd			
		00	01	11	10
ab	00	1	1	1	1
	01	1	0	1	0
	11	0	1	0	0
	10	1	1	0	1

$$f(a,b,c,d) = (\bar{a} + \bar{b} + d)(\bar{a} + \bar{c} + \bar{d}) \\ (\bar{b} + \bar{c} + d)(a + \bar{b} + c + \bar{d})$$

Segment c

		cd			
		00	01	11	10
ab	00	1	1	1	0
	01	1	1	1	1
	11	0	1	0	0
	10	1	1	1	1

$$f(a,b,c,d) = ab + \bar{a}b + \bar{a}\bar{c} + \bar{a}d + \bar{c}\bar{d}$$

Segment c

cd

	00	01	11	10
00	1	1	1	0
ab	01	1	1	1
11	0	1	0	0
10	1	1	1	1

$$f(a,b,c,d) = (\bar{a} + \bar{b} + \bar{c})(\bar{a} + \bar{b} + d)(a + b + \bar{c} + d)$$

Segment d

cd

	00	01	11	10
00	1	0	1	1
ab	01	0	1	0
11	1	1	0	1
10	1	1	1	0

$$f(a,b,c,d) = a\bar{c} + b\bar{c}d + bcd + \bar{b}cd + \bar{a}\bar{b}d$$

Segment d

cd

	00	01	11	10
00	1	0	1	1
ab	01	0	1	0
11	1	1	0	1
10	1	1	1	0

$$f(a,b,c,d) = (\bar{b} + \bar{c} + d)(\bar{a} + b + \bar{c} + d)$$

$$(a + \bar{b} + c + d)(a + b + c + \bar{d})$$

Segment e cd

	00	01	11	10
ab	1	0	0	1
00	1	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	0	1	1

$$f(a,b,c,d) = cd + ac + ab + \bar{b}d$$

Segment e cd

	00	01	11	10
ab	1	0	0	1
00	1	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	0	1	1

$$f(a,b,c,d) = (a+d)(a+b+c)(b+c+d)$$

Segment f

	00	01	11	10
ab	1	0	0	0
00	1	1	0	1
01	1	1	0	1
11	1	0	1	1
10	1	1	1	1

$$f(a,b,c,d) = \bar{cd} + \bar{ab} + ac + \bar{a}\bar{b}\bar{c} + b\bar{d}$$

Segment f cd

	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	1	0	1	1
10	1	1	1	1

$$f(a,b,c,d) = (\bar{a} + \bar{b} + c + \bar{d})(a + b + d)$$

$$\quad \quad \quad (a + \bar{c} + \bar{d})(a + b + \bar{c})$$

Segment g cd

	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

$$f(a,b,c,d) = \bar{c}\bar{d} + \bar{a}\bar{b} + \bar{b}c + \bar{a}b\bar{c} + ad$$

