Laboratorium 1
Transkoder liczb

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

marzec 2024

Spis treści

1 Cel zadania

Celem zadania było zaprojektować, zbudować i przetestować układ kombinacyjny realizujący transkoder czterobitowej liczby naturalnej (wraz z zerem) na sześciobitową liczbę pierwszą, bazując wyłącznie na bramkach NAND.

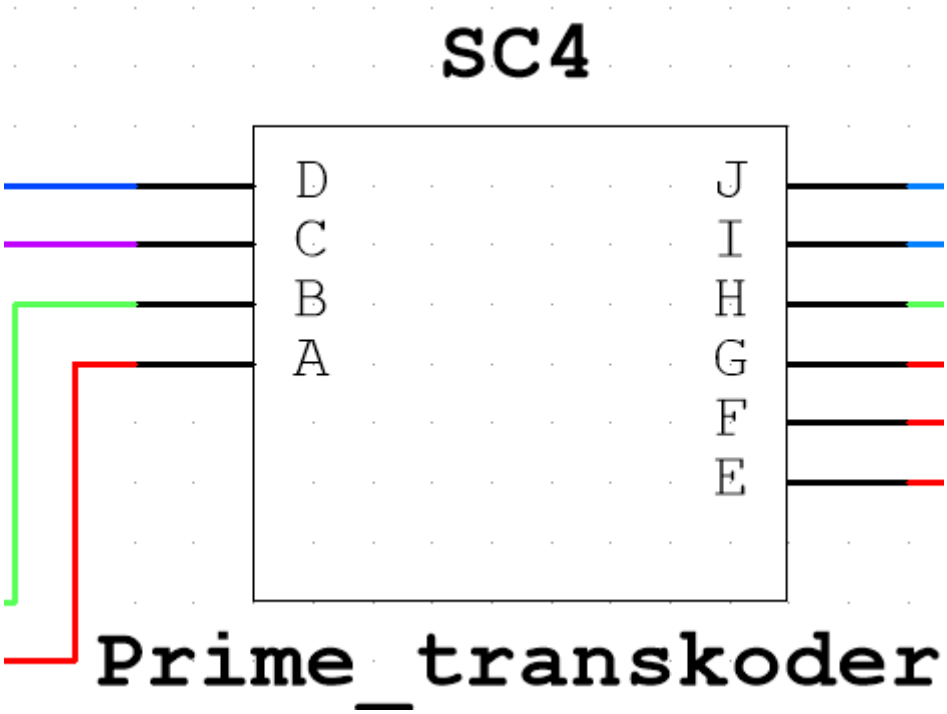
2 Idea rozwiązania

Nasze rozwiązanie opiera się na przekształcaniu 4 bitów wejściowych za pomocą transkoderów, generując w rezultacie 6 bitów wyjściowych. Aby uzyskać konkretne kombinacje bitów wyjściowych, zastosowaliśmy funkcje logiczne opracowane z wykorzystaniem tablic Karnaugh.

3 Układ transkodera liczb pierwszych

3.1 Black box

Pierwszym krokiem w projektowaniu układu jest przedstawienie go jako tzw. ”Black Box”. Czyli taką czarną skrzynkę dla której określamy tylko wejście i oczekiwany wynik działania dla danego wejścia ale nie wgłębiamy się w implementację.



Rysunek 3.1: Czarna skrzynka transkodera

Wejście do układu stanowią 4 piny ABCD kodujące binarnie wejściową liczbę 0-15. Stan wysoki na pinach stanowi logiczną jedynkę (1), a stan niski logiczne zero (0).

Numer bitu	3	2	1	0
Bit	A	B	C	D
Mnożnik	2^3	2^2	2^1	2^0

Tabela 3.1: Kodowanie pinów ABCD

Wyjście układu stanowi 6 pinów EFGHIJ kodujące pierwsze 16 liczb pierwszych. Tak samo jak na wejściu stan wysoki na pinach stanowi logiczną jedynkę (1), a stan niski logiczne zero (0).

Numer bitu	5	4	3	2	1	0
Bit	E	F	G	H	I	J
Mnożnik	2^5	2^4	2^3	2^2	2^1	2^0

Tabela 3.2: Kodowanie pinów EFGHIJ

Układ mapuje wejście na wyjście w następujący sposób - zakładając notację binarną zapisanego mapowania.

Wejście	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Oczekiwane wyjście	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53

Tabela 3.3: mapowanie wejścia na wyjście

3.2 Tabela prawdy

Poniżej zapisaliśmy tabelę prawd dla projektowanego układu:

Wejście				Wyjście					
A	B	C	D	E	F	G	H	I	J
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 3.4: Tabela prawdy dla układu

3.3 Tablice Karnaugh

Poniżej przedstawione są tabele Karnaugh których użyliśmy do zbudowania transkoderów poszczególnych bitów. W poniższym zapisie $A+B$ oznacza bramkę $OR(A,B)$, AB oznacza $AND(A,B)$ i $\overline{A}$ oznacza $NOT(A)$.

3.3.1 Człon J

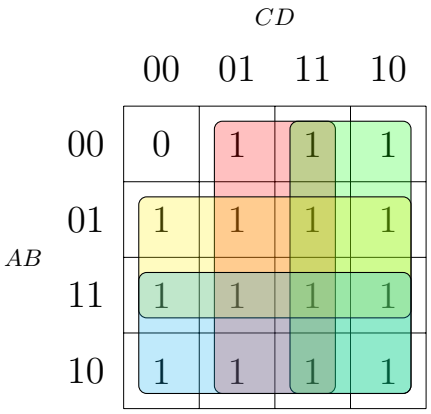


Tabela 3.5: Tabela Karnaugh dla członu J

$$f_{(1)} = A + C + D + B$$

3.3.2 Człon I

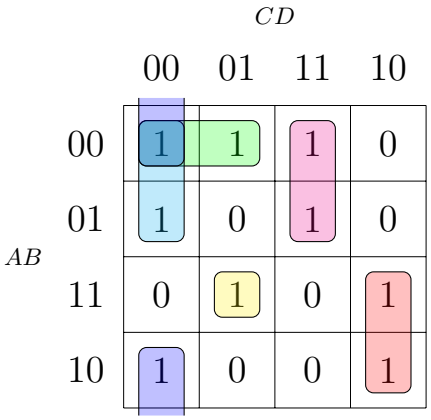


Tabela 3.6: Tabela Karnaugh dla członu I

$$f_{(1)} = AC\overline{D} + \overline{A}C\overline{D} + \overline{ABC} + \overline{ACD} + \overline{BCD} + AB\overline{C}D$$

3.3.3 Człon H

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 3.7: Tabela Karnaugh dla członu H

$$f_{(1)} = A\overline{B} + AC + \overline{B}C + \overline{A}B\overline{C}D$$

3.3.4 Człon G

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 3.8: Tabela Karnaugh dla członu G

$$f_{(1)} = B\overline{C} + AC\overline{D} + A\overline{C}D$$

3.3.5 Człon F

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 3.9: Tabela Karnaugh dla członu F

$$f_{(1)} = A\overline{BD} + A\overline{BC} + \overline{A}BC + BCD$$

3.3.6 Człon E

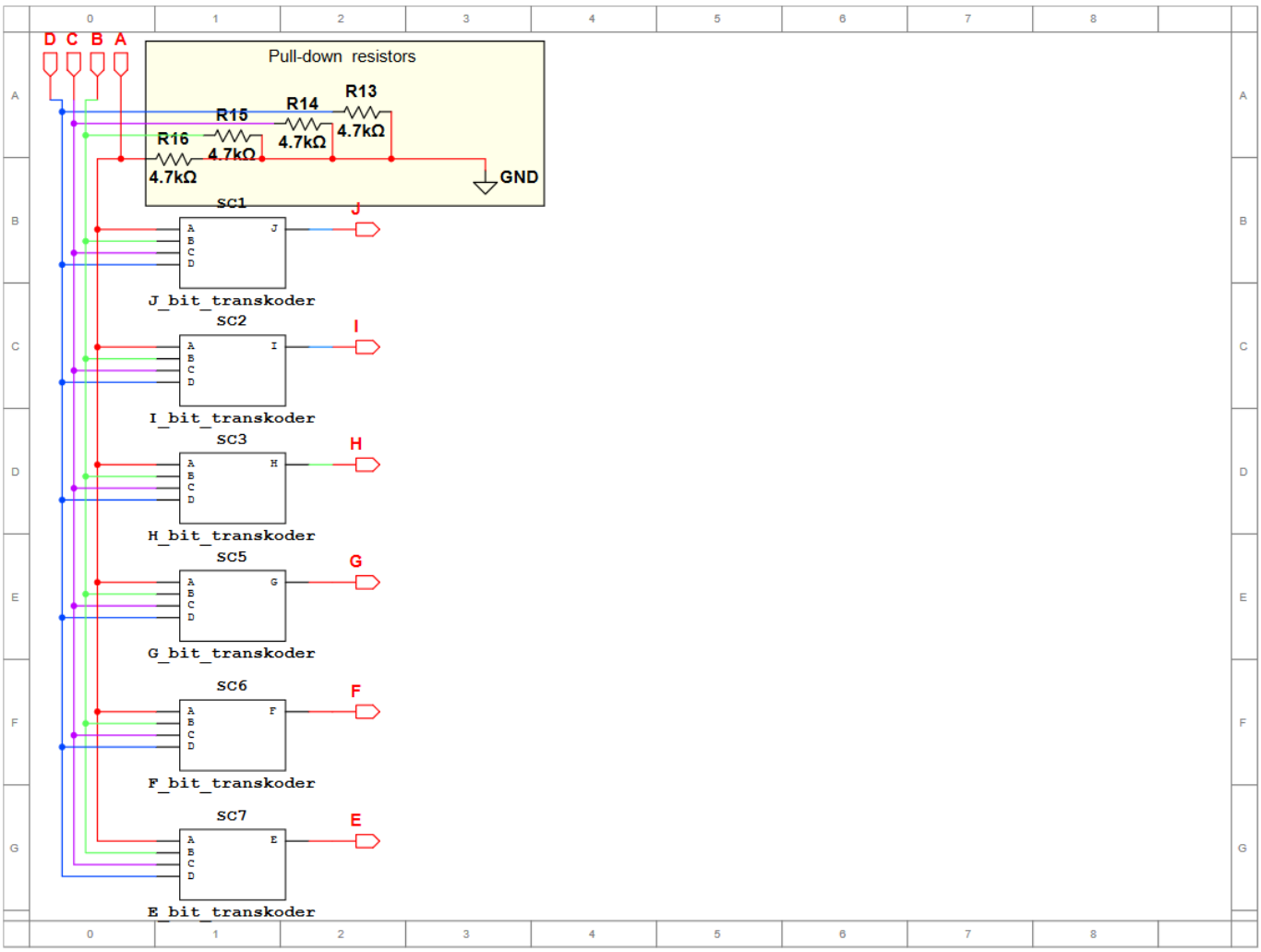
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	0

