Laboratorium 3
Sterownik windy

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1 Cel zadania

Proszę zaproponować, zbudować i przetestować układ sterujący windą w przykładowym trzykondygnacyjnym budynku.

Winda posiada:

- wskaźnik ruchu windy
- wskaźnik kierunku ruchu windy
- trzy czujniki otwarcia drzwi, po jednym na każdej kondygnacji
- trzy przyciski przywołania windy, po jednym na każdej kondygnacji
- trzy przyciski wyboru piętra w kabinie windy.

Winda powinna posiadać stale aktualizowany wskaźnik aktualnego piętra.

Rzeczy niedopowiedziane w treści zadania, proszę ustalić, doprecyzować i opisać samodzielnie.

2 Rozwiązanie

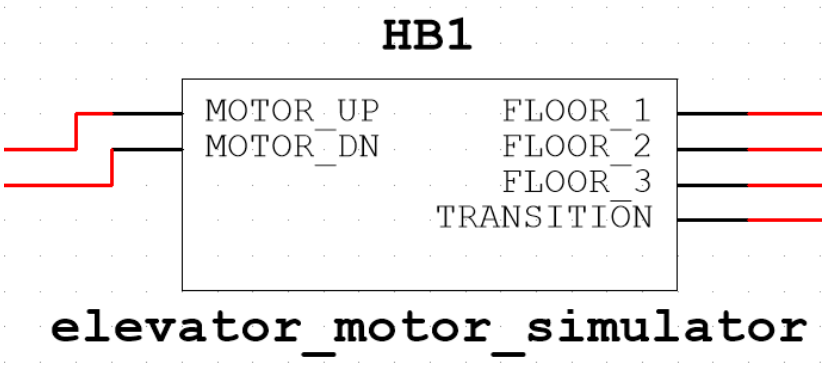
Postanowiliśmy rozbić problem na wiele mniejszych problemów i rozwiązać je osobno aby na końcu połączyć je w jeden działający system. Poniżej zamieszczamy schemat blokowy przedstawiający poszczególne systemy.

2.1 Symulator silnika windy

Aby zasymulować ruch windy wraz z czasem przemieszczania się między piętrami, zdecydowaliśmy się zaimplementować układ bazujący na liczniku i demultiplexerze.

2.1.1 Black box

Poniżej zamieszczamy schemat wyjść i wejść oraz opis logiki układu.



Rysunek 2.1: Black box układu symulującego silnik windy

2.1.2 Wejścia

- `MOTOR_UP` - sygnał wejściowy nakazujący poruszać się windzie w górę
- `MOTOR_DN` - sygnał wejściowy nakazujący poruszać się windzie w dół

2.1.3 Wyjścia

- `FLOOR_1` - sygnał wyjściowy informujący o tym, że winda znajduje się na 1 piętrze, aktywny w stanie wysokim
- `FLOOR_2` - sygnał wyjściowy informujący o tym, że winda znajduje się na 2 piętrze, aktywny w stanie wysokim
- `FLOOR_3` - sygnał wyjściowy informujący o tym, że winda znajduje się na 3 piętrze, aktywny w stanie wysokim
- `TRANSITION` - sygnał wyjściowy informujący o tym że winda jest obecnie w ruchu, aktywny w stanie wysokim

2.1.4 Realizacja układu

Do realizacji układu wykorzystaliśmy 4 bitowy licznik z biblioteki komponentów programu Multisim, układ `74LS193N`. Poniżej zamieszczamy tabelkę przedstawiającą działanie układu:

`CLR`	`~LOAD`	`Up`	`Down`	Mode
H	X	X	X	Reset(Async.)
L	L	X	X	Preset(Async.)
L	H	H	H	No Change
L	H	↑	H	Count Up
L	H	H	↑	Count Down

Tabela 2.1: Źródło: `https://www.multisim.com/help/components/binary-counters/`

Streszczenie

- H - stan wysoki na wejściu
- L - stan niski na wejściu
- X - dowolny stan na wejściu
- ↑ - narastające zbocze sygnału

Stworzyliśmy układ kombinacyjny, który mapuje wyjście zegara, na adres demultiplexera, który z kolei przekazuje jedynkę logiczną na odpowiednie wyjście. Przyjęliśmy, że:

- 0000 - winda jest na 1 piętrze

- 1000 - winda jest na 2 piętrze
- 1111 - winda jest na 3 piętrze
- każdy inny - winda porusza się między piętrami

Jako demultiplexer wykorzystaliśmy układ `U7A 4555BD_5V`. Jest to demultiplexer 1: 4, z 2 bitami adresowymi.

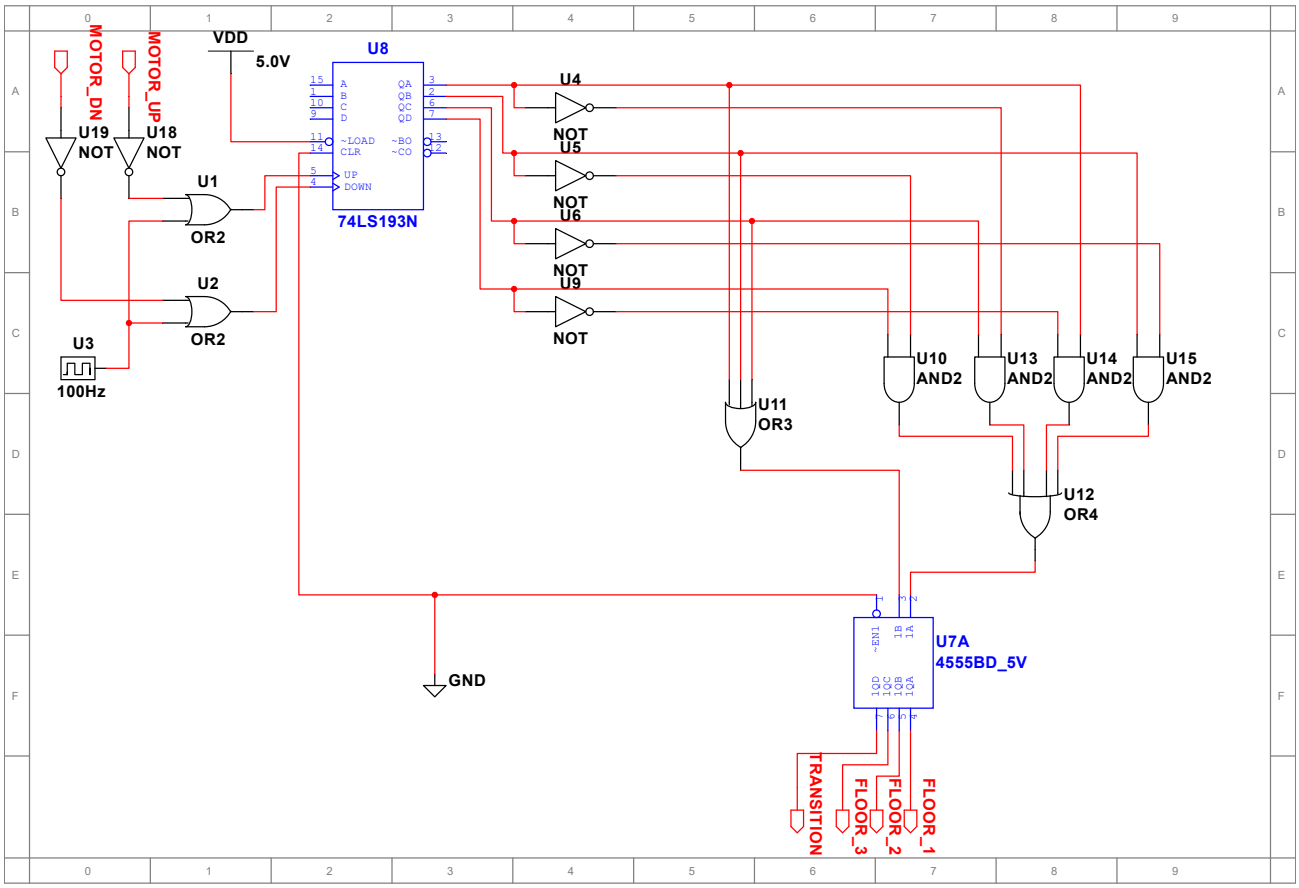
Do wyprowadzenia formuł wykorzystaliśmy skrypt w języku Python, który generuje tabelę prawdy oraz minimalizuje formuły logiczne. Poniżej zamieszczamy kod programu oraz wynik jego działania:

```python
import logicmin
elevator_motor_mux_tt = logicmin.TT(4, 2)

for i in range(16):
    permutation = bin(i).removeprefix("0b").rjust(4, '0')

    a1 = '1'
    b1 = '1'

    if i == 0:
        a1 = '0'
        b1 = '0'
    elif i == 8:
        a1 = '1'
        b1 = '0'
    elif i == 15:
        a1 = '0'
        b1 = '1'

    elevator_motor_mux_tt.add(permutation, [a1, b1])

print("--------------------------------elevator_motor_tt")
sols = elevator_motor_mux_tt.solve()
print(sols.printN(xnames=['QD', 'QC', 'QB', 'QA'],ynames=['1A', '1B']))
```

Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```
--------------------------------elevator_motor_tt
1B <= QA + QB + QC
1A <= QD'.QA + QC'.QB + QC.QA' + QD.QB'
```

Po zaimplementowaniu układu w programie Multisim, uzyskaliśmy następujący schemat:



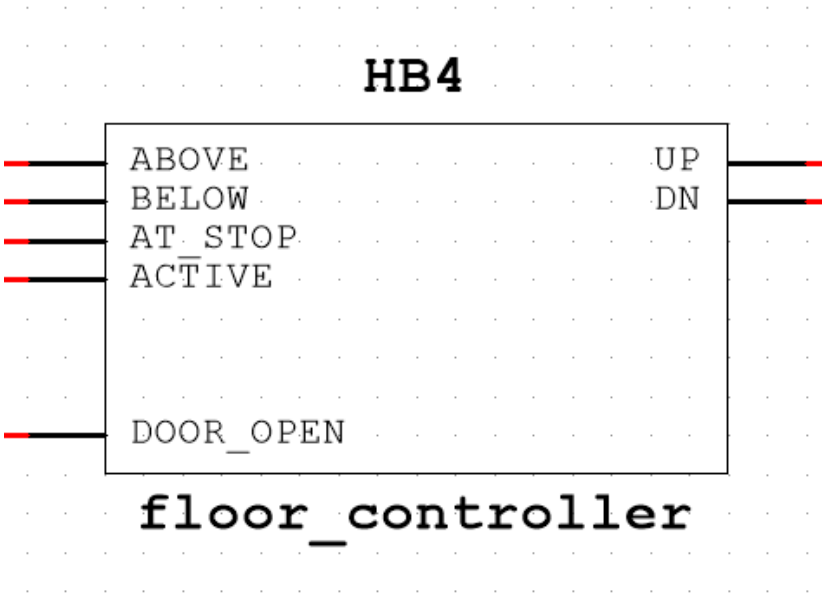
Rysunek 2.2: Schemat układu symulującego silnik windy

Bramki logiczne `NOT` i `OR` na wejściu układu służą do przekazywania stanu wysokiego lub zegara na wejście licznika. Zgodnie z tabelką działania układu zapewniają one, że w momencie gdy chcemy liczyć do góry, układ przekazuje stan wysoki na wejście licznika `DOWN` oraz sygnał zegara na wejście licznika `UP`.

2.2 Kontroler ruchu windy

Prosty układ kombinacyjny przekazujący silnikowi sygnał o ruchu w danym kierunku gdy wszystkie warunki na ruszenie windą zostaną spełnione.

2.2.1 Black box



Rysunek 2.3: Black box układu inicjującego ruch windy

2.2.2 Wejścia

- `DOOR_OPEN` - sygnał wejściowy informujący o stanie otwarcia drzwi
- `ACTIVE` - sygnał wejściowy informujący, że widna otrzymała wezwanie
- `BELOW` - sygnał wejściowy informujący, że piętro docelowe znajduje się poniżej obencej pozycji windy
- `ABOVE` - sygnał wejściowy informujący, że piętro docelowe znajduje się powyżej obencej pozycji windy
- `AT_STOP` - sygnał wejściowy informując, że widna znajduje się na piętrze na, które została wezwana

2.2.3 Wyjścia

- `UP` - sygnał wyjściowy informujący o gotowośći do ruchu w górę
- `DN` - sygnał wyjściowy informujący o gotowośći do ruchu w dół

2.2.4 Realizacja układu

W realizacji układu wykorzystaliśmy dwa przerzutniki SR, które są ustawiane w momencie gdy wszystkie warunki do ruchu w danym kierunku są spełnione i resetowane gdy spełnione są wszystkie warunki konieczne do zatrzymania windy.

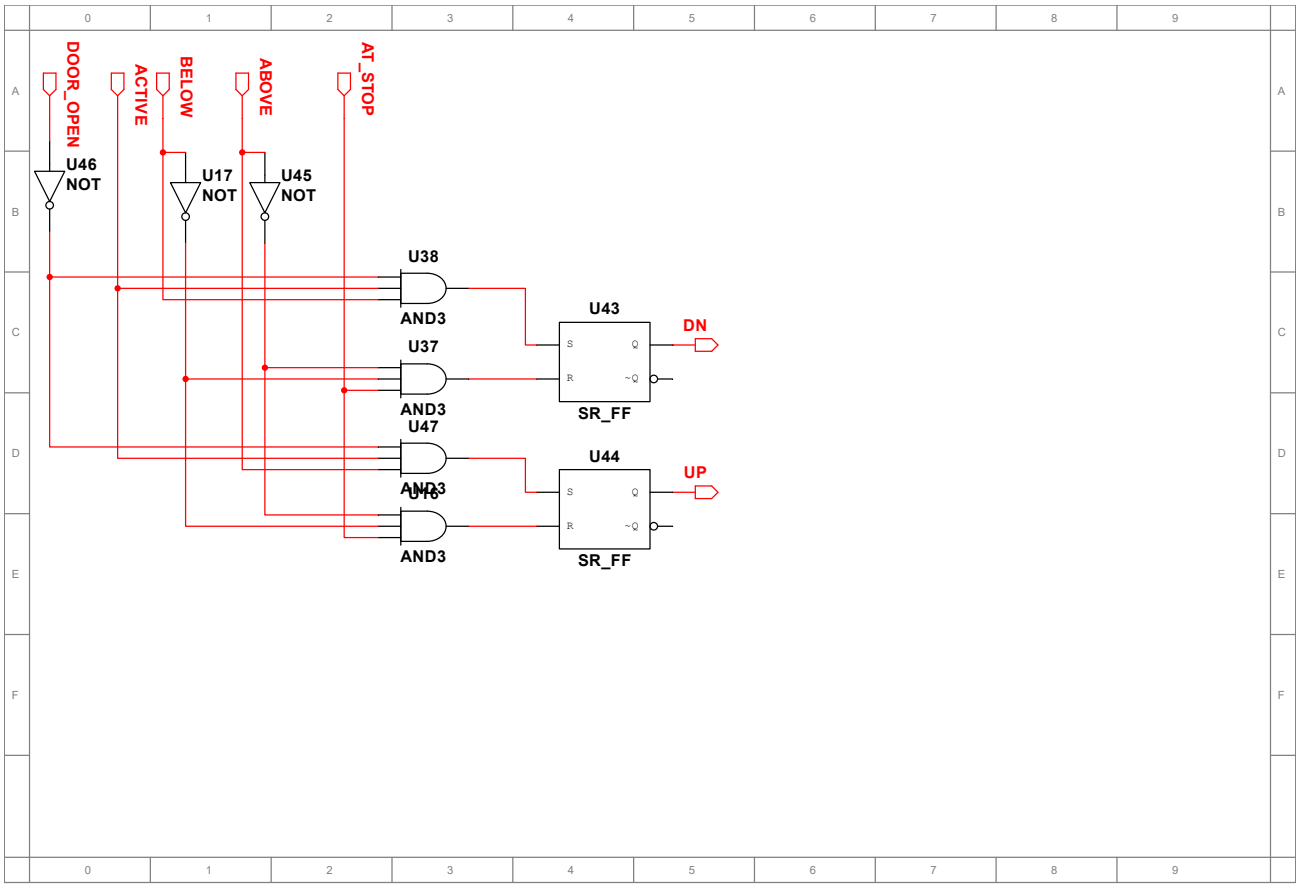
Poniżej przedstawiony został kod programu w języku Python, który wykorzystaliśmy do wyprowadzenia formuł logicznych.

```
import logicmin
floor_controller_tt = logicmin.TT(5, 4)

for i in range(32):
    permutation = bin(i).removeprefix("0b").rjust(5, '0')

    variables = {
        'DOOR_OPEN': int(permutation[0]),
        'ACTIVE': int(permutation[1]),
        'BELOW': int(permutation[2]),
        'ABOVE': int(permutation[3]),
        'AT_STOP': int(permutation[4])
    }

    set_up = '0'
    reset_up = '0'
    set_dn = '0'
    reset_dn = '0'

    if not variables['ABOVE'] and not variables['BELOW'] and variables['AT_STOP']:
        set_up = '0'
        reset_up = '1'
        set_dn = '0'
        reset_dn = '1'

    elif variables['ACTIVE'] and variables['BELOW'] and not variables['DOOR_OPEN']:
        set_up = '0'
        reset_up = '0'
        set_dn = '1'
        reset_dn = '0'

    elif variables['ACTIVE'] and variables['ABOVE'] and not variables['DOOR_OPEN']:
        set_up = '1'
        reset_up = '0'
        set_dn = '0'
        reset_dn = '0'

    floor_controller_tt.add(permutation, [set_up, reset_up, set_dn, reset_dn])

print("------------------------------floor_controller_tt")
sols = floor_controller_tt.solve()
print(sols.printN(xnames=['DOOR_OPEN', 'ACTIVE', 'BELOW', 'ABOVE', 'AT_STOP'],
                  ynames=['SET_UP', 'RESET_UP','SET_DN', 'RESET_DN']))
```

Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```
--------------------------------floor_controller_tt
RESET_DN <= BELOW'.ABOVE'.AT_STOP
SET_DN <= DOOR_OPEN'.ACTIVE.BELOW
RESET_UP <= BELOW'.ABOVE'.AT_STOP
SET_UP <= DOOR_OPEN'.ACTIVE.BELOW'.ABOVE
```

Na podstawie otrzymanych formuł zaimplementowaliśmy w programie Multisim, niżej przedstawiony schemat.

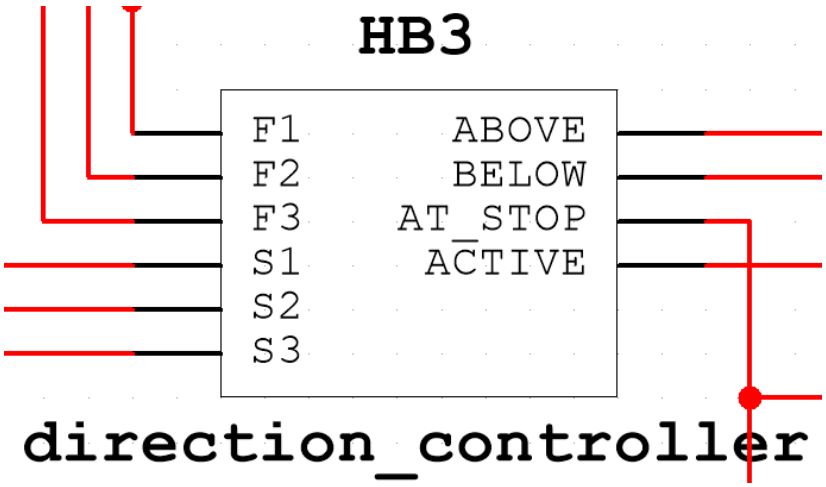


Rysunek 2.4: Schemat układu inicjującego ruch windy

2.3 Kontroler kierunku ruchu

Układ, który na podstawie otrzymanego wezwania rozpoznaje względną pozycję piętra docelowego w odniesieniu do obecnego położenia windy.

2.3.1 Black box



Rysunek 2.5: Black box układu przetwarzającego wezwanie

2.3.2 Wejścia

- `F1` - sygnał wejściowy informujący o tym, że winda znajduje się na 1. piętrze
- `F2` - sygnał wejściowy informujący o tym, że winda znajduje się na 2. piętrze
- `F3` - sygnał wejściowy informujący o tym, że winda znajduje się na 3. piętrze
- `S1` - sygnał wejściowy informujący o wezwaniu windy na 1. piętro
- `S2` - sygnał wejściowy informujący o wezwaniu windy na 2. piętro
- `S3` - sygnał wejściowy informujący o wezwaniu windy na 3. piętro

2.3.3 Wyjścia

- `ABOVE` - sygnał wyjściowy informujący, że piętro docelowe znajduje się powyżej obencej pozycji windy
- `BELOW` - sygnał wyjściowy informujący, że piętro docelowe znajduje się poniżej obencej pozycji windy
- `AT_STOP` - sygnał wyjściowy informujący, że widna znajduje się na piętrze docelowym
- `ACTIVE` - sygnał wyjściowy przekazujący dalej informację o otrzymaniu wezwania

