Automat mogący posłużyć do sterowania prostym odtwarzaczem plików muzycznych mp3

Bartosz Kaganiec, Julia Demitraszek, Adrian Krawczyk, Damian Chłus

12 maja 2025

Spis treści

1 Opis rozwiązania **3**

2 Tabela Karnaugh **5**

3 Oprogramowanie **5**

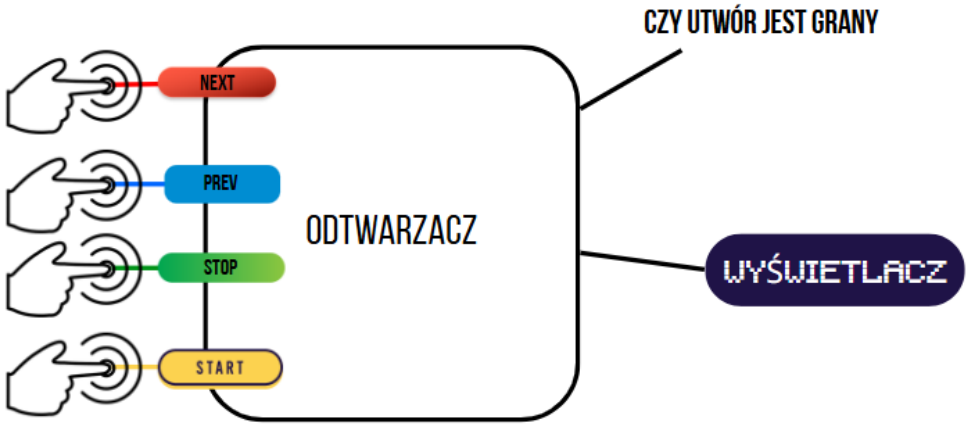
4 Testy **15**

5 Zastosowania **20**

6 Wnioski **23**

1 Opis rozwiązania

Rysunek 1: Inspiracja do wstępnego pomysłu

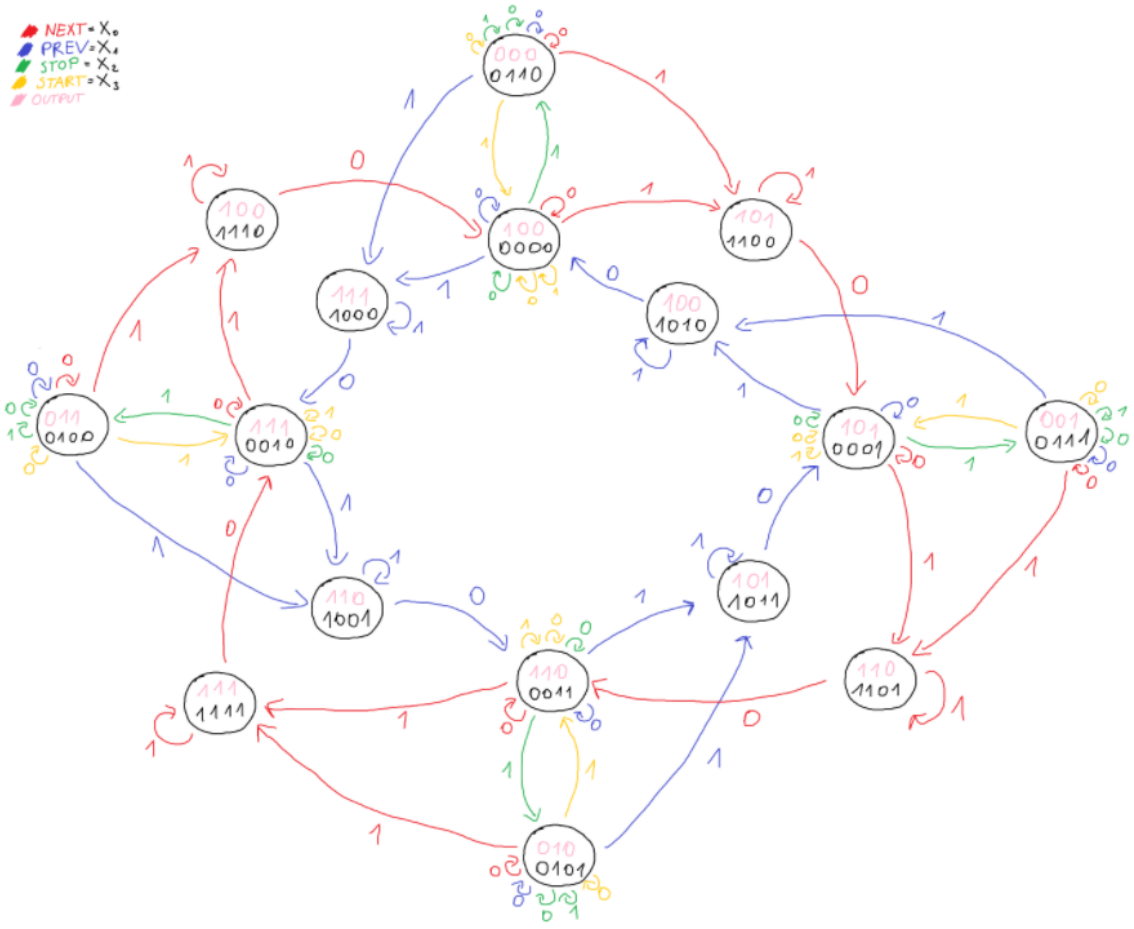


Rysunek 2: Wstępny pomysł

Naszym pomysłem było to, że aby jakiś konkretny stan umożliwić tzn go wykonać należy kliknąć i odkliknąć przycisk (tylko w przypadku PREV czy NEXT), dlatego też są te stany pośrednie, które umożliwiają jedynie zakończenie operacji (tzn jeżeli klikniemy na PREV to należy go odkliknąć, ale już gra poprzednia piosenka). W przypadku jednak przycisków START czy STOP jeżeli piosenka jest w trakcie gry i klikniemy

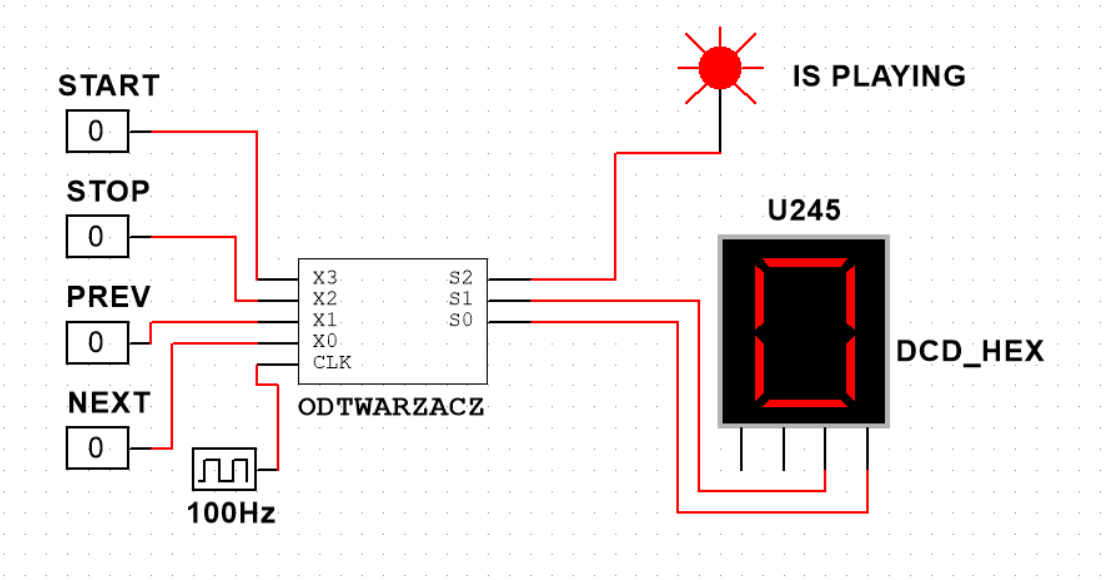
START nic się nie stanie, a jeżeli klikniemy STOP to się zatrzyma. Stany zatrzymania i grające oznaczyliśmy odpowiednio jako 1 w najbardziej znaczącym bicie stanu wyjściowego oraz 0 w najbardziej znaczącym bicie stanu wyjściowego. Oznaczyliśmy przyciski w następujący sposób:

- **NEXT** jako X_0
- **PREV** jako X_1
- **STOP** jako X_2
- **START** jako X_3



Rysunek 3: Nasz pomysł rozwiązania problemu - schemat automatu

Wobec tego należy mieć możliwość otrzymania 4 sygnałów oraz zapamiętywać poprzedni stan i w zależności od tego powinno być ukazywane, który utwór jest rozpatrywany i kiedy się zatrzymuje



Rysunek 4: Schemat w programie multisim

2 Tabela Karnaugh

Stan x jest tutaj sygnałem nieokreślonym.