Tabela 3.10: Tabela Karnaugh dla członu E

$$f_{(1)} = AB + ACD$$

3.4 Implementacja

Otrzymane z tablic Karnaugh funkcje zostały odpowiednio przekształcone tak, aby używały jedynie bramek NAND i następnie zostały one zaimplementowane. Zdecydowaliśmy się na użycie schematów hierarchicznych - subcircuit - dla zwiększenia czytelnosci schematu. Na pierwszym poziomie znajdują się same transkodery pojedynczych bitów - czarne skrzynki które otrzymują całe wejście, a w wyniku dają konkretny bit wyjścia, innymi słowy każda funkcja logiczna została zaimplementowana na oddzielnym schemacie, a następnie zgromadziliśmy je w jeden podschemat stanowiący cały projektowany układ "Prime transkoder"



Rysunek 3.2: Podschemat najwyższego rzędu, gromadzący w sobie podschematy właściwych funkcji logicznych

Użyliśmy rezystorów pull-down na wejściu układu, które ściągają niepodpięty sygnał do logicznego zera aby zapewnić poprawne funkcjonowanie układu nawet gdy wejście nie jest podpięte w całości.

Poniżej implementacje poszczególnych funkcji logicznych zgodnie z funkcjami otrzymanymi z tablic Karnaugh razem z przekształceniami aby używane były wy-

łącznie bramki NAND. Najczęściej używanymi przez nas przekształceniami będą:

$$A + B = NAND(\overline{A}, \overline{B})$$

$$AB = \overline{NAND(A, B)}$$

$$\overline{A} = NAND(A, A)$$

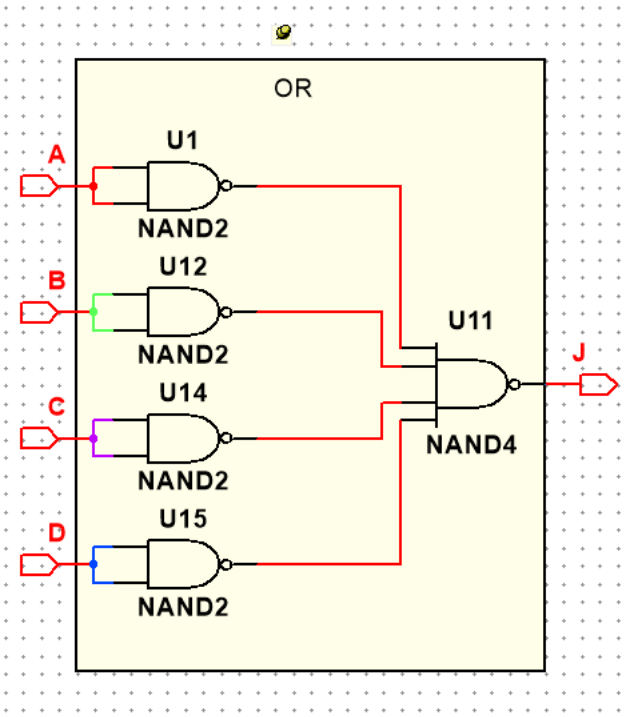
Warto zauważyć, że w następującej sytuacji:

$$AB + CD = NAND(\overline{AB}, \overline{CD}) = NAND(\overline{\overline{NAND(A, B)}}, \overline{\overline{NAND(C, D)}})$$

Co można uprościć do poniższej formuły:

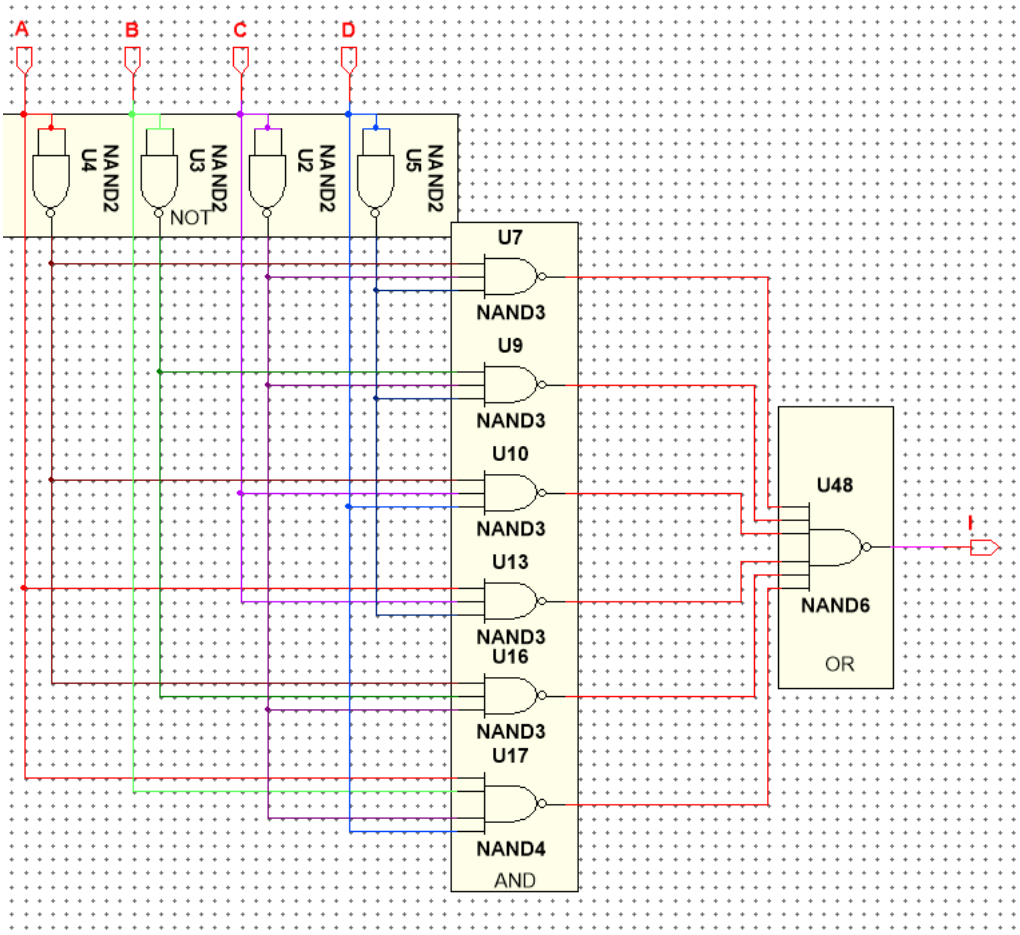
$$NAND(NAND(A, B), NAND(C, D))$$

3.4.1 Transkoder J



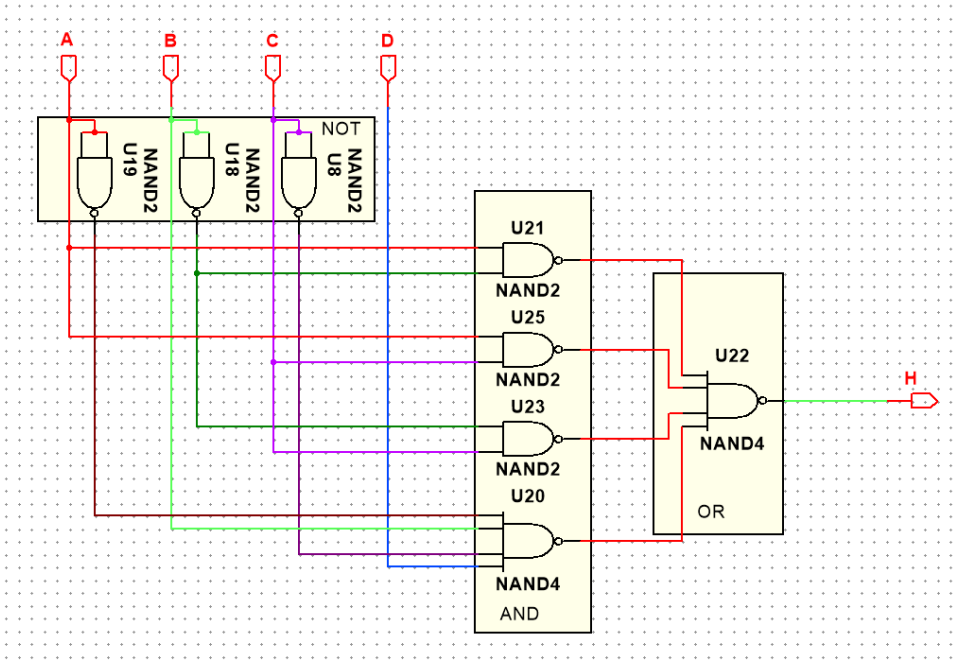
Rysunek 3.3: Podschemat funkcji logicznej dla członu J

