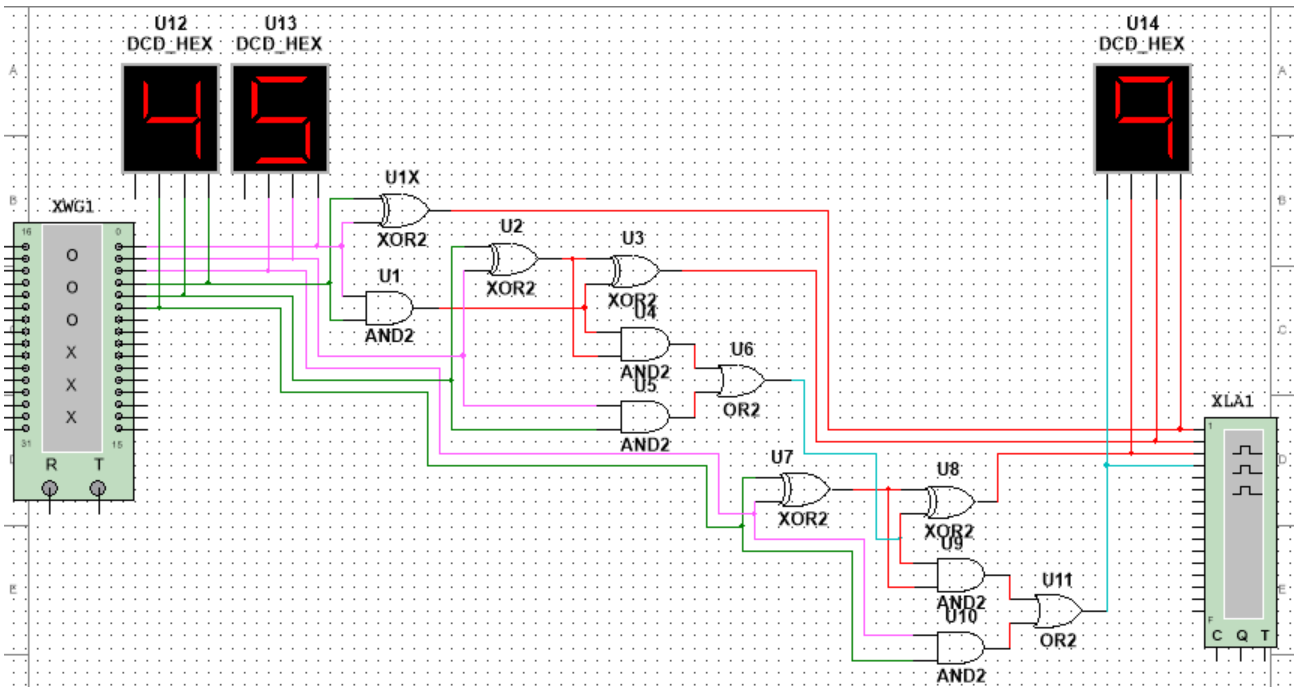
Technika Cyfrowa
Sprawozdanie - bramki logiczne

Maciej Trątnowiecki

AGH, Semestr Letni, 2020

1 Układ sumujący dwie liczby trzybitowe

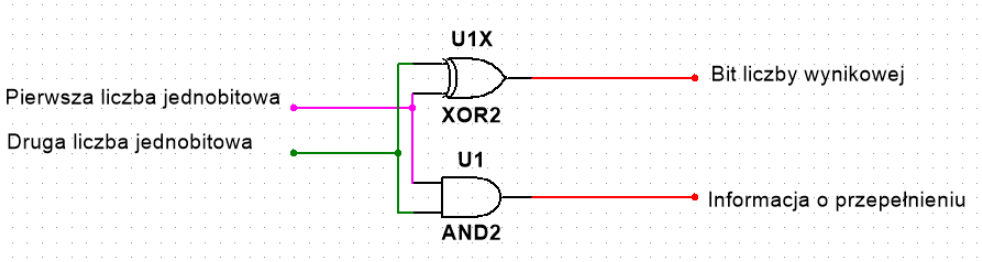
1.1 Projekt układu



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.

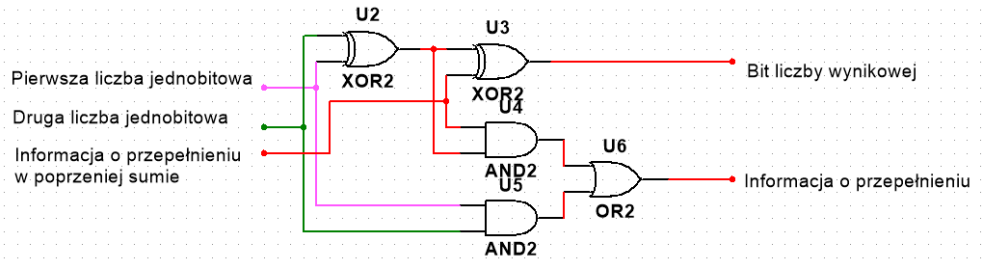
1.2 Sumator pełny i pół-pełny

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 wystarczą 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

Jednakże, w tej sytuacji 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. Do jego wykonania będziemy potrzebować aż pięciu bramek.



