Technika Cyfrowa

Sprawozdanie z ćwiczenia nr 3

Natalia Curzytek, Marta Stanisławska,
Karolina Nitsch, Szymon Kłodowski

Treść zadania

Proszę zaprojektować automat mogący posłużyć do sterowania jakimś prostym odtwarzaczem plików muzycznych mp3.

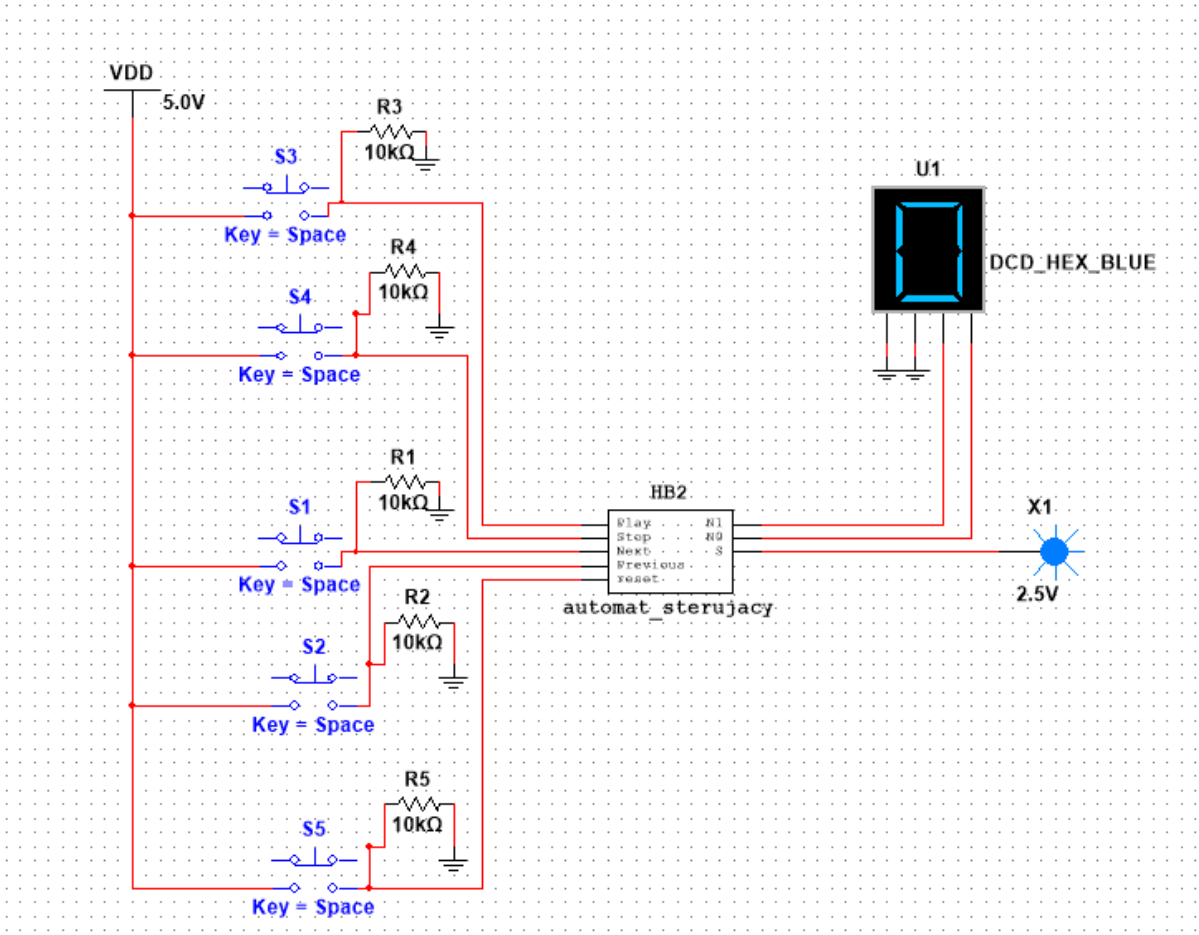
Układ powinien mieć następujące przyciski oraz odpowiadające im sygnały i wskaźniki:

- STOP
- PLAY
- NEXT
- PREVIOUS

oraz powinien posiadać dwubitowe wyjście binarne określające numer utworu.

Poglądowy schemat układu

Na wyświetlaczu wyświetlany będzie numer utworu, lampka sygnalizować będzie, że muzyka jest aktualnie odtwarzana (tryb play).
Zakładamy, że dwa przyciski nie mogą być wciśnięte jednocześnie.



Idea rozwiązania

Aby zrealizować zadanie zbudowaliśmy automat, którego stanem manipulujemy za pomocą czterech przycisków: STOP, PLAY, PREVIOUS, NEXT, umożliwiając przełączanie się cyklicznie pomiędzy czterema utworami oraz ich zatrzymywanie i odtwarzanie. Każdy stan tego automatu składa się z 3 bitów: S N1 N0.

- S oznacza, czy muzyka jest aktualnie odtwarzana- jasno oddziela tryb stop i play. Wartości bitu oznaczają:
  - 0- muzyka nie jest odtwarzana,
  - 1- muzyka jest odtwarzana.

- N1 N0 to dwa bity oznaczające numer aktualnie wybranego utworu:
  - 00- utwór 0,
  - 01- utwór 1,
  - 10- utwór 2,
  - 11- utwór 3.

Wejścia automatu:

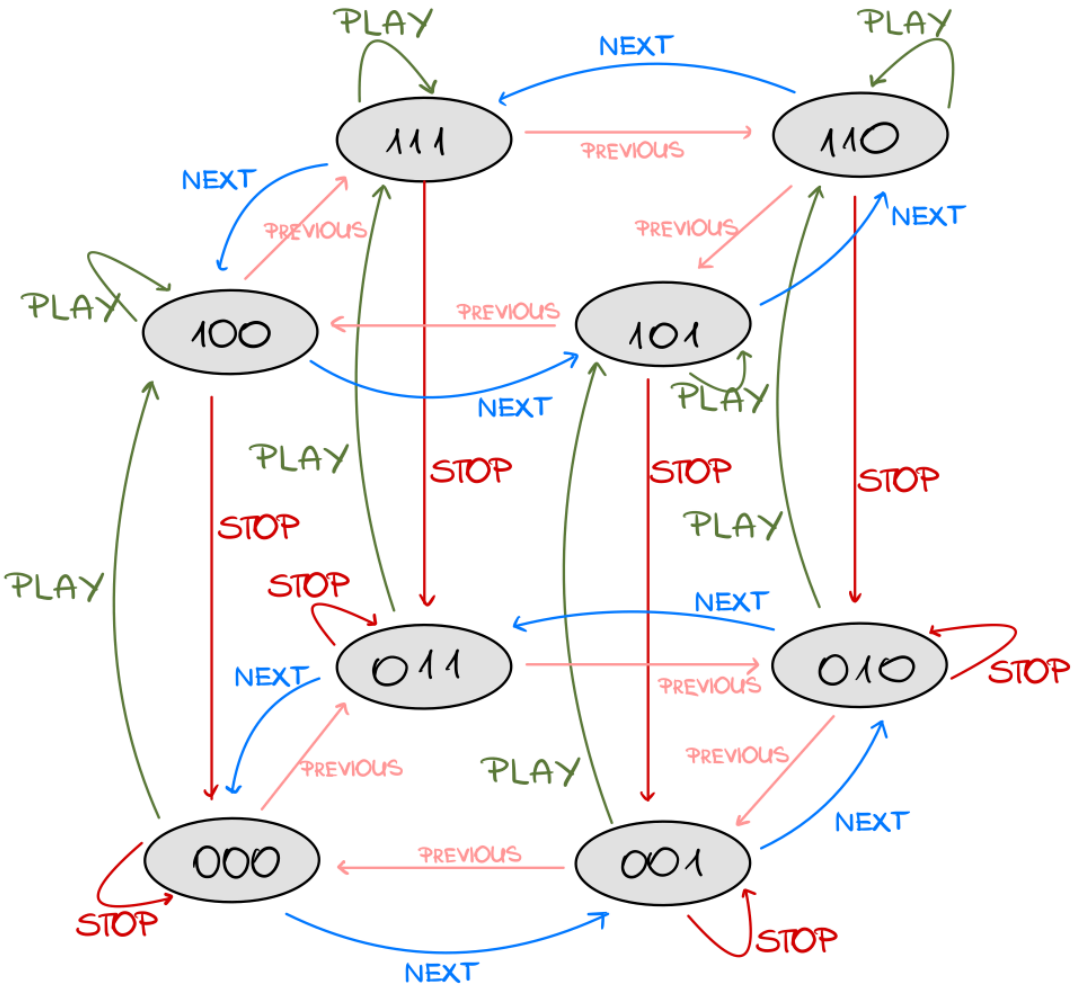
- Przycisk `PLAY` zawsze powoduje przejście z trybu `STOP` (S=0) do `PLAY` (S=1), bez zmiany numeru utworu.
- `STOP` działa odwrotnie – ustawia bit S=0, reszta się nie zmienia.
- `NEXT` i `PREVIOUS` zmieniają tylko numer utworu, zachowując tryb (PLAY/STOP).

