Technika Cyfrowa

Sprawozdanie z ćwiczenia nr. 1

Piotr Błaszczyk, Eryk Olejarz, Krzysztof Swędzioł, Filip Zieliński

30.04.2024

1 Wprowadzenie

Celem ćwiczenia było zaprojektowanie, zbudowanie i przetestowanie układu kombinacyjnego realizującego transkoder czterobitowej liczby naturalnej (wraz z zerem) na sześciobitową liczbę pierwszą jedynie przy wykorzystaniu bramek logicznych NAND.

Układ 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

2 Ogólna idea rozwiązania

Od tej chwili, kolejne bity wejścia oznaczane będa kolejno przez **A,B,C,D** (**D** jest najmłodszym bitem), a kolejne bity wyjścia będą oznaczane jako **W1,W2,W3,W4,W5,W6** (**W6** jest najmłodszym bitem).
Zdefiniujemy tabele zależności między bitami wejścia a bitami wyjścia. Następnie dla każdego bitu wyjścia zapiszemy zależność w postaci funkcji logicznej. Po doprowadzeniu jej do postaci kanonicznej, przy użyciu **Tablic Karnaugh** znajdziemy uproszczony jej zapis. Następnie utworzymy schemat układu dla każdego wejścia. W razie potrzeby zostanie podzielony on na podukłady, które będziemy osobno rozrysowywać już przy użyciu jedynie bramki logicznej NAND. Następnie połączymy podukłady w jeden, złożony układ.

3 Zależności logiczne Wejście-Wyjście.

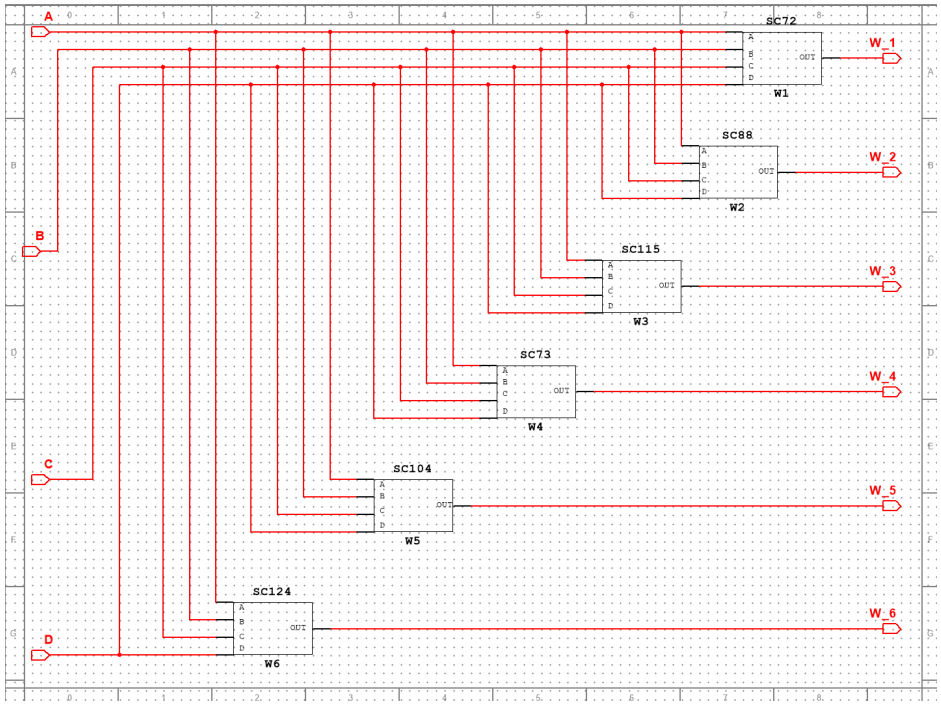
Wejście	**Wyjście**	**Wejście (binarnie)**	**Wyjście (binarnie)**
0	2	0000	000010
1	3	0001	000011
2	5	0010	000101
3	7	0011	000111
4	11	0100	001011
5	13	0101	001101
6	17	0110	010001
7	19	0111	010011
8	23	1000	010111
9	29	1001	011101
10	31	1010	011111
11	37	1011	100101
12	41	1100	101001
13	43	1101	101011
14	47	1110	101111
15	53	1111	110101

Tabela 1: Przekształcanie par liczb (wejście, wyjście) na wartości binarne.

A	B	C	D	W1	W2	W3	W4	W5	W6
0	0	0	0	0	0	0	0	1	0
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	1	1
0	1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	0	0	0	1
0	1	1	1	0	1	0	0	1	1
1	0	0	0	0	1	0	1	1	1
1	0	0	1	0	1	1	1	0	1
1	0	1	0	0	1	1	1	1	1
1	0	1	1	1	0	0	1	0	1
1	1	0	0	1	0	1	0	0	1
1	1	0	1	1	0	1	0	1	1
1	1	1	0	1	0	1	1	1	1
1	1	1	1	1	1	0	1	0	1

Tabela 2: Mapowanie bitów wejścia (A,B,C,D) na bity wyjścia (W1,W2,W3,W4,W5,W6)

4 Schemat Całego układu.



Rysunek 1: Schemat całego układu

5 Układ powiązany z Wyjściem 1 (W1)

5.1 Tablica Karnaugh dla W1

Aby stworzyć tablicę Karnaugh dla układu związanego z W1, należy utworzyć tabelę 4x4 układając wiersze i kolumny za pomocą **Kodu Grey'a**. Miejsca, w których znajdują się wartości 1 można odczytać z Tabeli 2. W

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 3: Tablica Karnaugh dla W1.

