Technika Cyfrowa
Sprawozdanie - bramki logiczne

Maciej Trątnowiecki

AGH, Semestr Letni, 2020

1 Układ sumujący dwie liczby trzybitowe

1.1 Realizowana funkcja logiczna

Układ ma trzy wejścia, ABC odpowiadające za bity pierwszej z liczb i DEF odpowiadające za bity drugiej, oraz cztery wyjścia GHIJ. Gdy sumujemy pojedyńczą parę bitów, możemy otrzymać cztery sytuacje opisane poniższą tabelą prawdy.

C	F	J
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 1: Tabela prawdy dla sumy dwóch bitów.

W ostatniej sytuacji, gdy oba sumowane bity miały stan wysoki otrzymaliśmy zero. Jednak powinniśmy otrzymać dwa w zapisie dziesiętnym, to jest 10 w zapisie binarnym. Dzieje się tak dlatego, że w tabeli przewidzieliśmy zaledwie pojedyncze wyjście. Nastąpiło tak zwane przepełnienie, to znaczy otrzymana liczba nie daje się reprezentować w przewidzianym zakresie bitów. Zostanie on zatem rozszerzony do dwóch bitów.

C	F	IJ
0	0	00
0	1	01
1	0	01
1	1	10

Tabela 2: Tabela prawdy dla sumy dwóch bitów.

W powyższej tabeli wyjście J odpowiada za wynik sumowania, a wyjście I za informację o przepełnieniu. Możemy ich wartości opisać poniższymi tabelami.

C / F	0	1
0	0	1
1	1	0

Tabela 3: Tabela prawdy dla wyjścia J (wynik dodawania).

C / F	0	1
0	0	0
1	0	1

Tabela 4: Tabela prawdy dla wyjścia I (informacja o przepełnieniu).

Z powyższych tabel odczytać możemy zależności opisujące stany na tych wyjściach.

$$I = FC$$

$$J = F\overline{C} + \overline{F}C = F \oplus C$$

Sumując liczby dwubitowe, będziemy musieli rozpatrzyć wynik z poprzedniej pary bitów. Oznacza to, że będziemy musieli rozpatrzyć trzy bity na wejściu. Tabelę prawdy takiej operacji przedstawiono poniżej. Małym i oznaczono przepełnienie z sumy poprzedniej pary bitów.

B	E	i	HI
0	0	0	00
0	0	1	01
0	1	0	01
0	1	1	10
1	0	0	01
1	0	1	10
1	1	0	10
1	1	1	11

Tabela 5: Tabela prawdy dla sumy dwóch bitów.

Dla takiej sumy również zapiszemy zależności konkretnych wyjść.

B / Ei	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Tabela 6: Tabela prawdy dla wyjścia I (wynik dodawania).

B / Ei	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Tabela 7: Tabela prawdy dla wyjścia H (informacja o przepełnieniu).

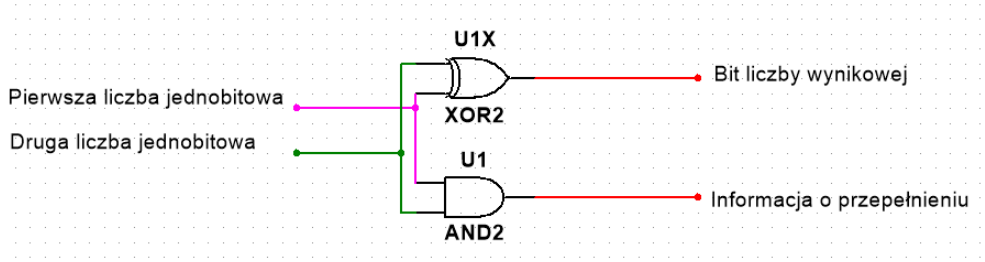
Skąd otrzymujemy poniższe zależności.

$$I = B\overline{E}\overline{i} + \overline{B}E\overline{i} + BEi + \overline{B}E\overline{i} = B(\overline{E}\overline{i} + Ei) + \overline{B}(\overline{E}i + E\overline{i}) = \overline{B}(E \oplus i) + B(\overline{(E \oplus i)}) = B \oplus (E \oplus i)$$