X3X2X1X0/Q3Q2Q1Q0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0000	0001	0011	0010	0110	0111	0101	0100	0001	0011	0010	0000	0000	0001	0011	0010
0001	1100	1101	1111	1110	1100	1101	1111	1110	1100	1101	1111	1110	x	x	x	x
0011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0010	1000	1010	1011	1001	1000	1010	1011	1001	x	x	x	x	1010	1011	1001	1000
0110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0100	0110	0111	0101	0100	0110	0111	0101	0100	x	x	x	x	x	x	x	x
1100	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1010	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1000	0000	0001	0011	0010	0000	0001	0011	0010	x	x	x	x	x	x	x	x

Tabela 1: Tabela przejść stanów po otrzymaniu danego sygnału

3 Oprogramowanie

Do realizacji naszego pomysłu wykorzystaliśmy trzy główne komponenty, automat, przerzutniki w celu zapamiętywania stanu poprzedniego oraz konwerter.

Automat

W celu nie wykonywania żmudnych obliczeń napisaliśmy program w języku programowania **Python** wykorzystujący bibliotekę **logicmin**, która umożliwia wyprowadzenie funkcji minimalizujących wykorzystując tabele Karnaugh oraz **pandas**, która z kolei umożliwia wczytanie danych z excela. Aby skorzysta z pełnej funkcjonalności biblioteki należy dla każdego wejcia przekaza odpowiadajce mu wyjście w tablicy.

```
import pandas as pd
import logicmin

df = pd.read_csv("excel.csv",sep=';', index_col=0)
t = logicmin.TT(8, 4)

first_variables = ["X3", "X2", "X1", "X0", "Q3", "Q2", "Q1", "Q0"]
second_variables = ["Q3'", "Q2'", "Q1'", "Q0'"]

# Iterujemy po wierszach i kolumnach
for row in df.index:
    for col in df.columns:
        input_bits = (str(row).strip() + str(col).strip()).zfill(8)
        output_bits = str(df.loc[row, col]).strip()

        if output_bits.lower() == 'x':
            output_bits = "----"
        else:
            output_bits = output_bits.zfill(4) # 4 bitowa liczba binarna
        t.add(input_bits, output_bits)

solution = t.solve()
print(solution.printN(xnames = first_variables,
ynames = second_variables))
```

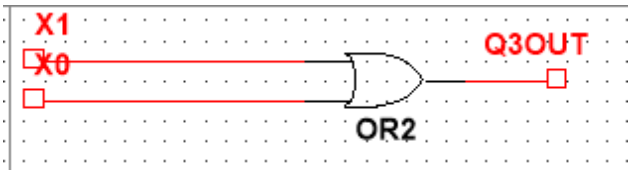
Powstałe wyjścia:

Przejście Q'_3

X3X2X1X0/Q3Q2Q1Q0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0001	1	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x
0011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0010	1	1	1	1	1	1	1	1	x	x	x	x	1	1	1	1
0110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0100	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x
1100	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1010	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1000	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x

Tabela 2: Tabela wartości bitu wyjściowego Q'_3 dla odpowiednich stanów i sygnałów wejściowych

$$Q'_3 = X_0 + X_1$$



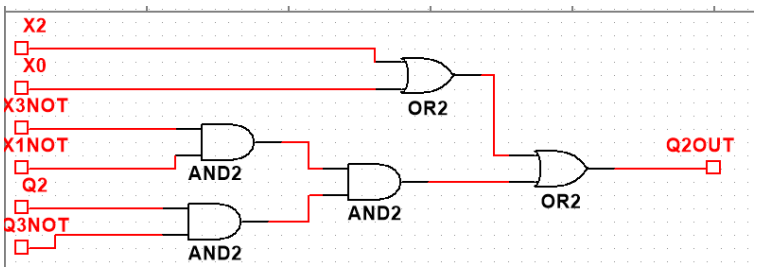
Rysunek 5: Wyjście $Q'_3(Q3OUT)$

Przejście Q'_2

X3X2X1X0/Q3Q2Q1Q0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0001	1	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x
0011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0010	0	0	0	0	0	0	0	0	x	x	x	x	0	0	0	0
0110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0100	1	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x
1100	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1010	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1000	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x

Tabela 3: Tabela wartości bitu wyjściowego Q'_2 dla odpowiednich stanów i sygnałów wejściowych

$$Q_2' = \overline{X_3} \cdot \overline{X_1} \cdot \overline{Q_3} \cdot Q_2 + X_0 + X_2$$



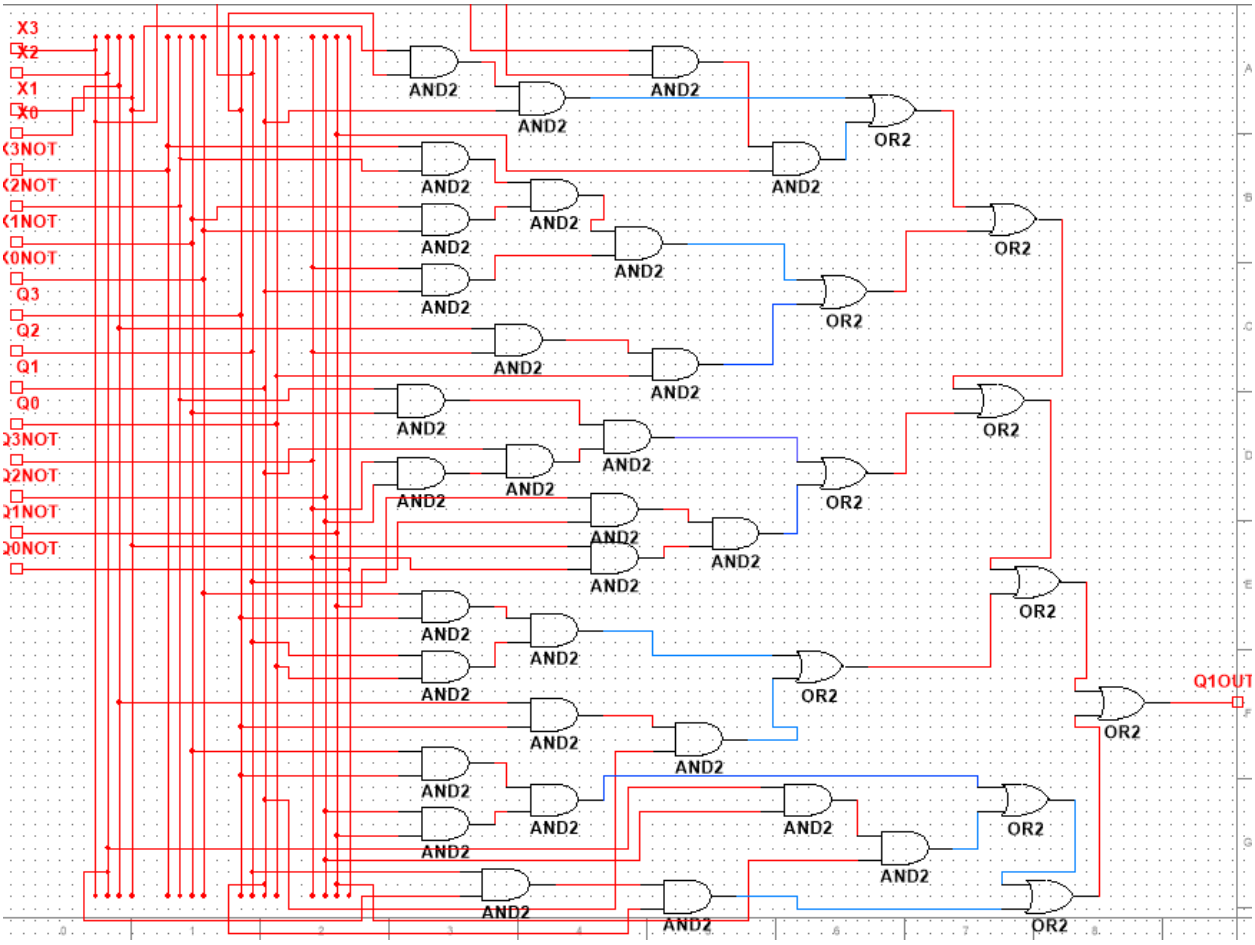
Rysunek 6: Wyjście $Q_2'(Q2OUT)$

Przejście Q_1'

X3X2X1X0/Q3Q2Q1Q0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0	0	1	1	1	1	0	0	0	1	1	0	0	0	1	1
0001	0	0	1	1	0	0	1	1	0	0	1	1	x	x	x	x
0011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0010	0	1	1	0	0	1	1	0	x	x	x	x	1	1	0	0
0110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0100	1	1	0	0	1	1	0	0	x	x	x	x	x	x	x	x
1100	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1010	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1000	0	0	1	1	0	0	1	1	x	x	x	x	x	x	x	x

Tabela 4: Tabela wartości bitu wyjściowego Q_1' dla odpowiednich stanów i sygnałów wejściowych

$$Q_1' = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0} \cdot \overline{Q_3} \cdot Q_1 + \overline{X_2} \cdot \overline{X_1} \cdot \overline{Q_3} \cdot \overline{Q_2} \cdot Q_1 + \overline{X_0} \cdot Q_3 \cdot Q_2 \cdot Q_0 + \overline{X_1} \cdot Q_3 \cdot \overline{Q_2} \cdot \overline{Q_1} + X_0 \cdot \overline{Q_3} \cdot Q_2 \cdot \overline{Q_1} +$$
$$X_0 \cdot Q_3 \cdot Q_1 + X_1 \cdot \overline{Q_3} \cdot Q_0 + X_1 \cdot Q_3 \cdot Q_1 + X_2 \cdot \overline{Q_2} \cdot \overline{Q_1} + X_2 \cdot Q_2 \cdot Q_1 + X_3 \cdot Q_2 \cdot \overline{Q_1}$$