Stanem początkowym automatu jest 000- zatrzymany utwór 0.

Tabela 1. Stany projektowanego automatu

S	N1	N0	Znaczenie
0	0	0	utwór 0, zatrzymany
1	0	0	utwór 0, odtwarzany
0	0	1	utwór 1, zatrzymany
1	0	1	utwór 1, odtwarzany
0	1	0	utwór 2, zatrzymany
1	1	0	utwór 2, odtwarzany
0	1	1	utwór 3, zatrzymany
1	1	1	utwór 3, odtwarzany

Schemat automatu i jego przejścia



Rys. 2. Diagram stanów automatu sterującego odtwarzaczem MP3

Tabela 2. Przejścia stanów automatu

Stan bieżący (S N1 N0)	**Wejście**	**Nowy stan (S N1 N0)**	**Opis przejścia**
000	PLAY	100	Start odtwarzania utworu 0
000	STOP	000	Nic się nie zmienia
000	NEXT	001	Zmiana utworu na 1, zatrzymany
000	PREVIOUS	011	Zmiana utworu na 3, zatrzymany
100	STOP	000	Zatrzymanie odtwarzania utworu 0
100	PLAY	100	Nic się nie zmienia
100	NEXT	101	Zmiana utworu na 1, odtwarzany
100	PREVIOUS	111	Zmiana utworu na 3, odtwarzany
001	PLAY	101	Start odtwarzania utworu 1
001	STOP	001	Nic się nie zmienia
001	NEXT	010	Zmiana utworu na 2, zatrzymany
001	PREVIOUS	000	Zmiana utworu na 0, zatrzymany
101	STOP	001	Zatrzymanie odtwarzania utworu 1
101	PLAY	101	Nic się nie zmienia
101	NEXT	110	Zmiana utworu na 2, odtwarzany
101	PREVIOUS	100	Zmiana utworu na 0, odtwarzany
010	PLAY	110	Start odtwarzania utworu 2
010	STOP	010	Nic się nie zmienia
010	NEXT	011	Zmiana utworu na 3, zatrzymany
010	PREVIOUS	001	Zmiana utworu na 1, zatrzymany
110	STOP	010	Zatrzymanie odtwarzania utworu 2
110	PLAY	110	Nic się nie zmienia
110	NEXT	111	Zmiana utworu na 3, odtwarzany
110	PREVIOUS	101	Zmiana utworu na 1, odtwarzany

011	PLAY	111	Start odtwarzania utworu 3
011	STOP	011	Nic się nie zmienia
011	NEXT	000	Zmiana utworu na 0, zatrzymany
011	PREVIOUS	010	Zmiana utworu na 2, zatrzymany
111	STOP	011	Zatrzymanie odtwarzania utworu 3
111	PLAY	111	Nic się nie zmienia
111	NEXT	100	Zmiana utworu na 0, odtwarzany
111	PREVIOUS	110	Zmiana utworu na 2, odtwarzany

Tabela prawdy automatu

- Zmienne wejściowe:
  - S, N1, N0- aktualny stan automatu:
    - S – bit trybu (0 = STOP, 1 = PLAY),
    - N1, N0 – bity numeru utworu (00–11).
  - PLAY, STOP, NEXT, PREVIOUS- wejścia sterujące. 1 jeśli aktywne, 0 jeśli nie,
    - Wciśnięcie STOP, gdy S=0 lub PLAY, gdy S=1 nie zmieni stanu automatu.
    - Zakładamy, że tylko jeden z przycisków PLAY, STOP, NEXT, PREVIOUS może być jednocześnie wciśnięty.
    - Jeśli żadne wejście nie jest aktywne, stan się nie zmienia.
- Zmienne wyjściowe:
  - S', N1', N0' – stan następny.

Tabela 3. Tabela prawdy automatu

S	N1	N0	PLAY	STOP	NEXT	PREVIOUS	S'	N1'	N0'
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	wszystkie pozostałe kombinacje				X	X	X
0	0	1	1	0	0	0	1	0	1
0	0	1	0	1	0	0	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	1
0	0	1	wszystkie pozostałe kombinacje				X	X	X
0	1	0	1	0	0	0	1	1	0
0	1	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1	1
0	1	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	1	0
0	1	0	wszystkie pozostałe kombinacje				X	X	X
0	1	1	1	0	0	0	1	1	1
0	1	1	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	0	0	1	1
0	1	1	wszystkie pozostałe kombinacje				X	X	X

1	0	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	1	0	1	0	1
1	0	0	0	0	0	1	1	1	1
1	0	0	0	0	0	0	1	0	0
1	0	0	wszystkie pozostałe kombinacje				X	X	X
1	0	1	1	0	0	0	1	0	1
1	0	1	0	1	0	0	0	0	1
1	0	1	0	0	1	0	1	1	0
1	0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	0	1	0	1
1	0	1	wszystkie pozostałe kombinacje				X	X	X
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	0	1	1	1
1	1	0	0	0	0	1	1	0	1
1	1	0	0	0	0	0	1	1	0
1	1	0	wszystkie pozostałe kombinacje				X	X	X
1	1	1	1	0	0	0	1	1	1
1	1	1	0	1	0	0	0	1	1
1	1	1	0	0	1	0	1	0	0
1	1	1	0	0	0	1	1	1	0
1	1	1	0	0	0	0	1	1	1
1	1	1	wszystkie pozostałe kombinacje				X	X	X

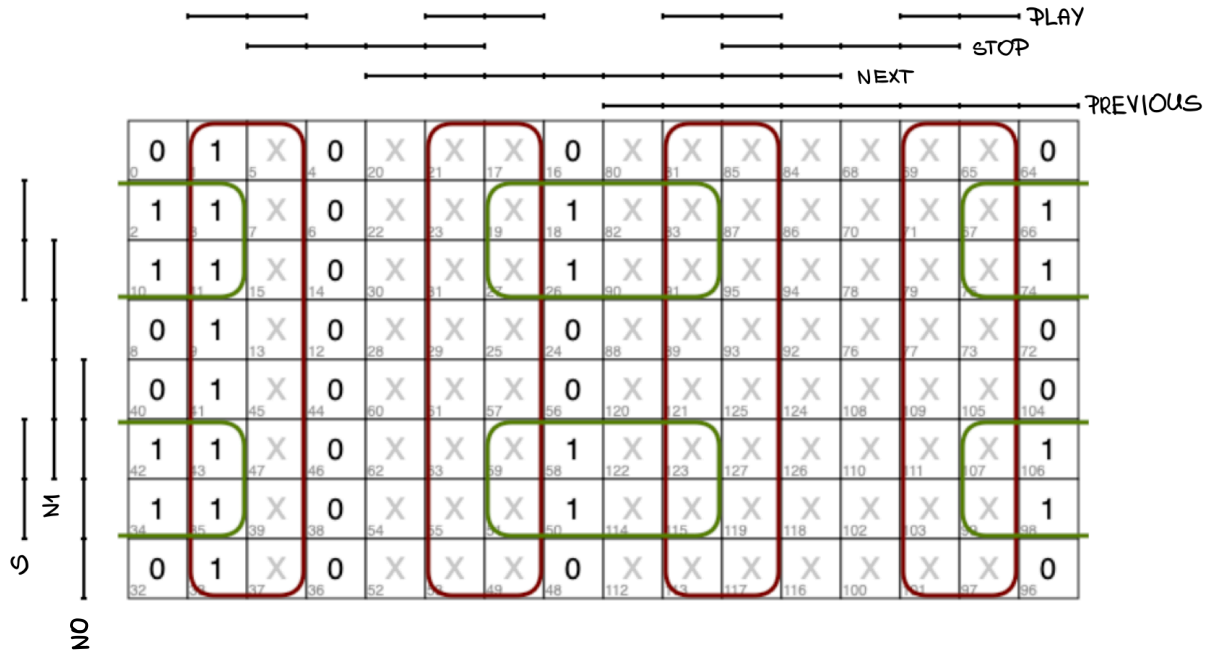
Wyprowadzenie funkcji logicznych dla wyjść

Wersja 1: 7-wejściowe tabele Karnaugh

Tabele zostały stworzone przy użyciu strony:
https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/

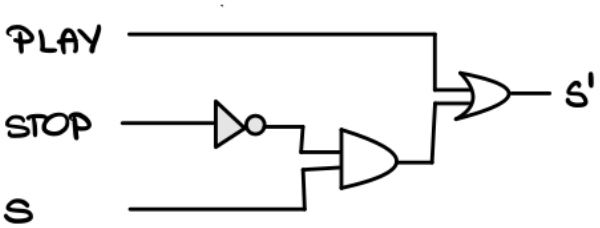
Wyprowadzenie funkcji logicznej dla wyjścia S'

Tabela 4. Tabela Karnaugh dla wyjścia S'



Otrzymana funkcja logiczna:

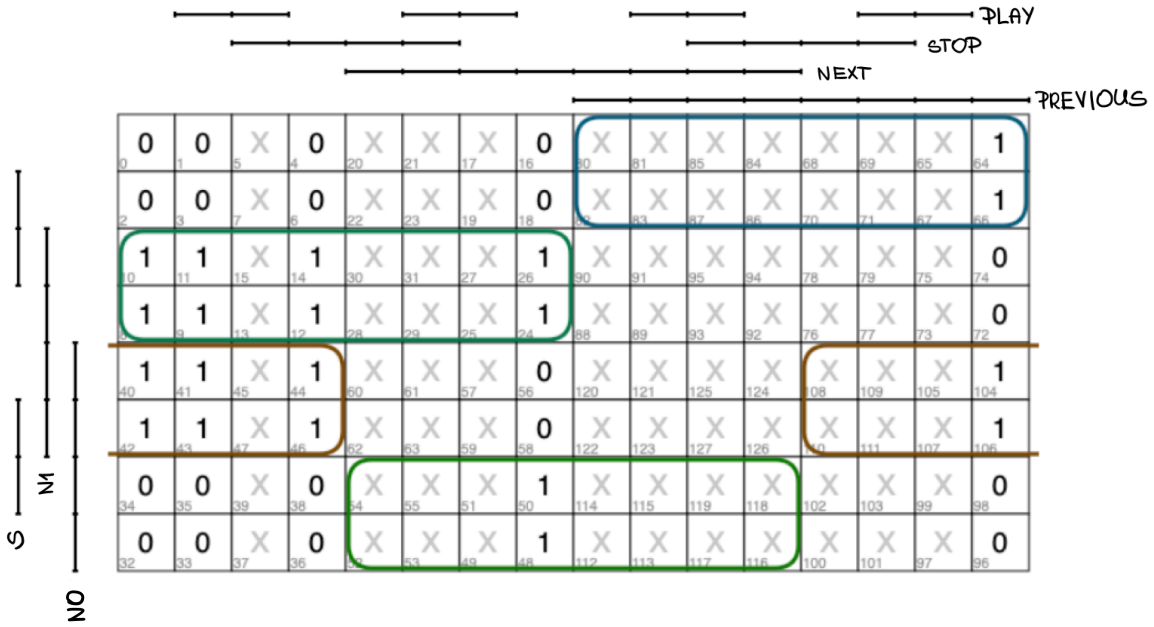
$$S' = PLAY + (\overline{STOP} \cdot S)$$



Rys. 3. Schemat dla wyjścia S2'

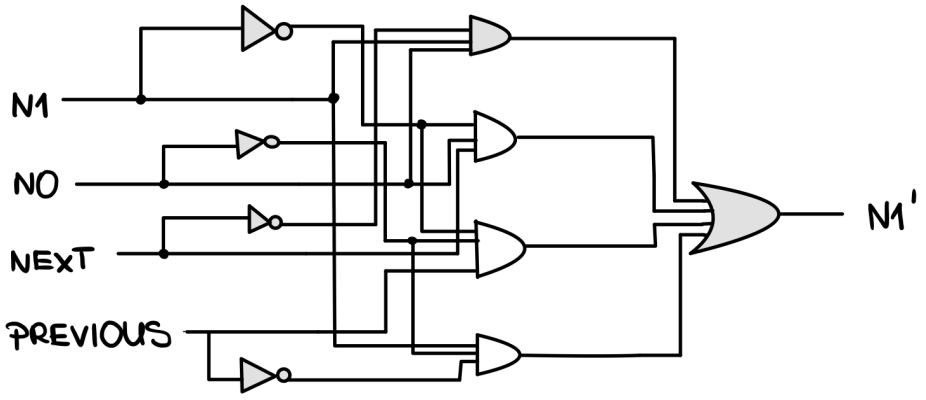
Wyprowadzenie funkcji logicznej dla wyjścia N1'

Tabela 5. Tabela Karnaugh dla wyjścia N1'



Otrzymana funkcja logiczna:

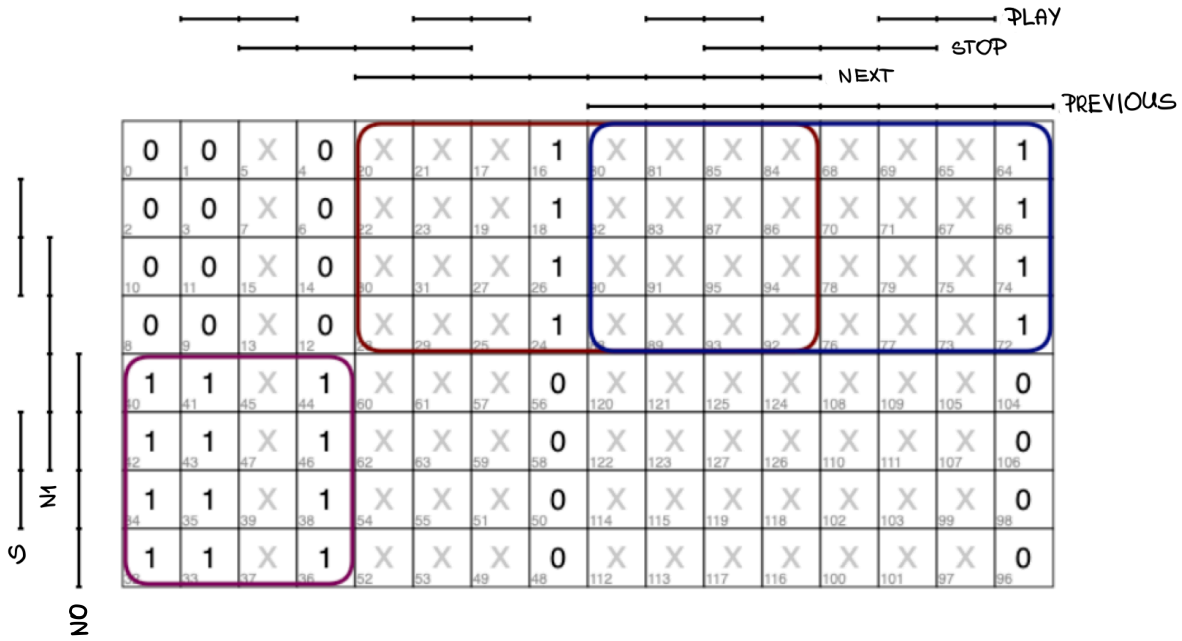
$$N1' = (NO \cdot NEXT \cdot \overline{N1}) + (PREVIOUS \cdot \overline{NO} \cdot \overline{N1}) + (\overline{PREVIOUS} \cdot \overline{NO} \cdot N1) + (NO \cdot \overline{NEXT} \cdot N1)$$



Rys. 4. Schemat dla wyjścia N1'

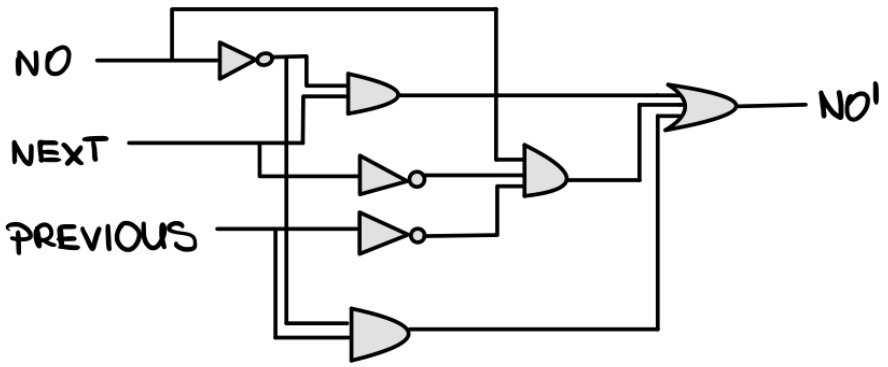
Wyprowadzenie funkcji logicznej dla wyjścia N0’

Tabela 6. Tabela Karnaugh dla wyjścia N0’



Otrzymana funkcja logiczna:

$$N0' = (\overline{N0} \cdot NEXT) + (\overline{PREVIOUS} \cdot N0 \cdot \overline{NEXT}) + (PREVIOUS \cdot \overline{N0})$$



Rys. 5. Schemat dla wyjścia N0’

Sprawdzenie poprawności wyprowadzeń

W celu sprawdzenia poprawności wyprowadzonych wzorów napisany został program w języku python, zaprezentowany poniżej.