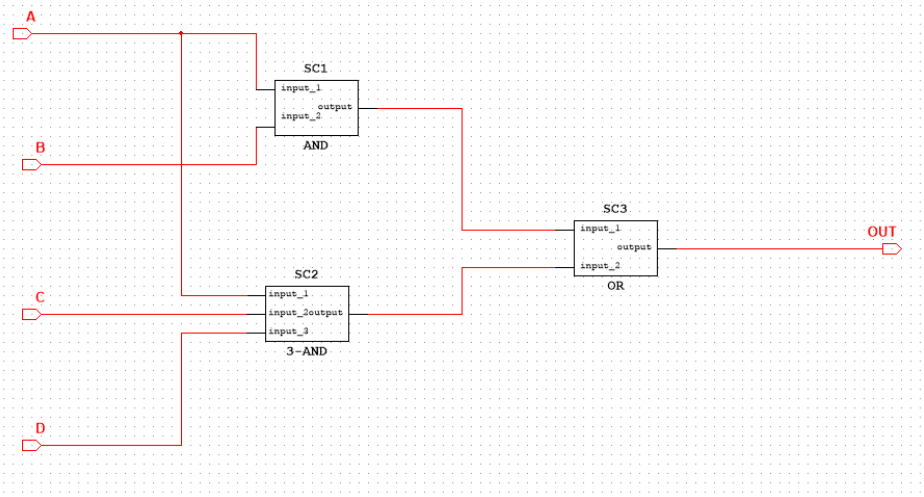
Tabeli 3 zaznaczono kolorem obszary, które umożliwiają minimalizację układów logicznych 1. Każde pojedyncze pole tabeli jednoznacznie przypisuje wartości A, B, C i D. Obszar składający się z sąsiednich pól (z wartością 1) o liczbie równającej się potędze dwójki pozwala na eliminację zmiennych z równania. Im większy obszar, tym więcej zmiennych można wyeliminować.

Warto zauważyć, że oprócz oczywistej redukcji dwóch zmiennych dzięki czerwonemu obszarowi, możemy również wyeliminować jedną zmienną z drugiego składnika sumy (powiązanego z niebieskim obszarem). Jest to możliwe dzięki stworzeniu spójnego obszaru o wymiarach 2x1, który obejmuje zarówno czerwone, jak i niebieskie pole.

Należy jednak zauważyć, że rysunek może wprowadzać w błąd, sugerując, że każde pole ma przypisany tylko jeden kolor, co nie zawsze jest prawdą w przypadku redukcji wykorzystującej tablice Karnaugh.
Dzięki temu zapisowi odczytujemy funkcję powiązaną z układem W1 jako :

$$W1 = AB + ACD$$

5.2 Implementacja w programie Multisim



Rysunek 2: Implementacja w Multisimie układu realizującego $W1 = AB + ACD$

Implementacje podukładów **sc1** (dwu wejściowa bramka AND), **sc2** (trzy wejściowa bramka AND), **sc3** (dwu wejściowa bramka or) za pomocą bramek **NAND** zaprezentowana jest kolejno w rozdziałach 11.1, 11.2, 11.5

6 Układ powiązany z Wyjściem 2 (W2)

6.1 Tablica Karnaugh dla W2

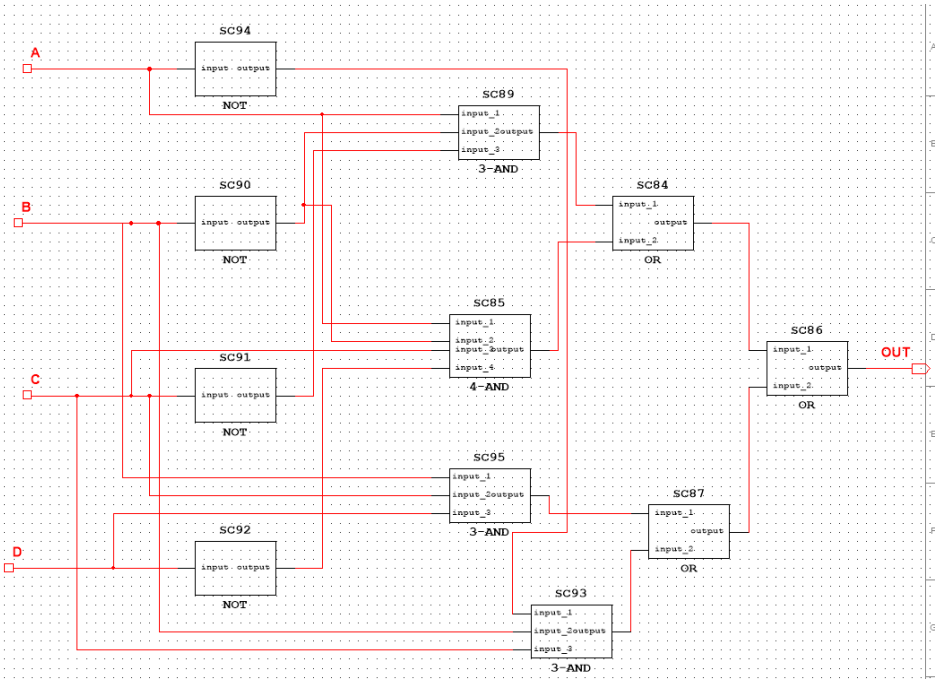
Analogicznie jak poprzednio dokonujemy minimalizacji i otrzymujemy funkcje postaci:

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 4: Tablica Karnaugh dla W2.

$$W2 = A\overline{B} \cdot \overline{C} + A\overline{B}C\overline{D} + BCD + \overline{A}BC$$

6.2 Implementacja w programie Multisim



Rysunek 3: Implementacja w Multisimie układu realizującego układ dla W2.