$$H = Ei + B(E + i) = Ei + BE + Bi$$

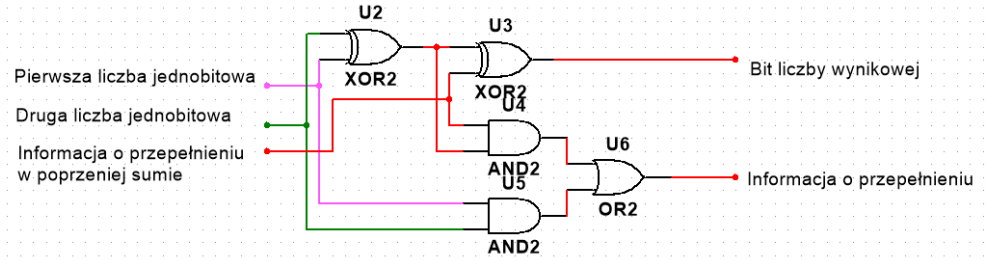
1.2 Sumator pełny i pół-pełny

Dla sumy najmniej ważnych bitów na wejściu sumatora otrzymujemy dwa sygnały - po jednym bicie od każdej z liczb. Jednakże, wynik dodawania dwóch liczb jednobitowych jest liczbą dwubitową. Dlatego, na wyjściu podajemy dwa sygnały - stan najmniej ważnego bitu liczby wynikowej i informacje o tym, czy wystąpiło przepełnienie.
Taki układ nazywamy sumatorem pół-pełnym, jednobitowym. Do jego wykonania wykorzystano dwie bramki logiczne - xor i and, do których podłączymy sygnały wejściowe. Wyjście bramki xor będzie stanem bitu w wyniku, a wyjście bramki and informacją o przepełnieniu.



Rysunek 1: Sumator pół-pełny

Podczas sumowania kolejnych bitów będziemy musieli rozpatrzyć trzy sygnały wejściowe - dwa jak poprzednio informujące o wartościach bitów w dodawanej liczbie, a trzeci informujący czy w poprzedniej operacji dodawania wystąpiło przepełnienie. Na wyjściu ponownie znajdą się dwa sygnały - jak w sumatorze pół-pełnym. Układ realizujący taką sumę nazywamy sumatorem pełnym.

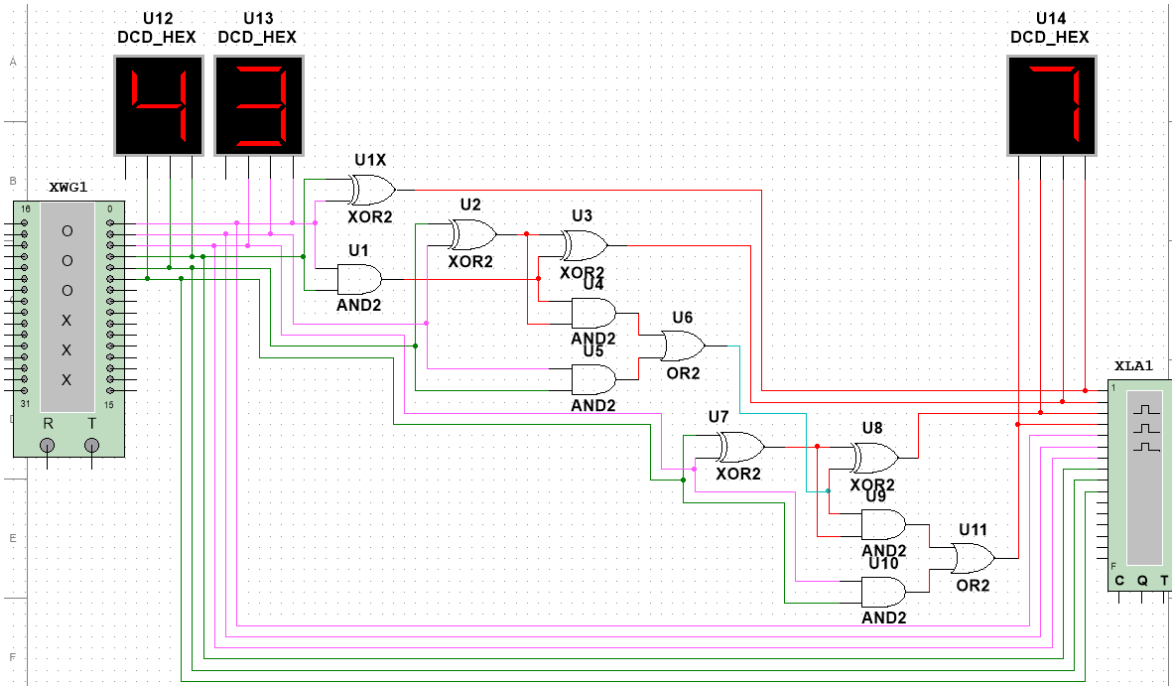


Rysunek 2: Sumator pełny

1.3 Konstrukcja układu sumującego

Zauważmy, że tak skonstruowane sumatory pełne można dowolnie długo łączyć, otrzymując na ich wyjściach poprawną sumę arytmetyczną liczb przekazanych na wejścia. W celu ograniczenia ilości użytych bramek logicznych do wykonania sumy najmniej znaczących bitów używamy sumatora pół-pełnego.
Aby wykonać sumę dwóch liczb trzybitowych potrzeba dwóch sumatorów pełnych i jednego pół-pełnego.