Rysunek 2: Sumator pełny

Podobnie jak poprzednio skierujemy wejścia pochodzące od bitów sumowanych liczb na bramkę xor oraz and. Teraz jednak musimy rozpatrzeć kilka przypadków. Jeśli oba bity mają stan wysoki (zatem stan wysoki pojawi się za bramką and), lub choć jeden z nich przyjmuje stan wysoki, a w poprzedniej sumie wystąpiło przepełnienie, to wtedy również i w tej operacji wystąpi przepełnienie. Jeśli zaś stan wysoki przyjmuje dokładnie jeden bit z wejścia, lub nie przyjmuje go żaden z bitów, za to w poprzedniej sumie wystąpiło przepełnienie, to wtedy stan wysoki przyjmuje także bit liczby wynikowej.
Układ realizujący tą prostą logikę nazywamy sumatorem pełnym.

1.3 Konstrukcja układu sumującego

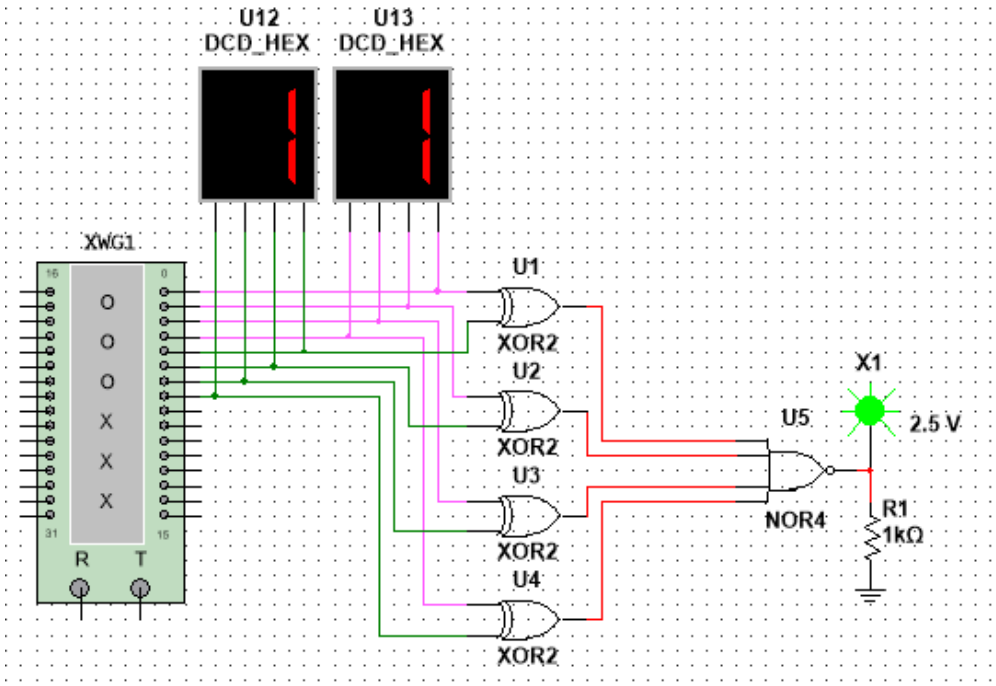
Zauważmy, że tak skonstruowane sumatory pełne możemy dowolnie długo łączyć, otrzymując na ich wyjściach poprawną sumę arytmetyczną liczb przekazanych na wejścia. W celu ograniczenia ilości użytych bramek logicznych do wykonania sumy najmniej znaczących bitów używamy sumatora pół-pełnego.
Aby wykonać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

1.4 Wnioski

Zauważmy, że w przedstawiony powyżej sposób możemy skonstruować sumator dla dwóch liczb o dowolnej (choć z góry ustalonej) liczby bitów. Z oczywistych względów (powszechność operacji dodawania) taki układ ma zastosowania praktyczne.

2 Układ sprawdzający równość dwóch liczb

2.1 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.

2.2 Konstrukcja układu

Zauważmy, że dwie liczby są równe wtedy i tylko wtedy, gdy wszystkie reprezentujące je bity, na odpowiednich pozycjach, są sobie równe. Zatem w celu wykazania ich równości, wystarczy wykorzystać bramki xor w ilości odpowiadającej liczbie bitów. Jeżeli wszystkie z bramek podadzą na wyjściu stan niski, to liczby są równe.

2.3 Wnioski

Zauważmy, że w przedstawiony powyżej sposób możemy skonstruować układ sprawdzający równość dwóch liczb o dowolnej (choć z góry ustalonej) liczby bitów. Z oczywistych względów (powszechność operacji porównania) taki układ ma zastosowania praktyczne. Dodatkowo, jest wyjątkowo prosty do skonstruowania, dzięki wysokiej dostępności bramek realizujących operację alternatywy wykluczającej.

