Laboratorium 2
Układ odliczający

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

kwiecień 2024

Spis treści

1 Cel zadania

Korzystając wyłącznie z wybranych przerzutników oraz dowolnych bramek logicznych, proszę zaprojektować czterobitowy układ TIMER, odmierzający ustawiany za pomocą przełączników czas (od 0 do 15).

Po wciśnięciu przycisku STRAT, układ rozpoczyna odmierzanie czasu do tyłu (proszę dobrać częstotliwość tak, aby efekt był dobrze widoczny na ekranie).

Po wyzerowaniu się licznika czasu, układ powinien się zatrzymać i włączyć alarm świetlny wykorzystujący diodę LED. Po ponownym wciśnięciu przycisku START, układ powinien wyłączyć alarm i ponownie rozpocząć odmierzanie ustawionego na przełącznikach czasu. Aktualny wskazywany przez układ czas proszę pokazywać na wyświetlaczach siedmiosegmentowych.

2 Idea rozwiązania

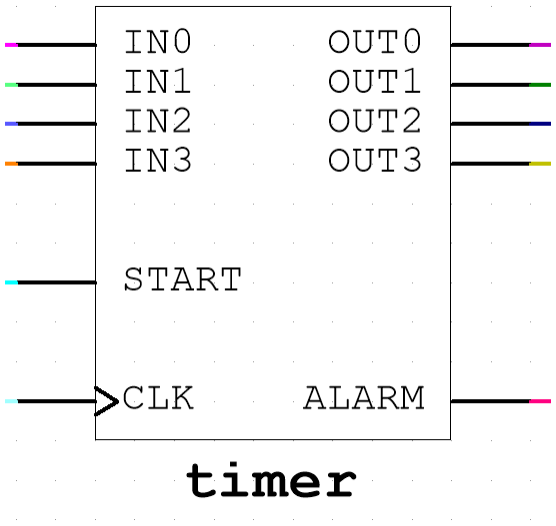
Do rozwiązania zadania wybraliśmy przerzutniki T - z powodu łatwości sterowania takim układem. Do sterowania przerzutnikami w trybie synchronicznym wykorzystaliśmy transkoder uruchamiający wejścia T kolejnych przerzutników na bazie obecnego stanu wyjścia układu. Dodatkowo dodaliśmy pojedynczy sygnał kontrolujący włączenie/wyłączenie układu.

Do początkowego zaprogramowania czasu odliczania na liczniku wykorzystaliśmy możliwość asynchronicznego ustawienia przerzutników w konkretny stan, również tutaj zaprojektowaliśmy transkoder, który porównuje stan przerzutników z wejściem do programowania czasu i odpowiednio ustawia układ.

3 Układ timer

3.1 Black box

Pierwszym krokiem w projektowaniu układu było zaprojektowanie czarnej skrzynki i określenie wejść i wyjść układu.



Rysunek 3.1: Czarna skrzynka timera

Poniżej przedstawimy specyfikację wejść i wyjść układu

3.1.1 Wejścia

- `INx` - wejścia programujące czas odliczania licznika - binarny zapis liczby od której licznik powinien zacząć odliczać. 4 wejścia łącznie pozwalają na odliczanie w zakresie 0-15. `IN0` oznacza najmniej znaczący bit, `IN3` oznacza najbardziej znaczący bit. Wejście jest używane do zaprogramowania w momencie gdy na wejściu `START` pojawi się stan wysoki.

Numer bitu	3	2	1	0
Bit	`IN3`	`IN2`	`IN1`	`IN0`
Mnożnik	2^3	2^2	2^1	2^0

Tabela 3.1: Kodowanie pinów wejściowych

- `START` - wejście aktywujące układ. Stan wysoki oznacza aktywację licznika, stan niski oznacza, że licznik dokończy liczenie do wyzerowania licznika.

  Jeśli wejście `START` będzie miało stan wysoki w czasie dojścia licznika do zera, na wyjściu `ALARM` pojawi się puls, po czym licznik zostanie zaprogramowany obecnym wejściem, a następnie uruchomiony ponownie.

Zmiany stanu na wejściu `START` w czasie gdy licznik jest w stanie liczenia, nie mają żadnego efektu.

- `CLK` - wejście zegara stanowiącego podstawę czasu licznika - określa jak szybko następować będą zmiany wyjścia i odliczanie licznika do zera.

3.1.2 Wyjścia

- `OUTx` - wyjścia stanowiące kolejne bity aktualnego stanu licznika. Zmiana wartości licznika, następuje na wznoszącym zboczu zegara wejściowego. `OUT0` stanowi najmniej znaczący bit, a `OUT3` najbardziej znaczący bit.

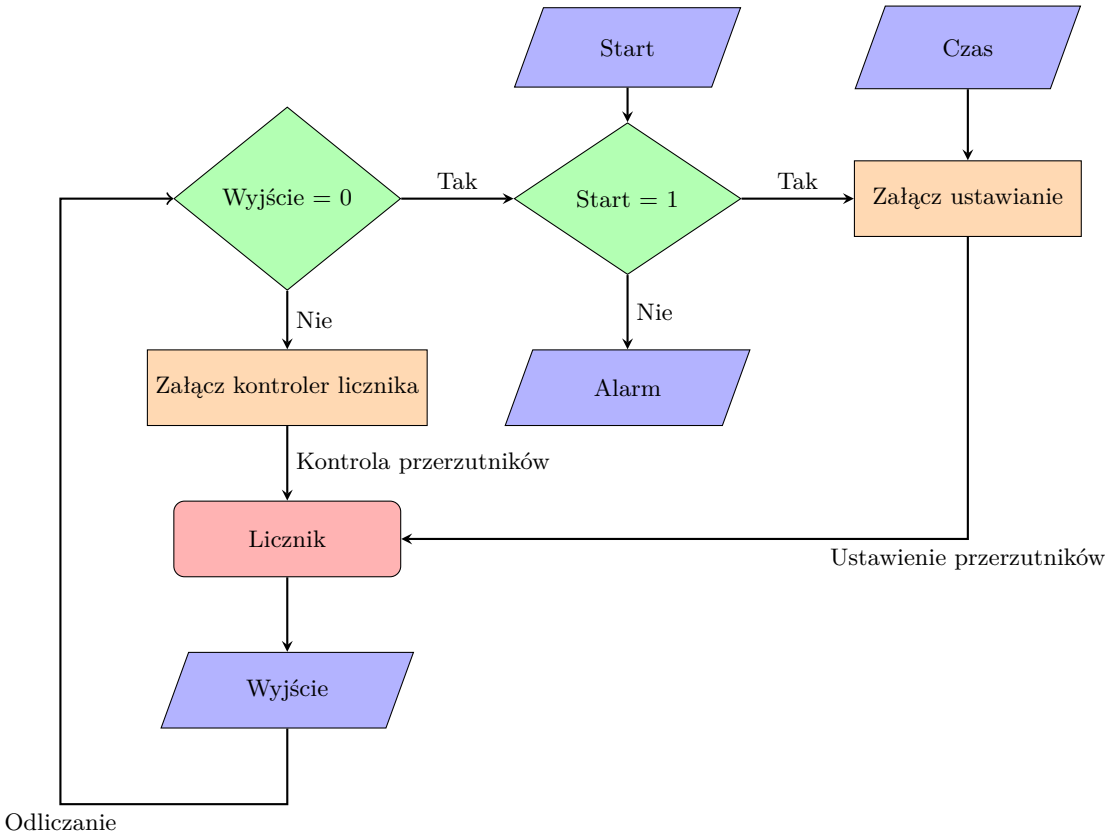
Numer bitu	3	2	1	0
Bit	`OUT3`	`OUT2`	`OUT1`	`OUT0`
Mnożnik	2^3	2^2	2^1	2^0

Tabela 3.2: Kodowanie pinów wejściowych

- `ALARM` - wyjście sygnalizujące zakończenie odliczania licznika. Stan wysoki oznacza, że obecny stan licznika jest równy 0.

3.2 Diagram załączania układów

Poniżej rozpisaliśmy diagram zależności stanu załączenia poszczególnych układów od siebie z którego wynikać będą tabele prawdy.



Rysunek 3.2: Diagram załączania układów

3.3 Kontrola działania licznika

Na najwyższym poziomie nasz układ `timer` składa się z dwóch podukładów: `timer_setter` - układu ustawiającego czas odliczania oraz `timer_driver` układu kontrolującego wejścia T przerzutników. Na tym samym poziomie znajdują się przerzutniki stanowiące faktyczny licznik oraz implementacja formuł załączających te układy opisanych tutaj.

Dla czytelności poniżej przyjmujemy następujące oznaczenia:

- `EN_SET` - wejście aktywujące w układzie `timer_setter`
- `EN_DRV` - wejście aktywujące w układzie `timer_driver`
- `EQ0` - wyjście układu `timer_driver` mówiące o tym czy obecny stan licznika to 0 (stan wysoki).
- `START` - wejście startowe timera

3.3.1 Tablice prawdy

Tabela prawdy wynika z schematu kontroli przedstawionego powyżej.

Wejście		Wyjście	
EQ0	START	EN_SET	EN_DRV
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	0

Tabela 3.3: Tabela prawdy dla stanów aktywacji podukładów

3.3.2 Wyprowadzenie formuł

Dla wyjścia EN_SET możemy odczytać formułę w prost z tabeli:

$$\texttt{EN\_SET} = \texttt{EQ0} \cdot \texttt{START}$$

Dla wyjścia EN_DRV możemy pokusić się o próbę optymalizacji formuły przy pomocy tablicy Karnaugh:

		START	
		0	1
EQ0	0	1	1
	1	0	0

Tabela 3.4: Tablica Karanugh dla formuły aktywującej układ kontrolujący licznik

Możemy z niej odczytać zoptymalizowaną formułę:

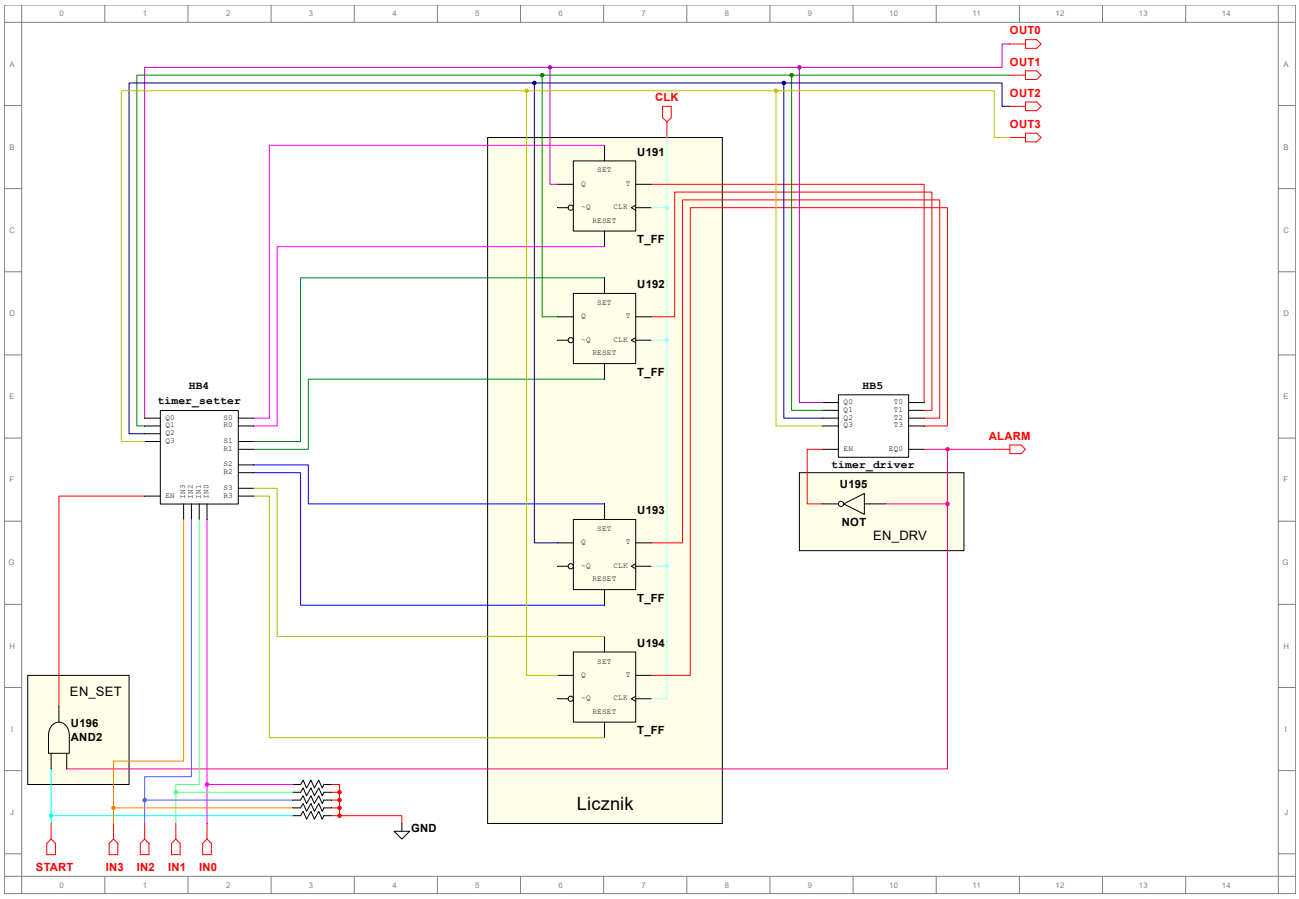
$$\texttt{EN\_DRV} = \overline{\texttt{EQ0}}$$