2.3.4 Realizacja układu

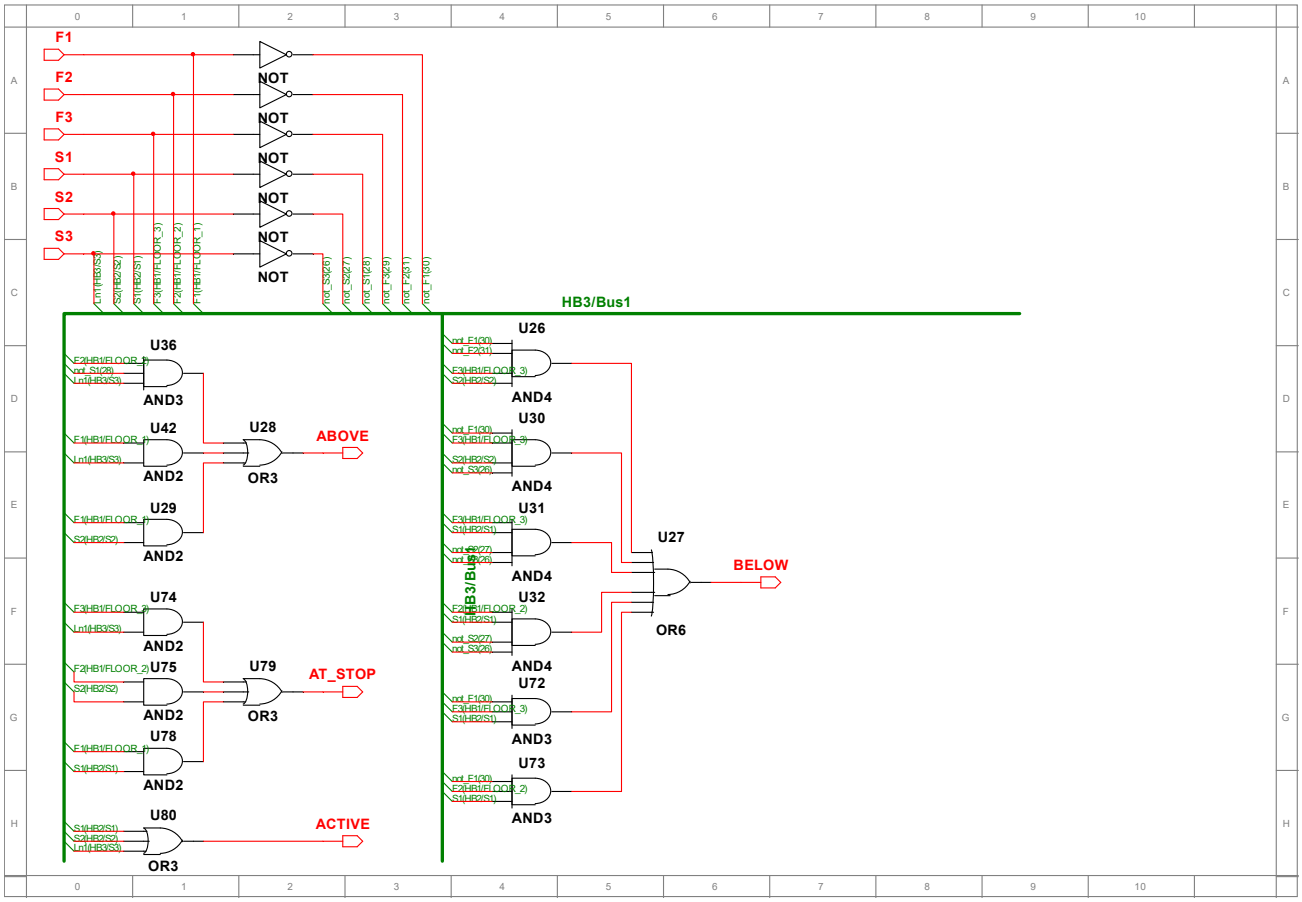
Poniżej przedstawiony został kod programu w języku Python, który wykorzystaliśmy do wyprowadzenia formuł logicznych.

```
import logicmin
direction_controller_tt = logicmin.TT(6, 4)

for i in range(64):
    permutation = bin(i).removeprefix("0b").rjust(6, '0')

    variables = {
        'F1': int(permutation[0]),
        'F2': int(permutation[1]),
        'F3': int(permutation[2]),
        'S1': int(permutation[3]),
        'S2': int(permutation[4]),
        'S3': int(permutation[5])
    }

    above = '0'
    below = '0'
    at_stop = '0'
    active = '0'

    if variables['F1'] and (variables['S2'] or variables['S3']):
        above = '1'

    elif variables['F2'] and variables['S1']:
        below = '1'

    elif variables['F2'] and variables['S3']:
        above = '1'

    elif variables['F3']and (variables['S1'] or variables['S2']):
        below = '1'

    if (variables['F1'] and variables['S1']) or (variables['F2'] and variables['S2'])
                or (variables['F3'] and variables['S3']):
        at_stop = '1'

    if variables['S1'] or variables['S2'] or variables['S3']:
        active = '1'

    direction_controller_tt.add(permutation, [above, below, at_stop, active])

print("-------------------------------direction_controller_tt")
sols = direction_controller_tt.solve()
print(sols.printN(xnames=['F1', 'F2', 'F3', 'S1', 'S2', 'S3'],ynames=['ABOVE','BELOW', 'AT_STOP', 'ACTIVE']))
```

Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```
--------------------------------direction_controller_tt
ACTIVE <= S3 + S2 + S1
AT_STOP <= F3.S3 + F2.S2 + F1.S1
BELOW <= F1'.F2'.F3.S2 + F1'.F3.S2.S3' + F3.S1.S2'.S3' + F2.S1.S2'.S3' +
+ F1'.F3.S1 + F1'.F2.S1
ABOVE <= F2.S1'.S3 + F1.S3 + F1.S2
```

Na podstawie otrzymanych formuł zaimplementowaliśmy w programie Multisim, niżej przedstawiony schemat.

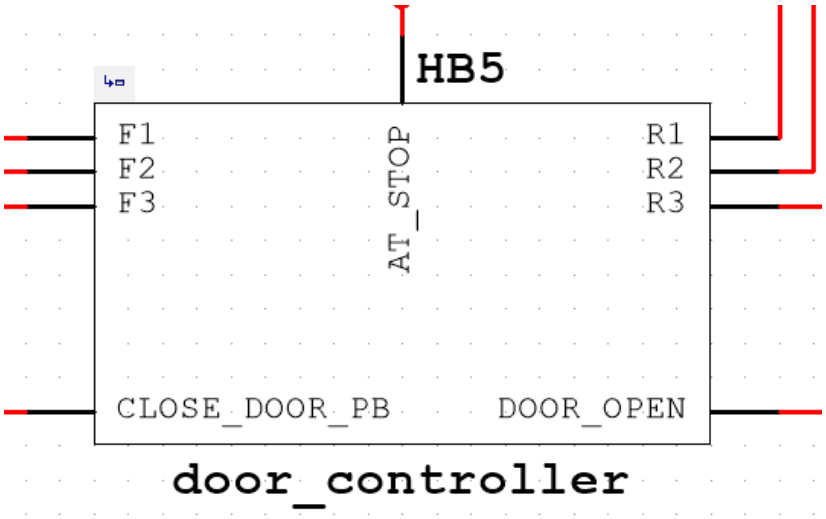


Rysunek 2.6: Schemat układu przetwarzającego wezwanie

2.4 Kontroler drzwi windy

Układ zajmujący się obsługą drzwi po znalezieniu się windy na piętrze docelowym. Przetwarza on również informację o wykonaniu obsługiwanego wezwania i wysyła sygnał do jego resetu.

2.4.1 Black box



Rysunek 2.7: Black box układu kontrolującego obsługę drzwi

2.4.2 Wejścia

- `F1` - sygnał wejściowy informujący o tym, że winda znajduje się na 1. piętrze
- `F2` - sygnał wejściowy informujący o tym, że winda znajduje się na 2. piętrze
- `F3` - sygnał wejściowy informujący o tym, że winda znajduje się na 3. piętrze
- `AT_STOP` - sygnał wejściowy informujący, że widna znajduje się na piętrze docelowym
- `CLOSE_DOOR_PB` - sygnał wejściowy nakazujący zamknięcie drzwi

2.4.3 Wyjścia

- `R1` - sygnał wyjściowy, informujący o wykonaniu wezwania na 1. piętro
- `R2` - sygnał wyjściowy, informujący o wykonaniu wezwania na 2. piętro
- `R3` - sygnał wyjściowy, informujący o wykonaniu wezwania na 3. piętro
- `DOOR_OPEN` - sygnał wyjściowy, informujący o otwarciu drzwi

2.4.4 Realizacja układu

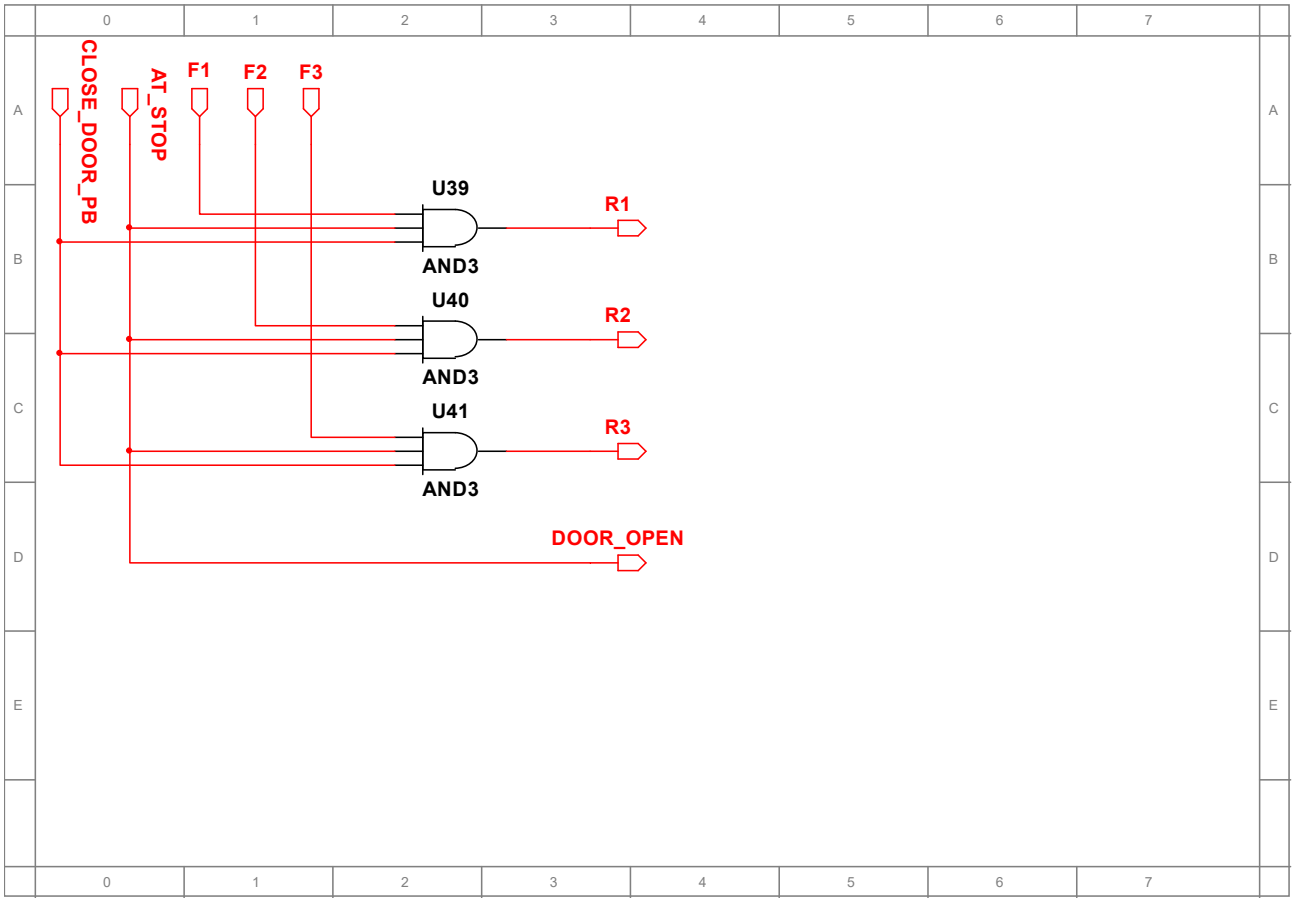
Poniżej przedstawiony został kod programu w języku Python, który wykorzystaliśmy do wyprowadzenia formuł logicznych.

```
import logicmin
door_controller_tt = logicmin.TT(5, 4)

for i in range(32):
    permutation = bin(i).removeprefix("0b").rjust(5, '0')

    variables = {
        'CLOSE_DOOR_PB': int(permutation[0]),
        'AT_STOP': int(permutation[1]),
        'F1': int(permutation[2]),
        'F2': int(permutation[3]),
        'F3': int(permutation[4])
    }

    r1 = '0'
    r2 = '0'
    r3 = '0'
    door_open = '0'

    if variables['AT_STOP']: door_open = '1'
    if variables['AT_STOP'] and variables['CLOSE_DOOR_PB']:
        if variables['F1']: r1 = '1'
        if variables['F2']: r2 = '1'
        if variables['F3']: r3 = '1'

    door_controller_tt.add(permutation, [r1, r2, r3, door_open])

print("--------------------------------door_controller_tt")
sols = door_controller_tt.solve()
print(sols.printN(xnames=['CLOSE_DOOR_PB', 'AT_STOP', 'F1', 'F2', 'F3'],ynames=['R1','R2', 'R3', 'DOOR_OPEN']))
```

Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```
--------------------------------door_controller_tt
DOOR_OPEN <= AT_STOP
R3 <= CLOSE_DOOR_PB.AT_STOP.F3
R2 <= CLOSE_DOOR_PB.AT_STOP.F2
R1 <= CLOSE_DOOR_PB.AT_STOP.F1
```

Na podstawie otrzymanych formuł zaimplementowaliśmy w programie Multisim, niżej przedstawiony schemat.