3.4.2 Transkoder I



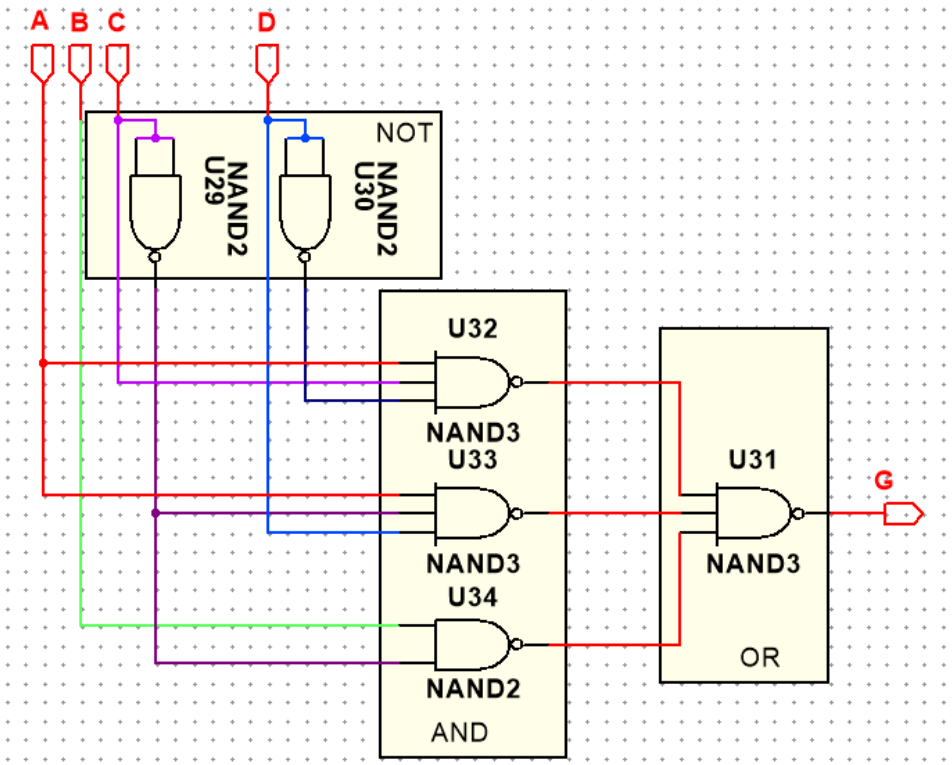
Rysunek 3.4: Podschemat funkcji logicznej dla członu I

3.4.3 Transkoder H



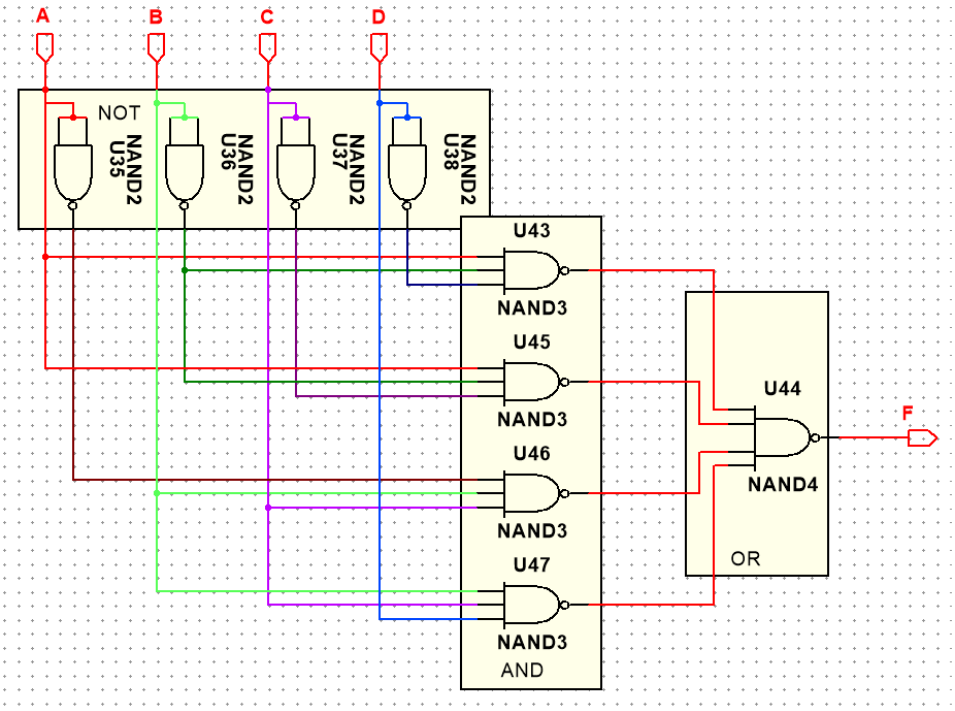
Rysunek 3.5: Podschemat funkcji logicznej dla członu H

3.4.4 Transkoder G



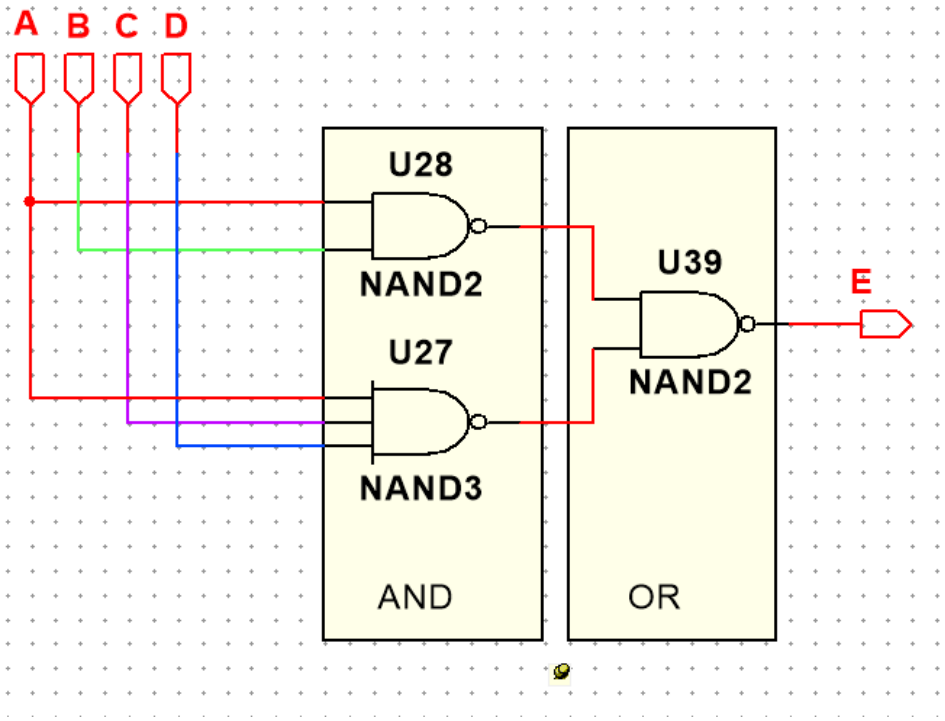
Rysunek 3.6: Podschemat funkcji logicznej dla członu G