3.3.3 Realizacja formuł

Poniżej przedstawiamy realizację wcześniej wyprowadzonych formuł:

$$\texttt{EN\_SET} = \texttt{EQ0} \cdot \texttt{START}$$

$$\texttt{EN\_DRV} = \overline{\texttt{EQ0}}$$



Rysunek 3.3: Ogólny schemat timera

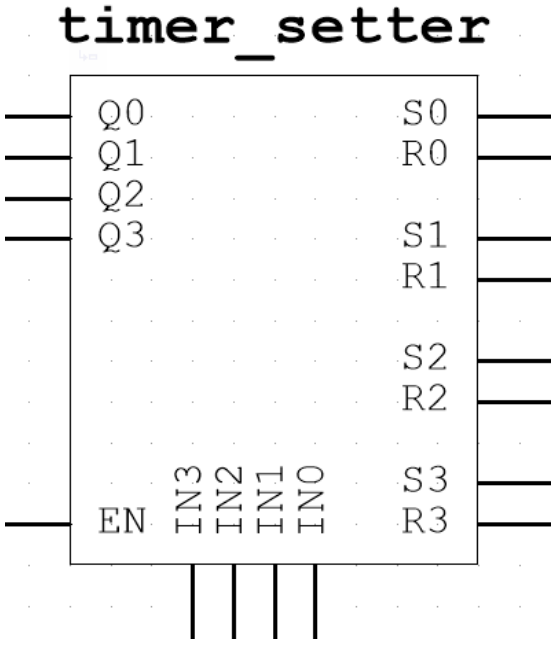
Na schemacie znajdują się również rezystory pull-down zabezpieczające wejścia układu przed nieokreślonym stanem wejść.

3.4 Podukład ustawiający czas licznika

Układ nazwany na naszych schematach `timer_setter` ustawia czas odliczania gdy układ zostanie załączony. Układ ma na celu wysterowanie asynchronicznych wejść przerzutników T poprzez odpowiednie wykonanie operacji SET lub RESET w zależności od obecnego stanu przerzutnika w porównaniu do odpowiadającego bitu programowania.

3.4.1 Wejścia i Wyjścia

Wejścia do układu stanowią bity oznaczające obecny stan poszczególnych wyjść przerzutnika, bity oznaczające stan wejścia programowania timera oraz sygnał załączający układ. Wyjścia natomiast stanowią pary pinów SET i RESET dla poszczególnych przerzutników.



Rysunek 3.4: Czarna skrzynka podukładu timer_setter

Poniżej opis wejść układu:

- `EN` - wejście aktywujące układ, gdy wejście jest w stanie wysokim, na wyjściach układu pojawiają się odpowiednie wartości
- `Qx` - wejścia obecnego stanu licznika, `Q0` stanowi najmniej znaczący bit obecnego stanu licznika, a `Q3` najbardziej znaczący bit.
- `INx` - wejścia programowania startowego stanu licznika, `IN0` stanowi najmniej znaczący bit wejścia, a `IN3` najbardziej znaczący bit.

Poniżej opis wyjść układu:

- `Sx` - wyjście SET ustawiające odpowiedni przerzutnik T, wartość `S0` obliczana jest na podstawie wejść `Q0` i `IN0`, a więc odpowiada ustawieniu przerzutnika T odpowiadającemu najmniej znaczącemu bitowi licznika.
- `Rx` - wyjście RESET resetujący odpowiedni przerzutnik T, wartość `R0` obliczana jest na podstawie wejść `Q0` i `IN0`, a więc odpowiada resetowaniu przerzutnika T odpowiadającemu najmniej znaczącemu bitowi licznika.

3.4.2 Tabele prawdy

Jako że, układ oblicza każdą parę wyjść dokładnie tak samo na podstawie odpowiadających sobie bitów, tabelę prawdy zapiszemy w postaci sparametryzowanej, tzn. parze wyjściowej `Sx`, `Rx` odpowiadają wejścia `INx`, `Qx` oraz sygnał enable. Finalnie ostateczny układ stanowią 4 powtórzone takie formuły dla każdego z bitów 0,1,2,3.

Tabela prawdy wynika z następujących faktów:

- jeśli `EN` $= 0$ to żadne wyjście nie jest aktywne
- jeśli `INx` $=$ `Qx` to nie musimy zmieniać stanu przerzutnika
- w pozostałych przypadków wykonujemy odpowiednio albo operację SET albo RESET

Wejście			Wyjście	
`EN`	`INx`	`Qx`	`Sx`	`Rx`
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	1
1	1	0	1	0
1	1	1	0	0

Tabela 3.5: Tabela prawdy dla układu programującego początkowy stan licznika

3.4.3 Wyprowadzenie formuł

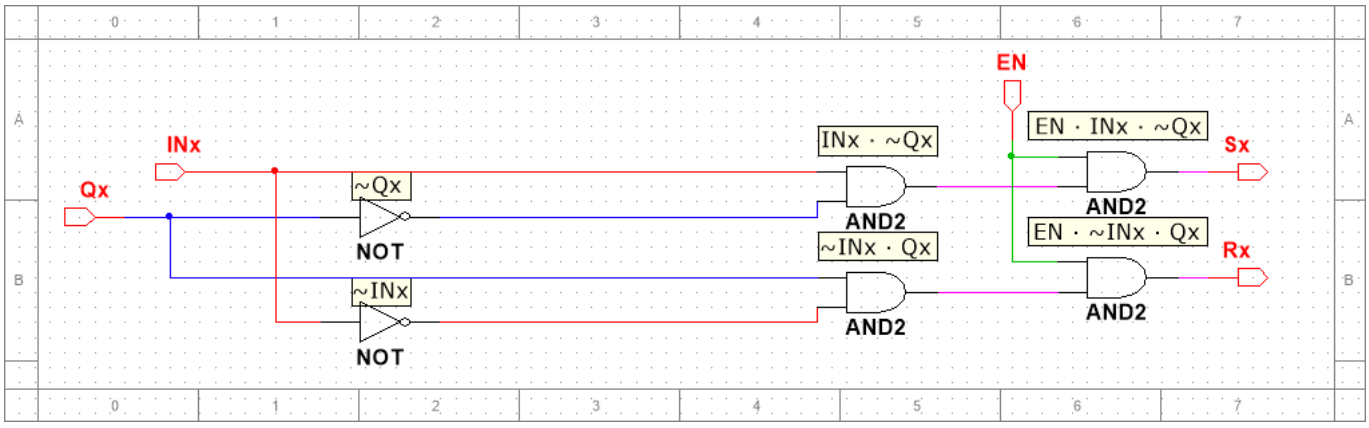
Na podstawie tabeli prawd możemy wyprowadzić formułę na wyjścia `Sx` i `Rx`

$$\texttt{Sx} = \texttt{EN} \cdot \texttt{INx} \cdot \overline{\texttt{Qx}}$$

$$\texttt{Rx} = \texttt{EN} \cdot \overline{\texttt{INx}} \cdot \texttt{Qx}$$

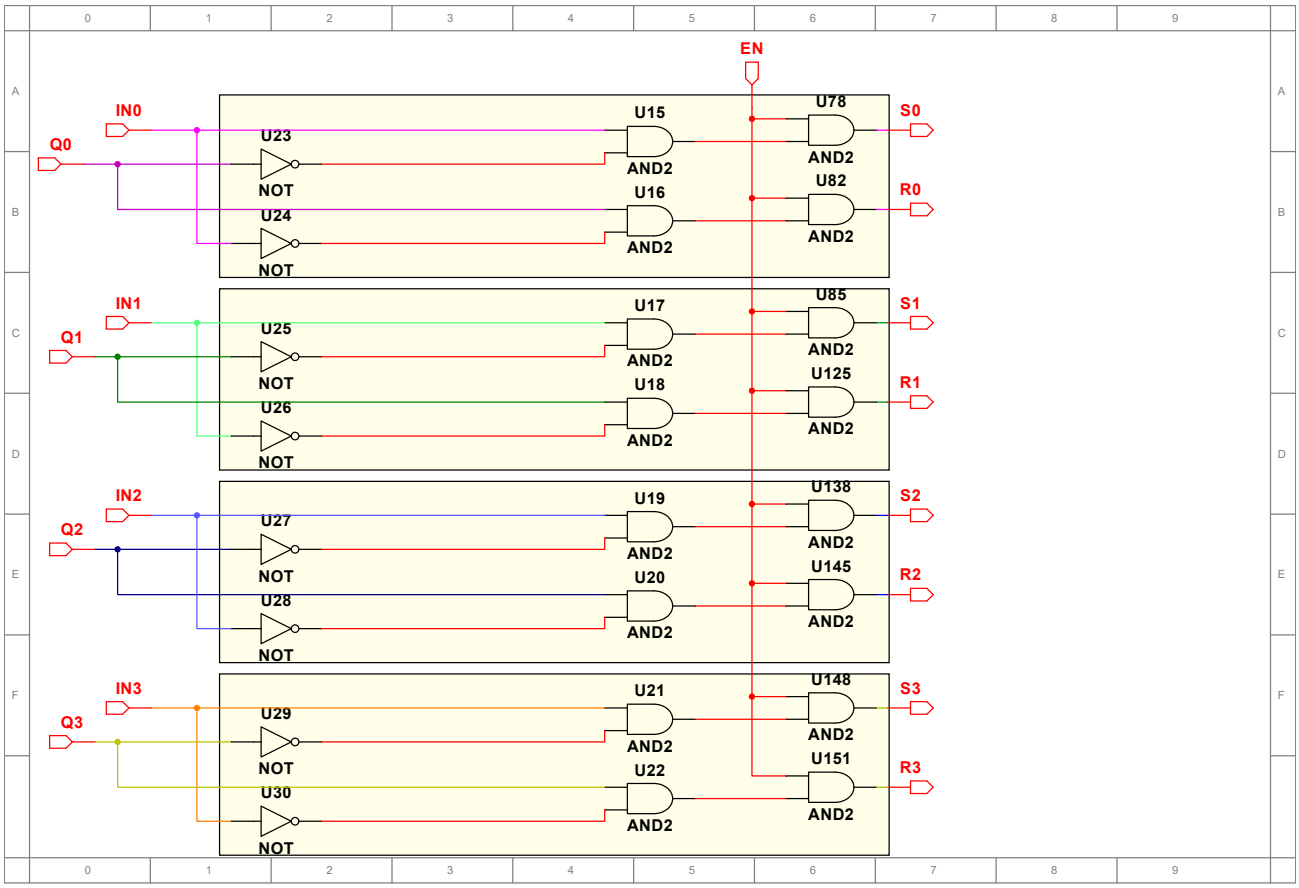
3.4.4 Realizacja formuł

Wyżej wymienione formuły można w multisimie przedstawić w następujący sposób:



Rysunek 3.5: Realizacja funkcji logicznych w Multisimie

Układ `timer_setter` został zaimplementowany jako czterokrotne powielenie powyższej struktury.