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 1: 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. Jeśli na tak skonstruowanych opisach grup użyjemy koniunkcji logicznej otrzymamy formułę opisującą poprawnie rozkład stanów wysokich i niskich względem sygnału wejściowego dla danego wejścia wyświetlacza. Analogiczną analizę przeprowadzimy dla każdego z pozostałych wejść wyświetlacza.
Skąd otrzymujemy (przez dopełnienie):

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

		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):

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

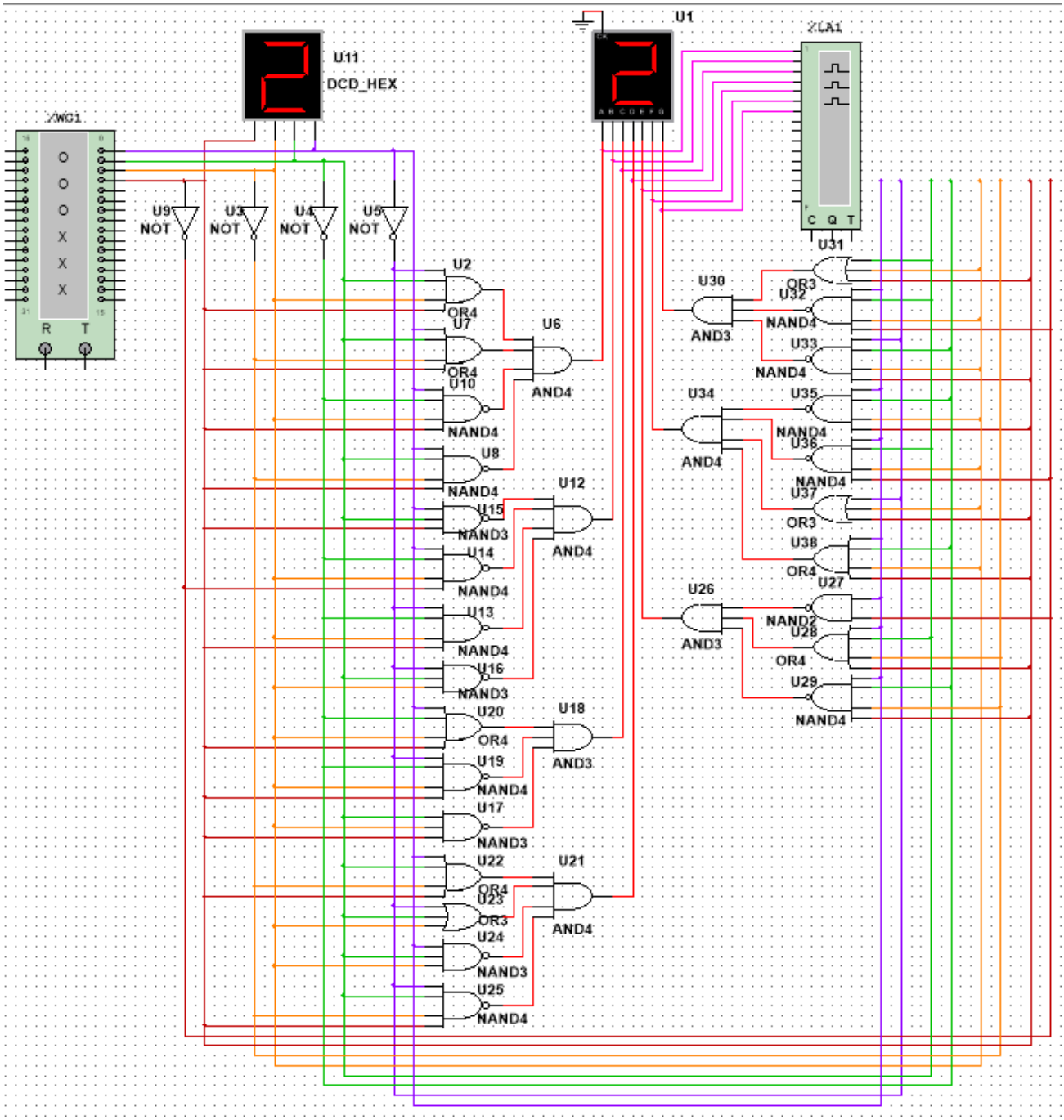
		x_8x_4			
		00	01	11	10
x_2x_1	00	0	1	0	1
	01	0	1	1	1
	11	1	0	1	1
	10	1	1	1	1

Tabela 8: Wyjście G

Skąd otrzymujemy (przez dopełnienie):

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

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 Wnioski

Minimalizacja funkcji logicznych metodą Karnaugha pozwala na konstrukcję układów realizujących funkcje logiczne w sposób intuicyjny. Daje ona nam łatwość konstrukcji złożonych wyrażeń w sposób graficzny przy zapewnionej poprawności działania. W efekcie proces projektowy 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.