3.4.5 Transkoder F



Rysunek 3.7: Podschemat funkcji logicznej dla członu F

3.4.6 Transkoder E

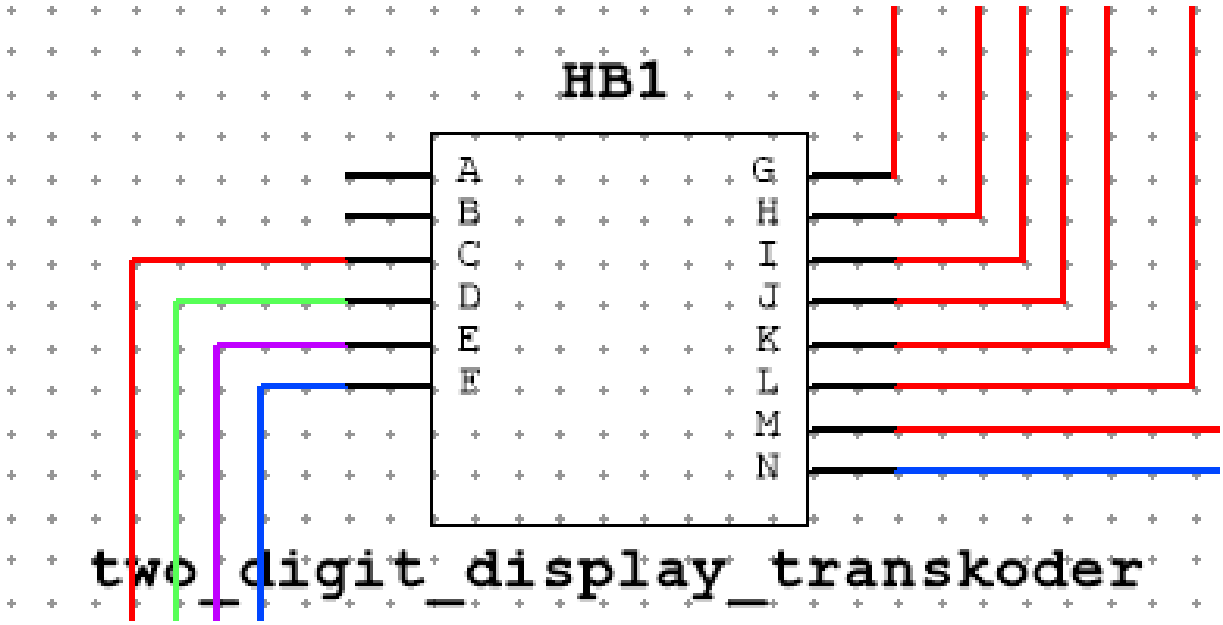


Rysunek 3.8: Podschemat funkcji logicznej dla członu E

4 Enkoder BCD

Aby wyświetlać liczby dwucyfrowe w poprawny sposób zdecydowaliśmy się na zaimplementowanie enkodera BCD (Binary Coded Decimal). Jest to format w którym liczba dziesiętna jest kodowana cyfra po cyfrze przez 4 bitowe liczby binarne. Pozwala to na łatwe wyświetlanie wielocyfrowych liczb. W programie Multisim nie udało nam się znaleźć gotowego układu z większą niż 4 bity liczbą wejść, dlatego sami go zaimplementowaliśmy. Jako, że nie jest to bezpośredni cel ćwiczenia, nie rozpisywaliśmy ręcznie tablic Karnaugh - dla 6 zmiennych wejściowych jest to bardziej złożona operacja, lecz skorzystaliśmy z gotowego kalkulatora w internecie (`http://www.32x8.com/var6.html`), aby wygenerować funkcję logiczną. Tabelę prawd wygenerowaliśmy samemu algorytmicznie, a następnie zaznaczyliśmy odpowiednie bity w kalkulatorze online w ten sposób otrzymując funkcje logiczne. Użyliśmy go dwukrotnie. Jeden służy do przedstawienia liczb 0-15, drugi zaś do przedstawienia kolejnych liczb pierwszych.

4.1 Black box



Rysunek 4.1: Czarna skrzynka enkodera BCD dla wejścia

Wejście do układu stanowią 6 pinów ABCDEF kodujące binarnie wejściową liczbę 0-63. Stan wysoki na pinach stanowi logiczną jedynkę (1), a stan niski logiczne zero (0).

Numer bitu	5	4	3	2	1	0
Bit	A	B	C	D	E	F
Mnożnik	2^5	2^4	2^3	2^2	2^1	2^0

Tabela 4.1: Kodowanie pinów ABCDEF

Wyjście układu stanowi 8 pinów GHIJKLMN kodujące odzielnie 2 cyfry składające się na liczbę 2 cyfrową. Tak samo jak na wejściu stan wysoki na pinach stanowi logiczną jedynkę (1), a stan niski logiczne zero (0).

Numer bitu	7	6	5	4	3	2	1	0
Bit	G	H	I	J	K	L	M	N
Mnożnik	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Tabela 4.2: Kodowanie pinów GHIJKLMN

Układ mapuje wejście na wyjście w następujący sposób - zakładając notację binarną zapisanego mapowania.

Wejście	0	1	2	3	4	5	6	7	8	9	10	...	63
Oczekiwane wyjście	0|0	0|1	0|2	0|3	0|4	0|5	0|6	0|7	0|8	0|9	1|0	...	6|3

Tabela 4.3: Mapowanie wejścia na wyjście

4.2 Tabela prawdy

Poniżej zapisaliśmy tabelę prawd dla projektowanego układu:

Wejście						Wyjście							
A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	1	1	0	0	0	0	0	0	1	1	0
0	0	0	1	1	1	0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	1	0	0	0	1
0	0	1	1	0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	0	0	1	0	0	1	1
0	0	1	1	1	0	0	0	0	1	0	1	0	0
0	0	1	1	1	1	0	0	0	1	0	1	0	1
0	1	0	0	0	0	0	0	0	1	0	1	1	0
0	1	0	0	0	1	0	0	0	1	0	1	1	1
0	1	0	0	1	0	0	0	0	1	1	0	0	0
0	1	0	0	1	1	0	0	0	1	1	0	0	1
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	1	0	0	1	0	0	0	0	1
0	1	0	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	1	0	0	1	0	0	0	1	1
0	1	1	0	0	0	0	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	1	0	0	1	0	1
0	1	1	0	1	0	0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	0	1	0	0	1	1	1
0	1	1	1	0	0	0	0	1	0	1	0	0	0
0	1	1	1	0	1	0	0	1	0	1	0	0	1
0	1	1	1	1	0	0	0	1	1	0	0	0	0
0	1	1	1	1	1	0	0	1	1	0	0	0	1

cd.

Wejście						Wyjście							
A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	0	0	0	0	0	0	0	1	1	0	0	1	0
1	0	0	0	0	1	0	0	1	1	0	0	1	1
1	0	0	0	1	0	0	0	1	1	0	1	0	0
1	0	0	0	1	1	0	0	1	1	0	1	0	1
1	0	0	1	0	0	0	0	1	1	0	1	1	0
1	0	0	1	0	1	0	0	1	1	0	1	1	1
1	0	0	1	1	0	0	0	1	1	1	0	0	0
1	0	0	1	1	1	0	0	1	1	1	0	0	1
1	0	1	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	1	0	1	0	0	1	0	0	0	0	1	0
1	0	1	0	1	1	0	1	0	0	0	0	1	1
1	0	1	1	0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	1	0	1	0	0	0	1	0	1
1	0	1	1	1	0	0	1	0	0	0	1	1	0
1	0	1	1	1	1	0	1	0	0	0	1	1	1
1	1	0	0	0	0	0	1	0	0	1	0	0	0
1	1	0	0	0	1	0	1	0	0	1	0	0	1
1	1	0	0	1	0	0	1	0	1	0	0	0	0
1	1	0	0	1	1	0	1	0	1	0	0	0	1
1	1	0	1	0	0	0	1	0	1	0	0	1	0
1	1	0	1	0	1	0	1	0	1	0	0	1	1
1	1	0	1	1	0	0	1	0	1	0	1	0	0
1	1	0	1	1	1	0	1	0	1	0	1	0	1
1	1	1	0	0	0	0	1	0	1	0	1	1	0
1	1	1	0	0	1	0	1	0	1	0	1	1	1
1	1	1	0	1	0	0	1	0	1	1	0	0	0
1	1	1	0	1	1	0	1	0	1	1	0	0	1
1	1	1	1	0	0	0	1	1	0	0	0	0	0
1	1	1	1	0	1	0	1	1	0	0	0	0	1
1	1	1	1	1	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	1	1	0	0	0	1	1

Tabela 4.4: Tabela prawdy dla układu

4.3 Funkcje logiczne

4.3.1 Funkcja N

$$f_{(1)} = F$$

4.3.2 Funkcja M

$$f_{(1)} = \overline{ABC}E + \overline{AC}DE + A\overline{BCE} + A\overline{C}D\overline{E} + A\overline{B}CE + ACDE + \overline{AB}CD\overline{E} + \overline{A}B\overline{CDE} + \overline{A}BC\overline{D}E + ABC\overline{DE}$$

4.3.3 Funkcja L

$$f_{(1)} = \overline{ABC}D+\overline{B}CDE+\overline{A}B\overline{DE}+\overline{A}BC\overline{D}+BC\overline{DE}+A\overline{B}D\overline{E}+A\overline{BCD}E+AB\overline{C}DE$$

4.3.4 Funkcja K

$$f_{(1)} = \overline{AB}C\overline{DE} + \overline{AB}\overline{C}DE + \overline{A}BCD\overline{E} + A\overline{BC}DE + AB\overline{CDE} + ABC\overline{D}E$$

4.3.5 Funkcja J

$$f_{(1)} = A\overline{BC} + A\overline{C}E + A\overline{C}D + \overline{AB}CE + \overline{AB}CD + \overline{A}CDE + \overline{A}B\overline{CD} + ABC\overline{D}$$