Rysunek 3.6: Schemat podukładu: `timer_setter`

3.5 Podukład kontrolujący przerzutniki

Układ nazwany na naszych schematach `timer_driver` kontroluje wejścia T przerzutników.

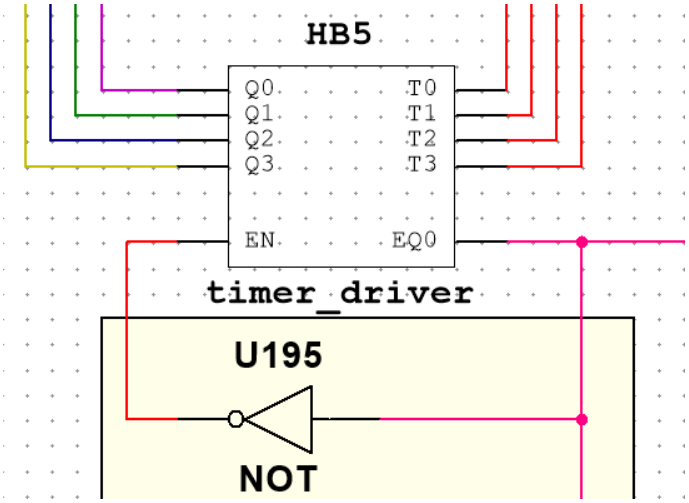
3.5.1 Wejścia i Wyjścia

Opis wejść układu:

- `EN` - Wejście aktywujące układ. Pozwala, aby na wyjściu pojawiały się odpowiednie wartości kiedy jest w stanie wysokim
- `Qx` - wejścia obecnego stanu licznika, `Q0` stanowi najmniej znaczący bit obecnego stanu licznika, a `Q3` najbardziej znaczący bit.

Opis wyjść układu:

- `EQ0` - Wyjście informujące o tym czy obecny stan licznika jest równy 0.
- `Tx` - wyjścia przerzutnika, `T0` stanowi najmniej znaczący bit przerzutnika, a `T3` najbardziej znaczący bit.



Rysunek 3.7: Czarna skrzynka podukładu: `timer_driver`

3.5.2 Tabele prawdy

Tabela prawdy została skonstruowana na podstawie formuły

$$T_x = \texttt{XOR}(Q_{n_x}, Q_{n_x+1})$$

gdzie Q_{n_x} oznacza stan obecny, a Q_{n_x+1} oznacza stan następny. Bierze się to z faktu, że musimy zmienić stan przerzutnika tylko w momencie kiedy obecny stan nie odpowiada stanowi następnemu co odpowiada funkcji `XOR`.

	Stan Obecny				Stan Następny				Przerzutniki			
	Q_{n_3}	Q_{n_2}	Q_{n_1}	Q_{n_0}	Q_{n_3+1}	Q_{n_2+1}	Q_{n_1+1}	Q_{n_0+1}	T_3	T_2	T_1	T_0
15	1	1	1	1	1	1	1	0	0	0	0	1
14	1	1	1	0	1	1	0	1	0	0	1	1
13	1	1	0	1	1	1	0	0	0	0	0	1
12	1	1	0	0	1	1	0	1	0	1	1	1
11	1	1	0	1	1	0	1	0	0	0	0	1
10	1	0	1	0	1	0	0	1	0	0	1	1
9	1	0	0	1	1	0	0	0	0	0	0	1
8	1	0	0	0	0	1	1	1	1	1	1	1
7	0	1	1	1	0	1	1	0	0	0	0	1
6	0	1	1	0	0	1	0	1	0	0	1	1
5	0	1	0	1	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	1	1	0	1	1	1
3	0	0	1	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0	0	1	1
1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	1	1	1	1	1

Tabela 3.6: Tabela prawdy z uwzględnieniem stanu następnego

Uwzględniając wejście `EN` tabela prawdy prezentuje się następująco:

Wejście					Wyjście				
EN	Q3	Q2	Q1	Q0	T3	T2	T1	T0	EQ0
0	1	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
1	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1	1	0
1	1	1	0	1	0	0	0	1	0
1	1	1	0	0	0	1	1	1	0
1	1	1	0	1	0	0	0	1	0
1	1	0	1	0	0	0	1	1	0
1	1	0	0	1	0	0	0	1	0
1	1	0	0	0	1	1	1	1	0
1	0	1	1	1	0	0	0	1	0
1	0	1	1	0	0	0	1	1	0
1	0	1	0	1	0	0	0	1	0
1	0	1	0	0	0	1	1	1	0
1	0	0	1	1	0	0	0	1	0
1	0	0	1	0	0	0	1	1	0
1	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	1	1	1	1

Tabela 3.7: Tabela prawdy dla podukładu: `timer_driver`

3.5.3 Wyprowadzenie formuł

Na podstawie powyższej tabeli stworzyliśmy tablice karnaugh i wyprowadziliśmy formuły

Q3Q2 \ Q1Q0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

EN= 0

Q3Q2 \ Q1Q0	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	0

EN= 1

Tabela 3.8: Tabela Karnaugh dla przerzutnika T3

Możemy z niej odczytać zoptymalizowaną formułę:

$$T3 = \overline{Q2} \cdot \overline{Q1} \cdot \overline{Q0} \cdot EN$$

Q3Q2 \ Q1Q0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

EN= 0

Q3Q2 \ Q1Q0	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	0	0
10	1	0	0	0

EN= 1

Tabela 3.9: Tabela Karnaugh dla przerzutnika T2

Możemy z niej odczytać zoptymalizowaną formułę:

$$T2 = \overline{Q1} \cdot \overline{Q0} \cdot EN$$

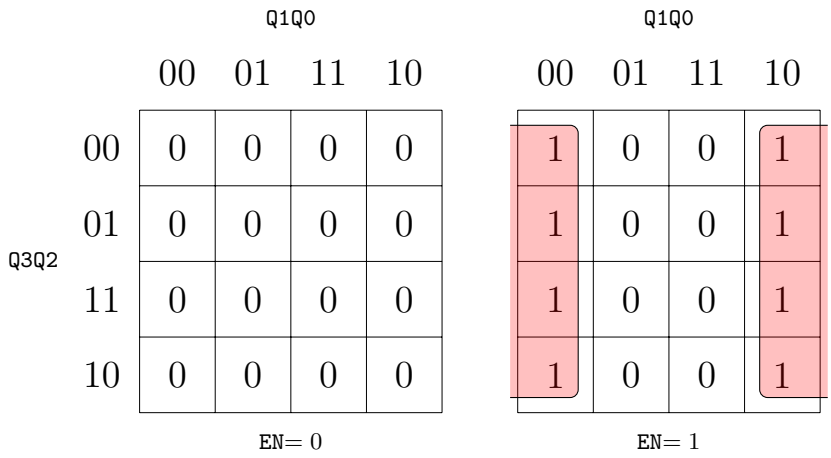


Tabela 3.10: Tabela Karnaugh dla przerzutnika T1

Możemy z niej odczytać zoptymalizowaną formułę:

$$T1 = \overline{Q0} \cdot EN$$

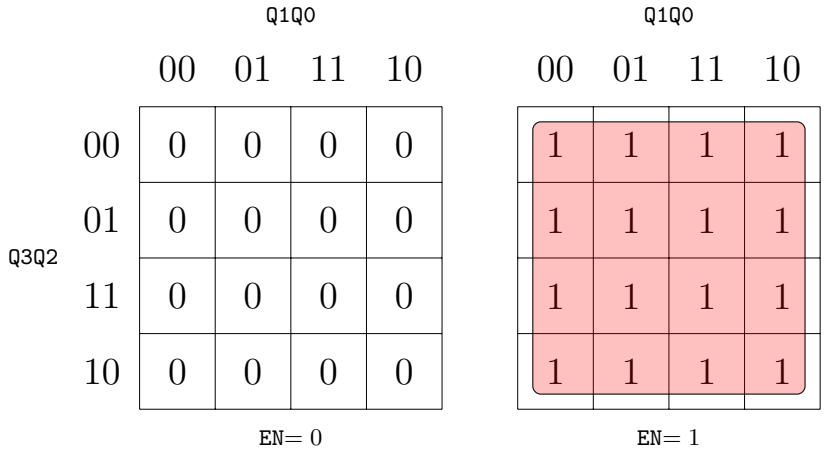


Tabela 3.11: Tabela Karnaugh dla przerzutnika T0

Możemy z niej odczytać zoptymalizowaną formułę:

$$T0 = EN$$

Q3Q2 \ Q1Q0	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

EN= 0

Q3Q2 \ Q1Q0	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

EN= 1

Tabela 3.12: Tabela Karnaugh dla przerzutnika EQ0

Możemy z niej odczytać zoptymalizowaną formułę:

$$EQ0 = \overline{Q0} \cdot \overline{Q1} \cdot \overline{Q2} \cdot \overline{Q3}$$