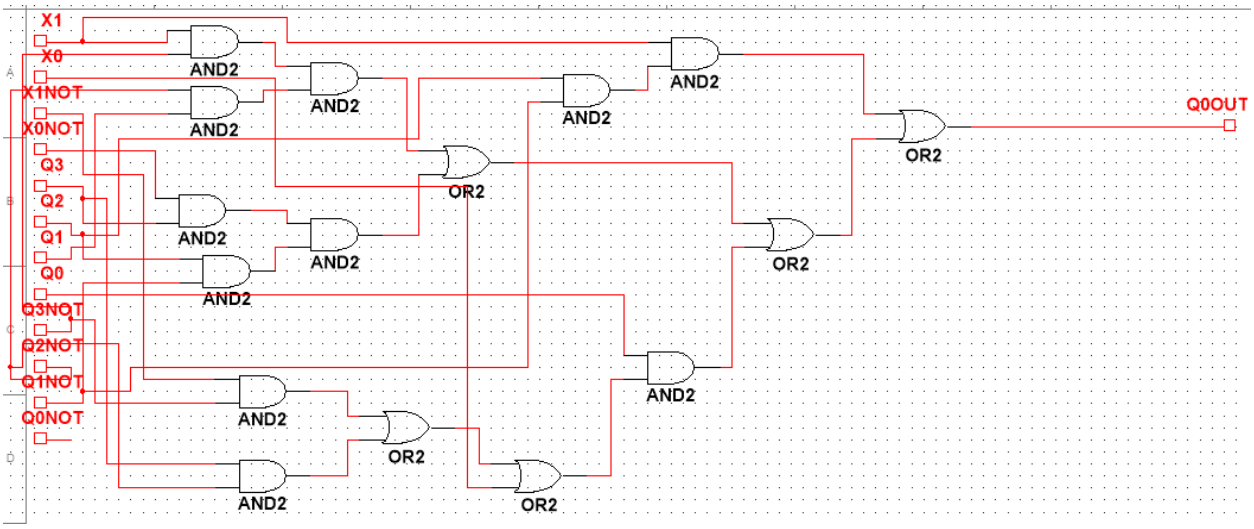
Rysunek 7: Wyjście $Q'_1(Q1OUT)$

Przejście Q'_0

X3X2X1X0/Q3Q2Q1Q0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0
0001	0	1	1	0	0	1	1	0	0	1	1	0	x	x	x	x
0011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0010	0	0	1	1	0	0	1	1	x	x	x	x	0	1	1	0
0110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0100	0	1	1	0	0	1	1	0	x	x	x	x	x	x	x	x
1100	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1101	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1111	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1010	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1011	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1000	0	1	1	0	0	1	1	0	x	x	x	x	x	x	x	x

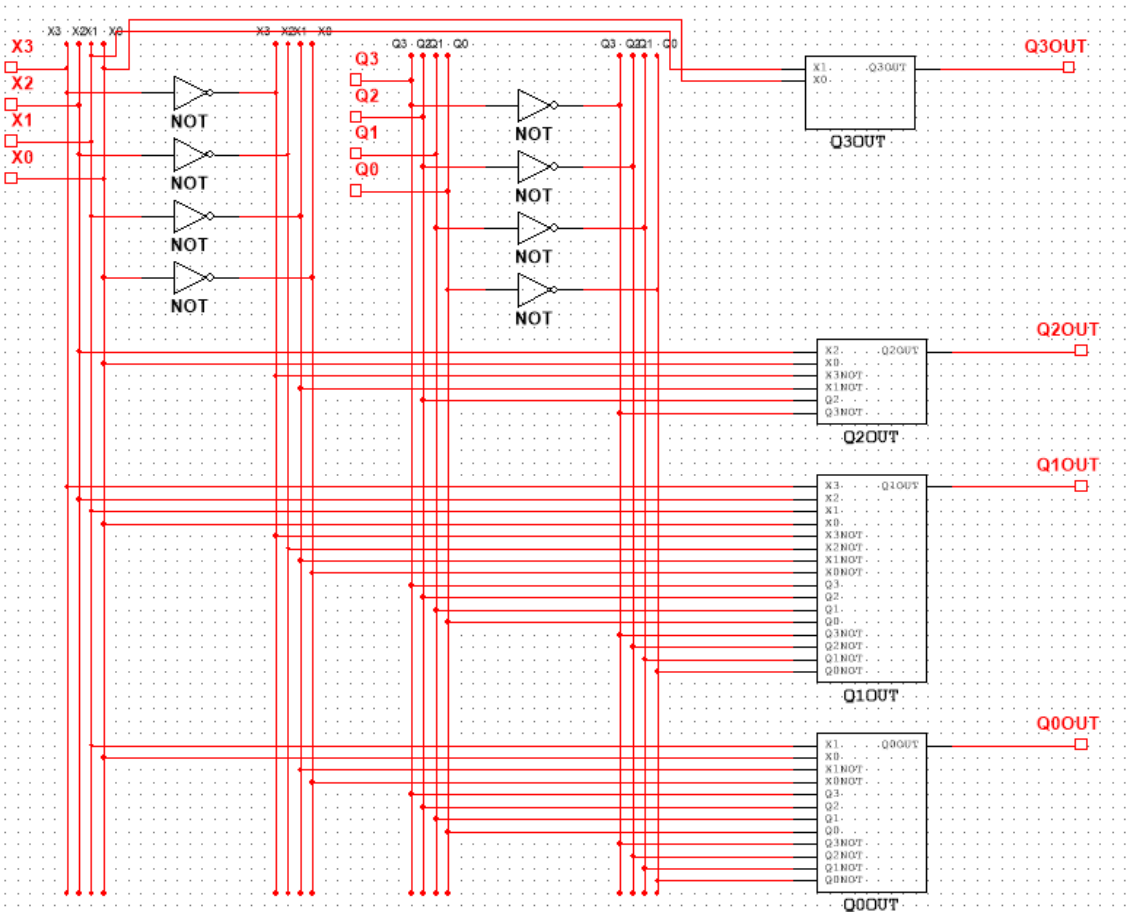
Tabela 5: Tabela wartości bitu wyjściowego Q'_0 dla odpowiednich stanów i sygnałów wejściowych

$$Q'_0 = \overline{X_0} \cdot Q_3 \cdot Q_2 \cdot \overline{Q_1} + X_1 \cdot \overline{Q_3} \cdot \overline{Q_2} \cdot Q_1 + \overline{X_1} \cdot \overline{Q_3} \cdot Q_0 + Q_3 \cdot \overline{Q_2} \cdot Q_0 + X_1 \cdot Q_2\overline{Q_1} + X_0 \cdot Q_0$$



Rysunek 8: Wyjście $Q'_0(Q0OUT)$

Powyższe wyjścia zostały opakowane w poniższy automat



Rysunek 9: Schemat automatu

Konwerter

Q3Q2/Q1Q0	00	01	11	10
00	100	101	110	111
01	011	010	001	000
11	101	110	111	100
10	111	110	101	100

Tabela 6: Tabela prawdy dla konwertera, która piosenka ma być w danym stanie rozpatrywana

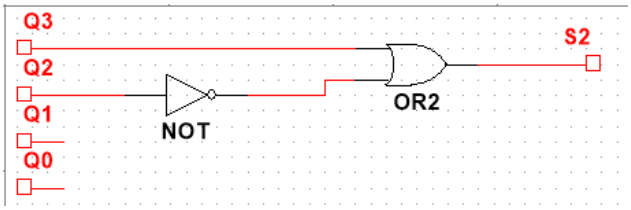
```
import pandas as pd
import logicmin

df = pd.read_csv("konwerter.csv",sep=';', index_col=0)
t = logicmin.TT(4, 3)

first_variables = ["Q3", "Q2", "Q1", "Q0"]
second_variables = ["S2", "S1", "S0"]

# Iterujemy po wierszach i kolumnach
for row in df.index:
    for col in df.columns:
        input_bits = str(row).strip().zfill(2) + str(col).strip().zfill(2)
        output_bits = str(df.loc[row, col]).strip()

        if output_bits.lower() == 'x':
            output_bits = "---"
        else:
            output_bits = output_bits.zfill(3) # 3 bitowa liczba binarna
        t.add(input_bits, output_bits)

solution = t.solve()
print(solution.printN(xnames = first_variables,
ynames = second_variables))
```

Wyniki:

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

Tabela 7: Tabela prawdy dla bitu wyjściowego S2

$$S2 = \overline{Q_2} + Q_3 \tag{1}$$

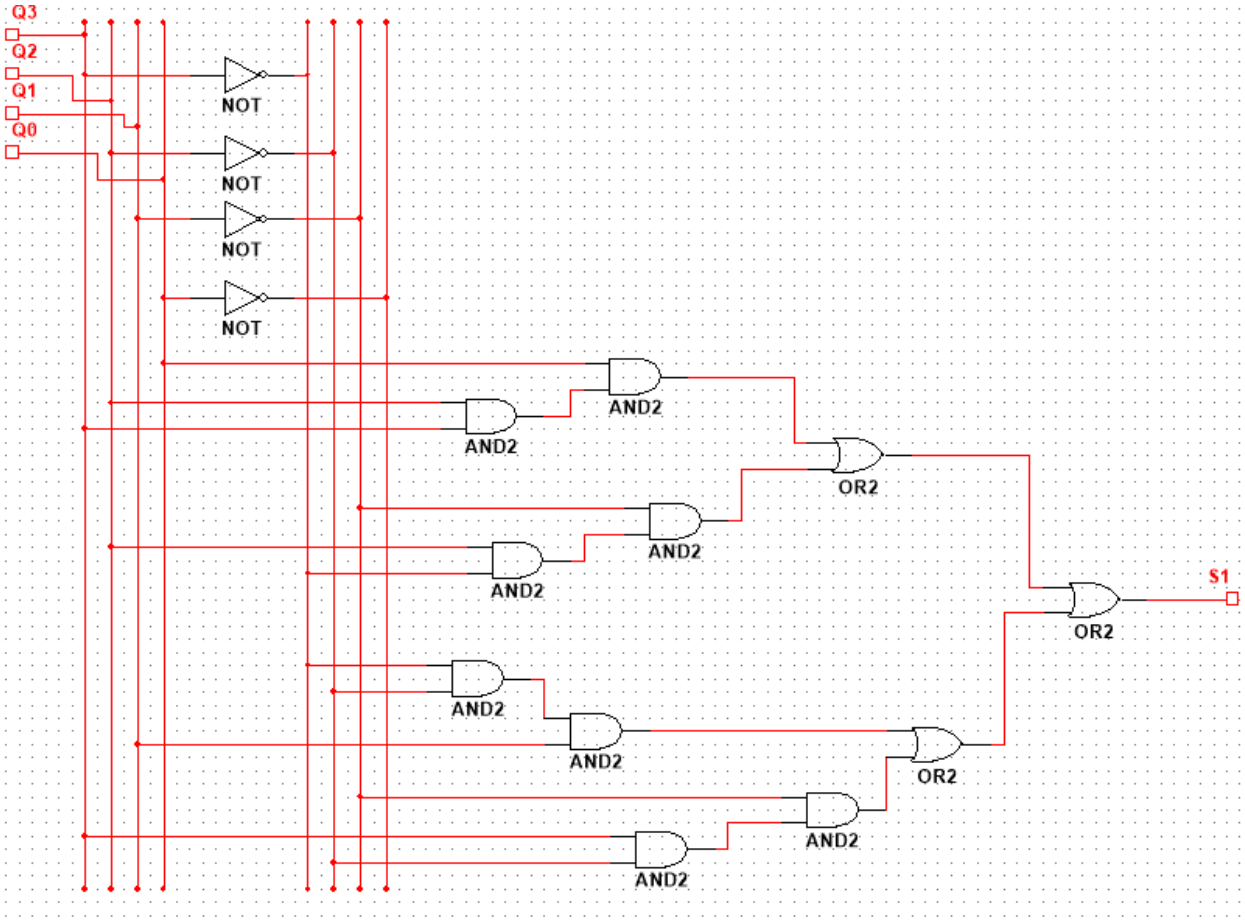


Rysunek 10: Wyjście S_2

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

Tabela 8: Tabela prawdy dla bitu wyjściowego S1

$$S1 = \overline{Q_3} \cdot \overline{Q_2} \cdot Q_1 + \overline{Q_3} \cdot Q_2 \cdot \overline{Q_1} + Q_3 \cdot Q_2 \cdot Q_0 + Q_3 \cdot \overline{Q_2} \cdot \overline{Q_1} \tag{2}$$

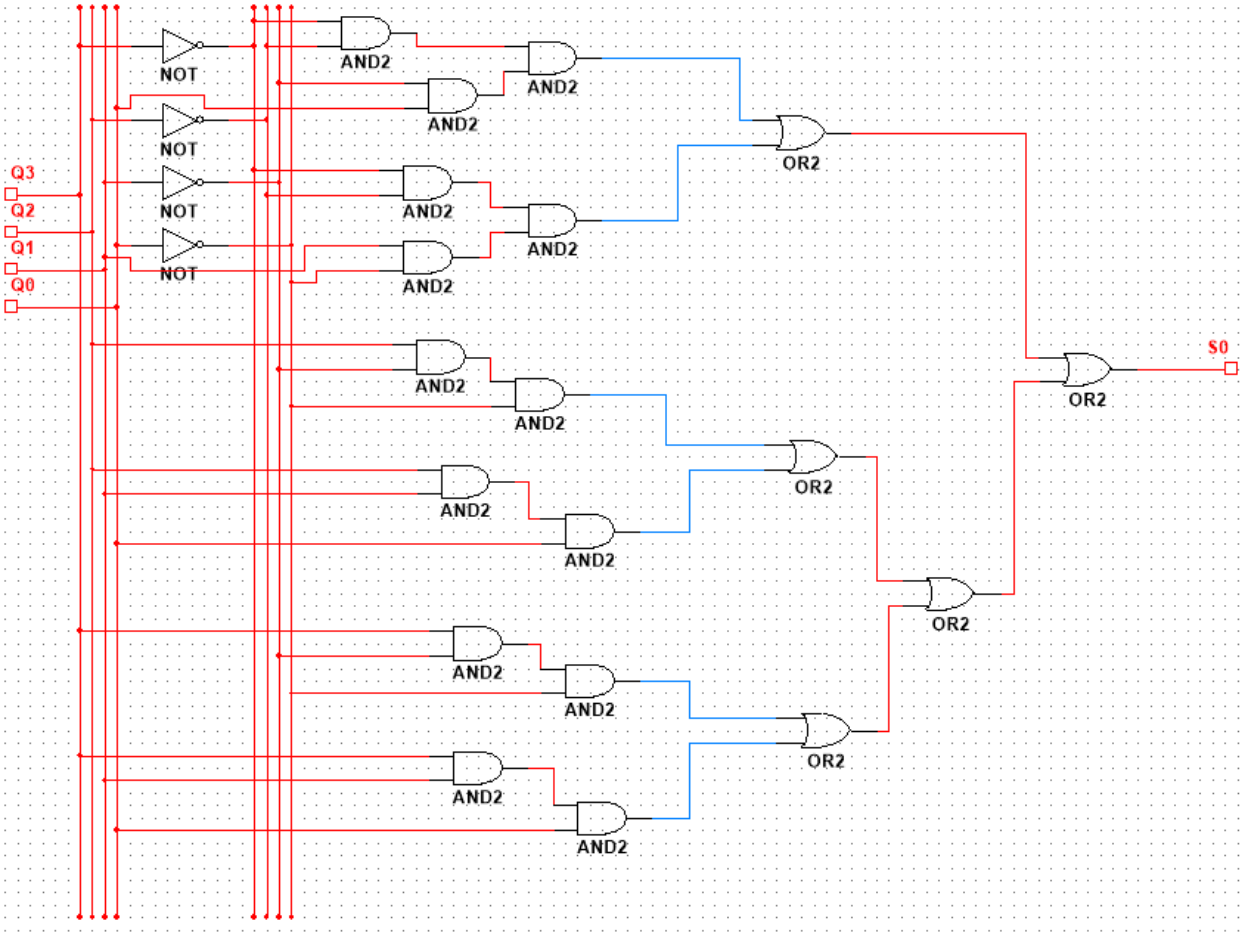


Rysunek 11: Wyjście S_1

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

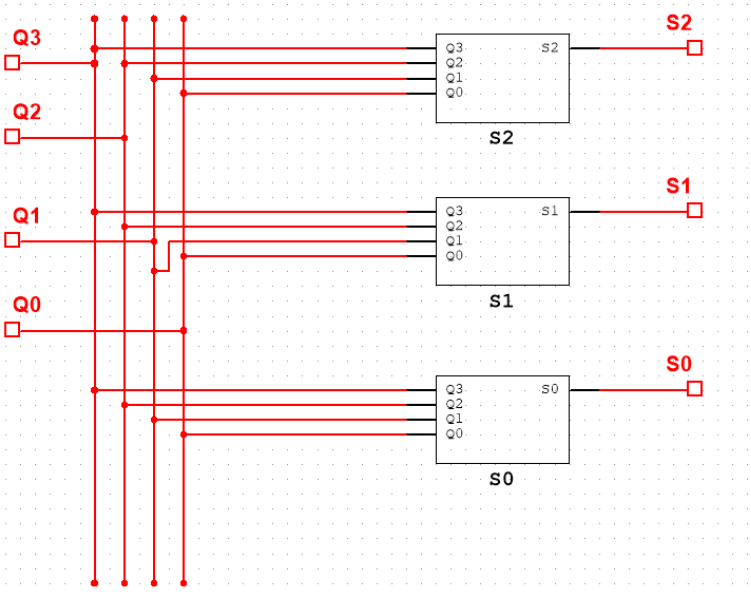
Tabela 9: Tabela prawdy dla bitu wyjściowego S0

$$S0 = \overline{Q_3} \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot Q_0 + \overline{Q_3} \cdot \overline{Q_2} \cdot Q_1 \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot Q_1 \cdot Q_0 + Q_3 \cdot \overline{Q_1} \cdot \overline{Q_0} + Q_3 \cdot Q_1 \cdot Q_0 \quad (3)$$