Segment g cd

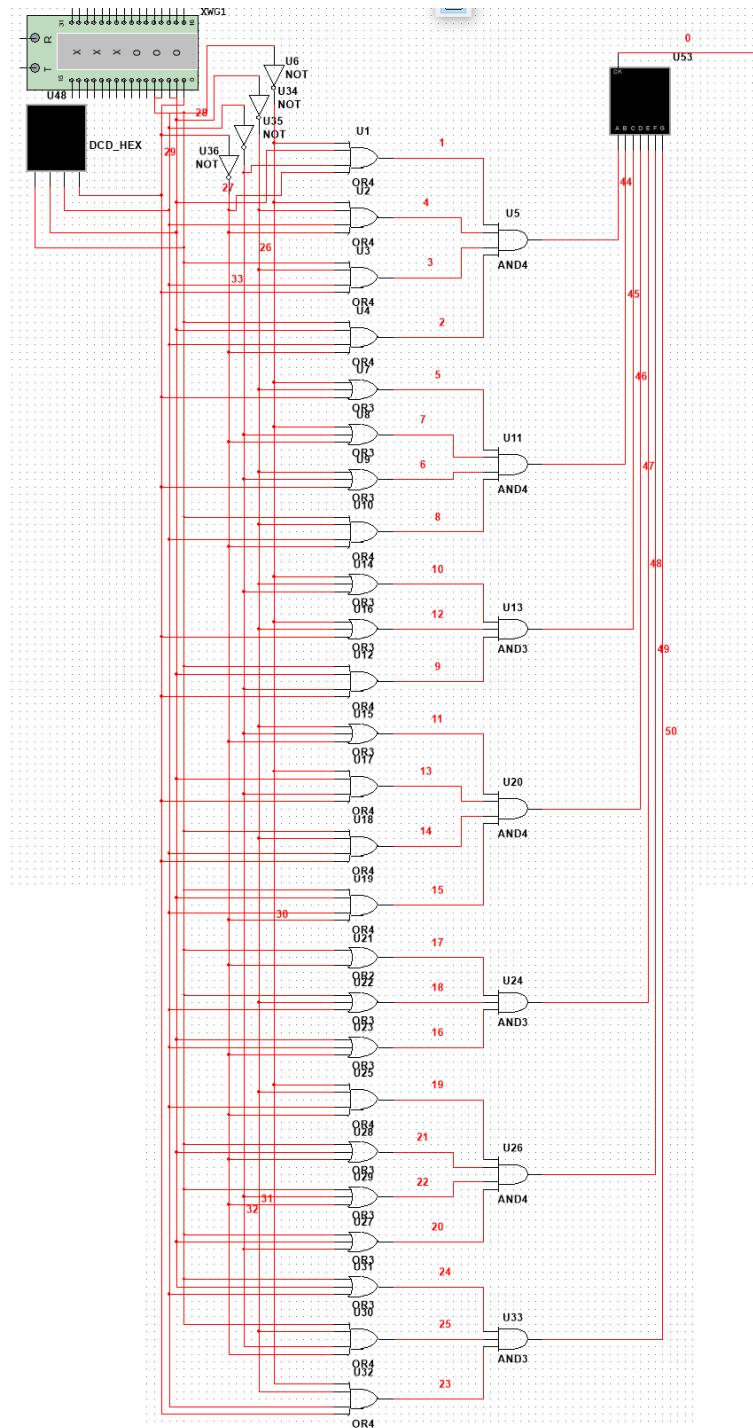
	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

$$f(a,b,c,d) = (a+b+c)(a+\bar{b}+\bar{c}+\bar{d})(\bar{a}+\bar{b}+c+d)$$

Jak widać, w każdym z możliwych przypadków produkt sum wymaga mniej bramek logicznych w jego implementacji. Takie układy zastosuje też w moim projekcie.