3.5.4 Realizacja formuł

Układ został stworzony na postawie poniższych formuł

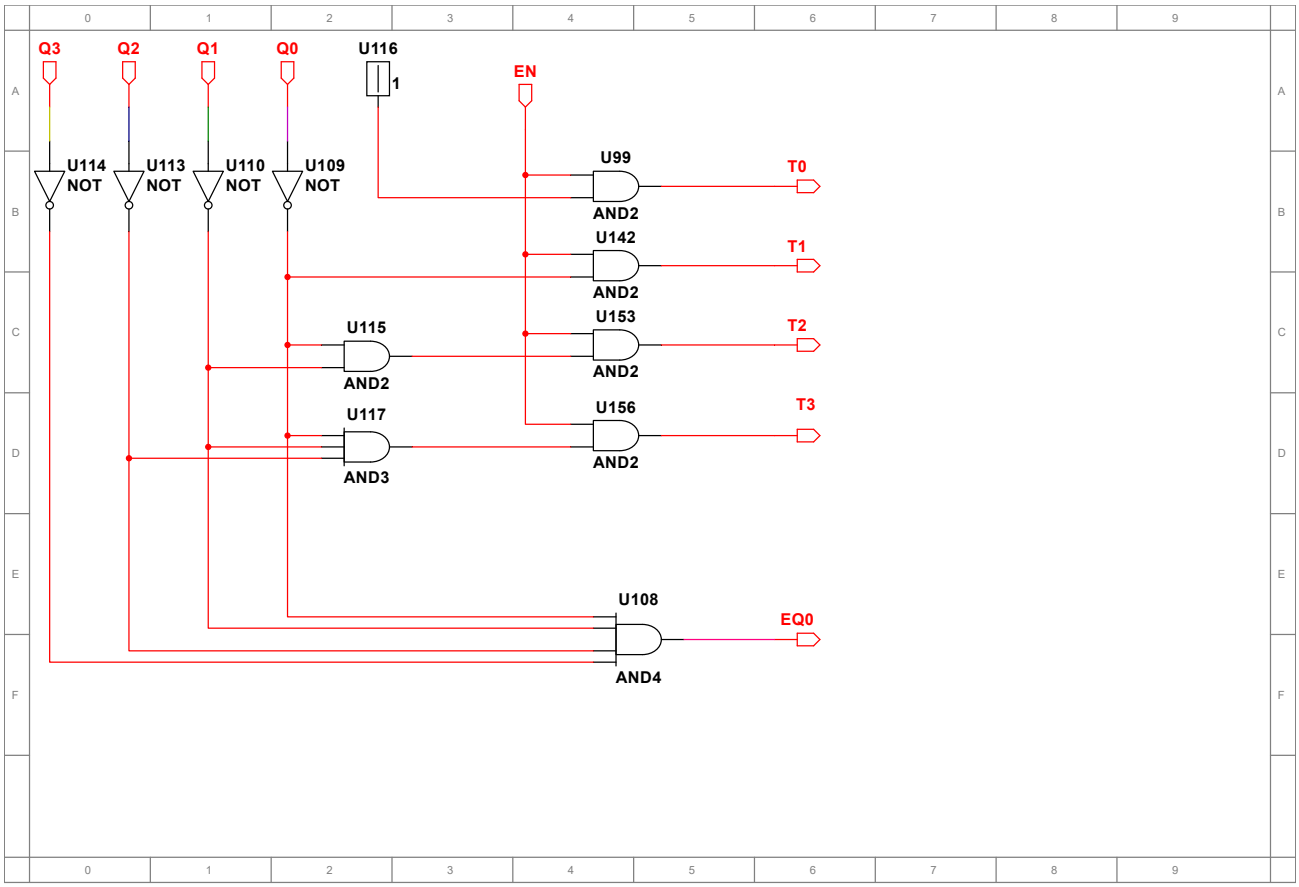
$$T0 = EN$$

$$T1 = \overline{Q0} \cdot EN$$

$$T2 = \overline{Q1} \cdot \overline{Q0} \cdot EN$$

$$T3 = \overline{Q2} \cdot \overline{Q1} \cdot \overline{Q0} \cdot EN$$

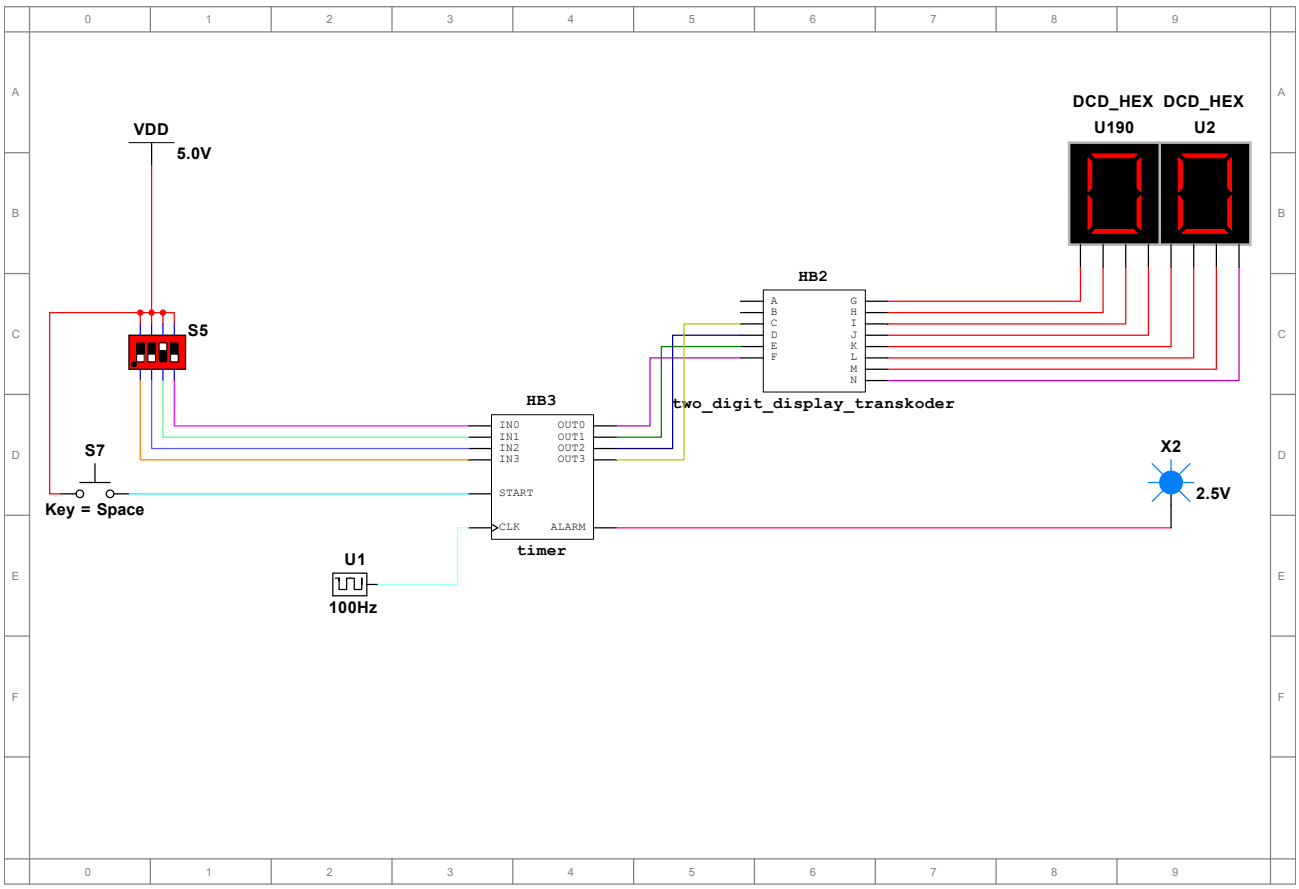
$$EQ0 = \overline{Q0} \cdot \overline{Q1} \cdot \overline{Q2} \cdot \overline{Q3}$$



Rysunek 3.8: Schemat podukładu: `timer_driver`

4 Przykład implementacji układu w obwodzie

Zestawiliśmy przykładowy układ prezentujący jak zaimplementować układ w obwodzie. Użyliśmy transkodera BCD który zaprojektowaliśmy podczas pierwszego laboratorium. Przekształca on 6 bitową liczbę zakodowaną binarnie na dwie 4 bitowe liczby binarne będące cyframi wejściowej liczby.



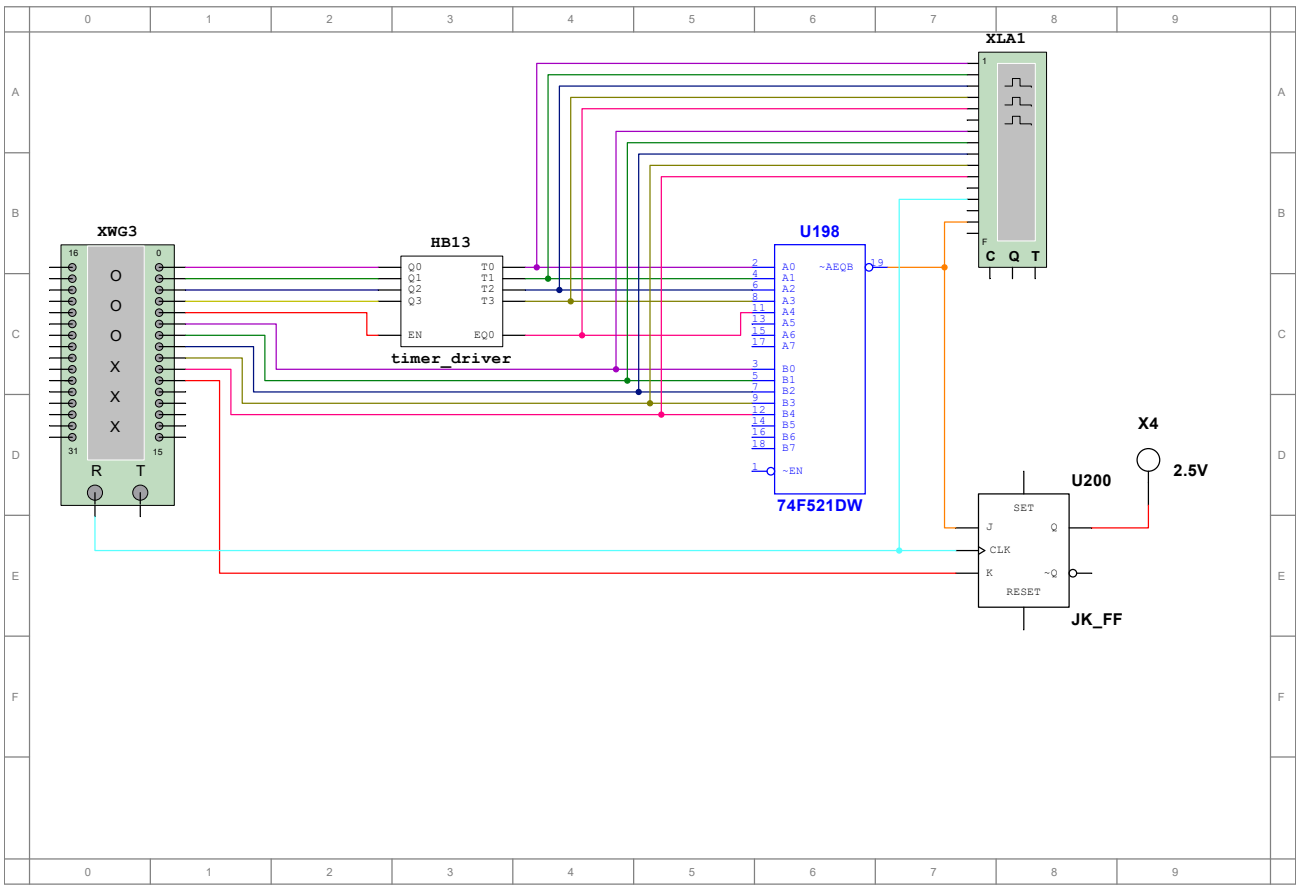
Rysunek 4.1: Przykładowa implementacja obwodu z wykorzystaniem układu `timer`

5 Testy

5.1 Testy podukładów

5.1.1 Testy timer_driver

Aby przetestować układ `timer_driver` użyliśmy układu z generatorem słów który nadaj dane testowe, komparatorem do wykrywania błędów, analizatorem stanów logicznych do przedstawienia przebiegu testu oraz przerzutnikiem JK do sygnalizowania, końcowego wyniku testu na podstawie wszystkich.