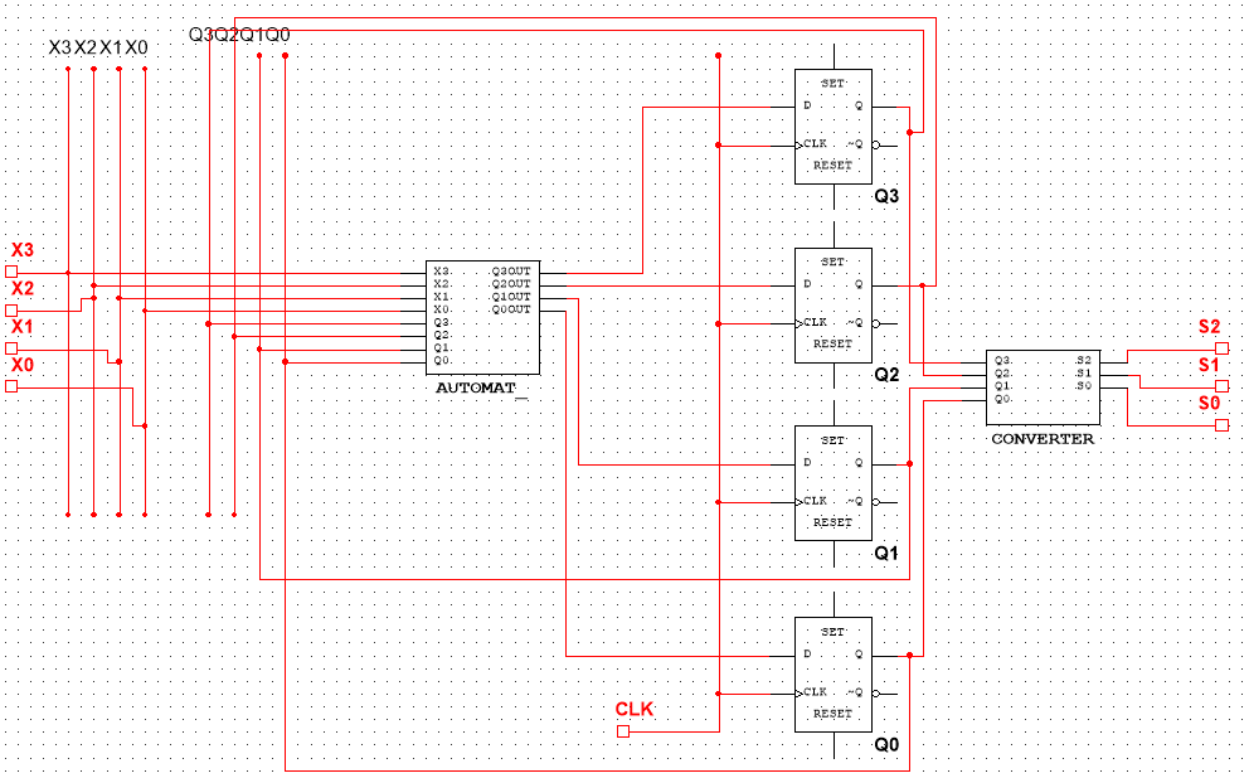
Rysunek 12: Wyjście S_0

Schemat konwertera prezentuje się następująco:

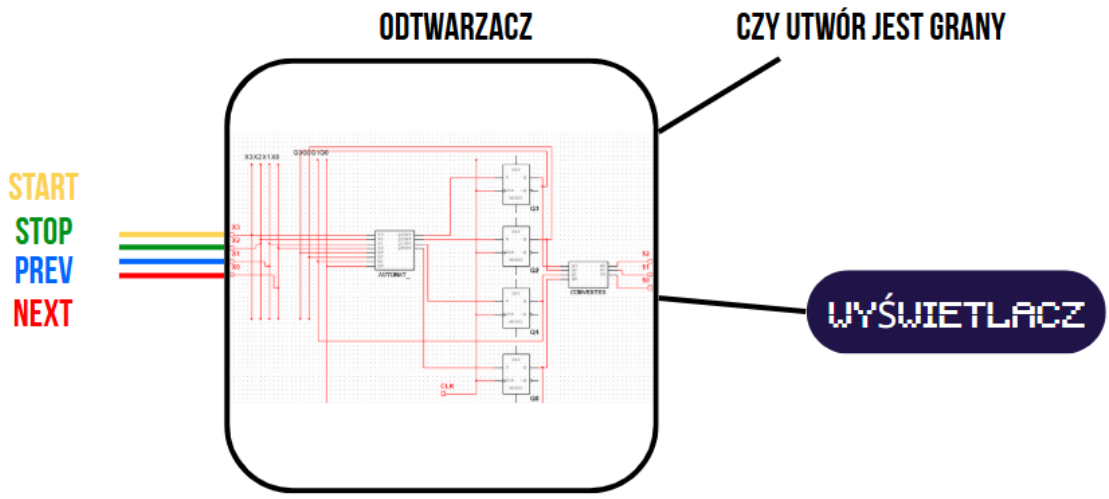


Rysunek 13: Schemat konwertera

Schemat odtwarzacza prezentuje się nastepująco:



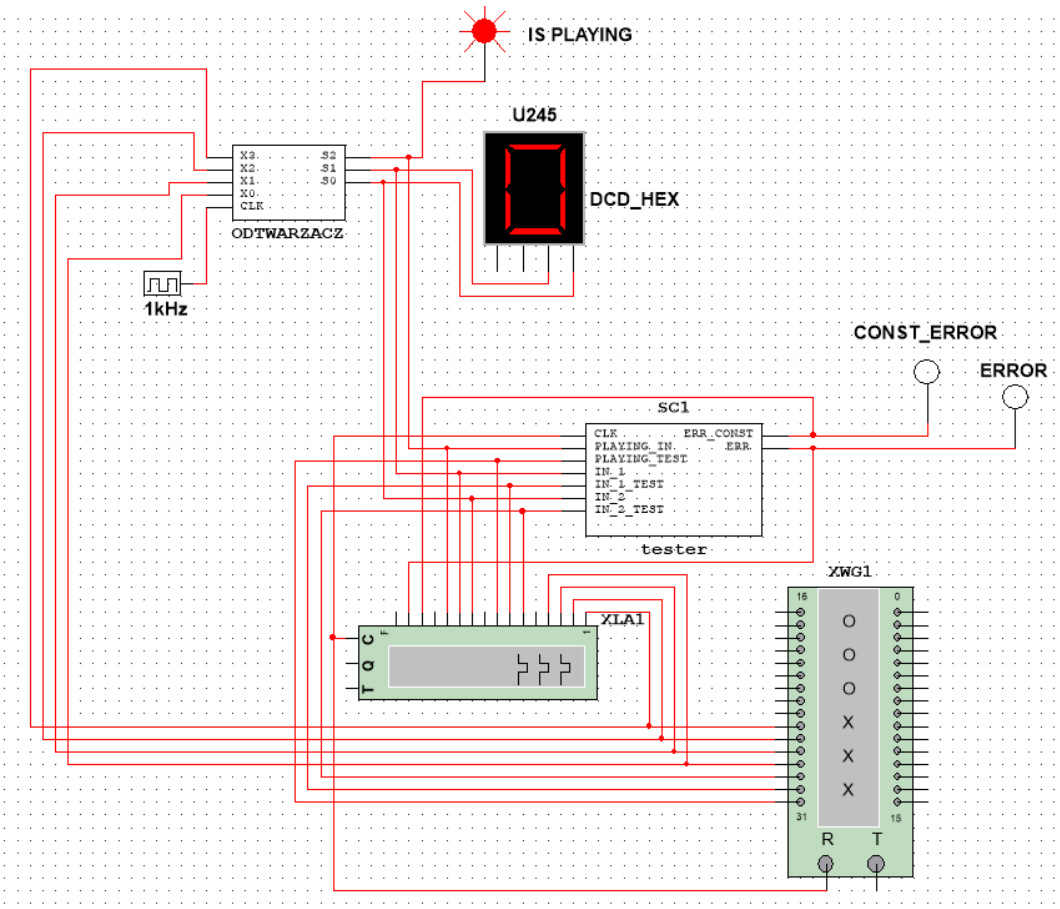
Rysunek 14: Schemat odtwarzacza



Rysunek 15: Schemat w kontekście naszego pomysłu

4 Testy

W celu przetestowania działania układu został stworzony dedykowany układ testowy, który wykorzystuje pełną logikę układu podstawowego oraz dodatkowe komponenty: *word generator*, *logic analyzer* oraz podukład *tester*.



Rysunek 16: Schemat układu testowego

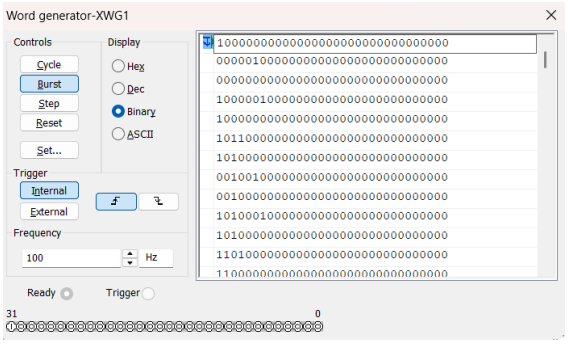
Word Generator

Word generator pełni funkcję źródła sygnałów wejściowych dla układu oraz generuje oczekiwane odpowiedzi wyjściowe. Posiada on własny zegar, który stanowi również zegar bazowy dla całego układu, ustawiony na częstotliwość $100\,Hz$.