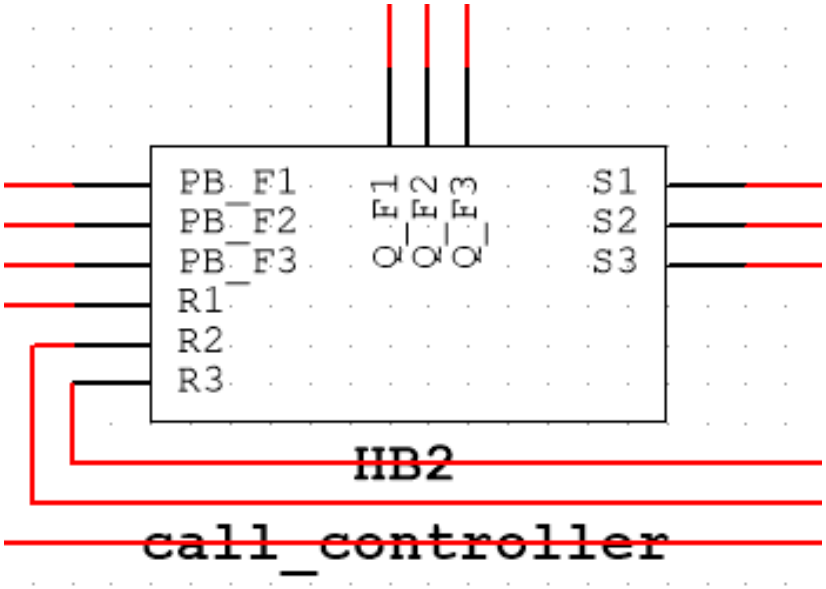


Rysunek 2.8: Schemat układu kontrolującego obsługę drzwi

2.5 Kontroler wezwań

Układ zajmujący się obsługiwaniem wezwań windy w odpowiedniej kolejności.

2.5.1 Black box



Rysunek 2.9: Black box układu obsługującego wezwania

2.5.2 Wejścia

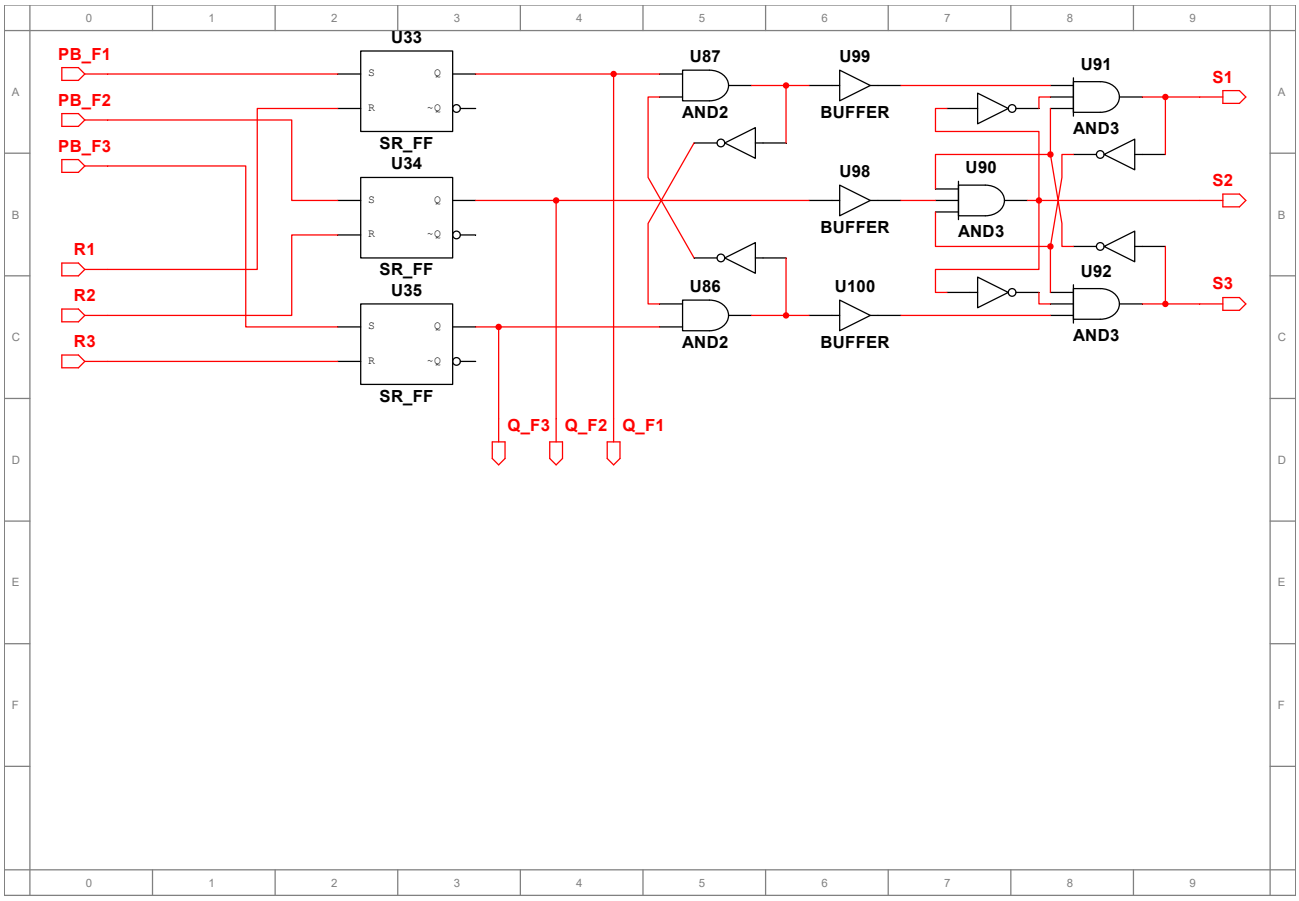
- `PB_F1` - sygnał informujący o wciśnięciu przycisku wzywającego windę na 1. piętro.
- `PB_F2` - sygnał informujący o wciśnięciu przycisku wzywającego windę na 2. piętro.
- `PB_F3` - sygnał informujący o wciśnięciu przycisku wzywającego windę na 3. piętro.
- `R1` - sygnał informujący o wykonaniu wezwania na 1. pietro.
- `R2` - sygnał informujący o wykonaniu wezwania na 2. pietro.
- `R3` - sygnał informujący o wykonaniu wezwania na 3. pietro.

2.5.3 Wyjścia

- `S1` - sygnał obecnie obsługiwanego wezwania na piętro 1.
- `S2` - sygnał obecnie obsługiwanego wezwania na piętro 2.
- `S3` - sygnał obecnie obsługiwanego wezwania na piętro 3.

- `Q_F1` - sygnał informujący, że wezwanie na pietro 1. znajduje się w kolejce do obsłużenia.
- `Q_F2` - sygnał informujący, że wezwanie na pietro 2. znajduje się w kolejce do obsłużenia.
- `Q_F3` - sygnał informujący, że wezwanie na pietro 3. znajduje się w kolejce do obsłużenia.

2.5.4 Realizacja układu



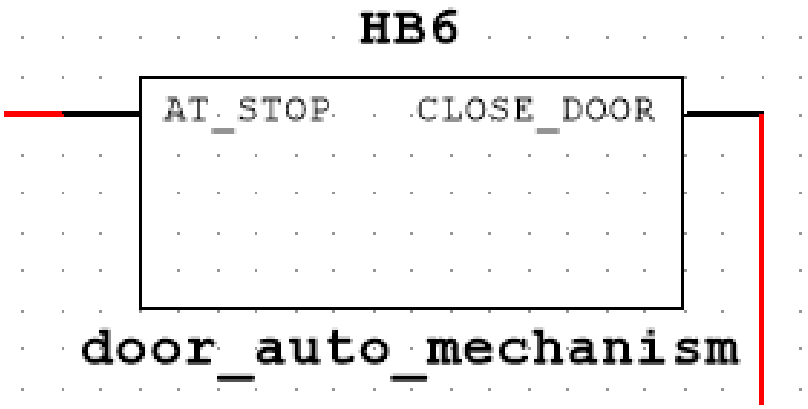
Rysunek 2.10: Schemat układu obsługującego wezwania

W naszej implementacji wykorzystaliśmy trzy przerzutniki SR służące do przechowywania informacji o wezwaniu na dane piętro. Dany przerzutnik jest ustawiany w momencie naciśnięcia odpowiedniego przycisku znajdującego się na piętrze lub wewnątrz windy i resetowany po obsłużeniu wezwania. Kolejną częścią implelementacji jest fragment układu, odpowiedzialny za przekazywanie dalej sygnału o wezwaniach. Sygnały są przekazywane pojedynczo z zachowaniem kolejności w jakiej się pojawiły.

2.6 Układ zamykający drzwi

Układ, który od czasu dotarcia na piętro docelowe odlicza czas po upłynięciu którego wysyła sygnał do zamknięcia drzwi.

2.6.1 Black box



Rysunek 2.11: Black box układu zamykającego drzwi automatycznie

2.6.2 Wejścia

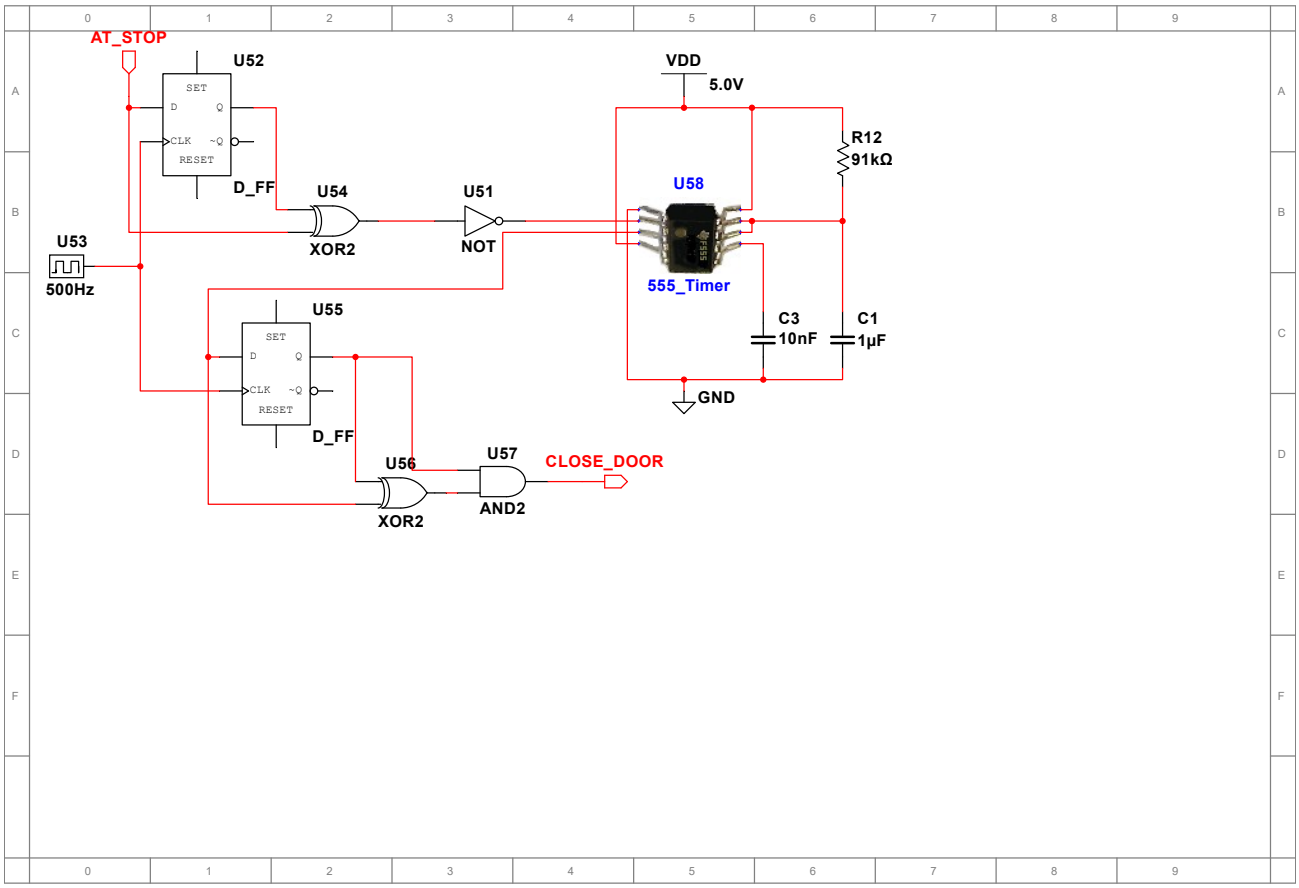
- `AT_STOP` - sygnał wejściowy informujący, że winda znalazła się na piętrze docelowym

2.6.3 Wyjścia

- `CLOSE_DOOR` - sygnał wyjściowy będący sygnałem dla zamknięcia drzwi, jest to krótki puls gdy licznik zakończy działanie.