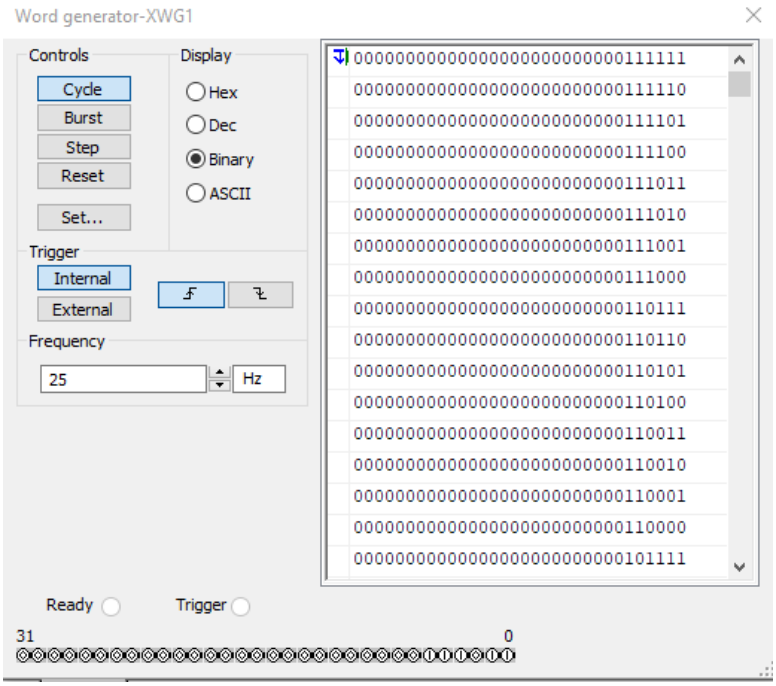
1.4 Projekt układu

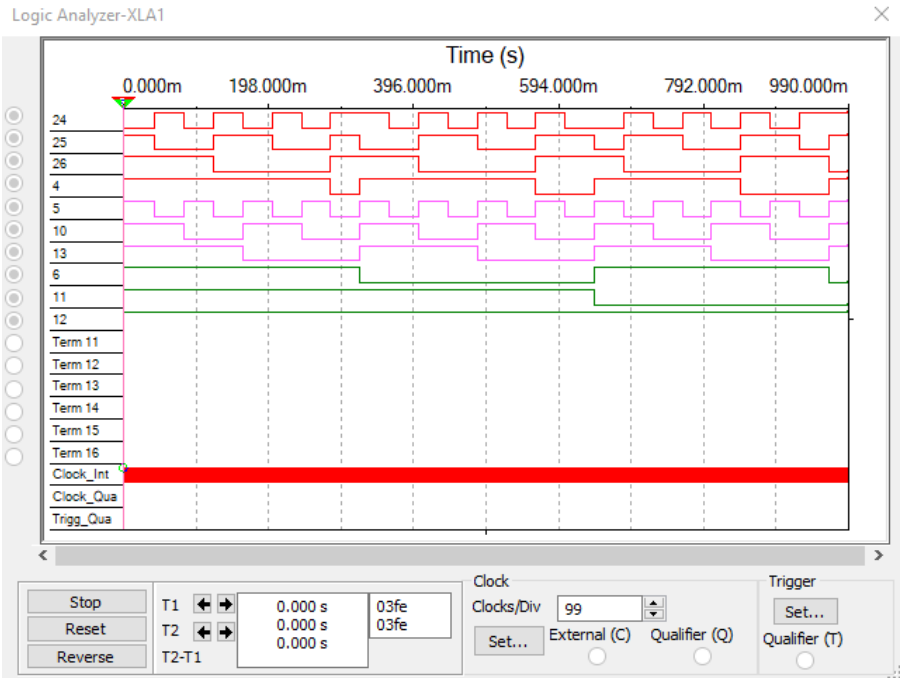


W układzie jako źródło sygnału wykorzystano generator słów. Wyjścia odpowiadające za bity pierwszej z liczb oznaczono kolorem zielonym, a za bity drugiej z nich kolorem różowym.
Do obliczenia sumy arytmetycznej wykorzystano jeden sumator pół-pełny (dla sumy najmniej ważnych bitów), oraz dwa sumatory pełne. Wyjścia odpowiadające za przepełnienie oznaczono kolorem lazurowym.
Wynik operacji zilustrowano poprzez wykorzystanie wyświetlacza HEX.