4.3.6 Funkcja I

$$f_{(1)} = \overline{A}BD + \overline{A}BC + BCD + A\overline{BC}$$

4.3.7 Funkcja H

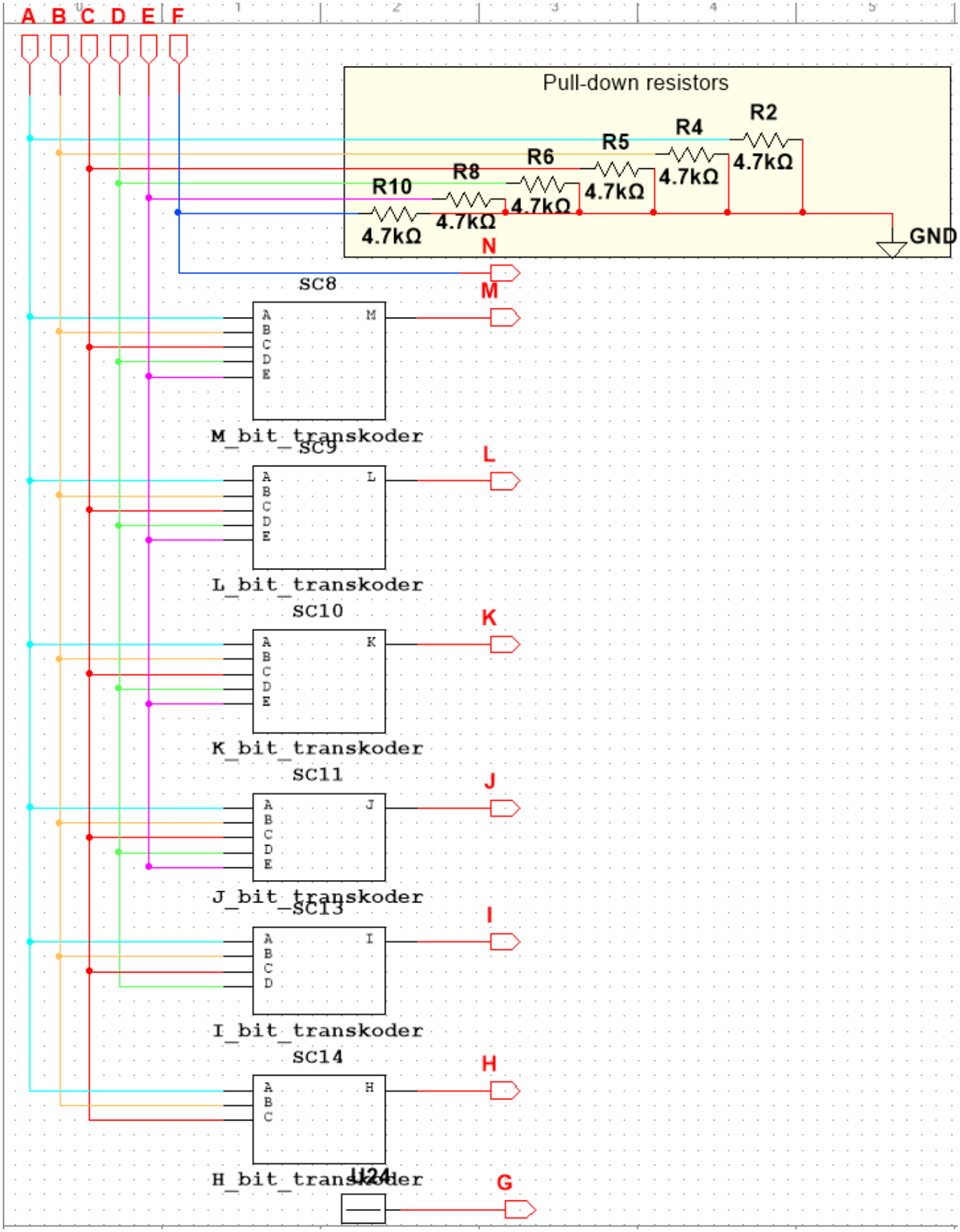
$$f_{(1)} = AB + AB$$

4.3.8 Funkcja G

$$f_{(1)} = 0$$

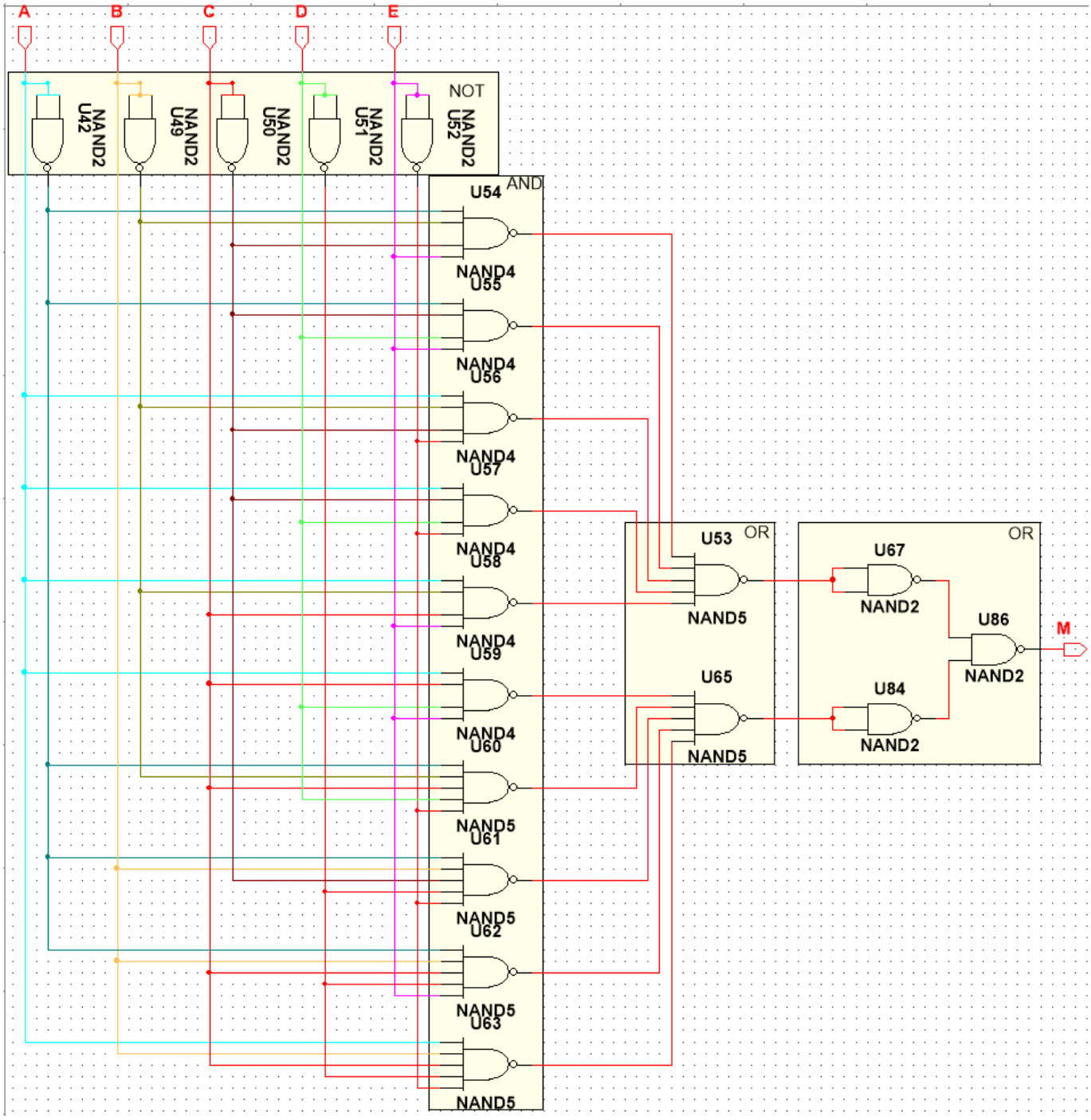
4.4 Implementacja

Tak jak poprzednio, otrzymane funkcje logiczne zostały w naszym układzie zaimplementowane wyłącznie przy użyciu bramek NAND. Tutaj również wykorzystaliśmy schematy hierarchiczne, zachowując, ten sam co wcześniej, podział na czerną skrzynkę, główny podschemat i transkodery dla poszczególnych bitów. Również tutaj zastosowaliśmy rezystory pull-down do ściągania wejść do logicznego '0' aby możliwe było korzystanie z układu przy nie pełnej liczbie wejść.



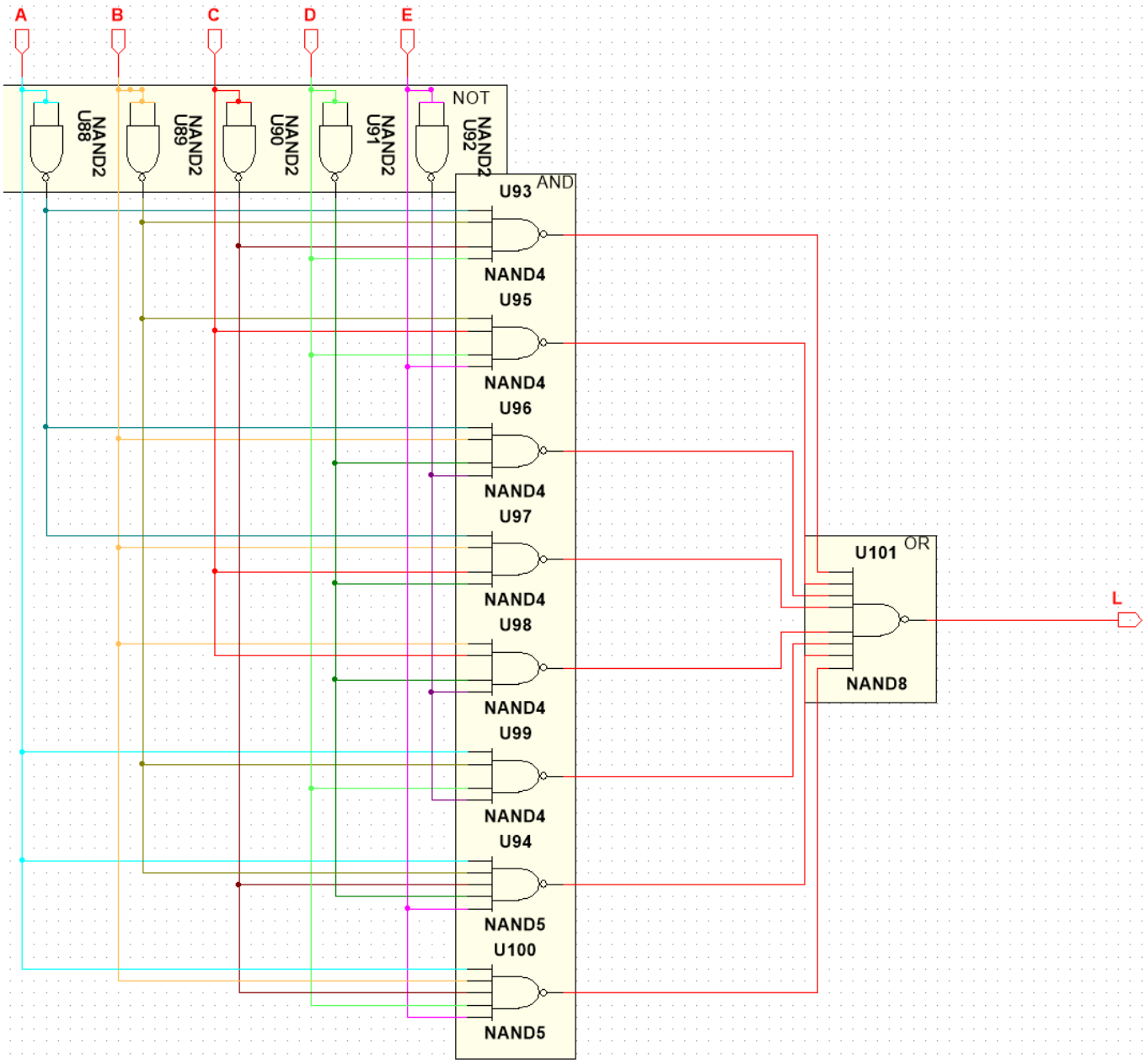
Rysunek 4.2: Podschemat najwyższego rzędu, gromadzący w sobie podschematy właściwych funkcji logicznych