Implementacja podukładów za pomocą samych bramek NAND znajduję się w 11.4 (NOT), 11.5 (OR), 11.2 (3-AND), 11.3 (4-AND).

7 Układ powiązany z Wyjściem W3

7.1 Tablica Karnaugh dla W3

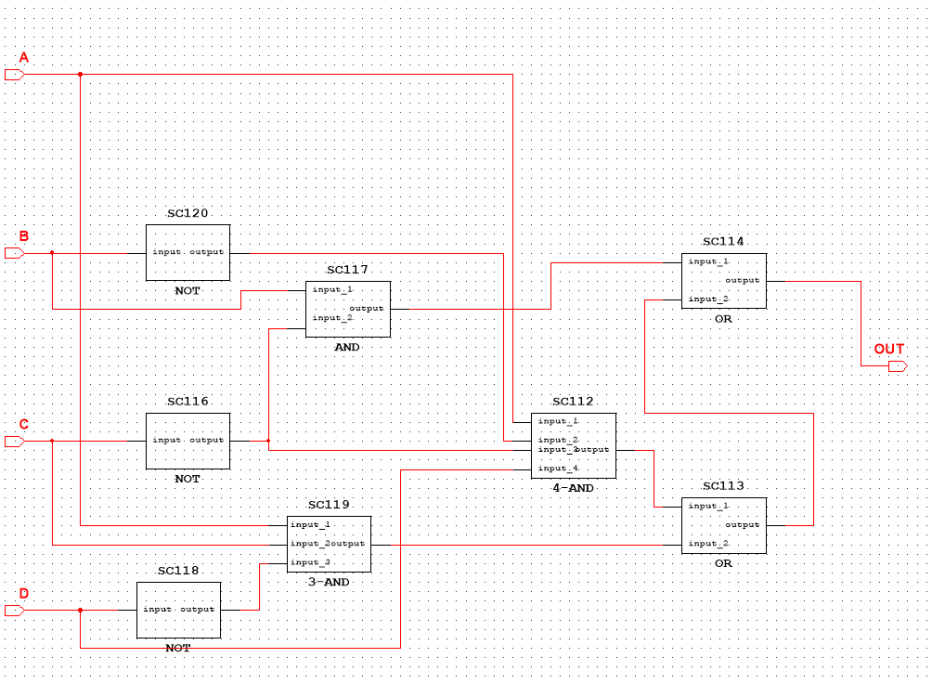
AB \CD	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	0	1
10	0	1	0	1

Tabela 5: Tablica Karnaugh dla W3.

Na podstawie tablicy Karnaugh otrzymujemy:

$$W3 = B\overline{C} + AC\overline{D} + A\overline{B} \cdot \overline{C}D$$

7.2 Implementacja w programie Multisim



Rysunek 4: Implementacja w Multisimie układu realizującego układ dla W3.

Implementacja podukładów za pomocą samych bramek NAND znajduję się w 11.4 (NOT), 11.5 (OR), 11.1 (AND), 11.2 (3-AND), 11.3 (4-AND).

8 Układ powiązany z Wyjściem W4

8.1 Tablica Karnaugh dla W4

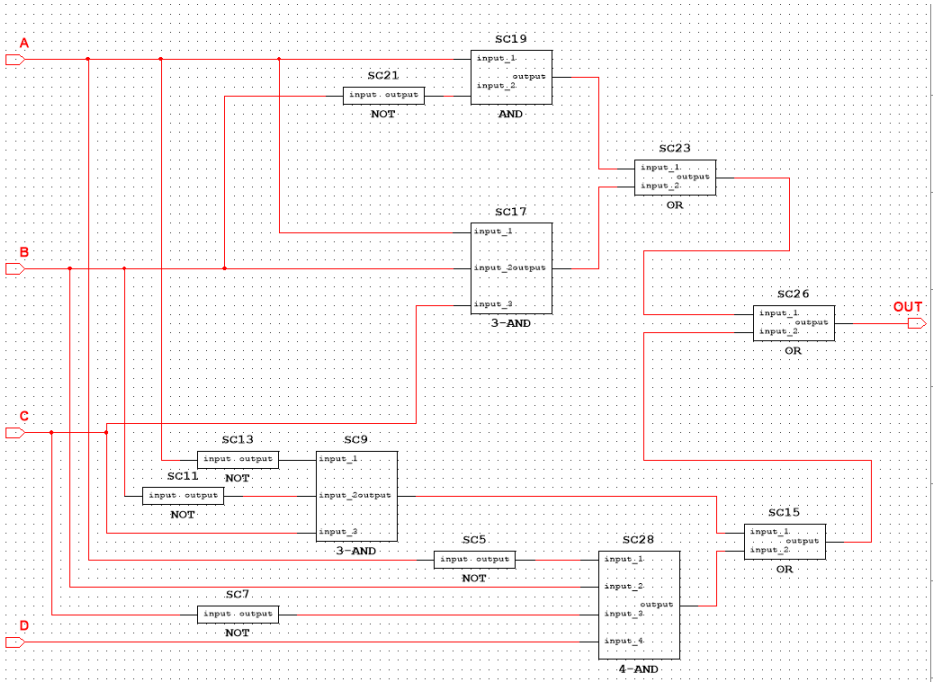
AB \CD	**00**	**01**	**11**	**10**
00	0	0	1	1
01	0	1	0	0
11	0	0	1	1
10	1	1	1	1

Tabela 6: Tablica Karnaugh dla W4.