Rysunek 5.1: Schemat układu testującego

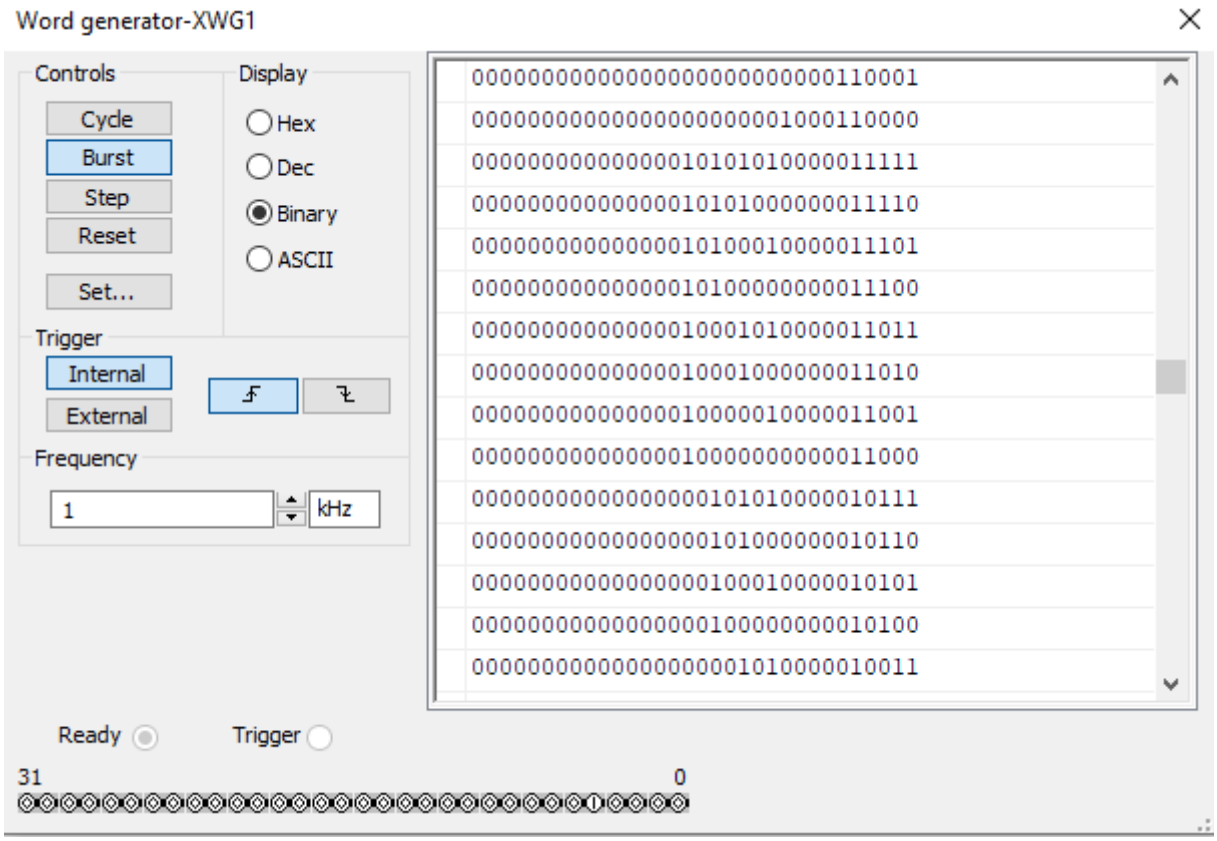
Aby łatwo wygenerować dane testowe napisaliśmy skrypt w języku python, który generuje każdą możliwość danych testowych oraz poprawny dla nich wynik. Kod skryptu:

```
f = open("driver_test_data.dp", "w")
f.write("Data:\n")

################################################################################
# |SR_RESET|EQ0|T3|T2|T1|T0|EN|Q3|Q2|Q1|Q0| #
################################################################################

class TestOutput:
    def __init__(self):
        self.in_data = 0
        self.en = 0
        self.output_data = 0
        self.eq0 = 0
        self.reset_sr = 0

    def to_bin_string(self):
        input_binary = str(bin(self.in_data)).removeprefix("0b").rjust(4, '0')
        en_binary = str(bin(self.en)).removeprefix("0b")
        output_binary = str(bin(self.output_data)).removeprefix("0b").rjust(4, '0')
        eq0_binary = str(bin(self.eq0)).removeprefix("0b")
        reset_sr_binary = str(bin(self.reset_sr)).removeprefix("0b")

        return reset_sr_binary + eq0_binary + output_binary + en_binary + input_binary


    def to_hex_string(self, pad):
        hex_val = hex(int(self.to_bin_string(), 2))
        return hex_val.removeprefix("0x").rjust(pad, '0')


###################################
# Test cycle to reset JK flip flop #
###################################

reset_to = TestOutput()
reset_to.eq0 = 1
reset_to.reset_sr = 1
f.write(reset_to.to_hex_string(8) + "\n")
f.write(reset_to.to_hex_string(8) + "\n")
reset_to.reset_sr = 0
f.write(reset_to.to_hex_string(8) + "\n")
data_count = 3

to = TestOutput()
for input in range(15, -1, -1):
    to.in_data = input
    to.eq0 = 1 if input == 0 else 0

    f.write(to.to_hex_string(8) + "\n")
    data_count += 1

to.en = 1
for input in range(15, -1, -1):
    to.in_data = input
    to.output_data = input ^ 15 if input == 0 else input ^ (input - 1)
    to.eq0 = 1 if input == 0 else 0
```

```
57
58          f.write(to.to_hex_string(8) + "\n")
59          data_count += 1
60
61
62      f.write("Initial:\n")
63      f.write("0000\n")
64      f.write("Final:\n")
65      f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
66
67      f.close()
```

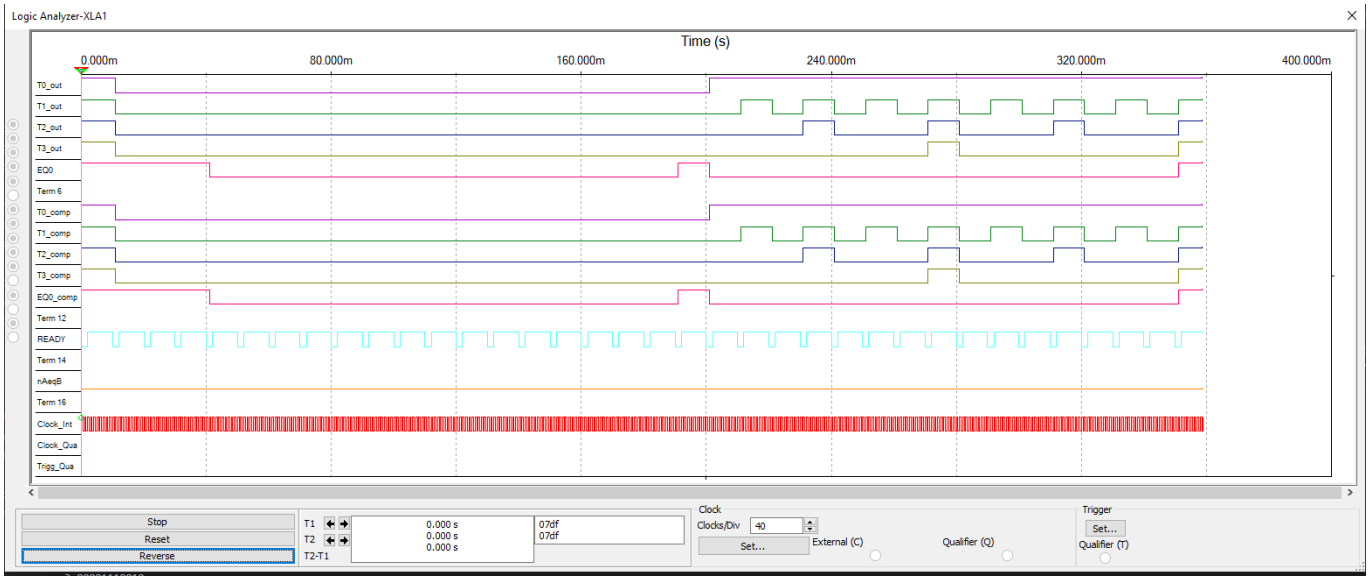
Rysunek 5.2: Skrypt generujący dane do testów, napisany w języku Python

Po zaimportowaniu pliku, który wygenerował skrypt generator słów przedstawia się następująco:



Rysunek 5.3: Dane zaimportowane do generatora słów

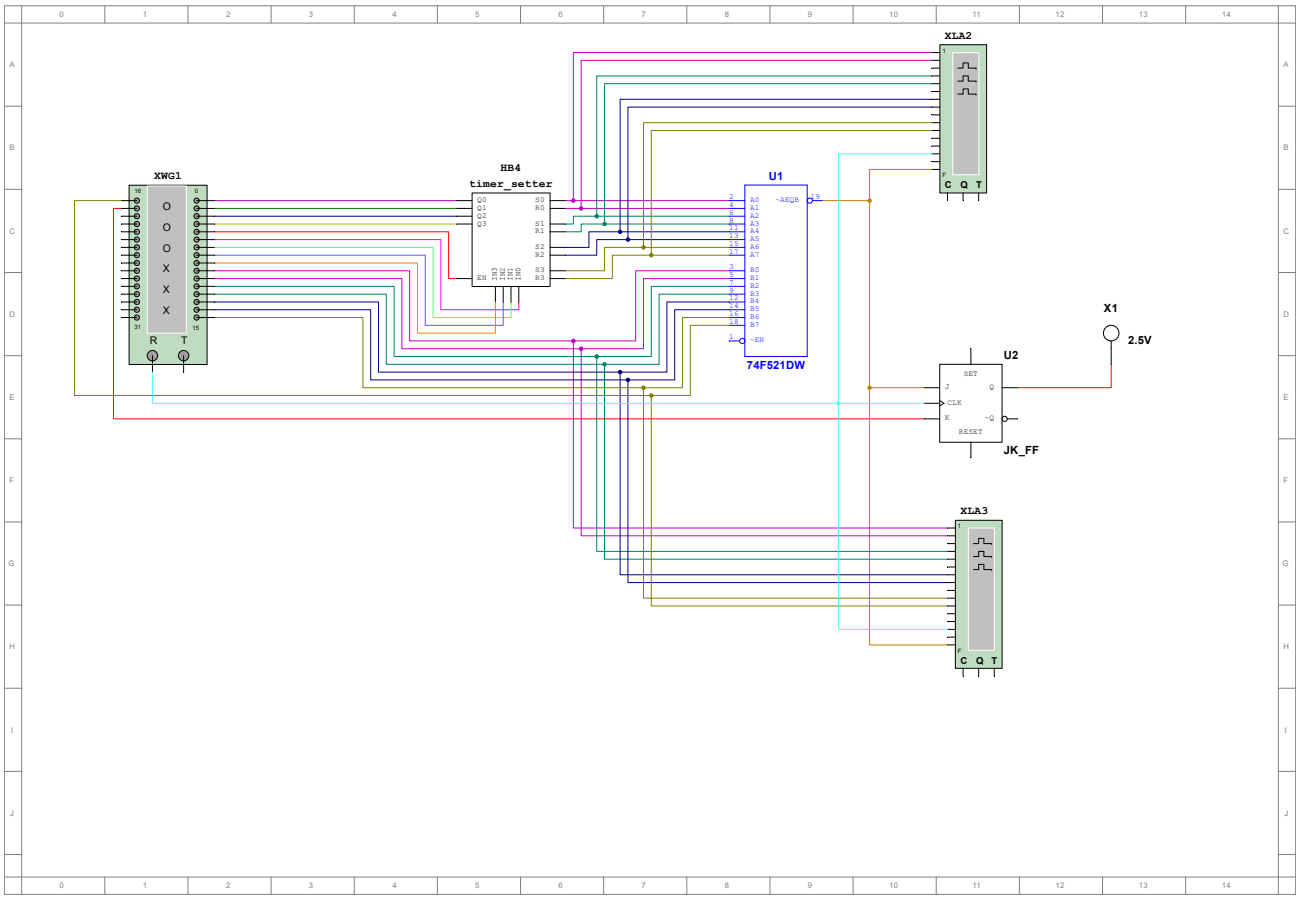
Poniżej wynik testu w postaci przebiegu analizatora stanów logicznych:



Rysunek 5.4: Przebieg sygnałów w analizatorze stanów logicznych podpiętego do wyjścia układu

5.1.2 Testy timer_setter

Układ dla testów układu `timer_setter` jest podobny do poprzedniego układu tylko z różnicą, że z powodu ilości wyjść i wejść musieliśmy użyć dwóch analizatorów stanów logicznych - ich wynik można zsynchronizować na podstawie przebiegu sygnału READY oraz czasu rozpoczęcia i zakończenia testu.