1.5 Symulacja

W celu przetestowania układu dla wszystkich par liczba trzybitowych generator ustawiony został w tryb generowania wszystkich liczb sześciobitowych od 111111 do 000000. Trzy najmniej znaczące bity odpowiadają za jedną z generowanych liczb, trzy najmniej znaczące za drugą.





Z ekranu analizatora logicznego odczytać możemy wynikową liczbę czterobitową, oraz trzybitowe liczby wejściowe od których jest zależna. Na tej podstawie możemy stwierdzić poprawność implementacji.

1.6 Wnioski

Zauważmy, że w przedstawiony powyżej sposób możemy niskim kosztem skonstruować sumator dla dwóch liczb o dowolnej (choć z góry ustalonej) liczby bitów. Taki układ może znaleźć zastosowanie w wielu sytuacjach codziennych. Dla przykładu, w układzie sterowania moją pralką. Na tryb prania składa się wiele etapów, jak moczenie, pranie, czy wirowanie. Długość każdego z tych etapów zależy od wybranych parametrów (jak temperatura wody, czy ocena stopnia zabrudzenia ubrań). Na wyświetlaczu pokazuje ona sumaryczny czas prania, będący w istocie sumą po czasach trwania każdego z tych etapów. Możemy zatem zastosować przygotowany powyżej sumator do obliczenia liczby przekazywanej na wejścia dekodera wyświetlacza.

2 Układ sprawdzający równość dwóch liczb

2.1 Konstrukcja układu

Układ komparatora dwóch liczb czterobitowych ma cztery wejścia, ABCD odpowiadające za bity pierwszej z liczb i EFGH odpowiadające za bity drugiej. Chcemy zrealizować funkcję logiczną

$$(A == E) \wedge (B == F) \wedge (C == G) \wedge (D == H)$$

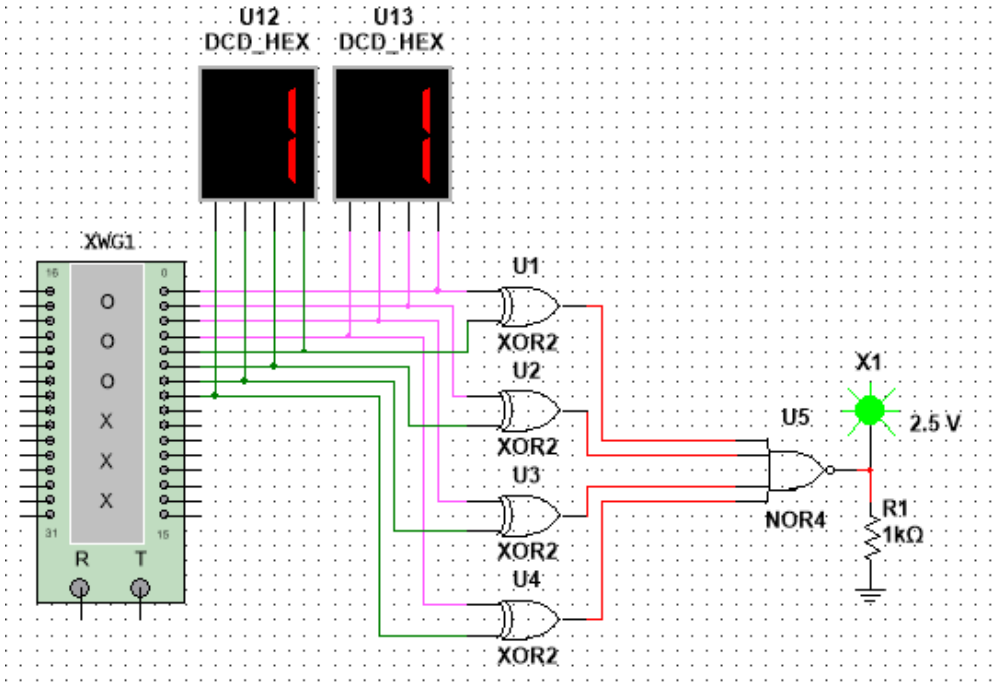
Co jest równoważne z

$$(\neg(A \oplus E)) \wedge (\neg(B \oplus F)) \wedge (\neg(C \oplus G)) \wedge (\neg(D \oplus H))$$

$$\neg((A \oplus E) \vee (B \oplus F) \vee (C \oplus G) \vee (D \oplus H))$$

Zatem w celu wykazania ich równości, wystarczy zastosować bramki XOR w ilości odpowiadającej liczbie bitów, których wyjścia zestawić należy bramką NOR.