Działanie generatora polega na umieszczaniu kolejnych słów testowych (symulujących np. naciśnięcia przycisków START, STOP, NEXT, PREV) na wyjściach 26–29. Każdy sygnał wejściowy jest symulowany jako chwilowe ustawienie danego bitu (symulacja wciśnięcia i natychmiastowego zwolnienia przycisku). Z kolei na wyjściach 30–32 zapisane są oczekiwane wartości sygnałów wyjściowych.

Wszystkie słowa testowe wraz z ich interpretacją

```
10000000000000000000000000000000 STAN POCZĄTKOWY
00000100000000000000000000000000 STOP
00000000000000000000000000000000
10000010000000000000000000000000 START
10000000000000000000000000000000
10110000000000000000000000000000 NEXT
10100000000000000000000000000000
00100100000000000000000000000000 STOP
00100000000000000000000000000000
10100010000000000000000000000000 START
10100000000000000000000000000000
11010000000000000000000000000000 NEXT
11000000000000000000000000000000
01000100000000000000000000000000 STOP
01000000000000000000000000000000
11000010000000000000000000000000 START
11000000000000000000000000000000
11110000000000000000000000000000 NEXT
11100000000000000000000000000000
01100100000000000000000000000000 STOP
01100000000000000000000000000000
11100010000000000000000000000000 START
11100000000000000000000000000000
10010000000000000000000000000000 NEXT
10000000000000000000000000000000
00000100000000000000000000000000 STOP
00000000000000000000000000000000
11101000000000000000000000000000 PREV
11100000000000000000000000000000
01100100000000000000000000000000 STOP
01100000000000000000000000000000
11001000000000000000000000000000 PREV
11000000000000000000000000000000
01000100000000000000000000000000 STOP
01000000000000000000000000000000
10101000000000000000000000000000 PREV
10100000000000000000000000000000
00100100000000000000000000000000 STOP
00100000000000000000000000000000
10001000000000000000000000000000 PREV
10000000000000000000000000000000
00000100000000000000000000000000 STOP
00000000000000000000000000000000
10110000000000000000000000000000 NEXT
10100000000000000000000000000000
00100100000000000000000000000000 STOP
00100000000000000000000000000000
11010000000000000000000000000000 NEXT
11000000000000000000000000000000
01000100000000000000000000000000 STOP
01000000000000000000000000000000
11110000000000000000000000000000 NEXT
11100000000000000000000000000000
01100100000000000000000000000000 STOP
01100000000000000000000000000000
10010000000000000000000000000000 NEXT
10000000000000000000000000000000
```



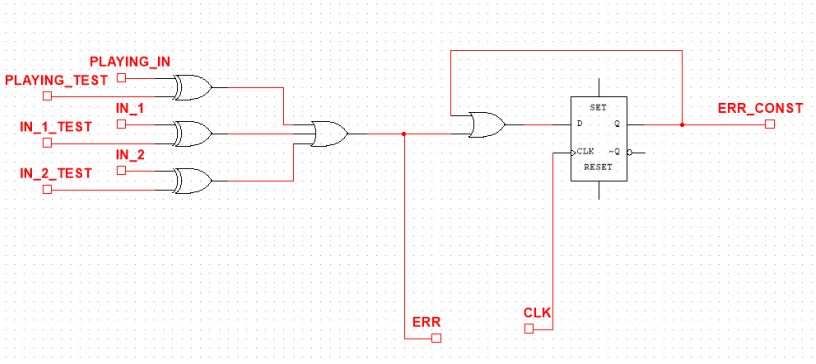
Rysunek 17: Schemat *word generatora*

Słowa te odpowiadają sekwencji działań, jakie wykonałby użytkownik. Zostały one przygotowane w celu sprawdzenia poprawności przejść stanów oraz odpowiedzi wyjściowych systemu.

Tester

Podukład *tester* odpowiada za automatyczne sprawdzanie poprawności działania układu. Składa się z trzech bramek XOR porównujących rzeczywiste wyjścia układu z oczekiwanymi wartościami generowanymi przez *word generator*. Wyniki porównań są następnie agregowane za pomocą bramki OR.

W przypadku niezgodności na któregokolwiek z wyjść, aktywowany jest sygnał `ERR`, wskazujący miejsce błędu, a także sygnał `ERR_CONST`, który pozostaje aktywny aż do ponownego uruchomienia układu — pełni funkcję trwałego znacznika błędu.

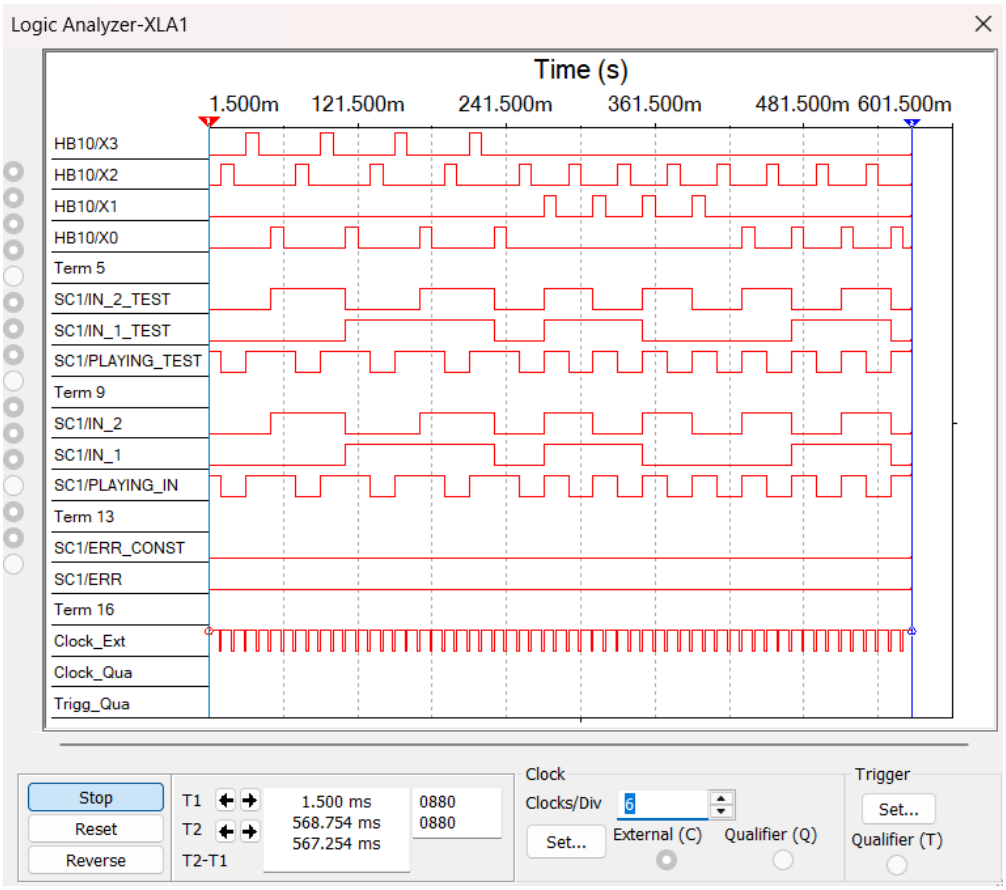


Rysunek 18: Schemat podukładu *tester*

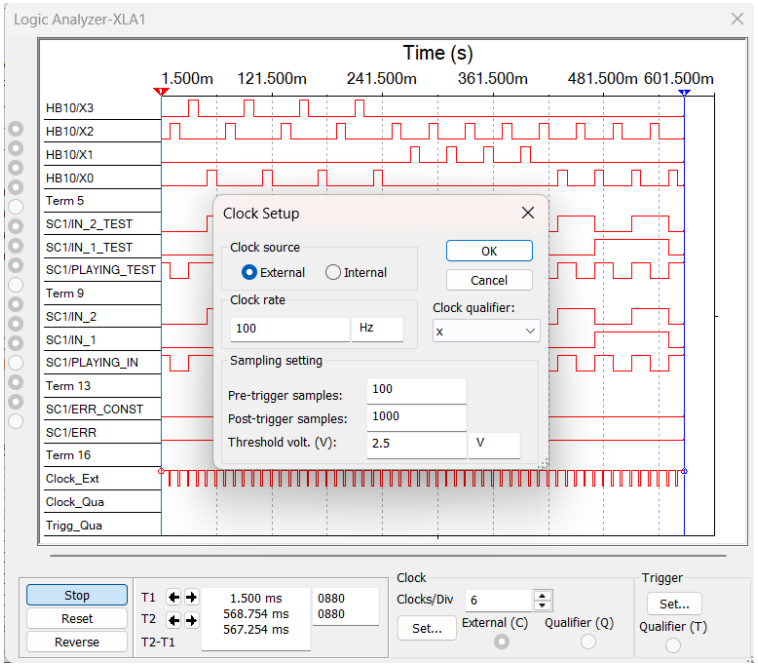
Logic Analyzer

Aby umożliwić dokładną analizę działania układu w czasie, wykorzystano *logic analyzer*. Do analizatora podłączono zarówno rzeczywiste sygnały wyjściowe, jak i wyjścia *word generatora*, co umożliwia wizualne porównanie rzeczywistych i oczekiwanych wartości.