1. Implementacja funkcji logicznych

```
def S_out(PLAY, STOP, S):
    return PLAY or (not(STOP) and S)

def N1_out(N0, NEXT, N1, PREVIOUS):
    term1 = N0 and NEXT and not(N1)
    term2 = PREVIOUS and not(N0) and not(N1)
    term3 = not(PREVIOUS) and not(N0) and N1
    term4 = N0 and not(NEXT) and N1
    return term1 or term2 or term3 or term4

def N0_out(N0, NEXT, PREVIOUS):
    term1 = not(N0) and NEXT
    term2 = not(PREVIOUS) and N0 and not(NEXT)
    term3 = PREVIOUS and not(N0)
    return term1 or term2 or term3
```

2. Test poprawności wyprowadzeń

   Wygenerowane zostały wszystkie możliwe kombinacje wartości logicznych dla wejść (z uwzględnieniem założenia, że jednocześnie tylko 1 z przycisków odtwarzacza może być wciśnięty). Wykorzystując funkcje logiczne stworzona została następnie tabela prawdy automatu, którą można porównać z zaprezentowaną wcześniej tabelą prawdy.

```
print("S N1 N0 PLAY STOP NEXT PREVIOUS | S' N1' N0'")
for PLAY, STOP, S, N1, N0, NEXT, PREVIOUS in product([0, 1], repeat=7):
    if PLAY+STOP+NEXT+PREVIOUS > 1: continue
    Sp = S_out(PLAY, STOP, S)
    N1p = N1_out(N0, NEXT, N1, PREVIOUS)
    N0p = N0_out(N0, NEXT, PREVIOUS)
    print(f"{S:>1} {N1:>2} {N0:>2} {PLAY:>4} {STOP:>4} {NEXT:>4} {PREVIOUS:>8} |
{Sp:>1}  {N1p:>2}  {N0p:>2}")
```

S	N1	N0	PLAY	STOP	NEXT	PREVIOUS	S'	N1'	N0'
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0	1
0	1	0	0	0	1	0	0	1	1
0	1	1	0	0	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	1	1	1	1
1	0	0	0	0	1	0	1	0	1
1	0	1	0	0	0	0	1	0	1
1	0	1	0	0	0	1	1	0	0
1	0	1	0	0	1	0	1	1	0
1	1	0	0	0	0	0	1	1	0
1	1	0	0	0	0	1	1	0	1
1	1	0	0	0	1	0	1	1	1
1	1	1	0	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1	0
1	1	1	0	0	1	0	1	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0	1
0	1	0	0	1	0	0	0	1	0
0	1	1	0	1	0	0	0	1	1
1	0	0	0	1	0	0	0	0	0
1	0	1	0	1	0	0	0	0	1
1	1	0	0	1	0	0	0	1	0
1	1	1	0	1	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	0	1
0	1	0	1	0	0	0	1	1	0
0	1	1	1	0	0	0	1	1	1
1	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	1	0	1
1	1	0	1	0	0	0	1	1	0
1	1	1	1	0	0	0	1	1	1

Rys. 6. Tabela prawdy uzyskana w programie testującym wyprowadzone funkcje

Po uruchomieniu programu otrzymaliśmy tabelę prawdy przedstawioną na rysunku 6. Uzyskane wyniki zgadzają się z przedstawioną wcześniej tabelą 3, co wskazuje na poprawność wyprowadzeń.

Wersja 2: wykorzystanie biblioteki logicmin w Pythonie

Spróbowaliśmy także wyznaczenia funkcji logicznych na inny sposób- za pomocą biblioteki logicmin w języku Python. Jest to narzędzie pozwalające na wygodne tworzenie oraz minimalizację funkcji logicznych poprzez wprowadzenie odpowiednich danych wejściowych i wyjściowych w postaci tabeli prawdy.

Poniżej przedstawiono fragment kodu w języku Python, który wykorzystaliśmy.

```
import pandas
import logicmin

df = pandas.read_csv("tabela.csv", sep=',')

t = logicmin.TT(7, 3)

input = df.iloc[:, 0:7].to_numpy().tolist()
output = df.iloc[:, 7::].to_numpy().tolist()

joined_input = ["".join(str(el) for el in row) for row in input]
joined_output = ["".join(str(el) for el in row) for row in output]

for i in range(len(joined_input)): t.add(joined_input[i], joined_output[i])

sols = t.solve()

xnames = ['S', 'N1', 'N0', 'PLAY', 'STOP', 'NEXT', 'PREV']
ynames = ["S'", "N1'", "N0'"]

print(sols.printN(xnames=xnames, ynames=ynames))
```

Otrzymane funkcje to:

N0' <= N0'.PLAY'.STOP'.NEXT'.PREV + N0'.PLAY'.STOP'.NEXT.PREV' + N0.PLAY'.NEXT'.PREV' + N0.STOP'.NEXT'.PREV'

N1' <= N1'.N0'.PLAY'.STOP'.NEXT'.PREV + N1'.N0.PLAY'.STOP'.NEXT.PREV' + N1.N0'.PLAY'.STOP'.PREV' + N1.N0.PLAY'.STOP'.NEXT' + N1.PLAY'.NEXT'.PREV' + N1.STOP'.NEXT'.PREV'

S' <= S.PLAY'.STOP'.NEXT' + S.PLAY'.STOP'.PREV' + PLAY.STOP'.NEXT'.PREV'

Sprawdzenie poprawności wyprowadzeń

W celu sprawdzenia poprawności wyprowadzonych z użyciem biblioteki wzorów napisany został program w języku python, zaprezentowany poniżej.

1. Implementacja funkcji logicznych

```
def S_out(PLAY, STOP, S, NEXT, PREVIOUS):
    term1 = S and not(PLAY) and not(STOP) and not(NEXT)
    term2 = S and not(PLAY) and not(STOP) and not(PREVIOUS)
    term3 = PLAY and not(STOP) and not(NEXT) and not(PREVIOUS)
    return term1 or term2 or term3

def N1_out(N0, NEXT, N1, PREVIOUS, STOP, PLAY):
    term1 = not(N1) and not(N0) and not(PLAY) and not(NEXT) and PREVIOUS
    term2 = not(N1) and N0 and not(PLAY) and NEXT and not(PREVIOUS)
    term3 = N1 and not(N0) and not(PLAY) and not(STOP) and not(PREVIOUS)
    term4 = N1 and N0 and not(PLAY) and not(STOP) and not(NEXT)
    term5 = N1 and not(PLAY) and not(NEXT) and not(PREVIOUS)
    term6 = N1 and not(STOP) and not(NEXT) and not(PREVIOUS)
    return term1 or term2 or term3 or term4 or term5 or term6

def N0_out(N0, NEXT, PREVIOUS, STOP, PLAY):
    term1 = not(N0) and not(PLAY) and not(STOP) and not(NEXT) and PREVIOUS
    term2 = not(N0) and not(PLAY) and not(STOP) and NEXT and not(PREVIOUS)
    term3 = N0 and not(PLAY) and not(NEXT) and not(PREVIOUS)
    term4 = N0 and not  (STOP) and not(NEXT) and not(PREVIOUS)
    return term1 or term2 or term3 or term4
```

2. Test poprawności wyprowadzeń

   Test działa w taki sam sposób, co przedstawiony wcześniej, przy wyprowadzeniu funkcji z tabel Karnaugh.

```
print("S N1 N0 PLAY STOP NEXT PREVIOUS | S' N1' N0'")
for PLAY, STOP, S, N1, N0, NEXT, PREVIOUS in product([0, 1], repeat=7):
    if PLAY+STOP+NEXT+PREVIOUS > 1: continue
    Sp = S_out(PLAY, STOP, S, NEXT, PREVIOUS)
    N1p = N1_out(N0, NEXT, N1, PREVIOUS, STOP, PLAY)
    N0p = N0_out(N0, NEXT, PREVIOUS, STOP, PLAY)
    print(f"{S:>1} {N1:>2} {N0:>2} {PLAY:>4} {STOP:>4} {NEXT:>4} {PREVIOUS:>8} |
{Sp:>1}  {N1p:>2}  {N0p:>2}")
```

```
S N1 N0 PLAY STOP NEXT PREVIOUS | S' N1' N0'
0  0  0    0    0    0        0 | 0   0   0
0  0  0    0    0    0        1 | 0   1   1
0  0  0    0    0    1        0 | 0   0   1
0  0  1    0    0    0        0 | 0   0   1
0  0  1    0    0    0        1 | 0   0   0
0  0  1    0    0    1        0 | 0   1   0
0  1  0    0    0    0        0 | 0   1   0
0  1  0    0    0    0        1 | 0   0   1
0  1  0    0    0    1        0 | 0   1   1
0  1  1    0    0    0        0 | 0   1   1
0  1  1    0    0    0        1 | 0   1   0
0  1  1    0    0    1        0 | 0   0   0
1  0  0    0    0    0        0 | 1   0   0
1  0  0    0    0    0        1 | 1   1   1
1  0  0    0    0    1        0 | 1   0   1
1  0  1    0    0    0        0 | 1   0   1
1  0  1    0    0    0        1 | 1   0   0
1  0  1    0    0    1        0 | 1   1   0
1  1  0    0    0    0        0 | 1   1   0
1  1  0    0    0    0        1 | 1   0   1
1  1  0    0    0    1        0 | 1   1   1
1  1  1    0    0    0        0 | 1   1   1
1  1  1    0    0    0        1 | 1   1   0
1  1  1    0    0    1        0 | 1   0   0
0  0  0    0    1    0        0 | 0   0   0
0  0  1    0    1    0        0 | 0   0   1
0  1  0    0    1    0        0 | 0   1   0
0  1  1    0    1    0        0 | 0   1   1
1  0  0    0    1    0        0 | 0   0   0
1  0  1    0    1    0        0 | 0   0   1
1  1  0    0    1    0        0 | 0   1   0
1  1  1    0    1    0        0 | 0   1   1
0  0  0    1    0    0        0 | 1   0   0
0  0  1    1    0    0        0 | 1   0   1
0  1  0    1    0    0        0 | 1   1   0
0  1  1    1    0    0        0 | 1   1   1
1  0  0    1    0    0        0 | 1   0   0
1  0  1    1    0    0        0 | 1   0   1
1  1  0    1    0    0        0 | 1   1   0
1  1  1    1    0    0        0 | 1   1   1
```

Rys. 7. Tabela prawdy uzyskana w programie testującym wyprowadzone funkcje

Po uruchomieniu programu otrzymaliśmy tabelę prawdy przedstawioną na rysunku 7. Uzyskane wyniki zgadzają się z przedstawioną wcześniej tabelą 3, co wskazuje na poprawność wyprowadzeń.