2.2 Projekt układu



W układzie jako źródło sygnału wykorzystano generator słów. Wyjścia odpowiadające za bity pierwszej z liczb oznaczono kolorem zielonym, a za bity drugiej z nich kolorem różowym.
Do porównania wartości liczb wykorzystano bramki xor, których wyniki zbiera bramka nor.
Równość liczb sygnalizuje zielona dioda.

2.3 Wnioski

Zauważmy, że w przedstawiony powyżej sposób możemy skonstruować układ sprawdzający równość dwóch liczb o dowolnej (choć z góry ustalonej) liczbie bitów. Domofon wejściowy w bloku w którym mieszkam można otworzyć wprowadzając specjalny kod mający postać liczby naturalnej. Za porównanie go z przewidzianą w układzie sekwencją bitów odpowiadać może układ podobny do przedstawionego powyżej.

3 Transkoder czterobitowych cyfr szesnastkowych na wyświetlacz siedmiosegmentowy

3.1 Minimalizacja funkcji boolowskich

Wyświetlaczem sterować można poprzez 7 wyjść, ponumerowanych od A do G, każde z nich odpowiada za inny segment wyświetlacza. W poniższej tabeli zebrano opis ustawień wyjść odpowiedzialnych za wyświetlanie każdej z liczb czterobitowych.

Liczba w systemie szesnastkowym	Liczba w systemie binarnym	Wyjścia wyświetlacza siedmiosegmentowego						
		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	1	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	0	0	1	1
A	1010	1	1	1	0	1	1	1
B	1011	0	0	1	1	1	1	1
C	1100	1	0	0	1	1	1	0
D	1101	0	1	1	1	1	0	1
E	1110	1	0	0	1	1	1	1
F	1111	1	0	0	0	1	1	1

Tabela 6: Wyjścia wyświetlacza

W celu uproszczenia układu wykonać należy minimalizację metodą Karnaugh dla każdego z wyjść wyświetlacza. Niech x_1, x_2, x_4, x_8 określają kolejne bity, czterobitowej liczby x w kolejności od najmniej do najbardziej znaczącego bitu. Wtedy:

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	0	1	1
	01	0	1	0	1
	11	1	1	1	0
	10	1	1	1	1

Tabela 2: Wyjście A

Zauważmy, że wartości zera logicznego w powyższej tabeli jest zdecydowanie mniej, niż stanów wysokich. Dlatego to po nich będziemy grupować, zatem otrzymamy wynik poprzez dopełnienie.
Każdą z grup zer możemy opisać poprzez określenie w postaci alternatywy logicznej wszystkich pozostałych wartości w tablicy, za wyjątkiem tej grupy. Taką grupę konstruujemy poprzez negację logiczną. Jeśli na tak przygotowanych grupach użyjemy koniunkcji logicznej otrzymamy formułę opisującą poprawnie rozkład stanów wysokich i niskich względem sygnału wejściowego dla danego wejścia wyświetlacza.
Skąd otrzymujemy (przez dopełnienie):

$$A = (\overline{x_1} + x_2 + x_4 + x_8)(x_1 + x_2 + \overline{x_4} + x_8)(\overline{x_1} + x_2 + \overline{x_4} + \overline{x_8})(\overline{x_1} + \overline{x_2} + x_4 + \overline{x_8})$$

Analogiczną analizę przeprowadzimy dla każdego z pozostałych wejść wyświetlacza.

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	1	0	1
	01	1	0	1	1
	11	1	1	0	0
	10	1	0	0	1

Tabela 3: Wyjście B

Skąd otrzymujemy (przez dopełnienie):

$$B = (\overline{x_1} + x_2 + \overline{x_4} + x_8)(\overline{x_1} + \overline{x_2} + \overline{x_8})(x_1 + x_2 + \overline{x_4} + \overline{x_8})(x_1 + \overline{x_2} + \overline{x_4})$$

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	1	0	1
	01	1	1	1	1
	11	1	1	0	1
	10	0	1	0	1

Tabela 4: Wyjście C

Skąd otrzymujemy (przez dopełnienie):

$$C = (x_1 + \overline{x_2} + x_4 + x_8)(\overline{x_2} + \overline{x_4} + \overline{x_8})(x_1 + x_2 + \overline{x_4} + \overline{x_8})$$

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	0	1	1
	01	0	1	1	0
	11	1	0	0	1
	10	1	1	1	0

Tabela 5: Wyjście D

Skąd otrzymujemy (przez dopełnienie):

$$D = (x_1 + x_2 + \overline{x_4} + x_8)(\overline{x_1} + x_2 + x_4)(\overline{x_1} + \overline{x_2} + \overline{x_4})(x_1 + \overline{x_2} + x_4 + \overline{x_8})$$