Na podstawie tablicy Karnaugh otrzymujemy:

$$W4 = A\overline{B} + ABC + \overline{A}B\overline{C}D + \overline{A} \cdot \overline{B}C$$

8.2 Implementacja w programie Multisim



Rysunek 5: Implementacja w Multisimie układu realizującego układ dla W4.

Implementacja podukładów za pomocą samych bramek NAND znajduję się w 11.4 (NOT), 11.5 (OR), 11.1 (AND), 11.2 (3-AND), 11.3 (4-AND).

9 Układ powiązany z Wyjściem W5

9.1 Tablica Karnaugh dla W5

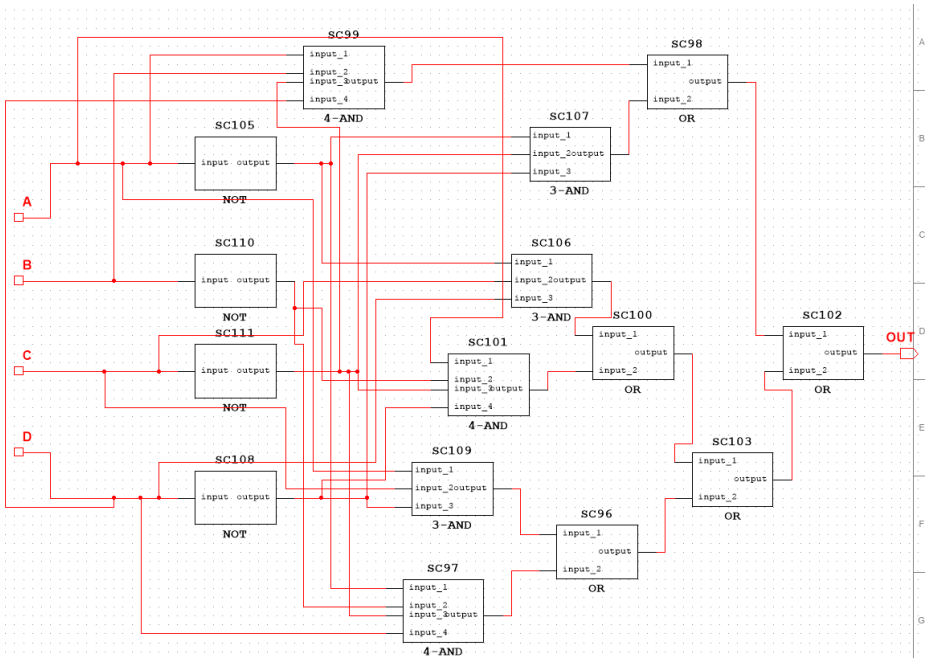
AB \CD	00	01	11	10
00	1	1	1	0
01	1	0	1	0
11	0	1	0	1
10	1	0	0	1

Tabela 7: Tablica Karnaugh dla W5.

Na podstawie tablicy Karnaugh otrzymujemy:

$$W5 = \overline{A} \cdot \overline{C} \cdot \overline{D} + \overline{A}CD + AC\overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C}D + AB\overline{C}D + A\overline{B} \cdot \overline{C} \cdot \overline{D}$$

9.2 Implementacja w programie Multisim



Rysunek 6: Implementacja w Multisimie układu realizującego układ dla W5.

Implementacja podukładów za pomocą samych bramek NAND znajduję się w 11.4 (NOT), 11.5 (OR), 11.2 (3-AND), 11.3 (4-AND).

10 Układ powiązany z Wyjściem W6

10.1 Tablica Karnaugh dla W6

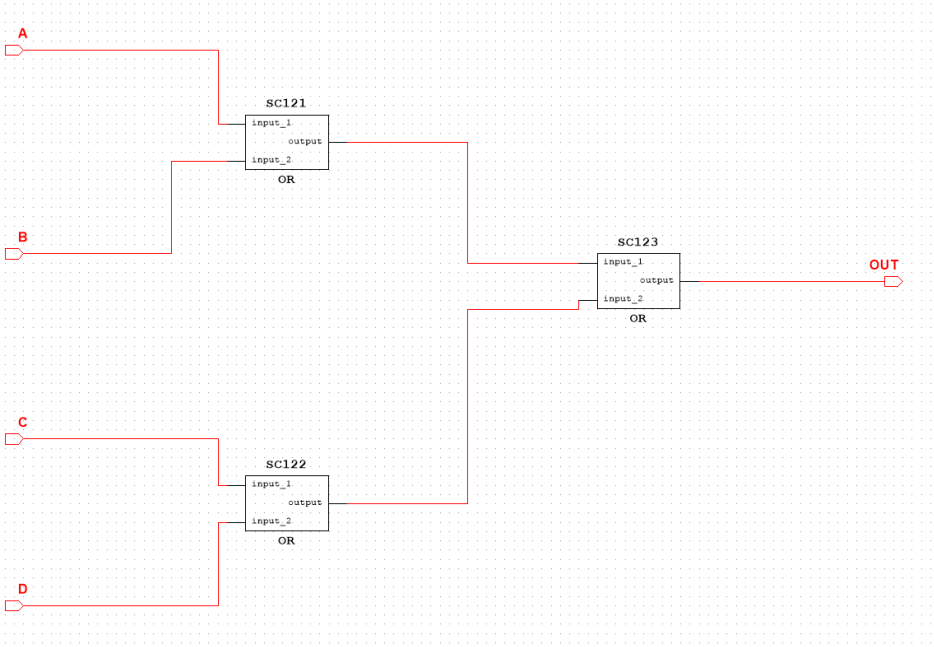
AB \CD	**00**	**01**	**11**	**10**
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Tabela 8: Tablica Karnaugh dla W6.