Wzory otrzymane przy wykorzystaniu biblioteki logicmin są jednak znacznie bardziej skomplikowane, wymagają większej liczby bramek. Zdecydowaliśmy się więc w dalszej implementacji wykorzystać wzory wyprowadzone wcześniej przy użyciu tabel Karnaugh.

Symulacja działania odtwarzacza

Przed przystąpieniem do implementacji układu w środowisku Multisim, przygotowaliśmy program w języku Python, który symuluje działanie projektowanego automatu. Program ten wykorzystuje uprzednio wyprowadzone funkcje logiczne odpowiadające przejściom stanów oraz generacji wyjść.

Dzięki tej symulacji możliwe było przetestowanie poprawności logiki działania odtwarzacza oraz jego reakcji na różne kombinacje sygnałów wejściowych, takich jak PLAY, STOP, NEXT i PREVIOUS. Obsługa programu odbywa się za pośrednictwem wiersza poleceń, co pozwala w łatwy sposób weryfikować zachowanie systemu w kolejnych cyklach działania automatu.

```
def next_state(B, D, F, A, C, E, G):
    S_next = A or (B and not(C))
    N1_next = (F and E and not(D)) or (G and not(F) and not(D)) or (not(G) and
not(F) and D) or (F and not(E) and D)
    N0_next = (not(F) and E) or (not(G) and F and not(E)) or (G and not(F))
    return (int(S_next), int(N1_next), int(N0_next))


def bin_to_track(N1, N0):
    return (N1 << 1) | N0


# Mapa komend na sygnały wejściowe
COMMANDS = {
    'PLAY':     {'PLAY': 1, 'STOP': 0, 'NEXT': 0, 'PREV': 0},
    'STOP':     {'PLAY': 0, 'STOP': 1, 'NEXT': 0, 'PREV': 0},
    'NEXT':     {'PLAY': 0, 'STOP': 0, 'NEXT': 1, 'PREV': 0},
    'PREV':     {'PLAY': 0, 'STOP': 0, 'NEXT': 0, 'PREV': 1}
}

# Początkowy stan
state = (0, 0, 0)

print()

print("Symulacja sterowania odtwarzaczem.")
print(f"Początkowy stan: S={state[0]}, TRACK={state[1], state[2]}")
print("Dostępne komendy: PLAY, STOP, NEXT, PREV, EXIT")
```

```
while True:
    cmd = input("\nPodaj komendę: ").strip().upper()
    if cmd == 'EXIT':
        print("Zakończono symulację.")
        break
    elif cmd not in COMMANDS:
        print("Nieznana komenda. Spróbuj ponownie.")
        continue

    S, N1, N0 = state
    input_signal = COMMANDS[cmd]
    state = next_state(S, N1, N0,
                       input_signal['PLAY'],
                       input_signal['STOP'],
                       input_signal['NEXT'],
                       input_signal['PREV'])
    print(f"Stan: S={state[0]}, TRACK={state[1], state[2]}")
```

Przykładowe działanie programu:

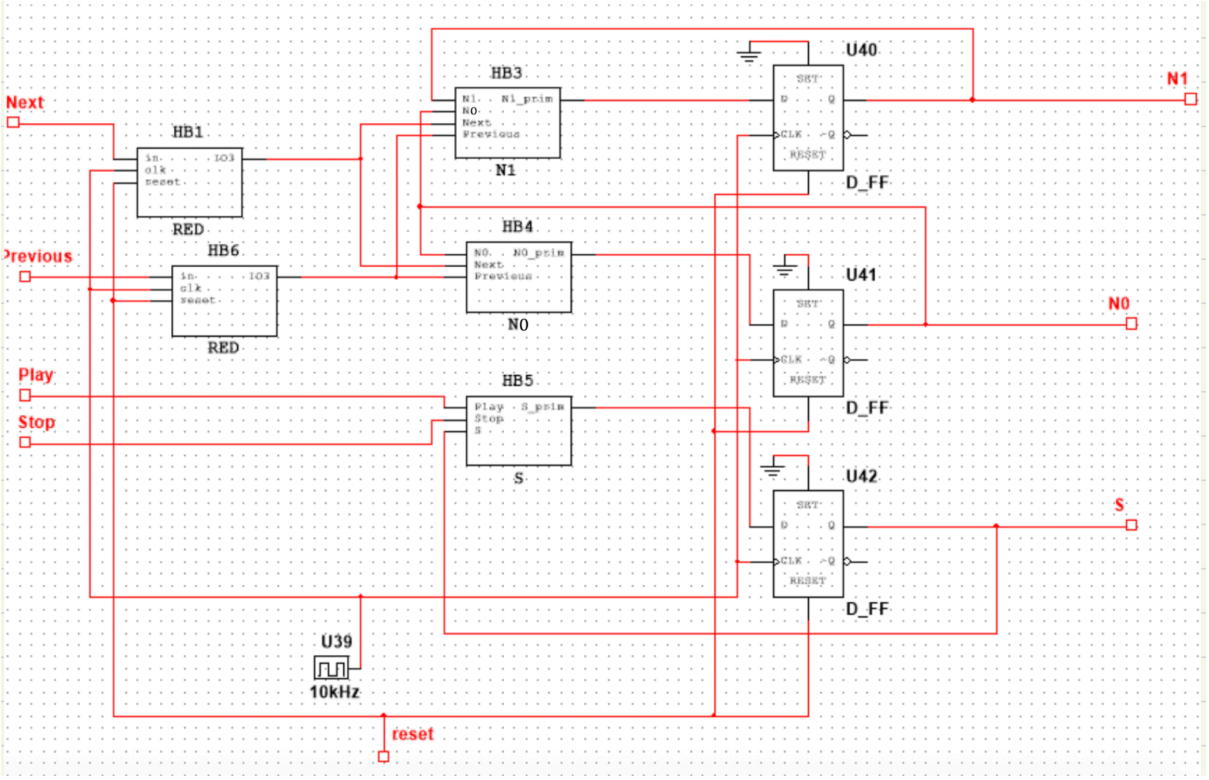
```
Symulacja sterowania odtwarzaczem.
Początkowy stan: S=0, TRACK=(0, 0)
Dostępne komendy: PLAY, STOP, NEXT, PREV, EXIT

Podaj komendę: PLAY
Stan: S=1, TRACK=(0, 0)

Podaj komendę: NEXT
Stan: S=1, TRACK=(0, 1)

Podaj komendę: NEXT
Stan: S=1, TRACK=(1, 0)

Podaj komendę: PREV
Stan: S=1, TRACK=(0, 1)

Podaj komendę: STOP
Stan: S=0, TRACK=(0, 1)

Podaj komendę: NEXT
Stan: S=0, TRACK=(1, 0)

Podaj komendę: NEXT
Stan: S=0, TRACK=(1, 1)

Podaj komendę: NEXT
Stan: S=0, TRACK=(0, 0)

Podaj komendę: STOP
Stan: S=0, TRACK=(0, 0)

Podaj komendę: PLAY
Stan: S=1, TRACK=(0, 0)

Podaj komendę: PREV
Stan: S=1, TRACK=(1, 1)

Podaj komendę: EXIT
Zakończono symulację.
```

Implementacja automatu sterującego

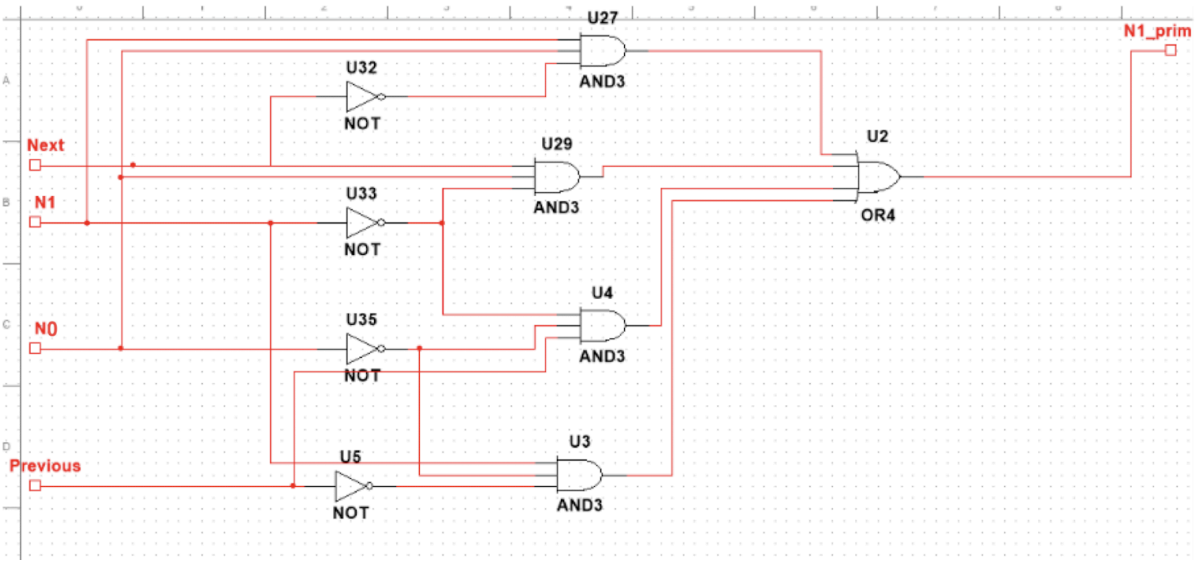
W implementacji wykorzystane zostały przerzutniki D.
Tabela 6. Tabela przejść przerzutnika D