Dodatkowo wykorzystano próbniki (*probes*) umożliwiające obserwację, w którym dokładnie cyklu zegara pojawiła się ewentualna rozbieżność między sygnałami, co ułatwia lokalizację błędu.



Rysunek 19: Widok z *logic analyzer*

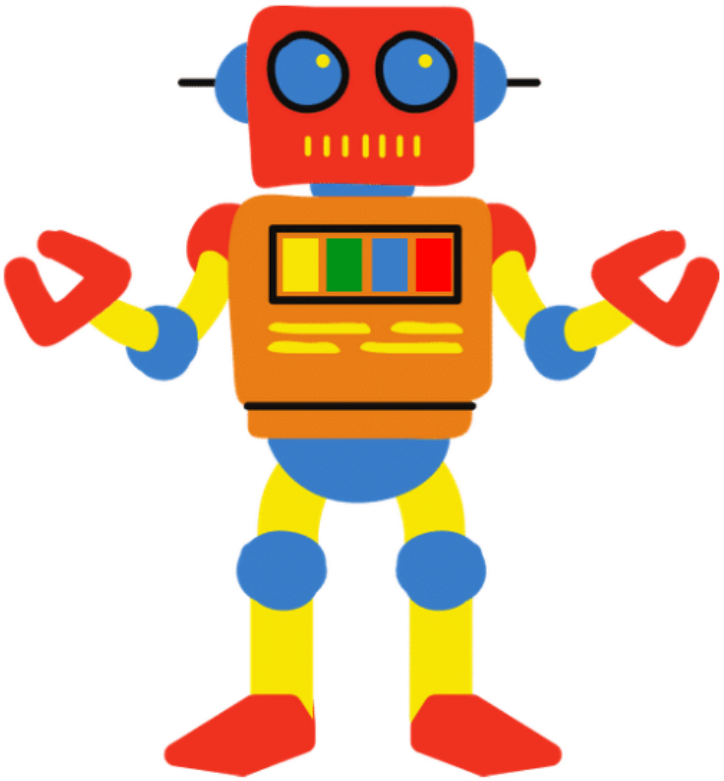


Rysunek 20: Zegar synchronizujący działanie całego układu testowego

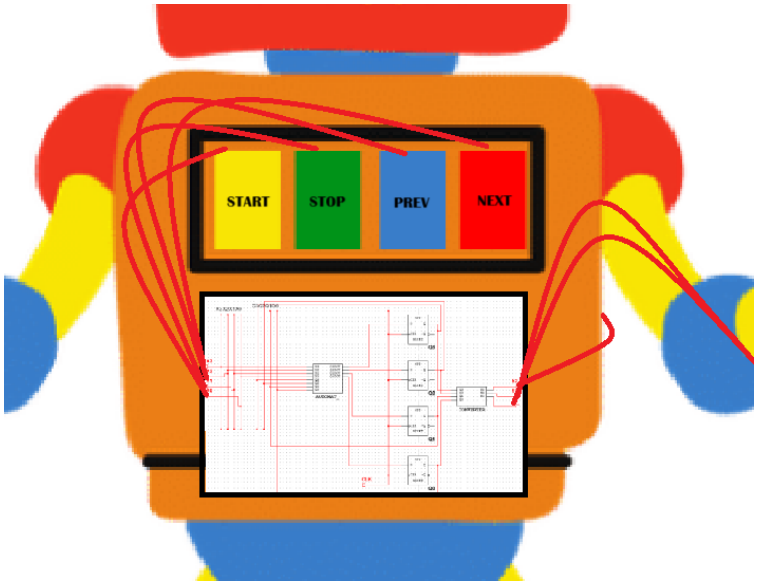
5 Zastosowania

Powyższy układ można zastosować na przykład w zabawkach dla dzieci, które odtwarzają konkretne dźwięki, gdzie przyciski to:

- **NEXT**
- **PREV**
- **STOP**
- **START**

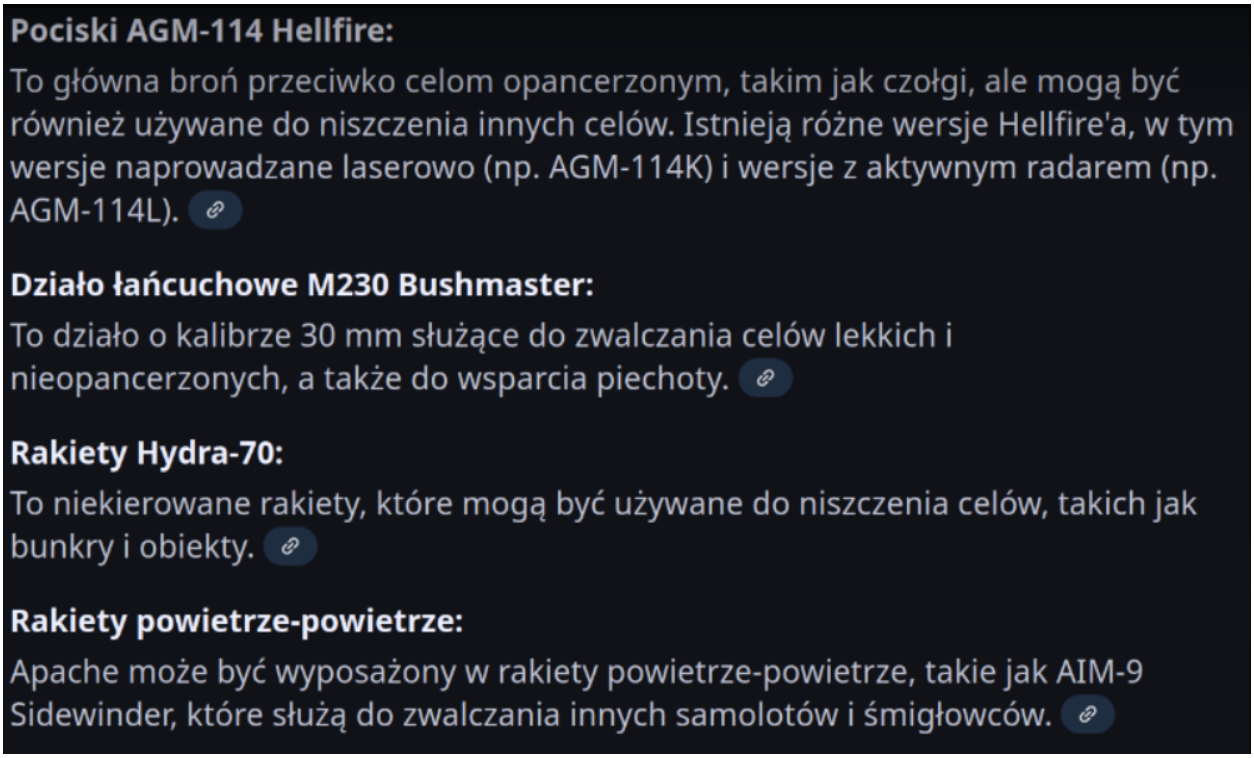


Rysunek 21: Robot



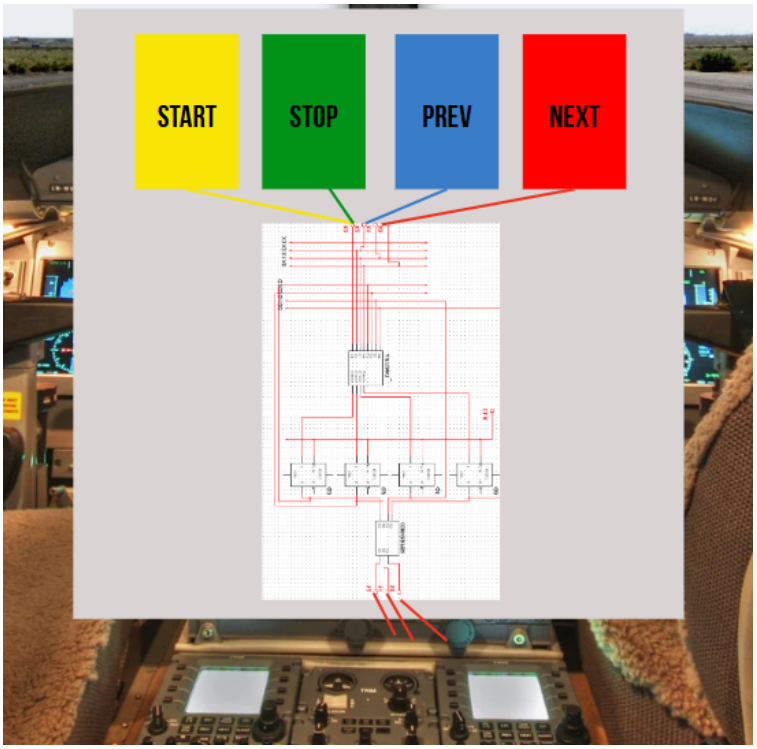
Rysunek 22: Układ w robocie

Innym możliwym zastosowaniem jest zarządzaniem ładowania amunicji w helikopterze szturmowym Apache, ponieważ są dokładnie 4 możliwe rodzaje pocisków (START wypuszcza pocisk, STOP zatrzymuje jego wypuszczanie, NEXT daje na następny w kolejce, a PREV daje zaś na poprzedni)