4.4.1 Enkoder M



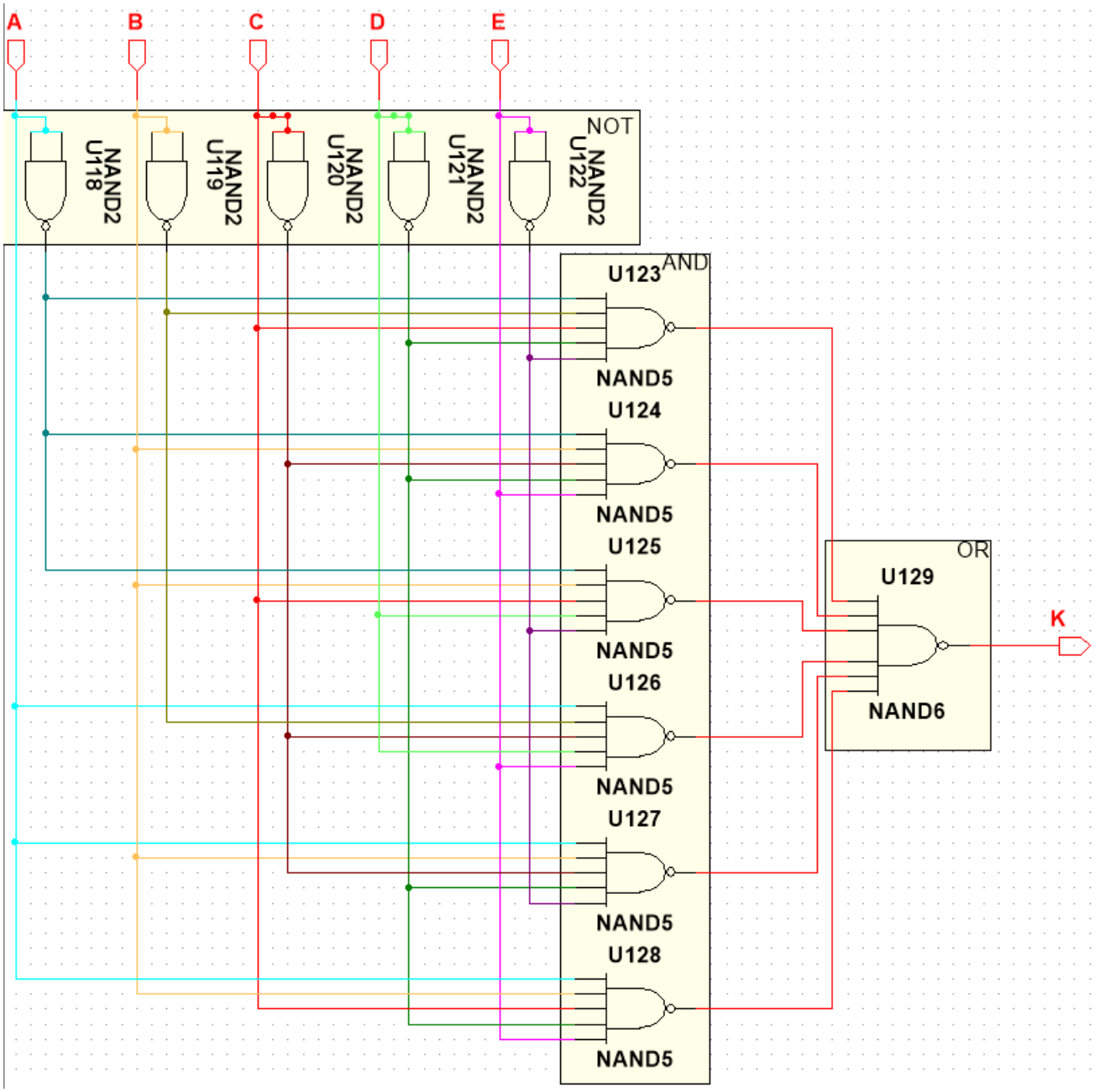
Rysunek 4.3: Podschemat funkcji logicznej dla członu M

4.4.2 Enkoder L



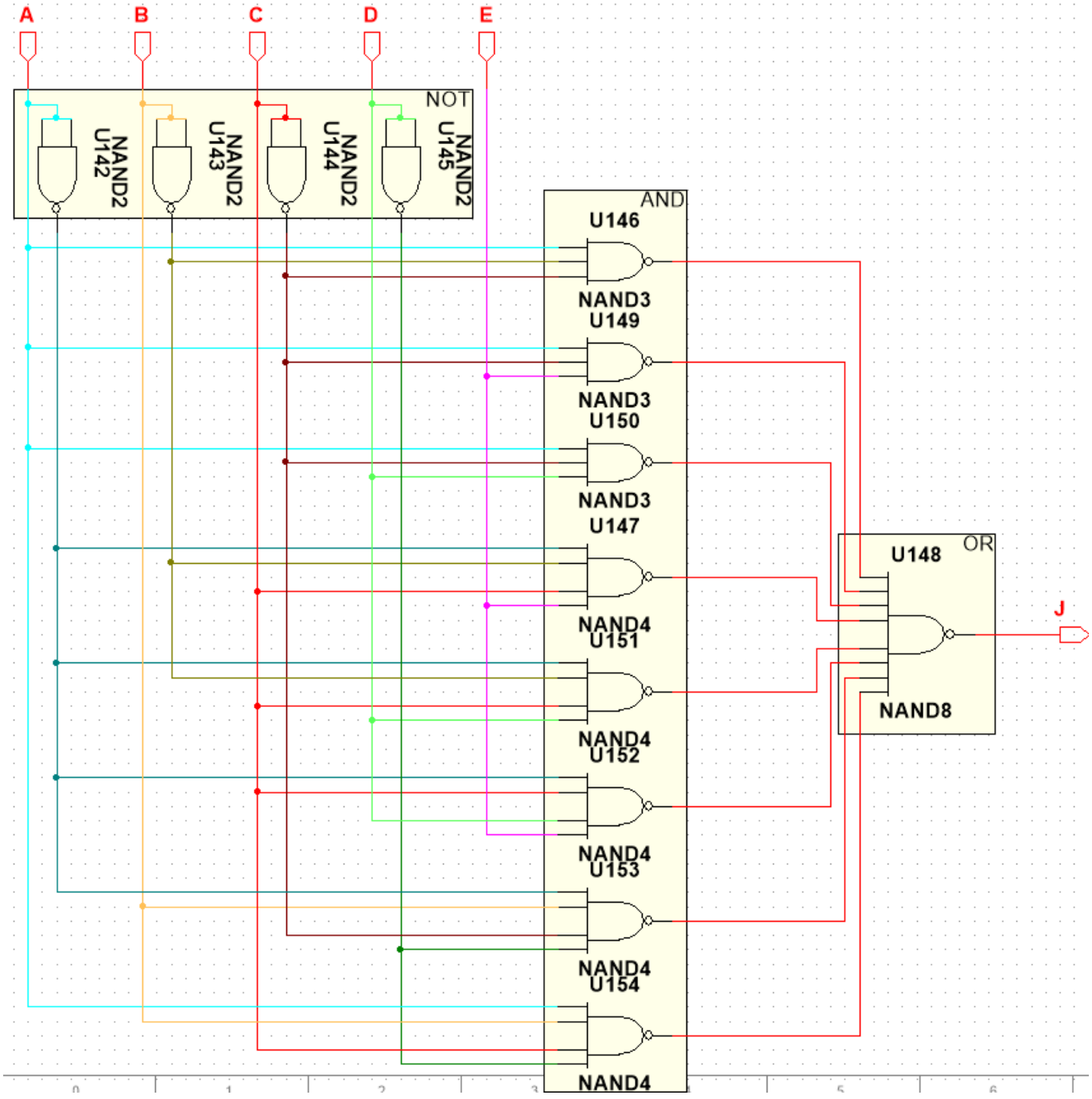
Rysunek 4.4: Podschemat funkcji logicznej dla członu L