CLK	D	Q(n+1)
↖	1	1
↖	0	0

Automat sterujący

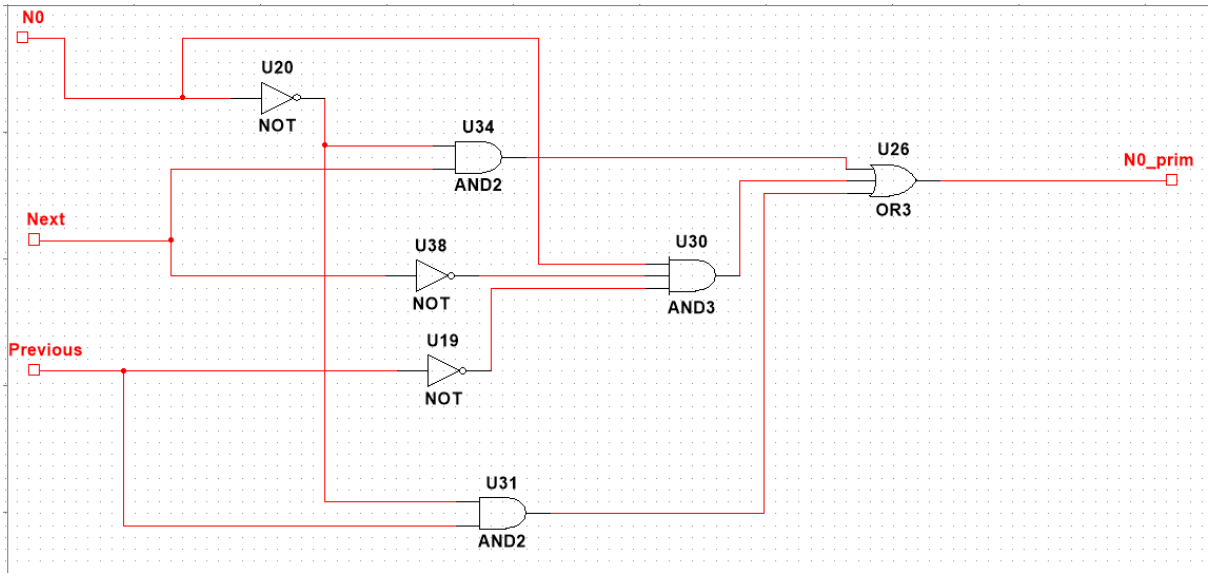


Układ N1



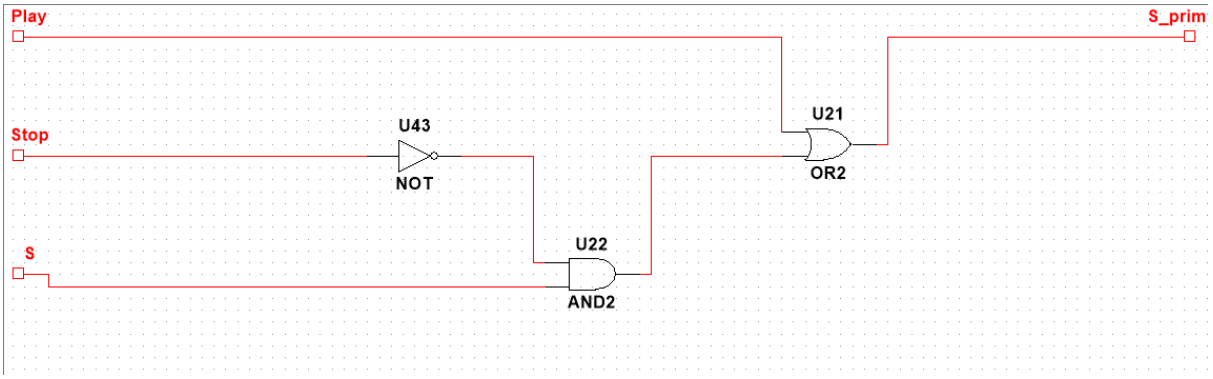
$$N1' = (N0 \cdot NEXT \cdot \overline{N1}) + (PREVIOUS \cdot \overline{N0} \cdot \overline{N1}) + (\overline{PREVIOUS} \cdot \overline{N0} \cdot N1) + (N0 \cdot \overline{NEXT} \cdot N1)$$

Układ N0



$$N0' = (\overline{N0} \cdot NEXT) + (\overline{PREVIOUS} \cdot N0 \cdot \overline{NEXT}) + (PREVIOUS \cdot \overline{N0})$$