3.3 Układ w Multisimie

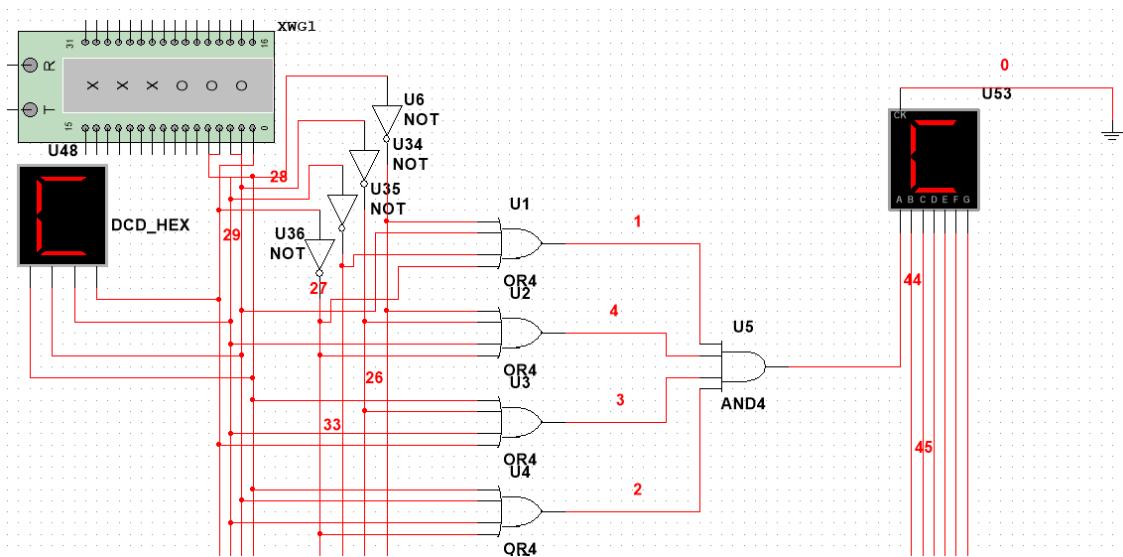
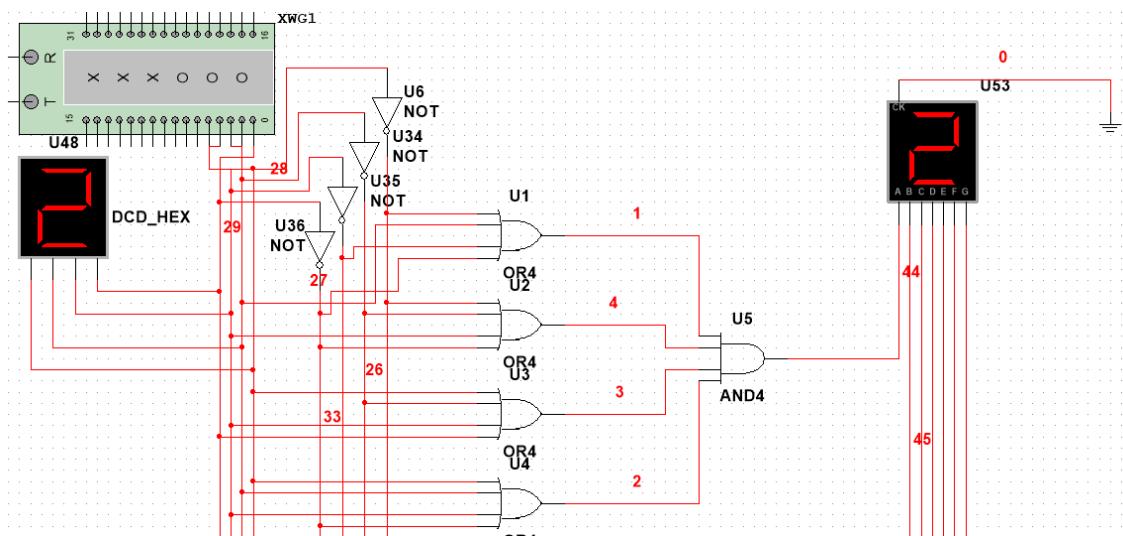
Kolejnym krokiem jest przelanie naszego pomysłu na odpowiedni układ w Multisimie w celu przetestowania poprawności jego działania. Poniżej widoczny zbudowany przeze mnie układ:



Word Generator generuje wszystkie liczby binarne czterobitowe (Up Counter → 16)

3.4 Testowanie

Po przetestowaniu układu stwierdzam, iż jest on wykonany poprawnie. Poniżej pokazuje przykładowe wyniki:



3.5 Wnioski

3.5.1 Czy układ działa poprawnie?

Przedstawione przeze mnie rozumowanie pozwoliło mi dojść do poprawnego wyniku końcowego, którym jest poprawnie zbudowany układ transkodera czterobitowych cyfr szesnastkowych na wyświetlacz siedmiosegmentowy. Przedstawione powyżej zdjęcia układów pokazują kilka przykładów transkodowania cyfr na wyświetlacz. Na podstawie przeprowadzonych testów stwierdzam, iż układ jest poprawny.

3.5.2 Co można było zrobić inaczej?

- być może istniała możliwość zmniejszenia ilości użytych bramek logicznych.
- zbudowanie bardziej przejrzystego układu.
- zastosowanie sum produktów w bramkach zamiast produktu sum (większa ilość bramek).
- zaplanować inaczej wygląd poszczególnych cyfr szesnastkowych.

3.5.3 Gdzie to można zastosować?

- wyświetlacze mikrofalówek, kalkulatorów, radia, pralek, zegarków
- wyświetlacze o większej ilości segmentów działają na bardzo podobnej zasadzie
- wyświetlacze takie tracą powoli zastosowanie, ponieważ są wypierane przez bardziej zaawansowane technologicznie odpowiedniki