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	0	1	1
	01	0	0	1	0
	11	0	0	1	1
	10	1	1	1	1

Tabela 6: Wyjście E

Skąd otrzymujemy (przez dopełnienie):

$$E = (\overline{x_1} + x_8)(x_1 + x_2 + \overline{x_4} + x_8)(\overline{x_1} + x_2 + x_4 + \overline{x_8})$$

		x_8x_4			
		00	01	11	10
x_2x_1	00	1	1	1	1
	01	0	1	0	1
	11	0	0	1	1
	10	0	1	1	1

Tabela 7: Wyjście F

Skąd otrzymujemy (przez dopełnienie):

$$F = (\overline{x_1} + x_2 + \overline{x_4} + \overline{x_8})(\overline{x_1} + \overline{x_2} + \overline{x_4} + x_8)(\overline{x_1} + x_4 + x_8)(x_1 + \overline{x_2} + x_4 + x_8)$$

		x_8x_4			
		00	01	11	10
x_2x_1	00	0	1	0	1
	01	0	1	1	1
	11	1	0	1	1
	10	1	1	1	1

Tabela 8: Wyjście G

Skąd otrzymujemy (przez dopełnienie):

$$G = (x_2 + x_4 + x_8)(\overline{x_1} + \overline{x_2} + \overline{x_4} + x_8)(x_1 + x_2 + \overline{x_4} + \overline{x_8})$$

Za pomocą powyższych obliczeń wyprowadziliśmy minimalną funkcję logiczną opisującą niezależnie każde z wyjść układu dekodera w zależności od bitów liczby wejściowej. Spróbujmy poszukać zależności pomiędzy wyjściami, które pozwoliłyby nam dalej zminimalizować układ (zgodnie z intuicją, że jeśli przykładowo dwa segmenty wyświetlacza świeciłyby się zawsze w tych samych momentach, nie potrzebowalibyśmy dwóch niezależnych dekoderów dla obu z tych segmentów).

		EFG							
		000	001	010	011	100	101	110	111
BCD	000	-	-	-	-	-	-	-	1
	001	-	-	-	-	-	-	-	1
	010	-	-	-	-	-	-	-	-
	011	-	-	-	1	-	-	1	1/0
	100	-	-	-	-	-	-	-	-
	101	-	-	-	-	-	1	-	-
	110	0/1	-	-	1	-	-	-	1
	111	-	1	-	-	-	0	1	1

Tabela 7: Opis stanów wyjścia A w zależności od pozostałych wyjść dekodera

Niestety, podczas próby konstrukcji tablicy stanów opisującej stan wyjścia A w zależności od stanu pozostałych wyjść odkryliśmy, że nie istnieje deterministyczna funkcja opisująca taką zależność - znaleźliśmy dwie sytuacje, w których wyjście A przyjmuje różne wartości dla tych samych stanów pozostałych bramek. Oznacza to, że otrzymana w poprzednim punkcie zależność dla wyjścia A od bitów liczby wejściowej jest optymalna.
Dla pozostałych wyjść układu, z wyjątkiem wyjścia F, również wskazać możemy takie sytuacje konfliktowe. Dla porządku wypiszmy je poniżej.

- Dla złącza B gdy pozostałe wyjścia przyjmują wartość 1 (to jest dla liczb 1000 i 0110).
- Dla złącza C gdy złącze B przyjmuje wartość 0, a pozostałe złącza 1 (to jest dla liczb 1110, 0110).
- Dla złącza D gdy wszystkie pozostałe złącza przyjmują wartość 1 (to jest dla liczb 1010 i 1000).
- Dla złącza E gdy złącze B przyjmuje wartość 0, a pozostałe złącza 1 (to jest dla liczb 0101 i 0110).
- Dla złącza G gdy wszystkie pozostałe wyjścia przyjmują wartość 1 (to jest dla liczb 0000, 1000).

Nie udało nam się znaleźć konfliktu dla wyjścia F. Rozpiszmy zatem jego tabelę prawdy.

		DEG							
		000	001	010	011	100	101	110	111
ABC	000	-	-	-	-	-	-	-	-
	001	-	-	-	-	-	-	-	1
	010	-	-	-	-	-	-	-	-
	011	0	-	-	-	-	-	-	0
	100	-	-	-	1	-	-	1	1
	101	-	-	-	-	-	1	-	1
	110	-	-	-	-	-	-	-	0
	111	0	1	-	1	-	0	1	1