Rysunek 5.5: Schemat układu testującego

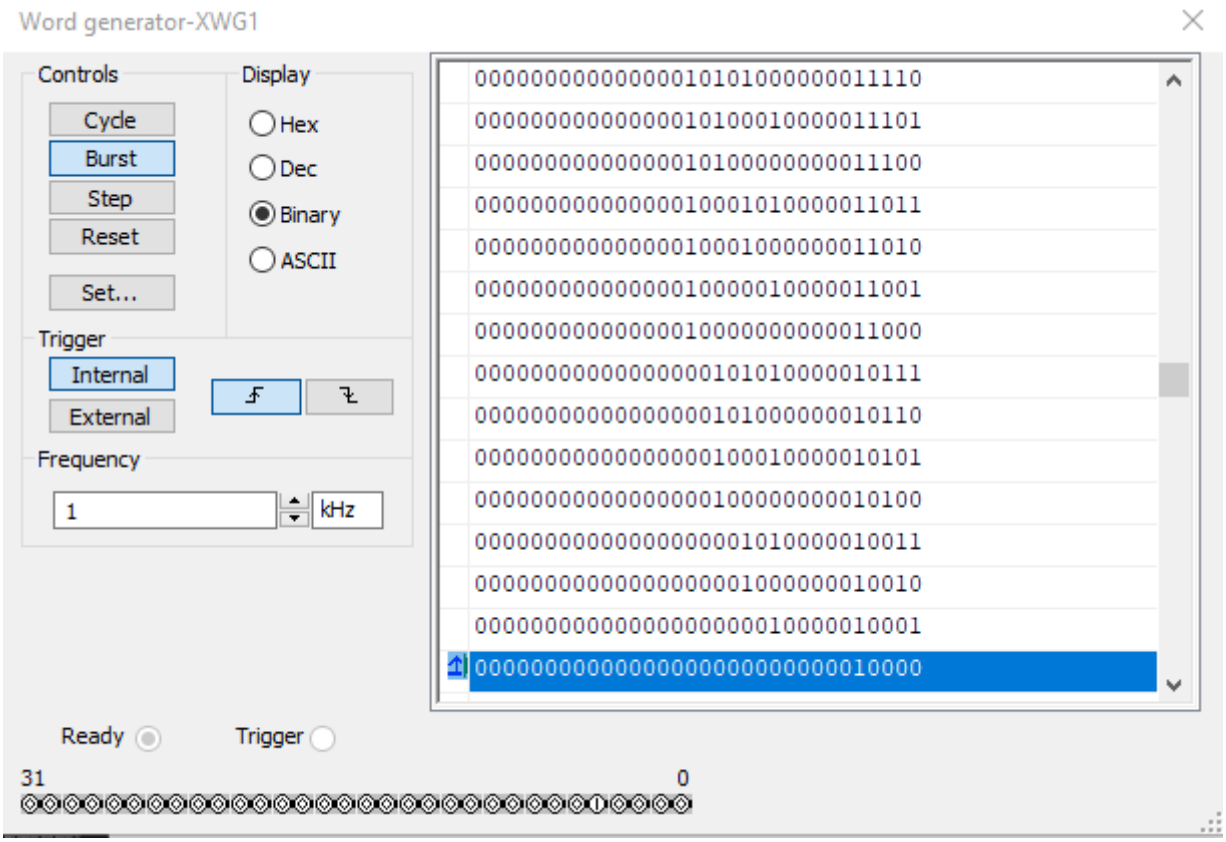
Podobnie jak wcześniej napisaliśmy skrypt generujący dane:

```
    f = open("setter_test_data.dp", "w")
    f.write("Data:\n")

    ################################################################################
    # |SR_RESET|S3|R3|S2|R2|S1|R1|S0|R0|IN3|IN2|IN1|IN0|EN|Q3|Q2|Q1|Q0| #
    ################################################################################

    class TestOutput:
        def __init__(self):
            self.q_in_data = 0
            self.en = 0
            self.in_data = 0

            self.s0 = 0
            self.r0 = 0
            self.s1 = 0
            self.r1 = 0
            self.s2 = 0
            self.r2 = 0
            self.s3 = 0
            self.r3 = 0

            self.reset_sr = 0

        def to_bin_string(self):
            q_input_binary = str(bin(self.q_in_data)).removeprefix("0b").rjust(4, '0')
            en_binary = str(bin(self.en)).removeprefix("0b")
            input_binary = str(bin(self.in_data)).removeprefix("0b").rjust(4, '0')

            s0_binary = str(bin(self.s0)).removeprefix("0b")
            r0_binary = str(bin(self.r0)).removeprefix("0b")
            s1_binary = str(bin(self.s1)).removeprefix("0b")
            r1_binary = str(bin(self.r1)).removeprefix("0b")
            s2_binary = str(bin(self.s2)).removeprefix("0b")
            r2_binary = str(bin(self.r2)).removeprefix("0b")
            s3_binary = str(bin(self.s3)).removeprefix("0b")
            r3_binary = str(bin(self.r3)).removeprefix("0b")

            reset_sr_binary = str(bin(self.reset_sr)).removeprefix("0b")

            return (reset_sr_binary +
                    r3_binary +
                    s3_binary +
                    r2_binary +
                    s2_binary +
                    r1_binary +
                    s1_binary +
                    r0_binary +
                    s0_binary +
                    input_binary +
                    en_binary +
                    q_input_binary)


        def to_hex_string(self, pad):
            hex_val = hex(int(self.to_bin_string(), 2))
            return hex_val.removeprefix("0x").rjust(pad, '0')


```

```
    ###################################
    # Test cycle to reset JK flip flop #
    ###################################

    reset_to = TestOutput()
    reset_to.reset_sr = 1
    f.write(reset_to.to_hex_string(8) + "\n")
    f.write(reset_to.to_hex_string(8) + "\n")
    reset_to.reset_sr = 0
    f.write(reset_to.to_hex_string(8) + "\n")
    data_count = 3

    to = TestOutput()
    for input in range(15, -1, -1):
        for q_input in range(15, -1, -1):
            to.in_data = input
            to.q_input = q_input


            f.write(to.to_hex_string(8) + "\n")
            data_count += 1

    to.en = 1
    for input in range(15, -1, -1):
        for q_input in range(15, -1, -1):
            to.in_data = input
            to.q_in_data = q_input

            in_bin = bin(input).removeprefix("0b").rjust(4, '0')
            q_in_bin = bin(q_input).removeprefix("0b").rjust(4, '0')

            to.s0 = 1 if in_bin[3] == "1" and q_in_bin[3] == "0" else 0
            to.r0 = 1 if in_bin[3] == "0" and q_in_bin[3] == "1" else 0

            to.s1 = 1 if in_bin[2] == "1" and q_in_bin[2] == "0" else 0
            to.r1 = 1 if in_bin[2] == "0" and q_in_bin[2] == "1" else 0

            to.s2 = 1 if in_bin[1] == "1" and q_in_bin[1] == "0" else 0
            to.r2 = 1 if in_bin[1] == "0" and q_in_bin[1] == "1" else 0

            to.s3 = 1 if in_bin[0] == "1" and q_in_bin[0] == "0" else 0
            to.r3 = 1 if in_bin[0] == "0" and q_in_bin[0] == "1" else 0

            f.write(to.to_hex_string(8) + "\n")
            data_count += 1


    f.write("Initial:\n")
    f.write("0000\n")
    f.write("Final:\n")
    f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))

    f.close()
```

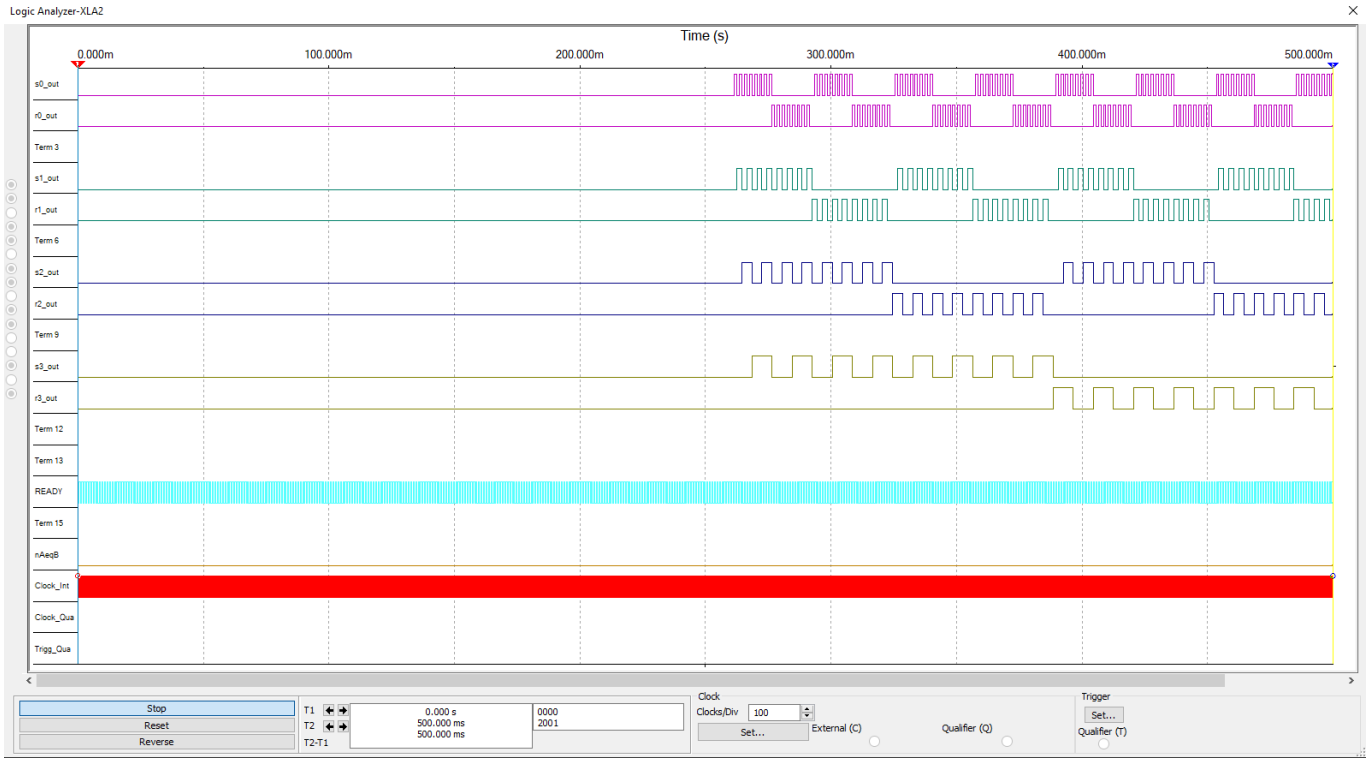
Rysunek 5.6: Skrypt generujący dane do testów, napisany w języku Python

Po zaimportowaniu pliku, który wygenerował skrypt generator słów przedstawia się następująco:



Rysunek 5.7: Dane zaimportowane do generatora słów

Poniżej przebieg obu analizatorów stanów logicznych:

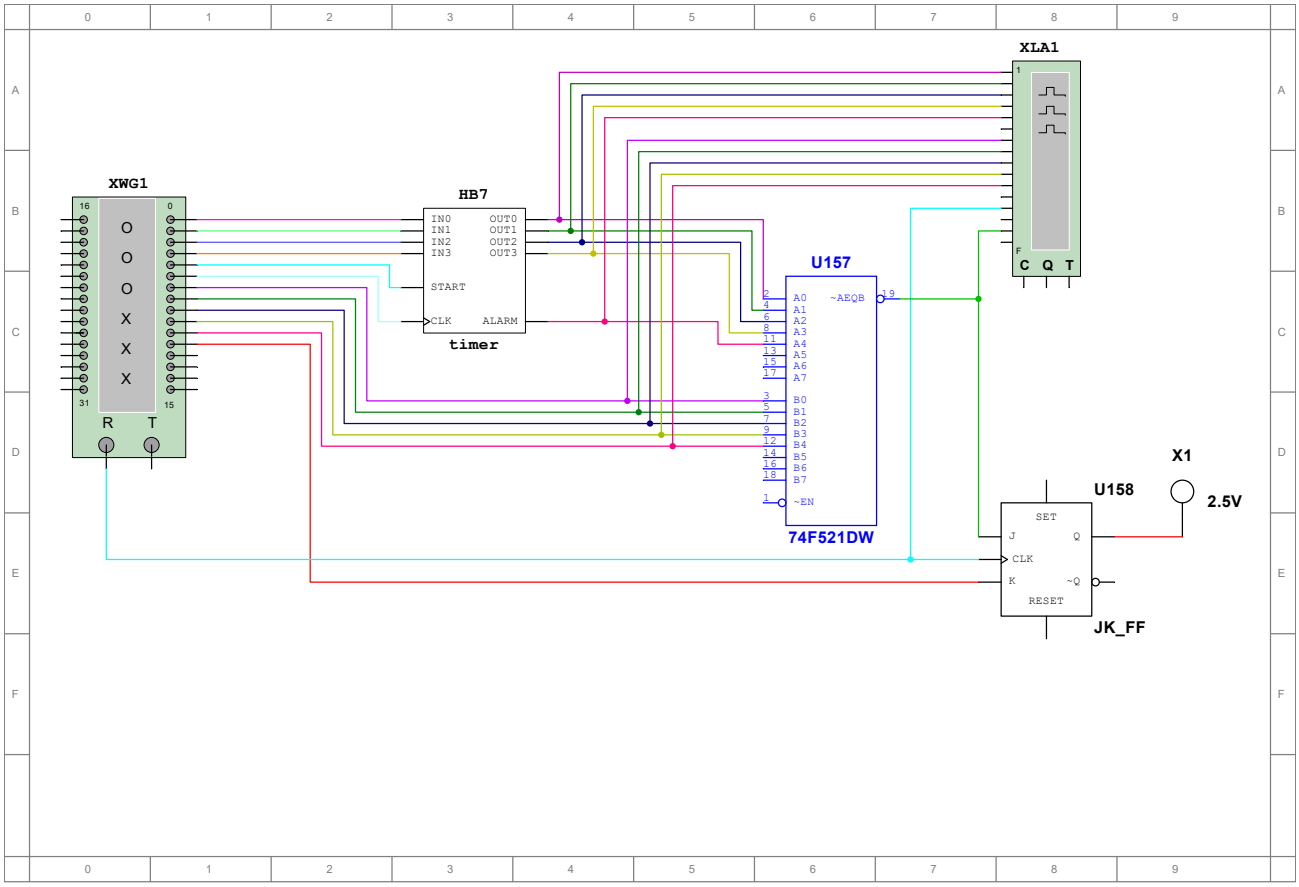


Rysunek 5.8: Przebieg sygnałów w analizatorze stanów logicznych podpiętego do wyjścia układu

Rysunek 5.9: Przebieg sygnałów w analizatorze stanów logicznych podpiętego do wyjścia danych porównawczych

5.2 Test timera

Zdecydowaliśmy się przeprowadzić ogólny test układu timer - na najwyższym poziomie abstrakcji. W tym celu zestawiliśmy układ składający się z generatora słów, naszego układu `timer`, komparatora oraz analizatora stanów logicznych. Generator słów obsługuje nasz timer zapewniając mu cykle zegara w kolejnych generowanych słowach oraz generujący dane porównawcze dla każdego cyklu.



Rysunek 5.10: Schemat układu testującego

Zastosowaliśmy przerzutnik JK próbkujący wyjście komparatora, gdy generator słów generuje wyjście R(READY) oznaczające, że skończył on generować dane słowo. Pojawienie się stanu wysokiego na wejściu J przerzutnika JK powoduje jego ustawienie, a pojawienie się stanu wysokiego na wyjściu K resetuje ten przerzutnik.
Do wygenerowanie pliku wejściowego dla generatora słów, napisaliśmy skrypt w języku python, który symuluje kolejne kroki odliczania timera w zmiennych i zapisuje je do pliku w formacie, który można zaimportować do multisima.

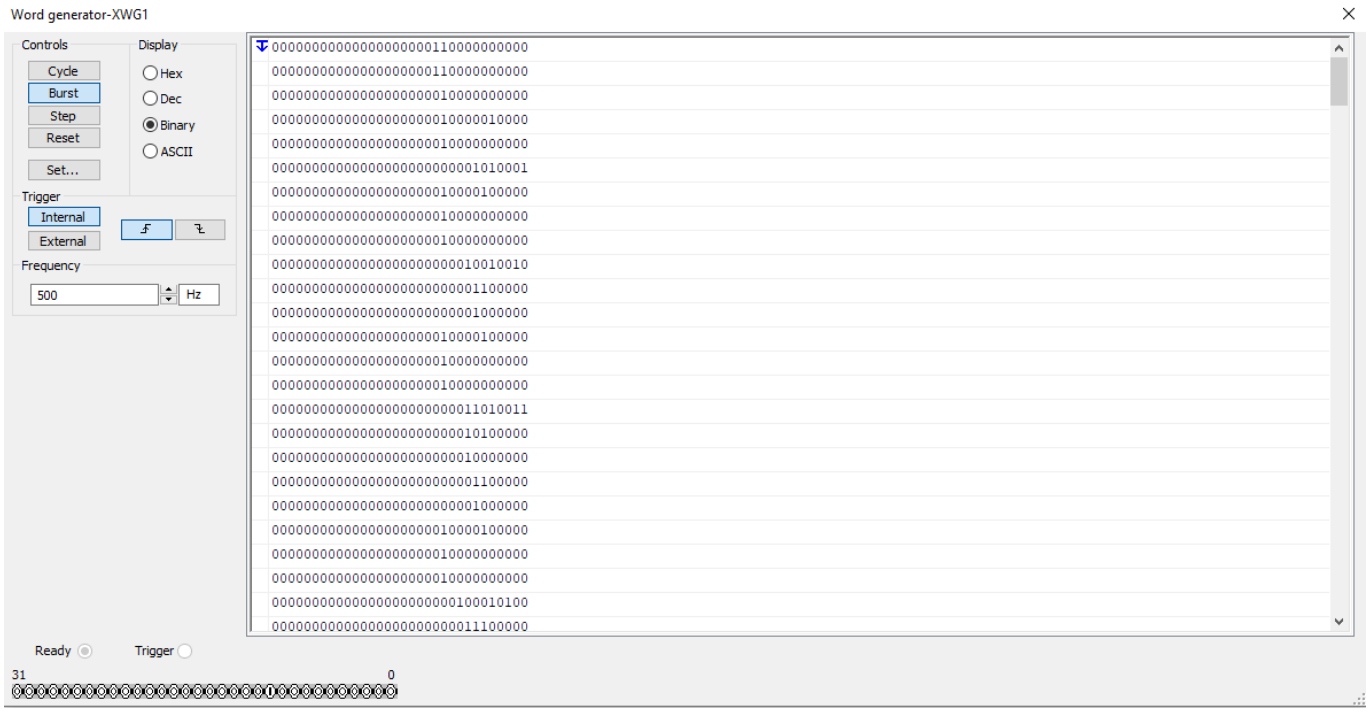
Poniżej kod skryptu generującego dane:

```
f = open("test_data.dp", "w")
f.write("Data:\n")

######################################################################
# |SR_RESET|ALARM|OUT3|OUT2|OUT1|OUT0|CLK|START|IN3|IN2|IN1|IN0| #
######################################################################

class TestOutput:
    def __init__(self):
        self.in_data = 0
        self.start = 0
        self.clk = 0
        self.output_data = 0
        self.alarm = 0
        self.reset_sr = 0

    def to_bin_string(self):
        input_binary = str(bin(self.in_data)).removeprefix("0b").rjust(4, '0')
        start_binary = str(bin(self.start)).removeprefix("0b")
        clk_binary = str(bin(self.clk)).removeprefix("0b")
        output_binary = str(bin(self.output_data)).removeprefix("0b").rjust(4, '0')
        alarm_binary = str(bin(self.alarm)).removeprefix("0b")
        reset_sr_binary = str(bin(self.reset_sr)).removeprefix("0b")

        return reset_sr_binary + alarm_binary + output_binary + clk_binary + start_binary + input_binary


    def to_hex_string(self, pad):
        hex_val = hex(int(self.to_bin_string(), 2))
        return hex_val.removeprefix("0x").rjust(pad, '0')


###################################
# Test cycle to reset JK flip flop #
###################################

reset_to = TestOutput()
reset_to.alarm = 1
reset_to.reset_sr = 1
f.write(reset_to.to_hex_string(8) + "\n")
f.write(reset_to.to_hex_string(8) + "\n")
reset_to.reset_sr = 0
f.write(reset_to.to_hex_string(8) + "\n")
data_count = 3

for i in range(16):
    to = TestOutput()
    to.in_data = i
    to.start = 1
    to.output_data = i
    to.alarm = int(i == 0)

    f.write(to.to_hex_string(8) + "\n")
    data_count += 1

    to.in_data = 0
    to.start = 0
    to.alarm = int(i == 0)

```

```
60      for k in range(i):
61          to.clk = 1
62          to.output_data -= 1
63          if to.output_data == 0:
64              to.alarm = 1
65
66          f.write(to.to_hex_string(8) + "\n")
67          data_count += 1
68
69          to.clk = 0
70
71          f.write(to.to_hex_string(8) + "\n")
72          data_count += 1
73
74      f.write(to.to_hex_string(8) + "\n")
75      data_count += 1
76
77  f.write("Initial:\n")
78  f.write("0000\n")
79  f.write("Final:\n")
80  f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
81
82  f.close()
```

Rysunek 5.11: Skrypt generujący dane do testów, napisany w języku Python

Po zaimportowaniu danych do generatora słów przedstawia się on następująco:



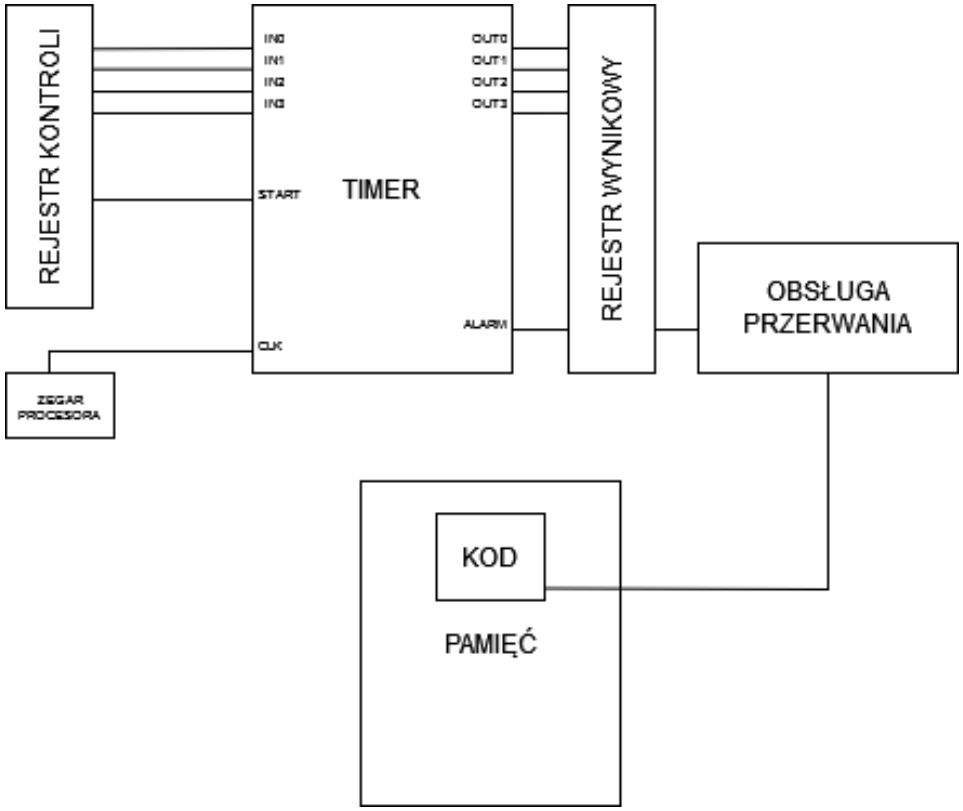
Rysunek 5.12: Dane zaimportowane do generatora słów