W tym wypadku, funkcja ma zwracać *0* wtedy i tylko wtedy gdy wszystkie bity są równe *0*, zatem jest to alternatywa A,B,C,D czyli:

$$W6 = A + B + C + D$$

10.2 Implementacja w programie Multisim



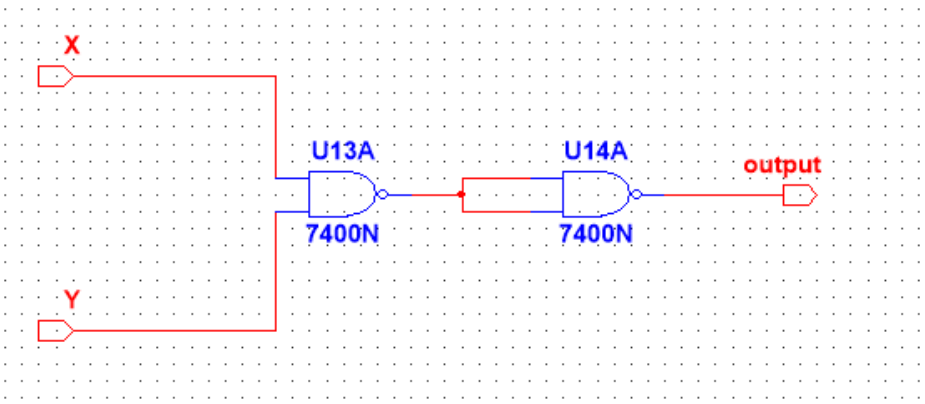
Rysunek 7: Implementacja w Multisimie układu realizującego układ dla W6.

Implementacja podukładu realizującego OR za pomocą samych bramek NAND znajduję się w 11.5

11 Implementacje podukładów

11.1 AND

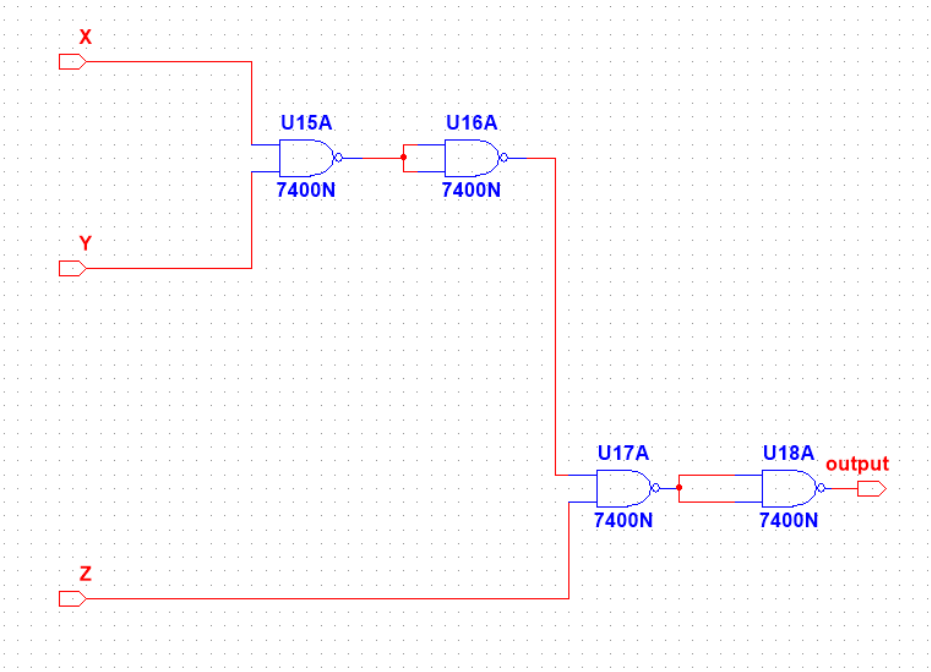
$$XY = \overline{\overline{XY}} = \overline{\overline{XY} \cdot \overline{XY}}$$



Rysunek 8: Implementacja w Multisimie bramki logicznej AND za pomocą bramek NAND.

11.2 3-AND

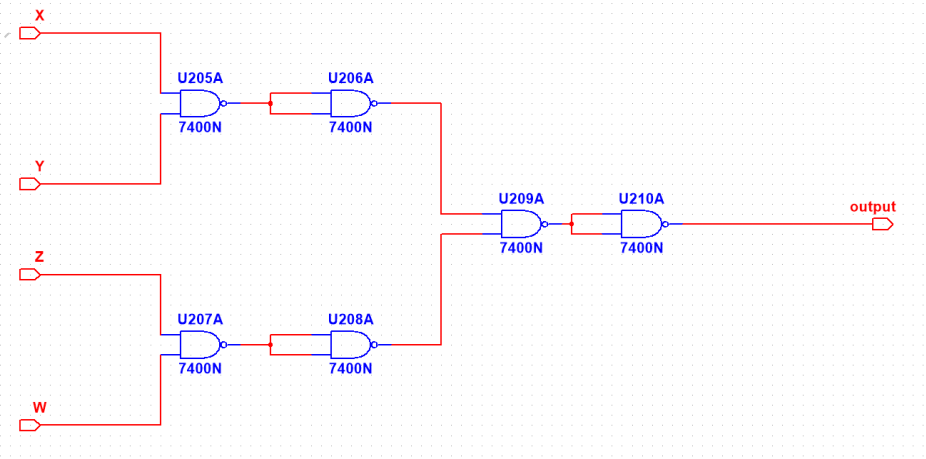
$$XYZ = XY \cdot Z = \overline{\overline{XY} \cdot \overline{XY}Z} = \overline{\overline{\overline{XY} \cdot \overline{XY}Z}} = \overline{(\overline{\overline{XY} \cdot \overline{XY}Z}) \cdot (\overline{\overline{XY} \cdot \overline{XY}Z})}$$