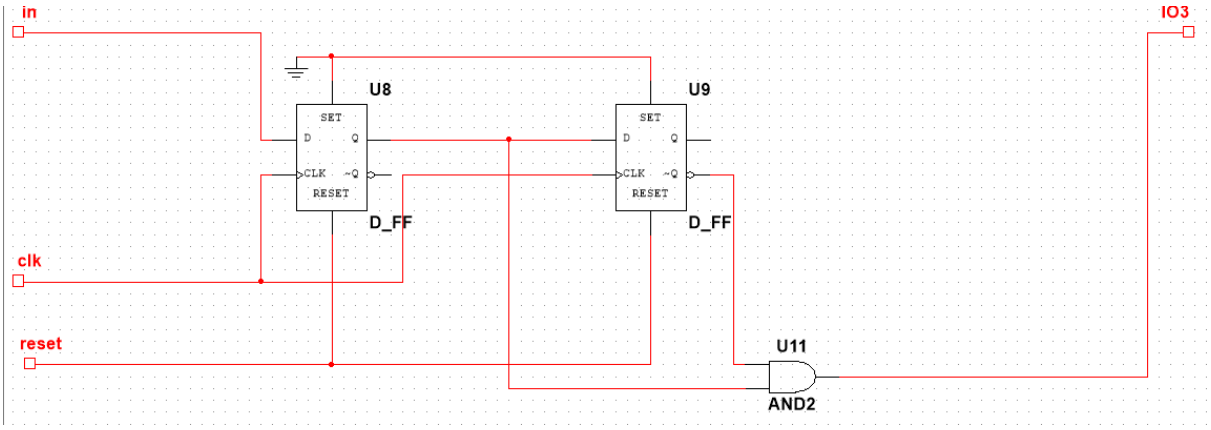
Układ S



$$S' = PLAY + (\overline{STOP} \cdot S)$$

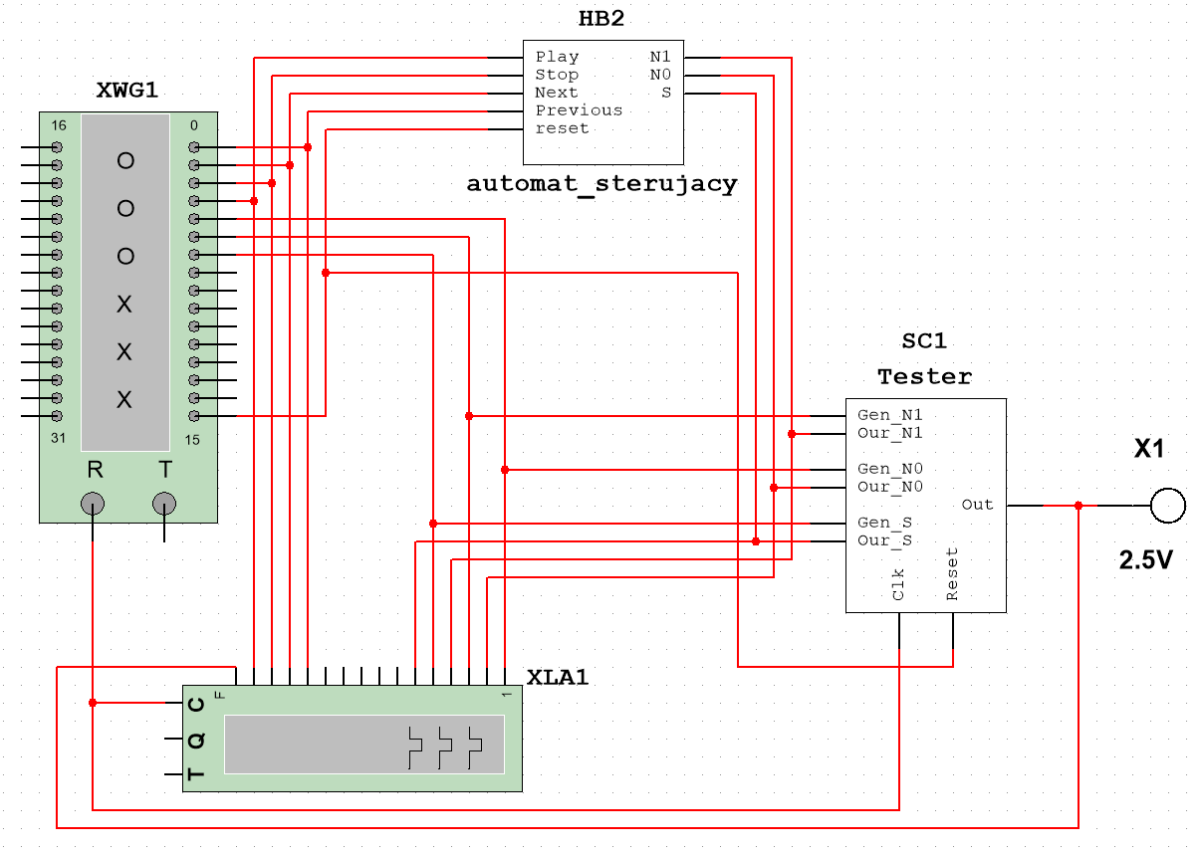
RED - Rising Edge Detector

Wykrywa zbocze narastające przycisku, generując impuls o długości jednego cyklu zegara, gdy na wejściu wystąpi zbocze narastające. Dzięki temu układ odpowiada na sam akt wciśnięcia przycisku, co pozwala lepiej zaobserwować zasadę jego działania (uniknięcie zbyt szybkiego przeskakiwania wartości na wyświetlaczu).



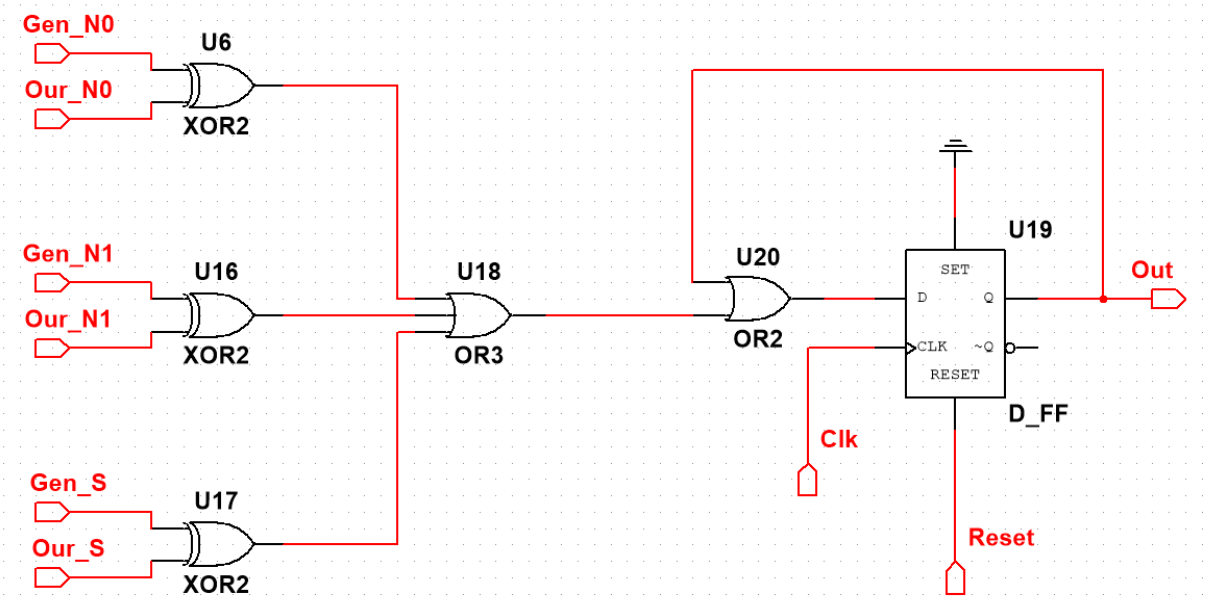
Testowanie układu

Do przetestowania naszego układu użyliśmy generatora słów, zbudowanego przez nas układu Tester, do którego podłączyliśmy "diodę" oraz analizatora stanów logicznych.



Implementacja układu Tester

Układ porównuje sygnały z automatu sterującego z odpowiadającymi im sygnałami z generatora słów. Gdy, któraś para jest niezgodna wysyłany jest sygnał do przełącznika, który zostaje w stanie “błędu” dopóki nie pojawi się sygnał Reset.

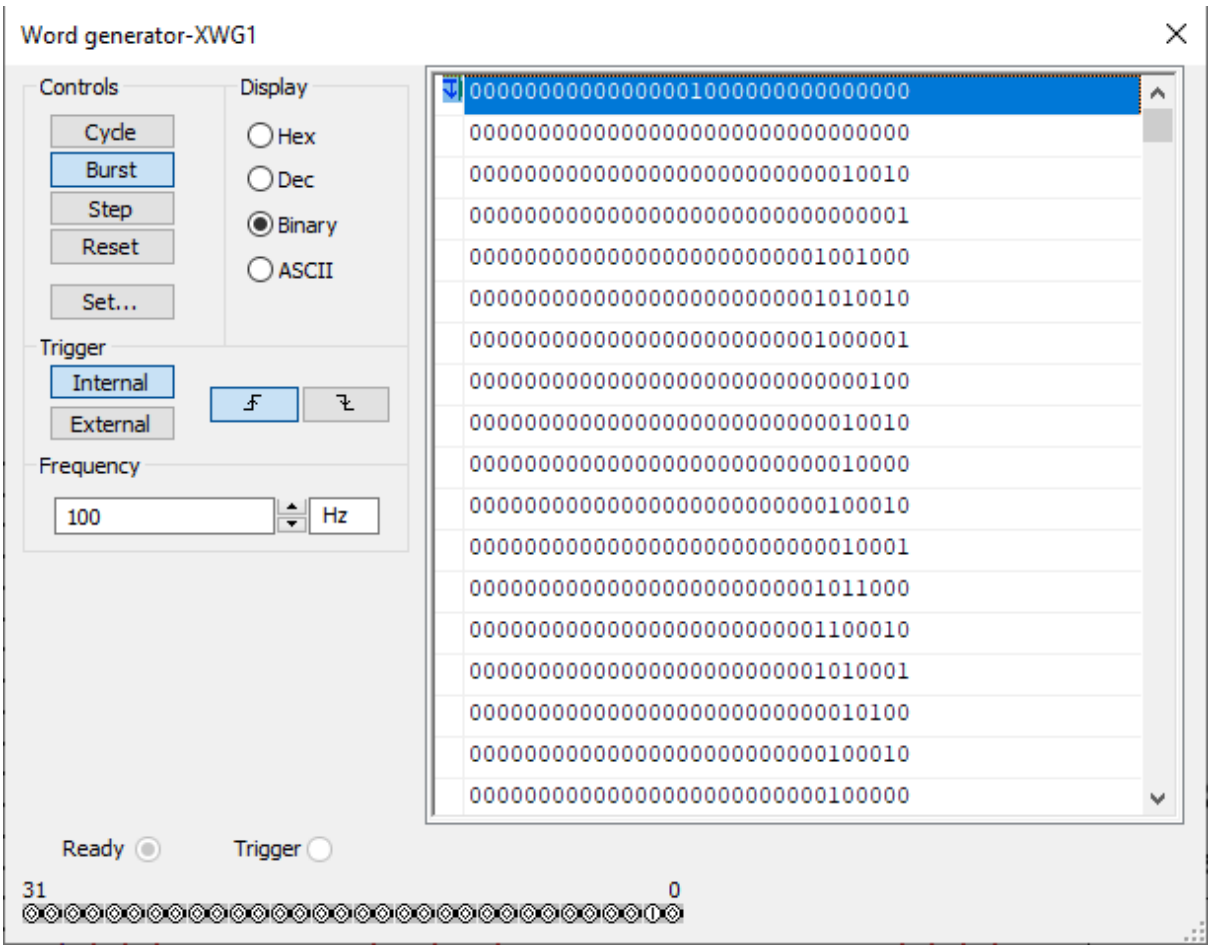


Ustawienia generatora słów

Nasz generator słów generuje na początku sygnał resetu dla układów a następnie kolejne instrukcje testujące każde przejścia. Gdzie w każdym słowie licząc od prawej 1 bit odpowiadał guzikowi Previous, 2 bit guzikowi Next, 3 bit guzikowi Stop, 4 bit guzikowi Play (gdy w słowie było “1” oznaczało wciśnięcie guzika), 5 bit oznacza stan N0, 6 bit oznacza stan N1, 7 bit oznacza stan S oraz 16 bit oznaczający RESET.