Tabela 8: Opis stanów wyjścia F w zależności od pozostałych wyjść dekodera

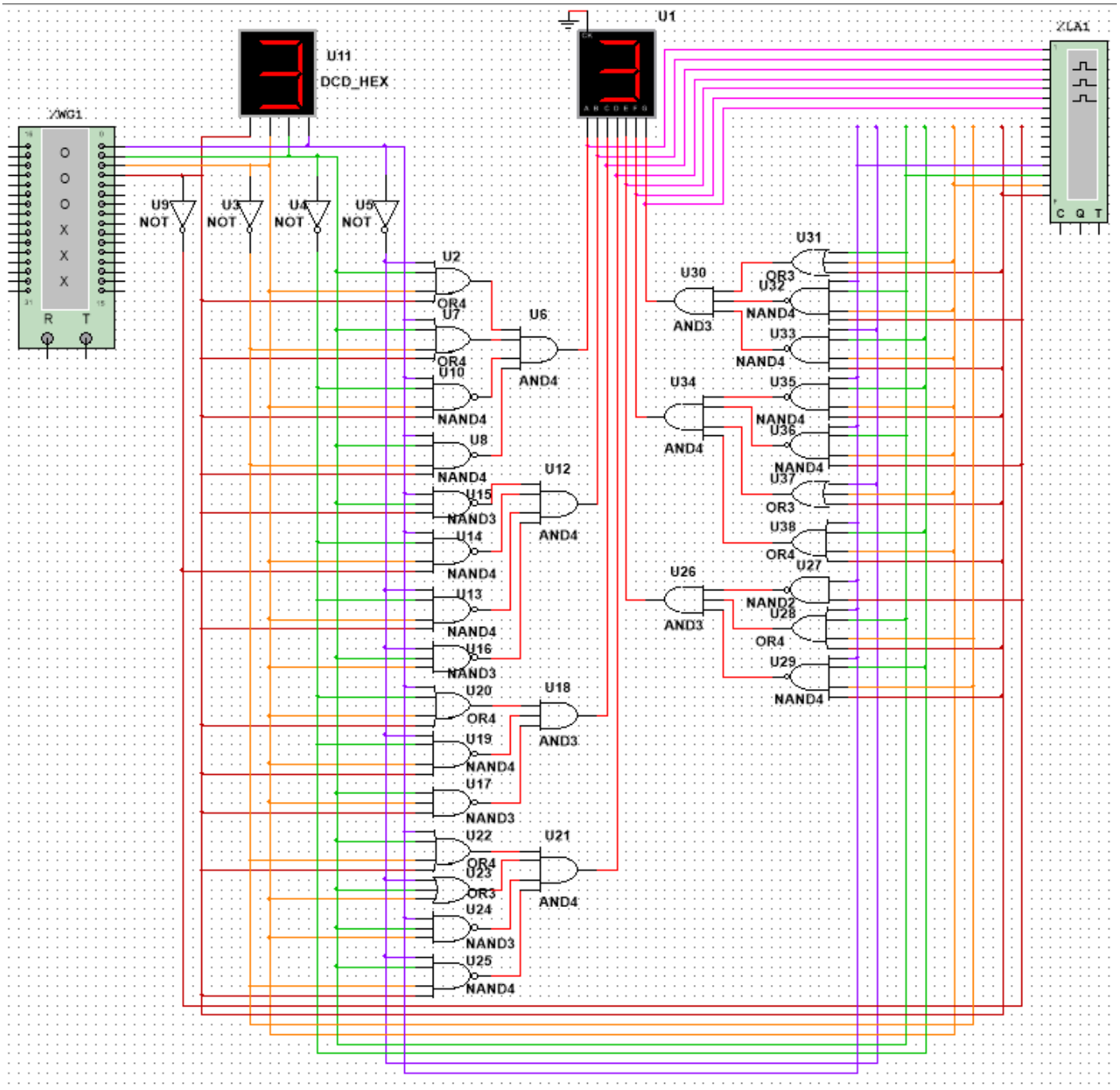
Korzystając z powyższej tabeli wyprowadzić możemy zależność:

$$F = A(G + E) + A\overline{B}D + DE(\overline{B} + ABC)$$

Uzyskaliśmy alternatywną zależność dla wyjścia F. Nie jest ona jednak realizowana za pomocą mniejszej ilości bramek logicznych, niż wskazana przez nas wcześniej zależność względem wejść układu.

Tym samym pokazaliśmy, że otrzymujemy układ zbudowany z optymalnej ilości bramek logicznych.