Rysunek 9: Implementacja w Multisimie 3-wejściowej bramki logicznej AND za pomocą bramek NAND.

11.3 4-AND

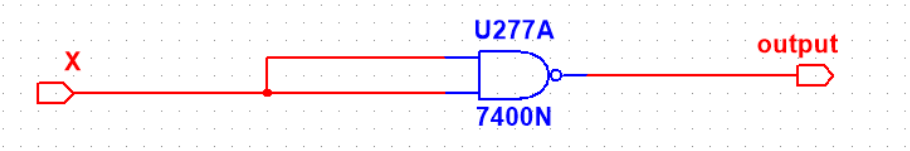
$$XYZW = XY \cdot ZW = \overline{\overline{XY} \cdot \overline{XY}} \cdot \overline{\overline{ZW} \cdot \overline{ZW}} = \overline{\overline{\overline{XY} \cdot \overline{XY}} \cdot \overline{\overline{ZW} \cdot \overline{ZW}}} \cdot \overline{\overline{\overline{XY} \cdot \overline{XY}} \cdot \overline{\overline{ZW} \cdot \overline{ZW}}}$$



Rysunek 10: Implementacja w Multisimie 4-wejściowej bramki logicznej AND za pomocą bramek NAND.

11.4 NOT

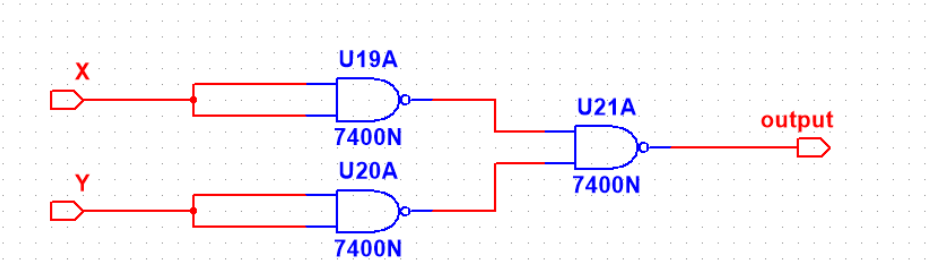
$$\overline{X} = \overline{XX}$$



Rysunek 11: Implementacja w Multisimie bramki logicznej NOT za pomocą bramek NAND.

11.5 OR

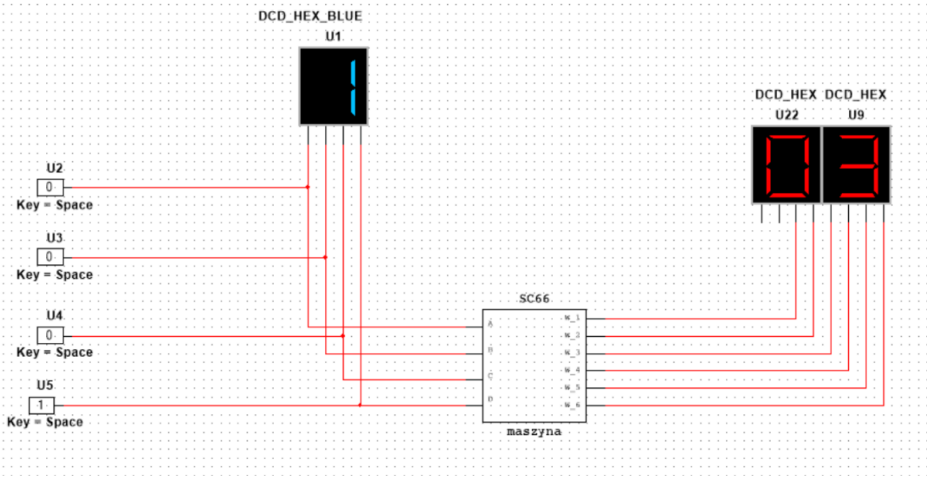
$$X + Y = \overline{\overline{X + Y}} = \overline{\overline{X} \cdot \overline{Y}} = \overline{\overline{X}\,\overline{X} \cdot \overline{Y}\,\overline{Y}}$$



Rysunek 12: Implementacja w Multisimie bramki logicznej OR za pomocą bramek NAND.