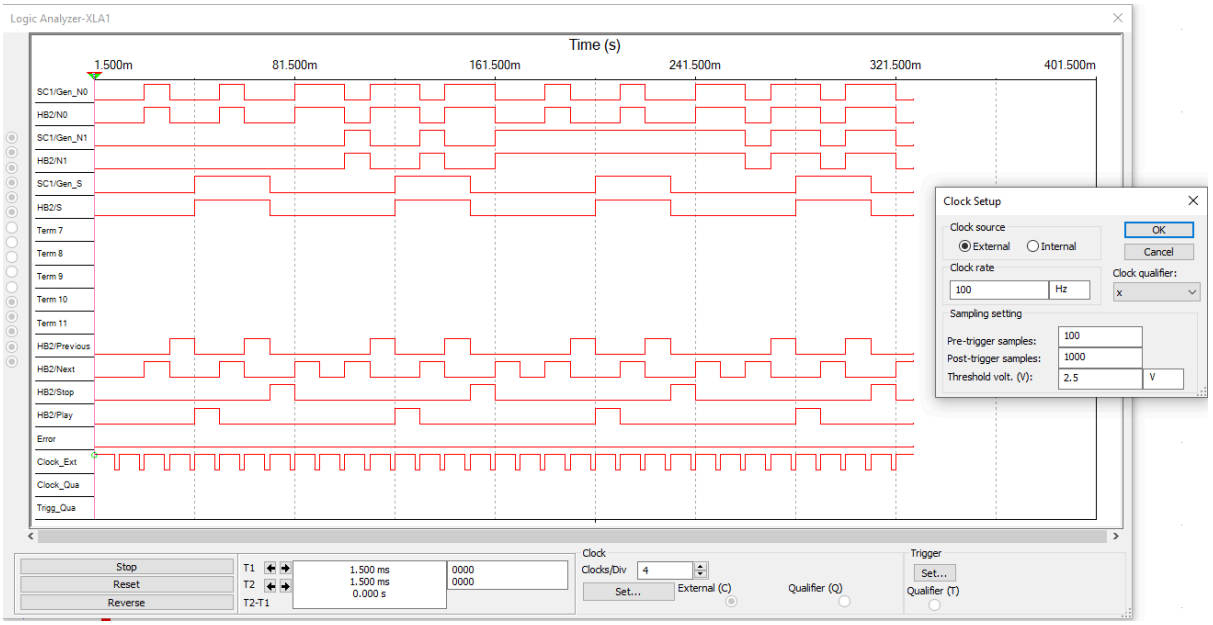
RESET	00000000000000001000000000000000
Zastopowany utwór 0	00000000000000000000000000000000
Zastopowany utwór 1	00000000000000000000000000010010
Zastopowany utwór 0	00000000000000000000000000000001
Odtwarzany utwór 0	00000000000000000000000001001000
Odtwarzany utwór 1	00000000000000000000000001010010
Odtwarzany utwór 0	00000000000000000000000001000001
Zastopowany utwór 0	00000000000000000000000000000100

Zastopowany utwór 1	00000000000000000000000000010010
“Odciśnięcie guzika Next”	00000000000000000000000000010000
Zastopowany utwór 2	00000000000000000000000000100010
Zastopowany utwór 1	00000000000000000000000000010001
Odtwarzany utwór 1	00000000000000000000000001011000
Odtwarzany utwór 2	00000000000000000000000001100010
Odtwarzany utwór 1	00000000000000000000000001010001
Zastopowany utwór 1	00000000000000000000000000010100
Zastopowany utwór 2	00000000000000000000000000100010
“Odciśnięcie guzika Next”	00000000000000000000000000100000
Zastopowany utwór 3	00000000000000000000000000110010
Zastopowany utwór 2	00000000000000000000000000100001
Odtwarzany utwór 2	00000000000000000000000001101000
Odtwarzany utwór 3	00000000000000000000000001110010
Odtwarzany utwór 2	00000000000000000000000001100001
Zastopowany utwór 2	00000000000000000000000000100100
Zastopowany utwór 3	00000000000000000000000000110010
“Odciśnięcie guzika Next”	00000000000000000000000000110000
Zastopowany utwór 0	00000000000000000000000000000010
Zastopowany utwór 3	00000000000000000000000000110001
Odtwarzany utwór 3	00000000000000000000000001111000
Odtwarzany utwór 0	00000000000000000000000001000010
Odtwarzany utwór 3	00000000000000000000000001110001
Zastopowany utwór 3	00000000000000000000000000110100
Zastopowany utwór 0	00000000000000000000000000000010


Word generator-XWG1
Controls
Cycle
Burst
Step
Reset
Set...
Display
Hex
Dec
Binary
ASCII
Trigger
Internal
External
Frequency
100
Hz
00000000000000001000000000000000
00000000000000000000000000000000
00000000000000000000000000010010
00000000000000000000000000000001
00000000000000000000000001001000
00000000000000000000000001010010
00000000000000000000000001000001
00000000000000000000000000000100
00000000000000000000000000010010
00000000000000000000000000010000
00000000000000000000000000100010
00000000000000000000000000010001
00000000000000000000000001011000
00000000000000000000000001100010
00000000000000000000000001010001
00000000000000000000000000010100
00000000000000000000000000100010
00000000000000000000000000100000
Ready
Trigger
31
0

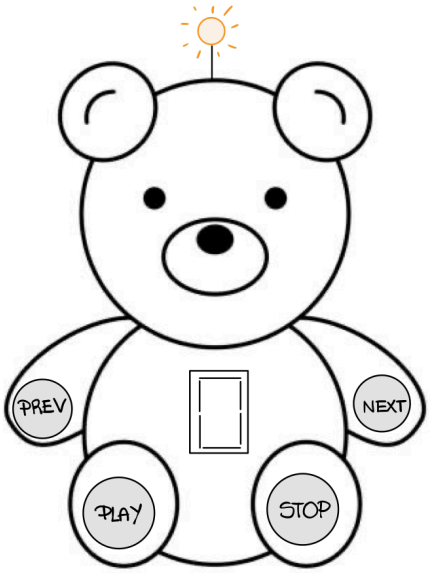
Analizator stanów logicznych

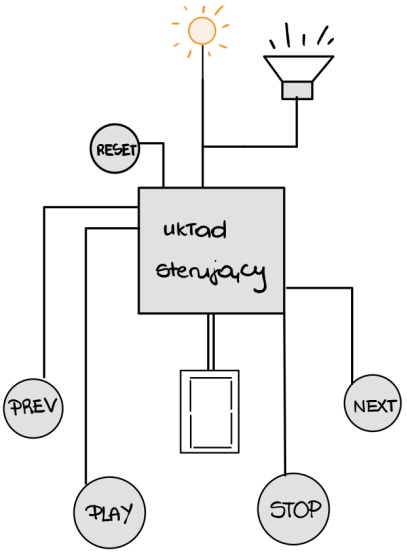
W analizatorze możemy zobaczyć przebieg wszystkich sygnałów oraz w razie błędu będziemy mogli zobaczyć kiedy dokładnie on wystąpił dzięki podpięciu również wyjścia z podukładu Tester.



Zastosowanie

Interaktywna przytulanka dla dzieci z funkcją odtwarzania uspokajających dźwięków tła (np. biały szum, bicie serca, szum morza, deszcz) z czterema podpisanymi przyciskami na łapkach pluszaka (jak na rysunku) oraz diodą LED informującą o odtwarzaniu (w praktyce mógłby to być nos misia). Na brzuszku / plecach można również umieścić wyświetlacz informujący o numerze odtwarzanego dźwięku.





Wnioski

Analiza wykonanego rozwiązania

- Automat został zaprojektowany w sposób przejrzysty i jednoznaczny – każdy przycisk ma jasno zdefiniowane działanie, co minimalizuje ryzyko błędów interpretacyjnych (z założeniem mechanicznym, że nie ma możliwości wciśnięcia dwóch przycisków jednocześnie)
- Wprowadzenie bitu **S** jako kontrolującego stan odtwarzania (STOP/PLAY) pozwoliło skutecznie oddzielić funkcję zmiany trybu od funkcji zmiany utworu (N1 N0), co uprościło logikę przejść między stanami.
- Użycie dwóch bitów do reprezentacji numeru utworu jest uzasadnione – pozwala na wygodne zakodowanie czterech utworów w sposób binarny (00 do 11).

Inne możliwe podejścia:

1. **Użycie automatu Mealy'ego zamiast Moore'a** – w obecnym rozwiązaniu stany niosą pełną informację o odtwarzaniu i numerze utworu (automaty typu Moore). Można byłoby zastosować automat Mealy'ego, w którym reakcje zależą także od wejść. To mogłoby zmniejszyć liczbę stanów, ale zwiększyłoby złożoność projektową (większa zależność od sygnałów wejściowych i potencjalne utrudnienie implementacji w prostych systemach cyfrowych).

2. **Zastosowanie dekodera utworu i flagi trybu jako osobnych bloków** – zamiast jednego rejestru 3-bitowego można byłoby rozdzielić reprezentację stanu na dwa osobne rejestry: jeden dla trybu (1 bit) i jeden dla numeru utworu (2 bity). Ułatwiłoby to testowanie i ewentualną rozbudowę, np. do większej liczby utworów. Na tym poziomie rozbudowy zwiększyłoby to złożoność układu bez wyraźnych korzyści funkcjonalnych.

3. **Zastosowanie liczników** – NEXT i PREVIOUS mogłyby inkrementować lub dekrementować licznik utworu (modulo 4), zamiast ręcznego kodowania stanów. Ułatwiłoby to skalowanie do większej liczby utworów, ale wymagałby osobnej logiki dla obsługi cyklicznego przełączania, co może wprowadzać błędy przy "nieprawidłowym" obrocie (np. z 11 na 00).