4.4.3 Enkoder K



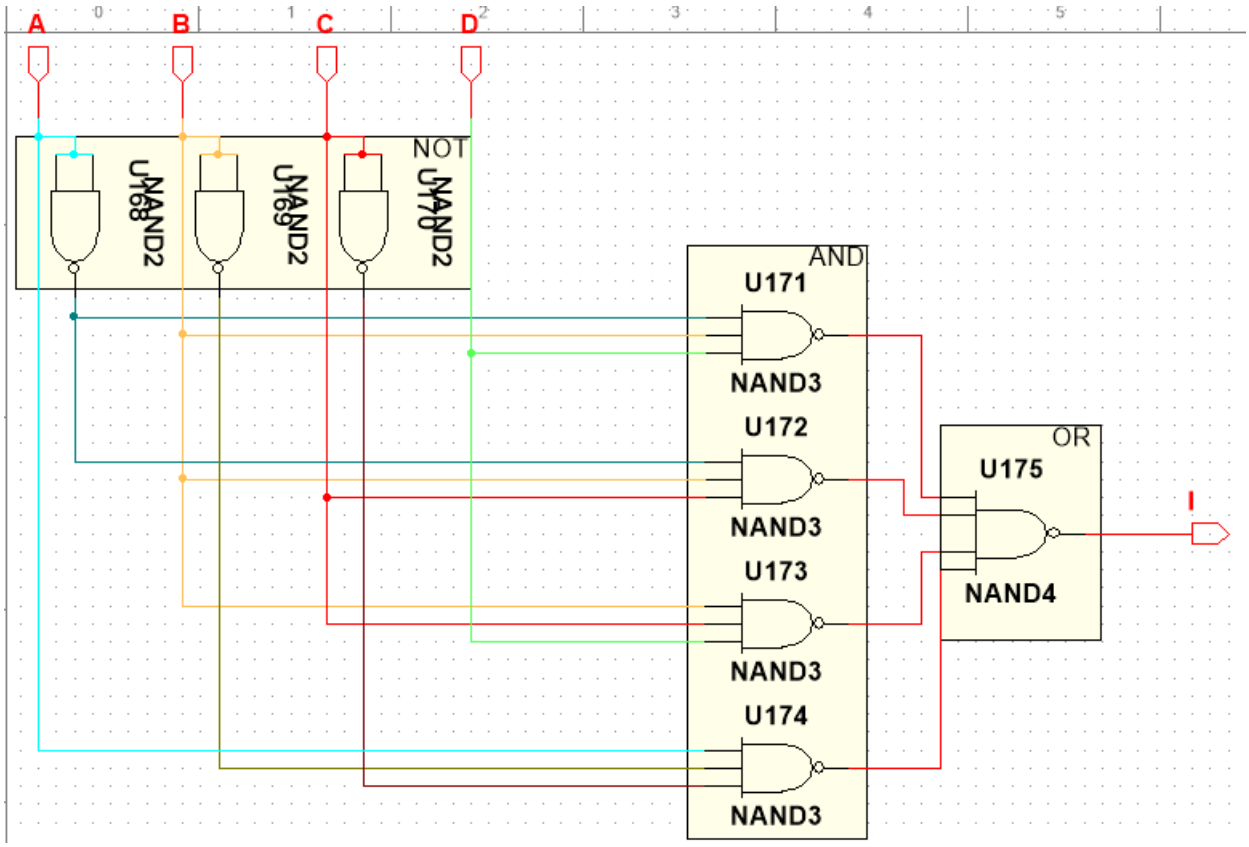
Rysunek 4.5: Podschemat funkcji logicznej dla członu K

4.4.4 Enkoder J



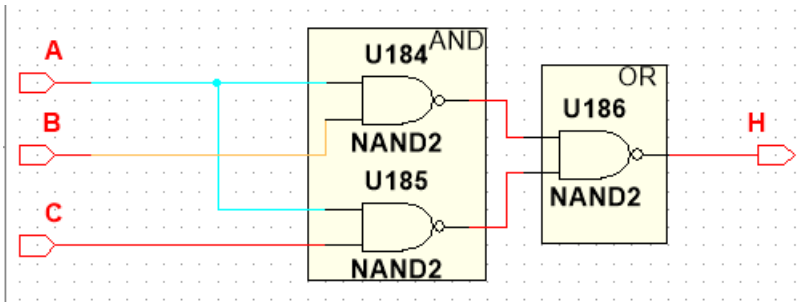
Rysunek 4.6: Podschemat funkcji logicznej dla członu J

4.4.5 Enkoder I



Rysunek 4.7: Podschemat funkcji logicznej dla członu I

4.4.6 Enkoder H



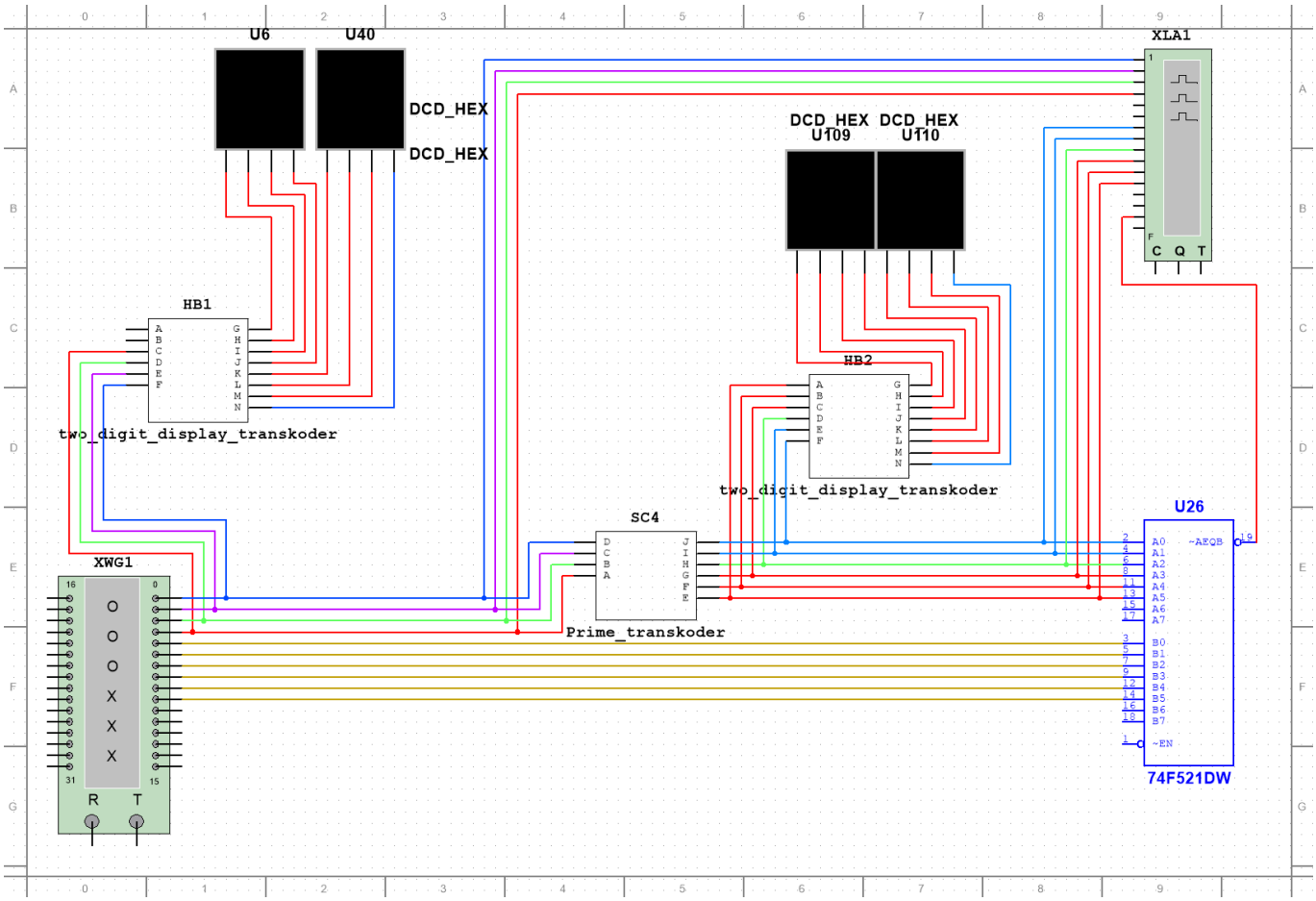
Rysunek 4.8: Podschemat funkcji logicznej dla członu H

5 Układ testowy

Aby przetestować projektowany element zestawiliśmy układ zawierający:

- Word Generator - generuje 4 bity wejściowe dla naszego układu, oraz 6 bitów oczekiwanego wyjścia - zakodowane liczby pierwsze
- Comparator - porównuje wyjście układu z zakodowanym wyjściem z generatora sygnałów

- Logic Analyzer - pozwala na wyświetlenie przebiegów wejścia i wyjścia układu, aby sprawdzić poprawność jego działania
- Wyświetlacze HEX - wyświetlacze 7 segmentowe z enkoderem pozwalające wyświetlić liczby 0-15 jako cyfry szesnastkowe
- Prime transkoder - nasz zaprojektowany układ transkodujący liczby 0-15 na liczby pierwsze
- two digit display transkoder - nasz autorski enkoder BCD, pozwalający lepiej zwizualizować działanie naszego układu



Rysunek 5.1: Schemat całego układu

5.1 Analiza przebiegu

- Word Generator

  Generuje 4 bity wejściowe dla układu, które są następnie przesyłane do Prime Transkodera oraz do Two Digit Display Transkodera.