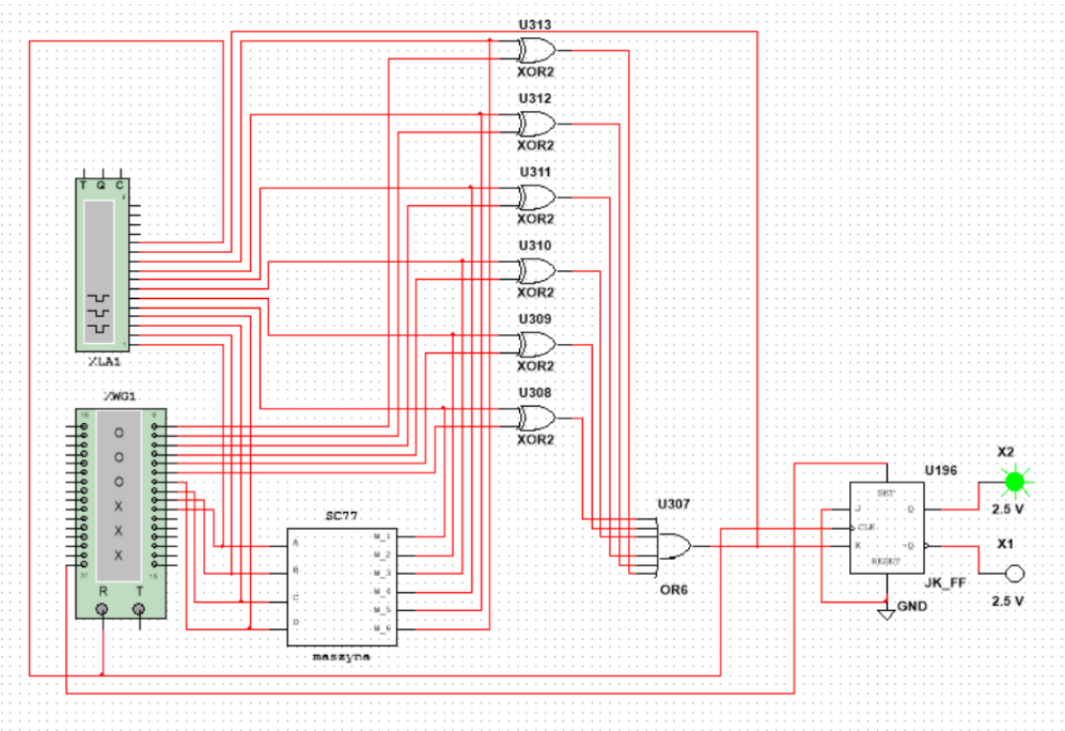
12 Testy

Wyświetlacze 7-segmentowe wyświetlające liczby w systemie szesnastkowym.



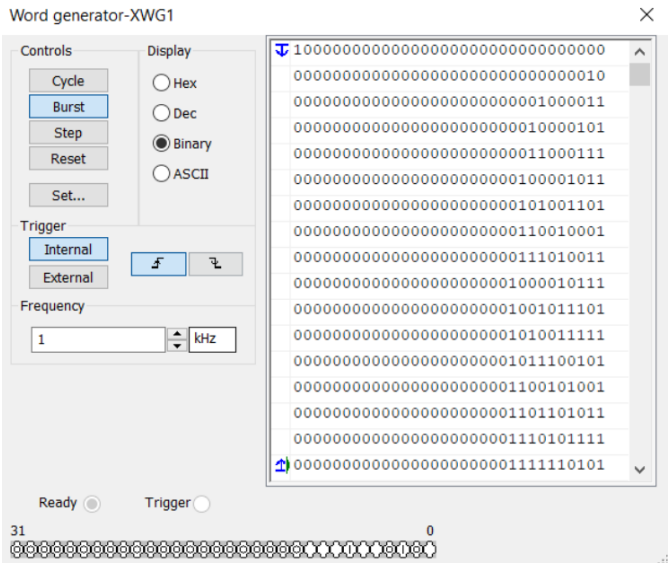
Rysunek 13: Przykład. Układ przekształca dziesiętną liczbę 1 na liczbę pierwszą 3.

12.1 Układ Testujący



Rysunek 14: Schemat układu testującego zaprojektowany w Multisimie.

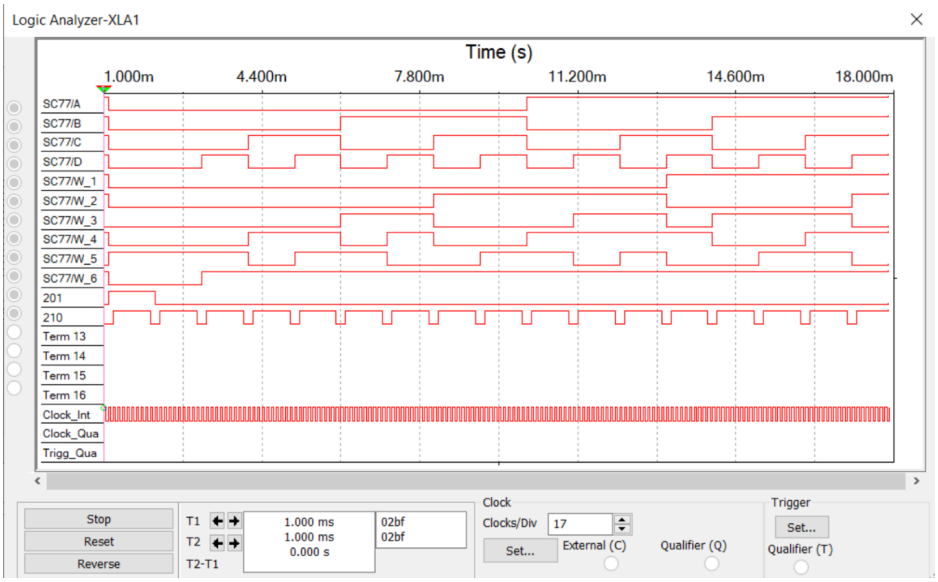
Generator słów poza pierwszym wyrazem, generuje 10 bitowe słowa. Pierwsze 4 bity odpowiadają wejściu, natomiast kolejne 6 to oczekiwane wyjście.