2.6.4 Realizacja układu

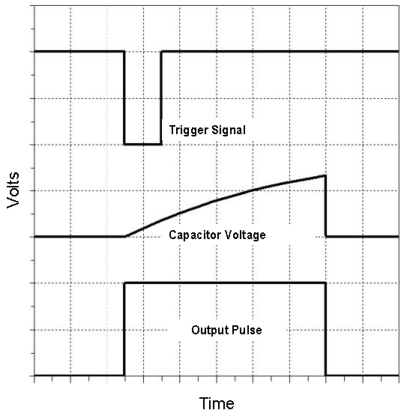
Do realizacji układu użyliśmy przerzutników D, które generują puls w momencie gdy wartość zmienia stan - wykorzystujemy je do wygenerowania pulsu startującego licznik oraz do wygenerowania sygnału wyjściowego po zakończeniu działania licznika. Jako licznik użyliśmy układu typu `555` w trybie monostabilnym (`https://en.wikipedia.org/wiki/555_timer_IC#Monostable`).



Rysunek 2.12: Schemat układu zamykającego drzwi automatycznie

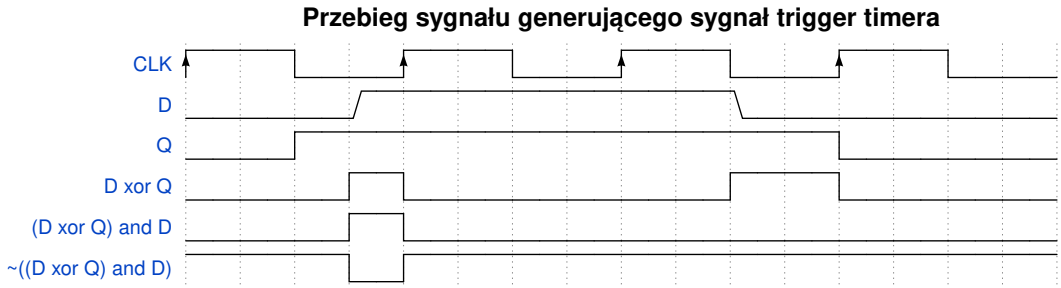
Czas odliczania licznika można regulować za pomocą pojemności kondensatora `C1` oraz rezystora `R12`. Nasze wartości dobraliśmy z tabelki zamieszczonej na Wikipedii układu, odpowiadające czasowi odliczania 100ms - co zapewnia dobrze widoczny efekt przy tempie symulacji w programie Multisim. Czas można też obliczyć ze wzoru:

$$t = ln(3) \cdot R_{12} \cdot C_1$$

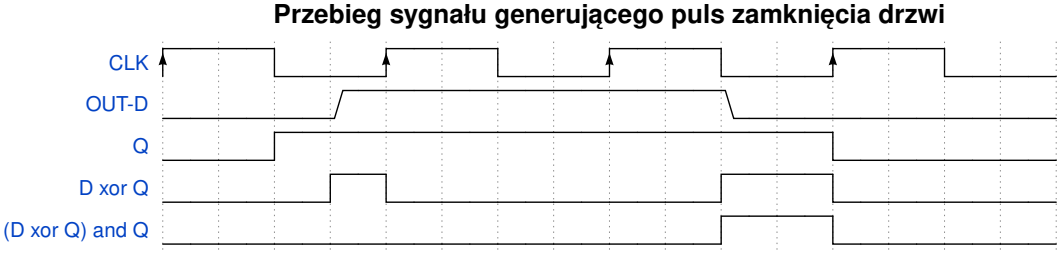


Rysunek 2.13: Wykres sygnałów w układzie monostabilnym timera 555

Za pomocą przerzutników D generujemy impuls na wejście wyzwalające licznika, a poprzez drugi przerzutnik D generujemy impuls przy opadającym zboczu wyjścia licznika.



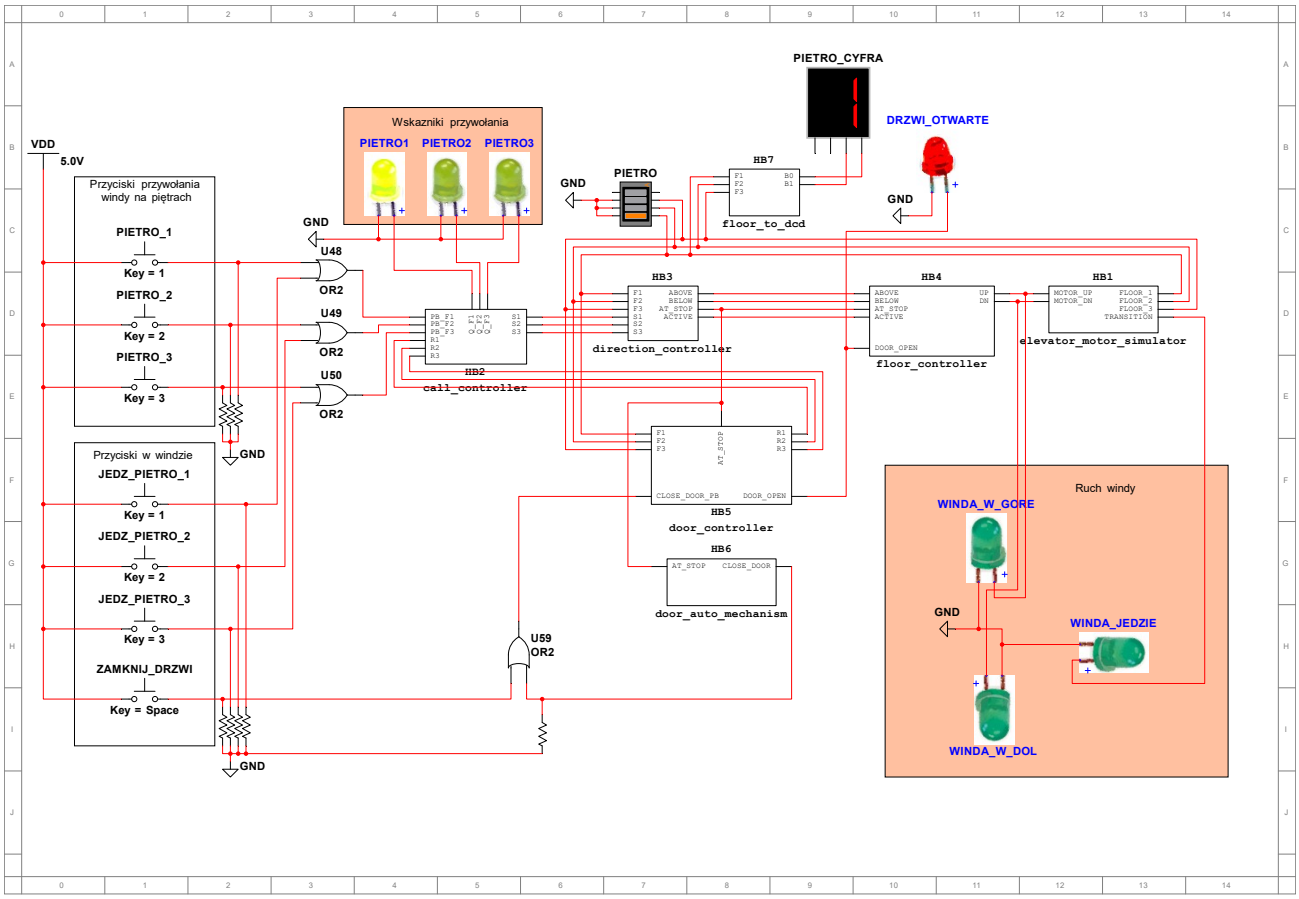
Rysunek 2.14: Przebieg sygnału ilustrujący działanie układu z przerzutnikiem D na wejściu układu



Rysunek 2.15: Przebieg sygnału ilustrujący działanie układu z przerzutnikiem D na wyjściu układu

3 Całość układu

Poniżej przedstawiamy schemat całego systemu z podpiętymi przyciskami i lampkami sygnalizującymi stan windy.



Rysunek 3.1: Schemat układu sterującego windą

Sterowanie windą odbywa się poprzez 3 przyciski na każdym piętrze przywołujące windę oraz 4 przycisków w kabinie windy: 3 przyciski odpowiadające piętrom docelowym oraz przycisk przyspieszający zamknięcie drzwi.

Stan windy sygnalizowany jest przez:

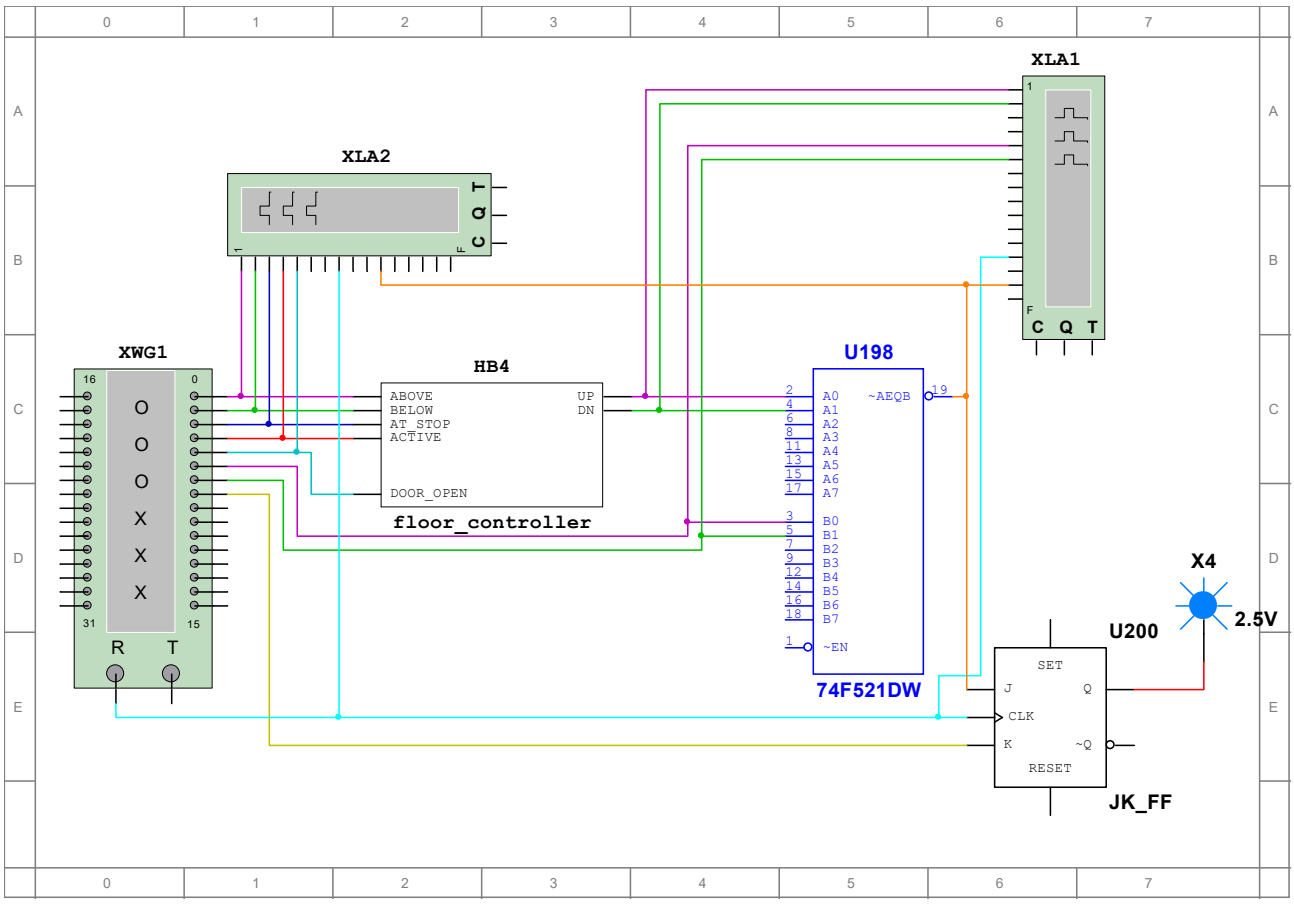
- 3 zielone ledy określające stan ruchu windy (czy jedzie, jeśli tak to w którą stronę)
- czerwonego LEDa sygnalizującego otwarcie drzwi
- 3 żółtych LED sygnalizujących na które piętro winda jest wezwana
- wyświetlacza 7 segmentowego wraz z paskiem LED pokazującym obecne piętro na którym jest winda