  Generuje 6 bitów oczekiwanego wyjścia, które są zakodowanymi liczbami

pierwszymi. Te bity są porównywane przez Comparator z wyjściem Prime Transkodera.

- Prime Transkoder

  Otrzymuje 4 bity wejściowe od Word Generator.

  Transkoduje otrzymane bity wejściowe na liczby pierwsze zgodnie z opisaną tabelą.

  Wygenerowane liczby pierwsze są przekazywane do Comparatora w celu porównania z oczekiwanym wyjściem.

- Comperator

  Porównuje wyjście z Prime Transkodera (liczby pierwsze) z zakodowanymi liczbami pierwszymi wygenerowanymi przez Word Generator.

  Jeśli wyjście układu jest zgodne z oczekiwanym wyjściem, generuje sygnał informujący o poprawności działania układu.

- Two Digit Display Transkoder

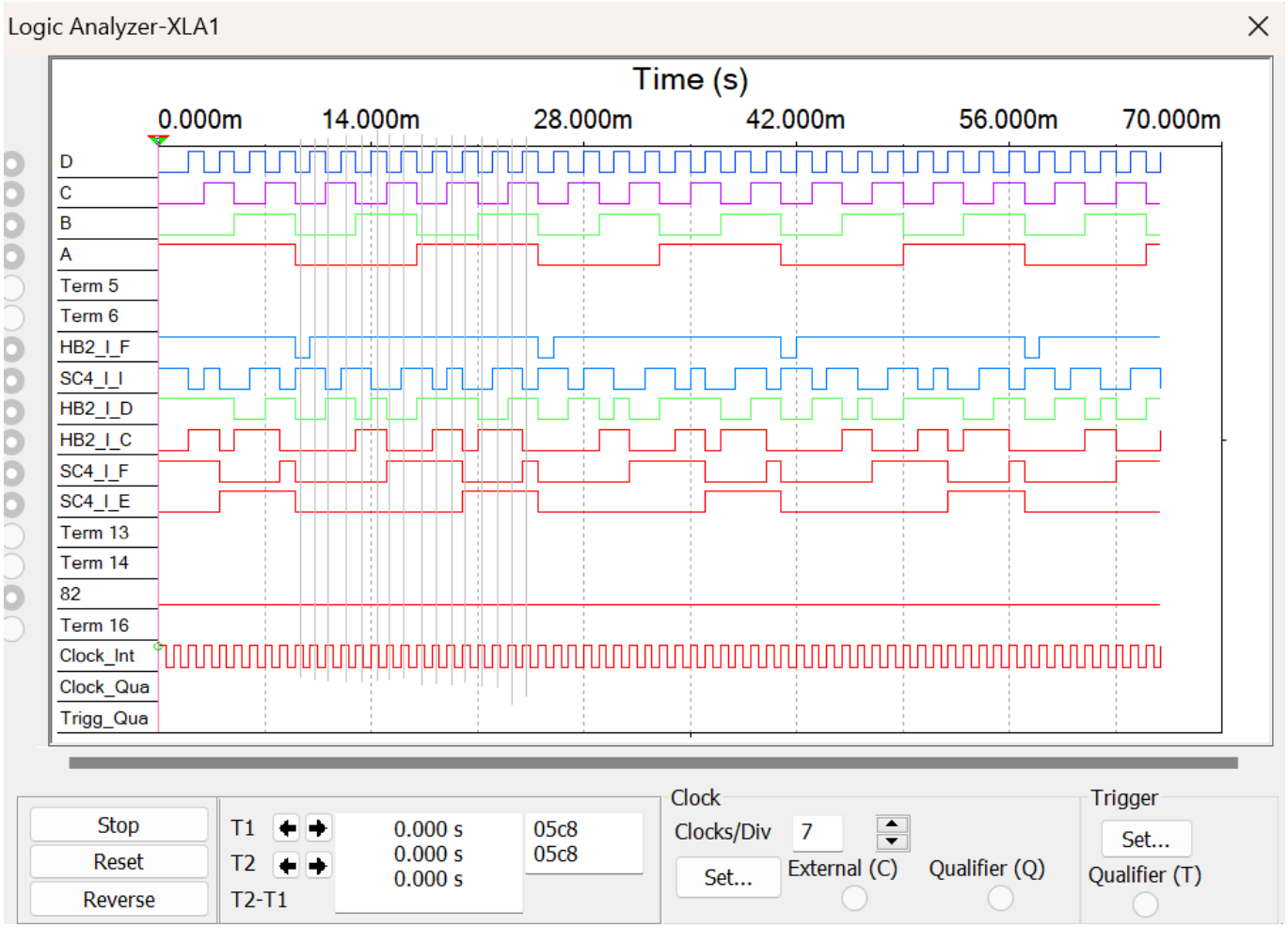
  Pierwszy z nich otrzymuje od Word Generatora 4 bity wejściowe i transkoduje je zgodnie z opisaną tabelą, a następnie przekazuje do wyświetlaczy HEX

  Drugi z nich otrzymuje od Prime Transkodera 6 bitów wejściowych i transkoduje je zgodnie z opisaną tabelą, a następnie przekazuje do wyświetlaczy HEX

- Wyświetlacze HEX

  Pomagają w lepszym zobrazowaniu działania układu poprzez wizualizację zadanych liczb.

- Logic Analyzer



Rysunek 5.2: Zrzut ekranu z analizatora logicznego w trakcie działania układu

Na Analizatorze widać 4 linie wejścia oznaczone A, B, C, D oraz 6 linii wyjścia układu oznaczone jako HB2_I_F (bit J), SC4_I_I (bit I), HB2_I_D (bit H), HB2_I_C (bit G), SC4_I_F (bit F), SC4_I_E (bit E). Dodatkow jako przebieg z etykietą ”82”, wyświetliliśmy wyjście z komparatora - sygnał AeqB - dający stan wysoki gdy strona A i B komparatora są różne i sygnał niski gdy obie strony są równe.

Jak widać na przebiegu sygnał z komparatora jest cały czas niski czyli zachodzi zgodność wyjścia z oczekiwanym wyjściem. Dla potwierdzenia działania układu można odczytać stany poszczególnych wyjść w czasie wzdłuż zaznaczonych przez nas pionowych linii.

6 Zastosowania

Poniżej wymieniamy przykładowe zastosowania zaprojektowanego układu:

- Transkoder generujący liczby pierwsze na podstawie prostej liczby może być zastosowany w urządzeniach szyfrujących. Dużo łatwiej jest wygenerować (pseudo)-losową liczbę z przedziału 0-15 i ją przekształcić na liczbę pierwszą za pomocą takiego transkodera niż wybierać losową liczbę pierwszą. Wiele algorytmów szyfrujących czy funkcji hashujących bazuje na liczbach pierwszych więc taki układ miałby tam zastosowanie.
- Innym zastosowaniem układów z rodziny transkoderów - lecz nie koniecznie tego konkretnego - może być przekształcenie kodu jakiegoś błędu reprezentowanego w systemie przez liczby 0-15 na jakiś inny kod błędu, który np. nadaje się do wyświetlenia użytkownikowi - bo mówi coś więcej o istocie problemu.
- W systemach do zarządzania danymi, zwłaszcza w przypadku baz danych, transkoder przekształcający liczby od 0 do 15 na pierwsze 16 liczb pierwszych może być używany do tworzenia unikalnych identyfikatorów dla rekordów lub kluczy głównych. Jest to przydatne, ponieważ liczby pierwsze są unikalne i trudne do odgadnięcia, co może pomóc w zabezpieczeniu danych przed próbami nieautoryzowanego dostępu lub manipulacji.
- W systemach wizyjnych, gdzie obrazy są przetwarzane i analizowane, transkoder taki może być wykorzystywany do mapowania odcieni kolorów lub innych cech obrazów na liczby pierwsze. Jest to szczególnie przydatne w przypadku algorytmów kompresji obrazu, gdzie zamiast bezpośrednio kodować odcienie kolorów, można przekształcać je na liczby pierwsze, co może zmniejszyć złożoność obliczeniową i objętość danych.

7 Wnioski

Dzięki ćwiczeniu mogliśmy się przekonać o łatwości z jaką bramki NAND mogą zastąpić funkcje logiczne. Jest to ważna cecha tych, że bramek gdyż jest je stosunkowo łatwiej wytworzyć sprzętowo niż klasyczne bramki AND i OR. Istotną cechą bramek NAND jest to, że stanowią one system funkcjonalnie pełny - można przez nimi zapisać wszystkie Boolowskie funkcje logiczne.

Dowód - stosujemy uproszczoną notację $NAND(A, B) \equiv A \uparrow B$:

- $\neg A \equiv A \uparrow A$
- $A \wedge B \equiv \neg(A \uparrow B) \equiv (A \uparrow B) \uparrow (A \uparrow B)$
- $A \vee B \equiv \neg(\neg A \wedge \neg B) \equiv (A \uparrow A) \uparrow (B \uparrow B)$

Kolejnym nasuwającym się wnioskiem jest to, że korzystając z tablic Karnaugh byliśmy w stanie w łatwy i relatywnie szybki sposób stworzyć formuły logiczne zapewniające poprawne funkcjonowanie układu minimalnym kosztem ilości bramek logicznych. Jest to stosunkowo proste narzędzie nawet dla bardzo skomplikowanych układów. Tak naprawdę nie ogranicza nas liczba wyjść układu, lecz liczba jego wejść. Dla układu z 4 bitami wejścia, zadanie było stosunkowo proste i wymagało głównie staranności, lecz tablice dla układu z 6 wejściami było to już dosyć skomplikowane i musieliśmy posłużyć się gotowym narzędziem.