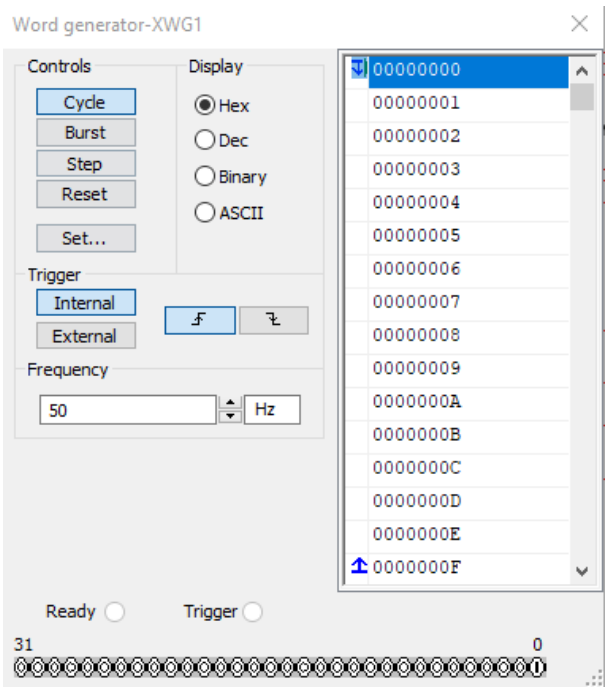
3.2 Projekt układu

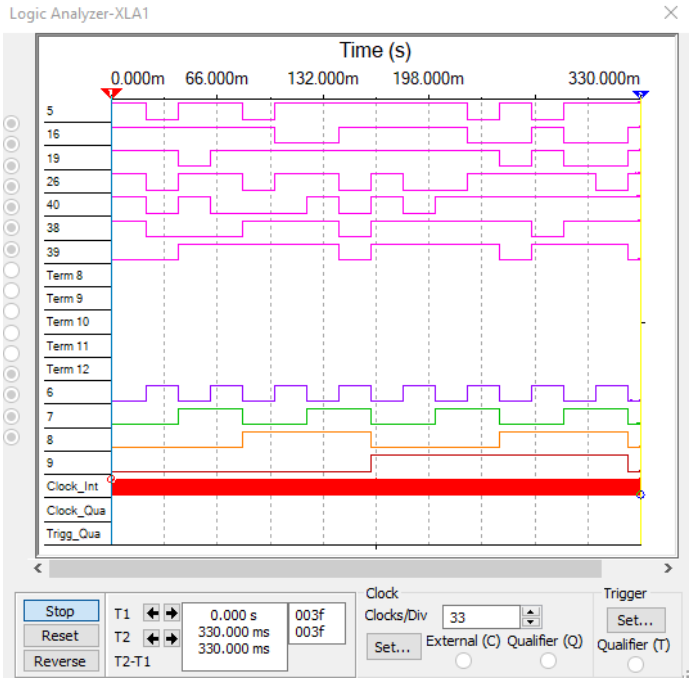


W układzie jako źródło sygnału wykorzystano generator słów. Wyjścia odpowiadające za kolejne bity, w kolejności od najmniej do najbardziej znaczącego, oznaczono odpowiednio kolorami: fioletowym, zielonym, pomarańczowym i bordowym.
Sygnał odpowiadający za kolejne wejścia wyświetlacza przetwarzany jest przy pomocy bramek logicznych, zgodnie z powyżej rozpisanymi formułami. Niektóre operacje alternatywy logicznej realizowane są poprzez użycie bramki NAND zgodnie z prawem de Morgana.
Do wejścia układu dołączono wyświetlacz DCD_Hex ze wbudowanym transkoderem dla zilustrowania poprawności działania układu.

3.3 Symulacja

Generator słów kierowany jest sygnałem wyzwalającym z wbudowanego zegara o częstotliwości 50HZ. Podaje on na wyjścia wszystkie liczby czterobitowe w cyklu.





Za pomocą analizatora logicznego, oraz podłączonego wyświetlacza siedmiosegmentowego możemy stwierdzić poprawność działania dekodera.

3.4 Wnioski

Minimalizacja funkcji logicznych metodą Karnaugha pozwala w sposób intuicyjny konstruować układy realizujące funkcje logiczne. Daje ona nam łatwość konstrukcji złożonych wyrażeń w sposób graficzny przy zapewnionej poprawności działania. W efekcie proces projektowania jest prostszy w porównaniu do próby minimalizacji klasycznego zapisu algebry booleowskiej.
Znacząco usprawnia to projektowanie układów których działanie określone jest względem wielu zmiennych logicznych za pomocą wielu funkcji logicznych. Przykładem takiego układu jest transkoder na wyświetlacz siedmiosegmentowy.
Zgodnie z prawem każdy sklep wyposażony musi być w kasę fiskalną. Taka kasa fiskalna wyświetla sumę cen wprowadzonych produktów za pomocą wyświetlacza zbudowanego z modułów siedmiosegmentowych. Każdy z takich modułów może być wyposażony w dekoder podobny do zaimplementowanego powyżej.