W wyniku testu otrzymujemy informację poprzez zaświecenie się lampki jeśli gdziekolwiek wystąpił błąd oraz przebieg sygnałów z analizatora stanów logicznych:

Rysunek 5.13: Przebieg sygnałów w analizatorze stanów logicznych

6 Zastosowania

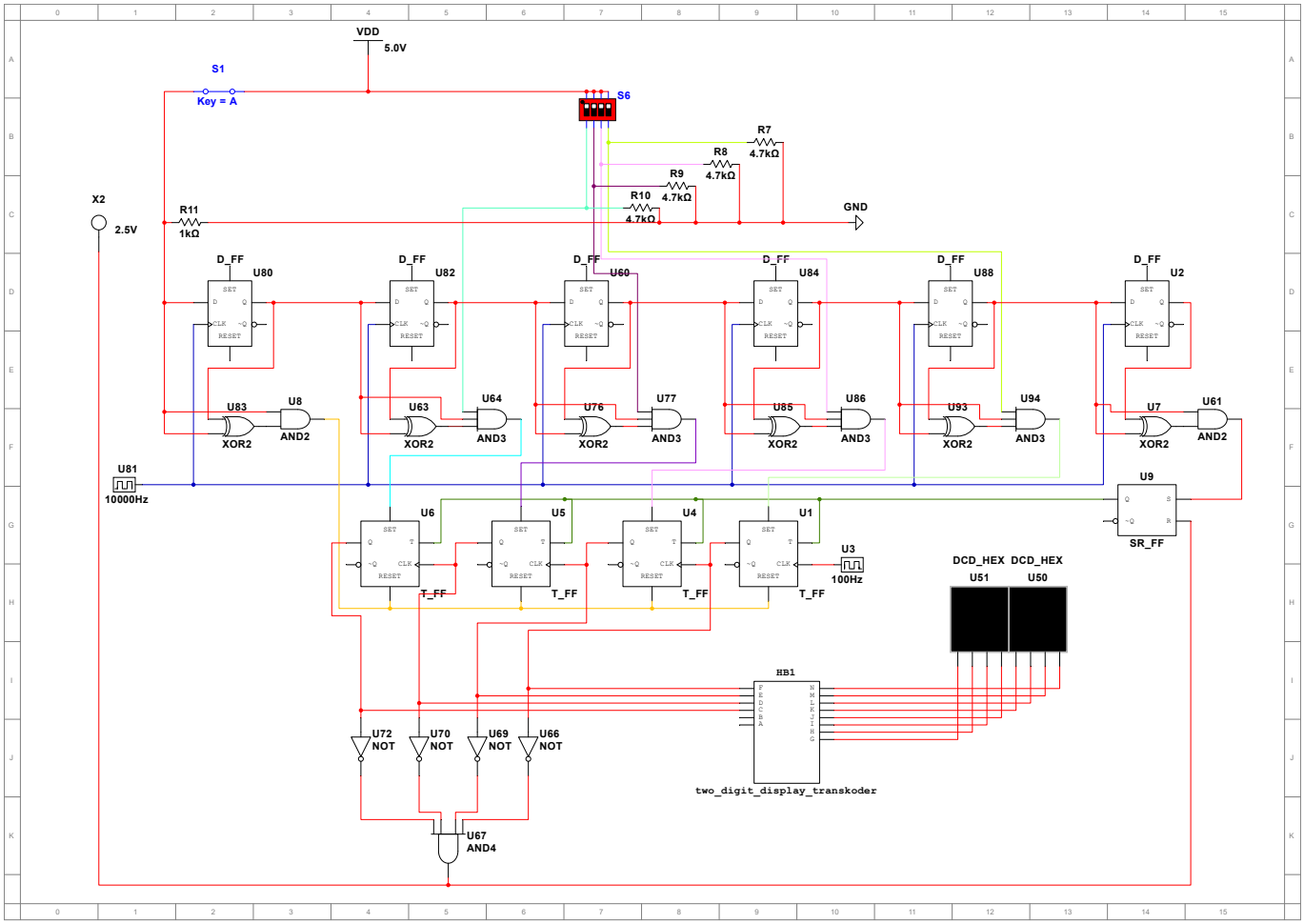
- Jednym z zastosowań układów timer jest umieszczanie ich w mikroprocesorach. Są one jednym z podstawowych układów peryferyjnych pozwalających na wykonywanie periodycznych zadań. Najbardziej podstawowym zadaniem może być periodyczne wykonywanie specjalnie zdefiniowanego fragmentu kodu aktywowanego sygnałem `ALARM` - przerwanie sprzętowe. W ten sposób nie musimy tracić czasu procesora na kontrolę czasu aby wykonywać kod. Taki mechanizm można też wykorzystać do periodycznej obsługi innych peryferiów jak np. przetwornik analogowo-cyfrowy czy kontrolery magistrali komunikacyjnych.



Rysunek 6.1: Przykładowe zastosowanie układu timer w mikroprocesorach

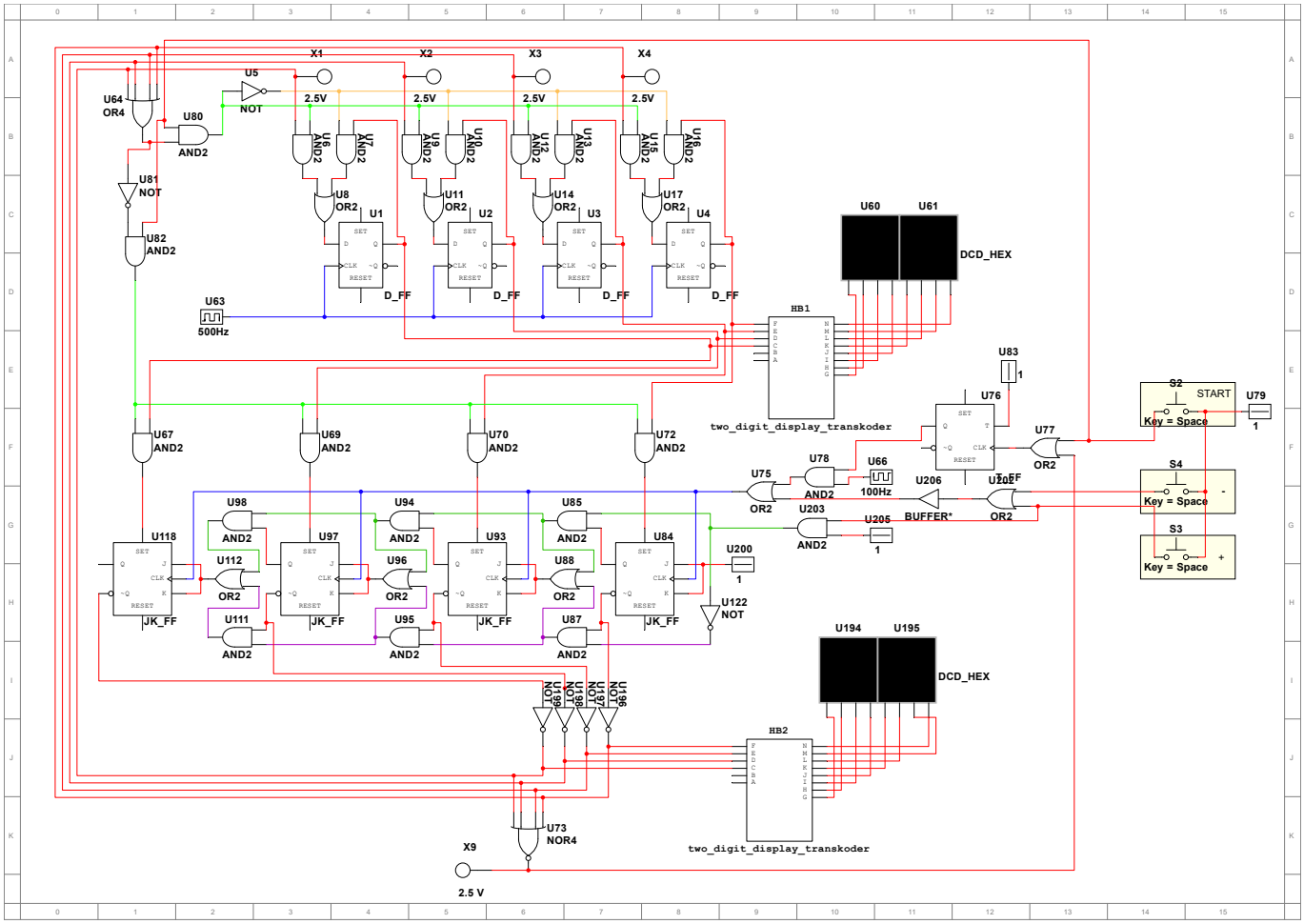
7 Wnioski

Dzięki przerzutnikom możemy projektować złożone układy z pamięcią swojego stanu.
W toku rozwiązywania postawionego problemu opracowaliśmy kilka możliwych rozwiązań. Ostatecznie zdecydowaliśmy się na przedstawienie tego konkretnego rozwiązania ze względu na jego prostotę i największą niezawodność. Zastosowanie przerzutników działających synchronicznie pozwoliło na zredukowanie ilości hazardów przy odliczaniu czasu.
Pierwotnym pomysłem było użycie przerzutników T asynchronicznie - łącząc je w szeregowo dzięki czemu nie było konieczne projektowanie układu kontrolującego przerzutniki. Największą wadą tego rozwiązania było to, że otoczka sterowania tak połączonych przerzutników była trudna do przedstawienia wzorami i uzasadnienia w prosty sposób. Rozwiązanie opierało się na idei generowania kolejnych impulsów synchronizujących kolejne etapy przygotowania i uruchomienia układu.



Rysunek 7.1: Schemat pierwszego rozwiązania

Kolejne rozwiązanie opierało się na wykorzystaniu przerzutników JK do zrobienia synchronicznego licznika. W tym układzie, czas był ustawiany bezpośrednio na przerzutnikach za pomocą przycisków (+) i (-). Było to możliwe dzięki zaimplementowaniu licznika w sposób pozwalający na jego odliczanie zarówno do przodu jak i do tyłu. Po wciśnięciu przycisku ”start” wybrany czas zostawał zapisywany w rejestrze o równoległych wejściach i wyjściach (PIPO), a licznik zaczynał odmierzać czas. Po dojściu licznika do zera, aktywował się alarm, a po kolejnym naciśnięciu przycisku ”start” wartość zapisana w rejestrze była wczytywana na licznik i odliczanie zaczynało się od początku. Ostatecznie zrezygnowaliśmy z tego rozwiązania, ponieważ było ono skomplikowane i pełne trudnych do rozwiązania błędów. Przede wszystkim jednak, znacznie odbiegało ono od wymogów zadania. Z tych właśnie powodów sama implementacja nigdy nie została całkowicie dokończona i na schemacie przedstawionym poniżej występują liczne błędy i niedociągnięcia.



Rysunek 7.2: Scheamt drugiego rozwiązania