Rysunek 23: Rodzaje pocisków

Rysunek 24: Helikopter szturmowy Apache

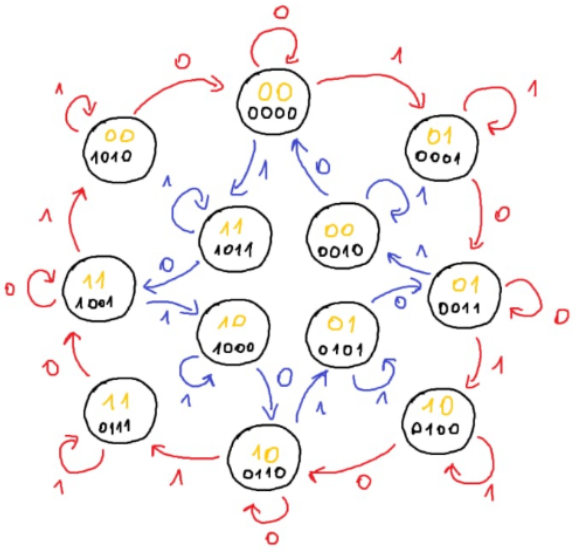


Rysunek 25: Układ w helikopterze

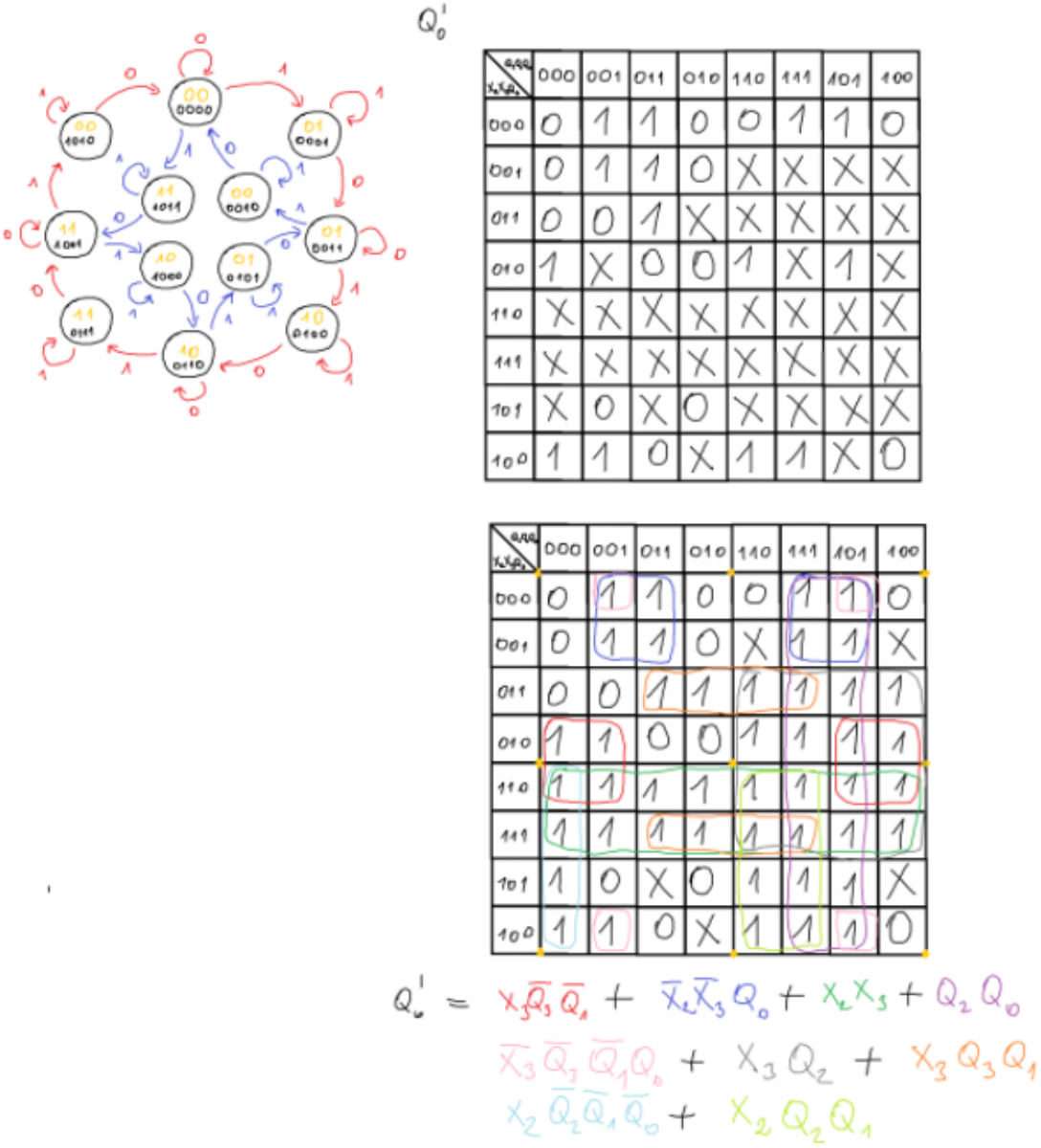
6 Wnioski

Można było wykonać zadanie wykorzystując dwa automaty, jeden osobno obsługujący STOP i START oraz PREV i NEXT, był to nasz początkowy pomysł w realizacji zadania.

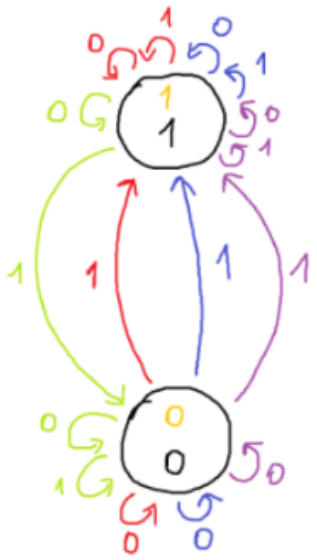
Rysunek 26: Oznaczenia



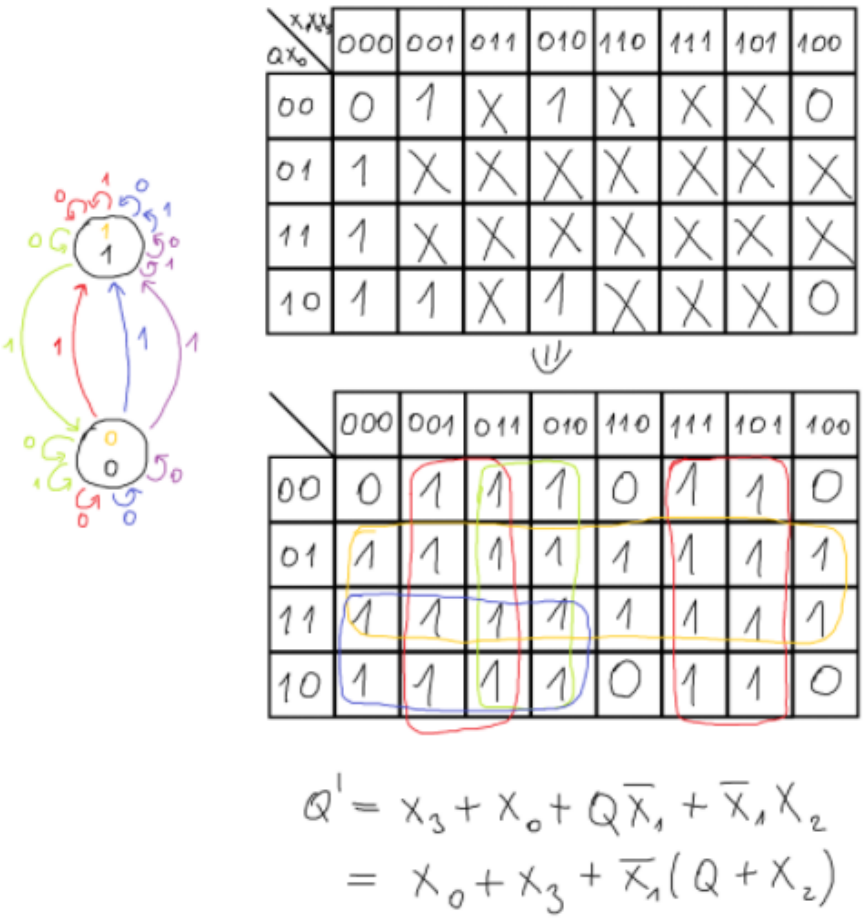
Rysunek 27: Automat do przycisków NEXT i PREV



Rysunek 28: Tabela Karnaugh dla wyjścia Q_0



Rysunek 29: STOP i START wraz z połączeniami



Rysunek 30: Tabela Karnaugh STOP i START wraz z połączeniami

W obliczeniach mogliśmy wykorzystać ręczne wyliczenia funkcji z tablic Karnaugh, jednak w celu przyspieszenia obliczeń zdecydowaliśmy się skorzystać z języka programowania Python.