Rysunek 15: Początkowe wartości słów generowanych przez generator.

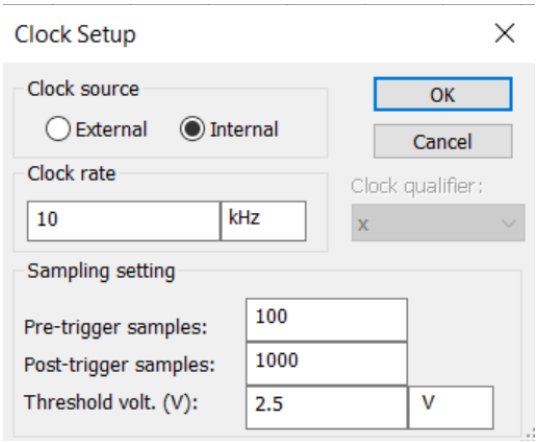
Po przekształceniu sygnału przez układ, bity prawdziwego wyjścia są porównywane z oczekiwanymi wynikami za pomocą bramek logicznych XOR. W przypadku wykrycia różnicy (czyli któryś XOR zwróci 1) stan przerzutnika zmieni się, zapalając czerwoną lampkę i pozostanie taki już do końca testu.
Przy teście następnej jednostki, pierwsze wygenerowane słowo, z powrotem ustawi wyjście Q przerzutnika na stan

logicznej jedynki, za pomocą wejścia SET. Aby uniknąć sytuacji, w której przerzutnik wychwyci niepoprawny sygnał pomiędzy zmianą stanów, zegar przerzutnika podpięty jest do sygnału ready generatora słów. Do układu dołączono również analizator stanów, dzięki któremu można zweryfikować, że sygnał ready generatora jest dobrym wyznacznikiem czasu działania przerzutnika (sygnał 210). Pozwala on też zobaczyć, dla których wejść wystąpił błąd (sygnał 201) oraz wejścia same w sobie i obliczone dla nich wyjścia.



Rysunek 16: Zwracane wykresy przez analizator stanów.

Aby poprawnie wyświetlać otrzymywane sygnały analizator pracuje z dziesięciokrotnie większą częstotliwością niż generator słów.



Rysunek 17: Prezentacja ustawień zegara.

13 Wnioski

- 1. Dzięki "modułowemu"podejściu jesteśmy w stanie stosunkowo łatwo realizować nawet skomplikowane układy cyfrowe. W każdym kroku schodząc z poziomem abstrakcji w dół, potrafimy rozkładać dany problem na mniejsze, prostsze problemy. Widać to w tym, że zaczynaliśmy od analizy 6bitowego wyjścia, następnie rozważaliśmy każdy bit wyjścia osobno, który rozkładaliśmy na wygodną postać kanoniczną funkcji, której to konkretne elementy (OR, AND, 3-AND, NOT itp) dopiero dekomponowaliśmy na bramki NAND.

- 2. W rzeczywistych układach często pojawia się konieczność zamiany początkowo znalezionych bramek, na takie prostsze, zbudowane na podstawie szybszych, tańszych czy bardziej dostępnych jak np. NAND.

- 3. Można by, po minimalizacji Karnaugh zamisat rozważać każdy składnik alternatywy osobno i dopiero go zapisywać za pomocą bramek *OR*, *AND* itp. i dopiero później konwertować te bramki na *NAND*, wcześniej przy postaci kanonicznej dokonywać równoważnych przekształceń tak, aby wykorzystać jedynie bramki *NAND*. Mogło by doprowadzić to do zmniejszenia liczby wykorzystanych łącznie bramek, a zatem uprościłoby układ.

- 4. Istotną kwestią w kontekście skalowalności jest nomenklatura i nazewnictwo. W naszym przypadku, najmłodszy bit był nazwany **D**. Jest to konwencja niewygodna w przypadku próby rozbudowania takiego układu. Przy próbie dodania dodatkowego bitu **E**, musielibyśmy modyfikować wszystkie wcześniejszy układy, tak, żeby najmłodszy bit był nazwany **E**, albo pogodzić się ze stratą spójności zapisu. Lepszym rozwiązaniem byłoby nazwanie najstarszego bitu **D**, a najmłodszego **A**. W takim wypadku, dołożenie dodatkowego bitu jest dużo prostsze i pozostajemy przy spójnym nazewnictwie

- 5. **Pomysły zastosowań**. Dla **n** kolejnych liczb naturalnych, statystycznie znajduje się wśród nich $ln(n)$ liczb pierwszych. Można więc wykorzystać mapowanie pierwszych n liczb naturalnych na pierwsze n liczb pierwszych jako małoprzewidywalne rzutowanie liczb z przedziału $\{1, .., n\} \longrightarrow \{2, .., [e^n]\}$. Charakterystyką liczb pierwszych jest ich mało regularny rozrzut, zatem może to być użyteczne w symulowaniu losowości pewnych układów. Dodatkowo można użyć rzutowania liter alfabetu A,B,C,D... na liczby pierwsze, aby korzystając z zasadniczego twierdzenia arytmetyki (jednoznaczność rozkładu liczby naturalnej na liczby pierwsze) aby zapisywać słowa jako permutacje konkretnych liter. Można to wykorzystać np. do zliczania permutacji tych samych słów w tekście. Liczby pierwsze i mapowanie na nie, wykorzystuje się również w kryptografii i różnego rodzaju szyfrowaniach, natomiast do tego potrzebowalibyśmy znacznie większych liczb.