Winda jest zabezpieczona przed ruszaniem w przypadku otwarcia drzwi. Winda pojedzie tylko wtedy, gdy upłynie czas automatycznego zamknięcia drzwi lub samemu wciśniemy wcześniej przycisk zamykający drzwi.

4 Testy

4.1 Testy układu floor_controller

Aby przetestować układ floor_controller użyliśmy układu z generatorem słów, który nadaje dane testowe, komparatorem do wykrywania błędów, analizatorami stanów logicznych do przedstawienia wejść układu i przebiegu testu oraz przerzutnikiem JK do sygnalizowania, końcowego wyniku testu.



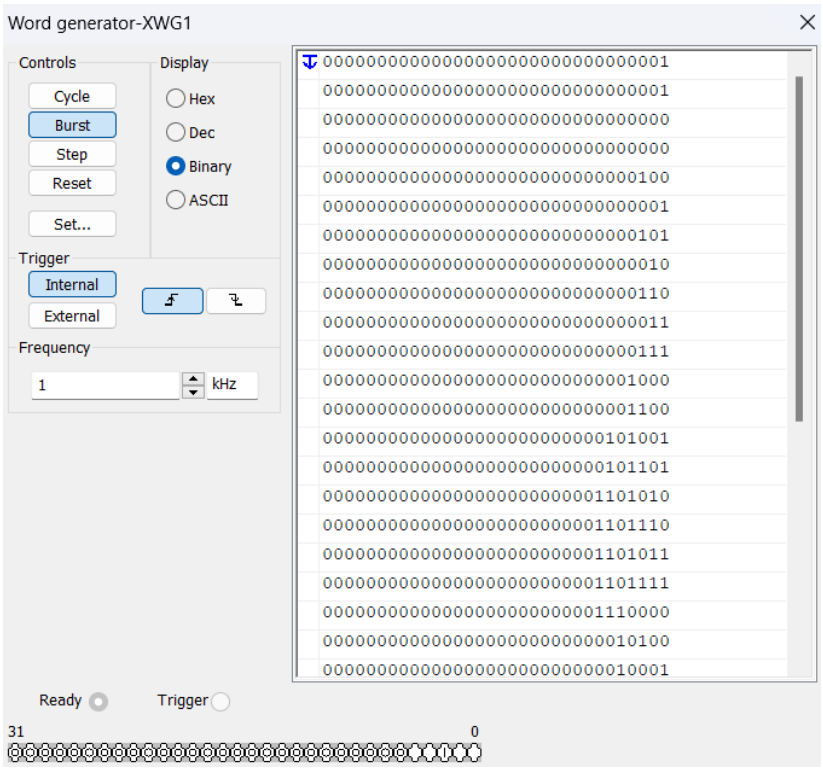
Rysunek 4.1: Schemat układu testującego

Do łatwego wygenerowania danych testowych wykorzystaliśmy zmodyfikowaną wersję wcześniej przedstawionego programu do upraszczania formuł logicznych.

```
f = open("floor_controller_test_data.dp", "w")
f.write("Data:\n")
reset_sr = 1
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
data_count = 3
up = 0
dn = 0
for i in range(32):

    permutation = bin(i).removeprefix("0b").rjust(5, '0')

    variables = {
        'DOOR_OPEN': int(permutation[0]),
        'ACTIVE': int(permutation[1]),
        'BELOW': int(permutation[2]),
        'ABOVE': int(permutation[3]),
        'AT_STOP': int(permutation[4])
    }

    set_up = '0'
    reset_up = '0'
    set_dn = '0'
    reset_dn = '0'

    if not variables['ABOVE'] and not variables['BELOW'] and variables['AT_STOP']:
        set_up = '0'
        reset_up = '1'
        set_dn = '0'
        reset_dn = '1'
        up = dn = 0

    elif variables['ACTIVE'] and variables['BELOW'] and not variables['DOOR_OPEN']:
        set_up = '0'
        reset_up = '0'
        set_dn = '1'
        reset_dn = '0'
        dn = 1

    elif variables['ACTIVE'] and variables['ABOVE'] and not variables['DOOR_OPEN']:
        set_up = '1'
        reset_up = '0'
        set_dn = '0'
        reset_dn = '0'
        up = 1

    door_open_b = str(bin(variables['DOOR_OPEN'])).removeprefix("0b")
    active_b = str(bin(variables['ACTIVE'])).removeprefix("0b")
    below_b = str(bin(variables['BELOW'])).removeprefix("0b")
    above_b = str(bin(variables['ABOVE'])).removeprefix("0b")
    at_stop_b = str(bin(variables['AT_STOP'])).removeprefix("0b")
    up_b = str(bin(up)).removeprefix("0b")
    dn_b = str(bin(dn)).removeprefix("0b")

    hex_val = hex(int(dn_b + up_b + door_open_b + active_b + at_stop_b + below_b + above_b, 2))
        .removeprefix("0x").rjust(8, '0')
    f.write(hex_val + "\n")
    data_count += 1

```

```
61 f.write("Initial:\n")
62 f.write("0000\n")
63 f.write("Final:\n")
64 f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
65
66 f.close()
```

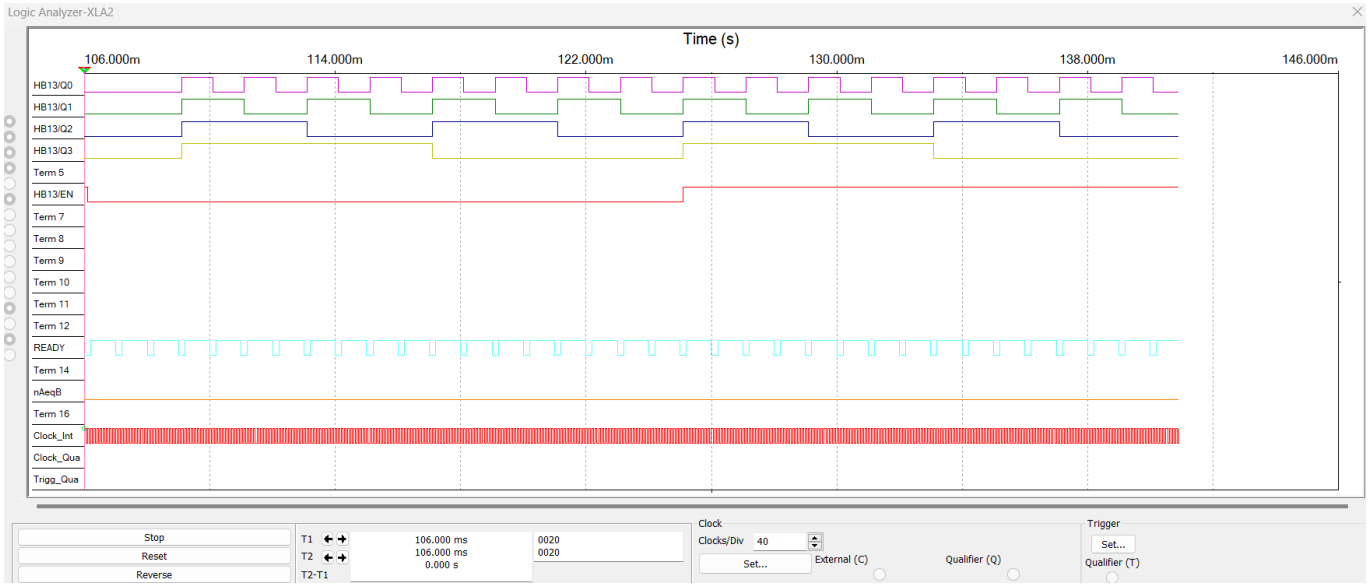
Po zaimportowaniu wygenerowanego pliku do generatora słów, wygląda on następująco:



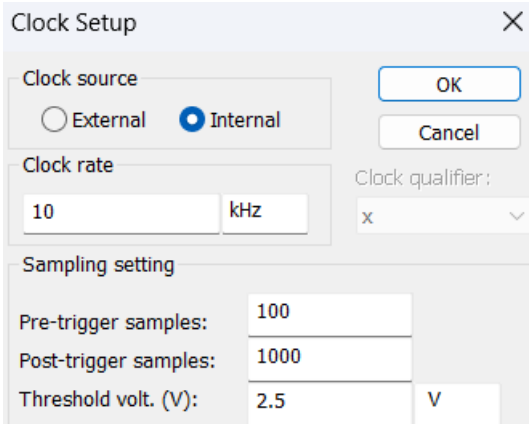
Rysunek 4.2: Dane zaimportowane do generatora słów

Poniżej wynik testu w postaci przebiegu analizatorów stanów logicznych:

Rysunek 4.3: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



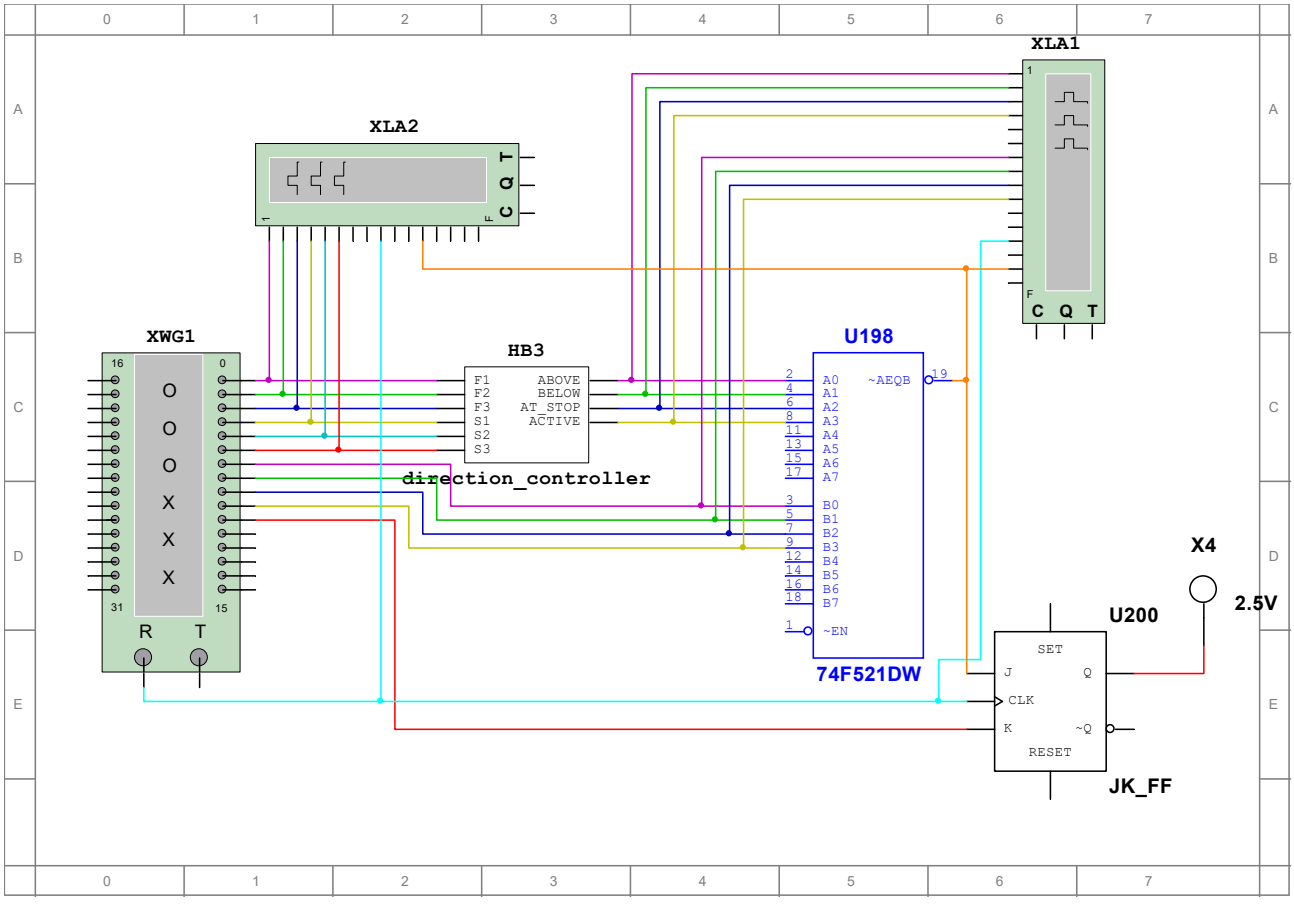
Rysunek 4.4: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wejścia układu



Rysunek 4.5: Ustawienia analizatorów

4.2 Testy układu direction_controller

Układ testujący dla układu direction_controller jest analogiczny do poprzedniego. Tutaj również wykorzystaliśmy generator słów, kompratator, dwa analizatory stanów logicznych i przerzutnik JK.



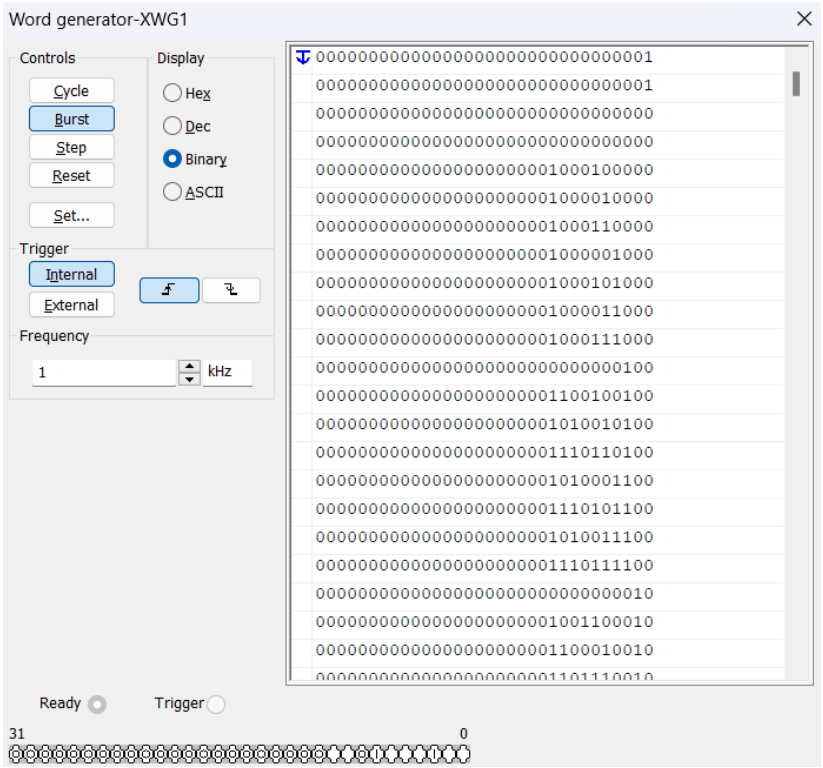
Rysunek 4.6: Schemat układu testującego

Podobnie jak wcześniej, wykorzystaliśmy zmodyfikowany kod do wyprowadzania formuł.

```
f = open("direction_controller_test_data.dp", "w")
f.write("Data:\n")
reset_sr = 1
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
data_count = 3
for i in range(64):
    permutation = bin(i).removeprefix("0b").rjust(6, '0')

    variables = {
        'F1': int(permutation[0]),
        'F2': int(permutation[1]),
        'F3': int(permutation[2]),
        'S1': int(permutation[3]),
        'S2': int(permutation[4]),
        'S3': int(permutation[5])
    }

    above = '0'
    below = '0'
    at_stop = '0'
    active = '0'

    if variables['F1'] and (variables['S2'] or variables['S3']):
        above = '1'

    elif variables['F2'] and variables['S1']:
        below = '1'

    elif variables['F2'] and variables['S3']:
        above = '1'

    elif variables['F3']and (variables['S1'] or variables['S2']):
        below = '1'

    if (variables['F1'] and variables['S1']) or (variables['F2'] and variables['S2'])
            or (variables['F3'] and variables['S3']):
        at_stop = '1'

    if variables['S1'] or variables['S2'] or variables['S3']:
        active = '1'

    F1_b = str(bin(variables['F1'])).removeprefix("0b")
    F2_b = str(bin(variables['F2'])).removeprefix("0b")
    F3_b = str(bin(variables['F3'])).removeprefix("0b")
    S1_b = str(bin(variables['S1'])).removeprefix("0b")
    S2_b = str(bin(variables['S2'])).removeprefix("0b")
    S3_b = str(bin(variables['S3'])).removeprefix("0b")
    above_b = str(bin(int(above))).removeprefix("0b")
    below_b = str(bin(int(below))).removeprefix("0b")
    at_stop_b = str(bin(int(at_stop))).removeprefix("0b")
    active_b = str(bin(int(active))).removeprefix("0b")

    hex_val = hex(int(active_b + at_stop + below_b + above_b + S3_b + S2_b + S1_b + F3_b + F2_b + F1_b, 2))
                    .removeprefix("0x").rjust(8, '0')
    f.write(hex_val + "\n")
```

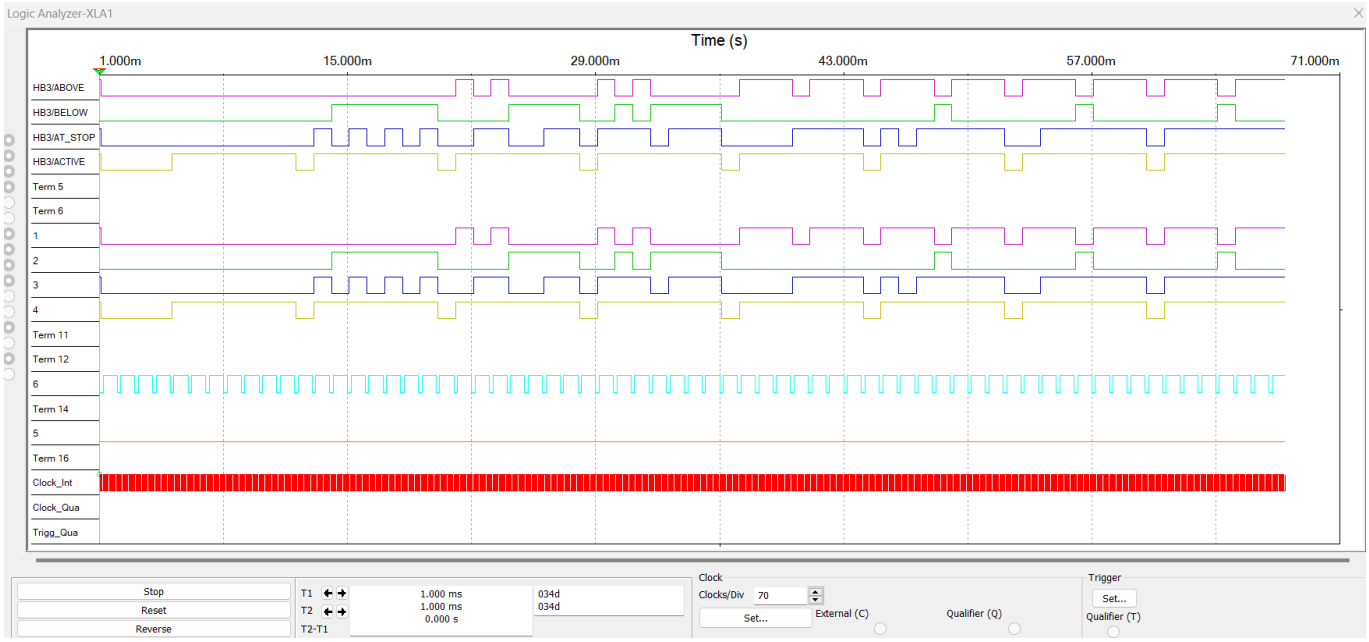
```
58      data_count += 1
59
60  f.write("Initial:\n")
61  f.write("0000\n")
62  f.write("Final:\n")
63  f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
64
65  f.close()
```

Po zaimportowaniu wygenerowanego pliku do generatora słów, wygląda on następująco:

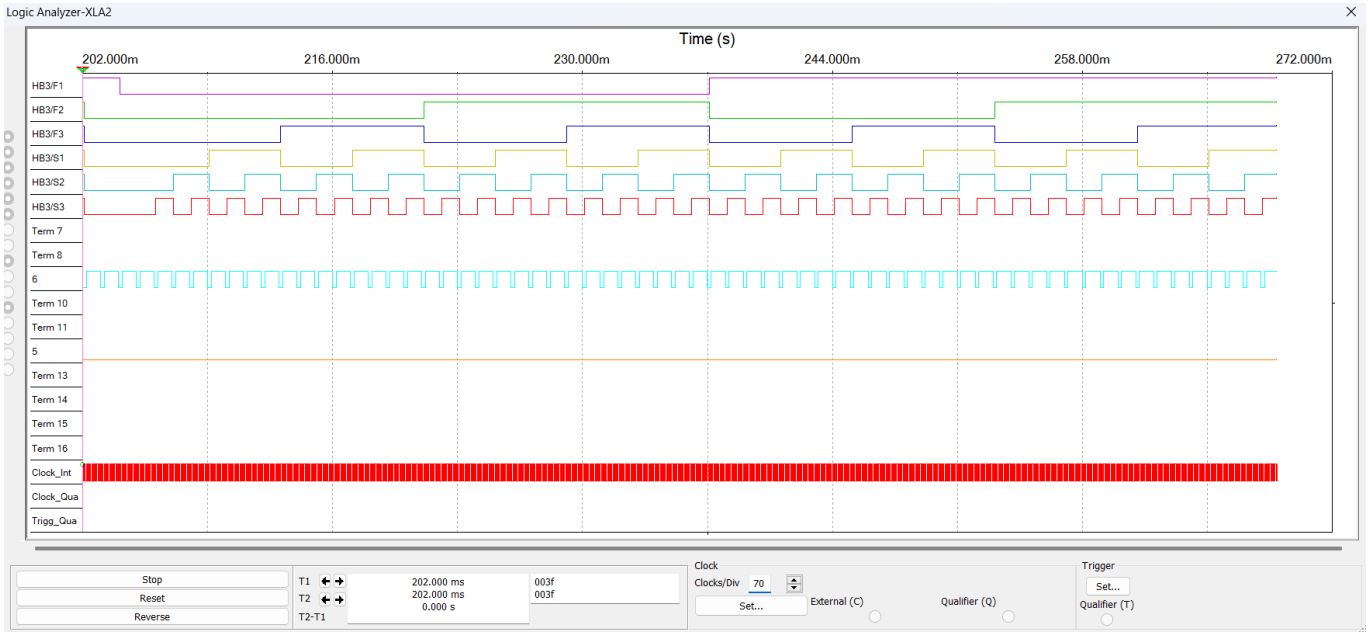


Rysunek 4.7: Dane zaimportowane do generatora słów

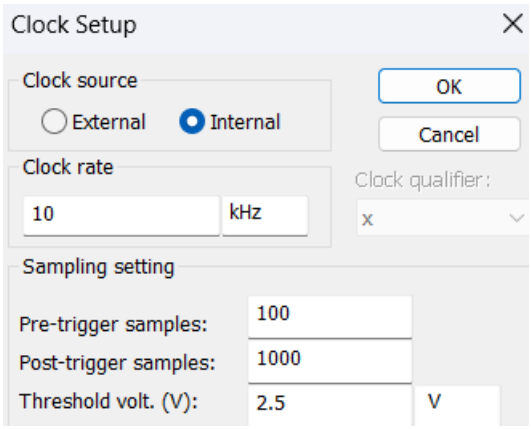
Poniżej wynik testu w postaci przebiegu analizatorów stanów logicznych:



Rysunek 4.8: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



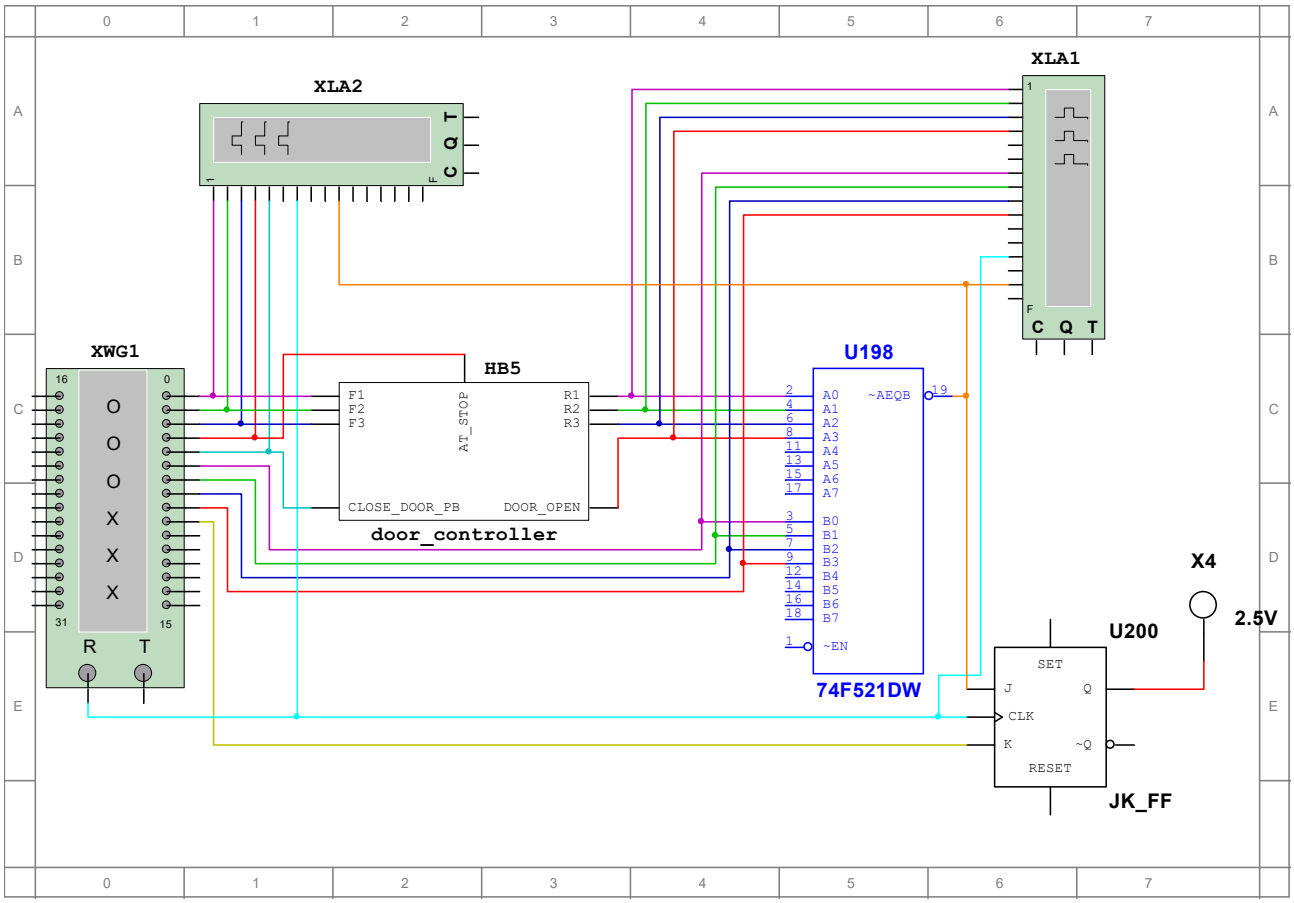
Rysunek 4.9: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wejścia układu



Rysunek 4.10: Ustawienia analizatorów

4.3 Testy układu door_controller

Układ testujący układu door_controller jest również całkowicie analogiczny do dwóch poprzednich.



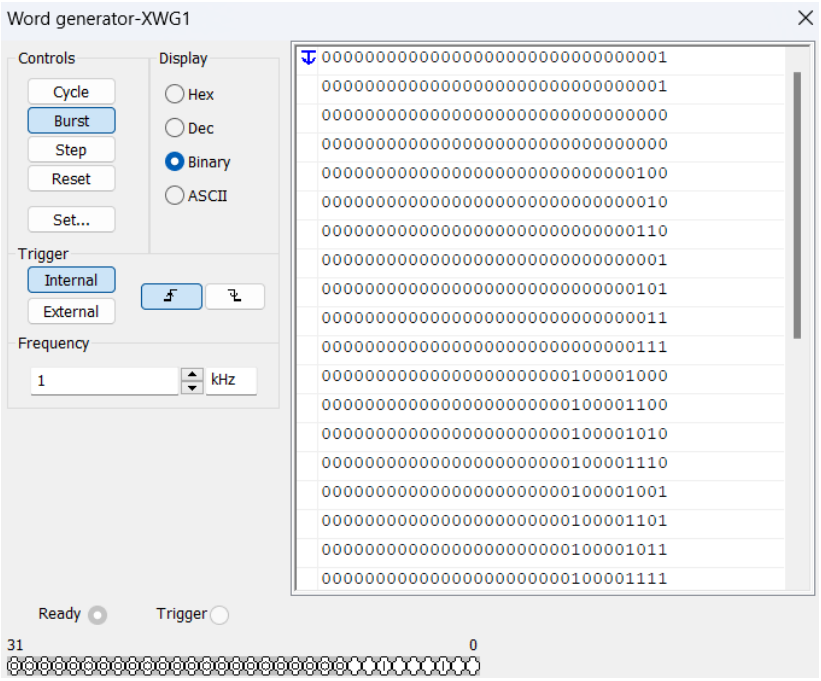
Rysunek 4.11: Schemat układu testującego

Podobnie jak wcześniej, wykorzystaliśmy zmodyfikowany kod do wyprowadzania

formuł.

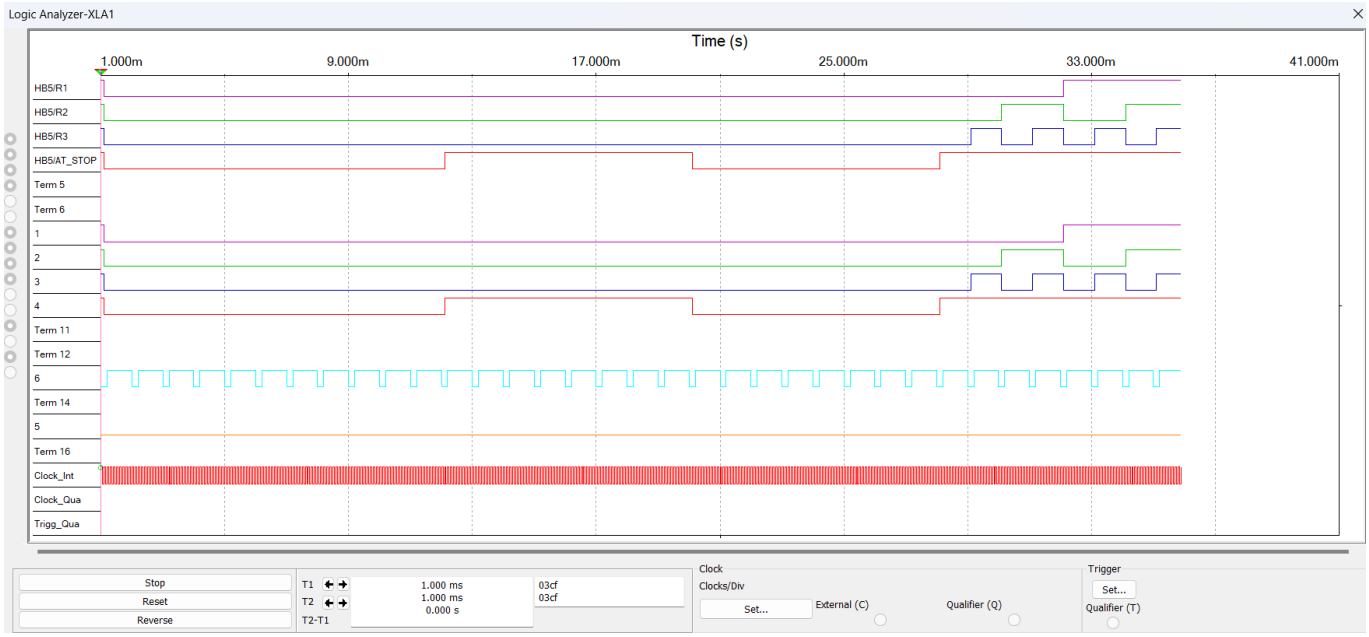
```
f = open("door_controller_test_data.dp", "w")
f.write("Data:\n")
reset_sr = 1
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
data_count = 3
for i in range(32):
    permutation = bin(i).removeprefix("0b").rjust(5, '0')

    variables = {
        'CLOSE_DOOR_PB': int(permutation[0]),
        'AT_STOP': int(permutation[1]),
        'F1': int(permutation[2]),
        'F2': int(permutation[3]),
        'F3': int(permutation[4])
    }

    r1 = '0'
    r2 = '0'
    r3 = '0'
    door_open = '0'

    if variables['AT_STOP']: door_open = '1'
    if variables['AT_STOP'] and variables['CLOSE_DOOR_PB']:
        if variables['F1']: r1 = '1'
        if variables['F2']: r2 = '1'
        if variables['F3']: r3 = '1'

    close_door_pb_b = str(bin(variables['CLOSE_DOOR_PB'])).removeprefix("0b")
    at_stop_b = str(bin(variables['AT_STOP'])).removeprefix("0b")
    F1_b = str(bin(variables['F1'])).removeprefix("0b")
    F2_b = str(bin(variables['F2'])).removeprefix("0b")
    F3_b = str(bin(variables['F3'])).removeprefix("0b")
    r1_b = str(bin(int(r1))).removeprefix("0b")
    r2_b = str(bin(int(r2))).removeprefix("0b")
    r3_b = str(bin(int(r3))).removeprefix("0b")
    door_open_b = str(bin(int(door_open))).removeprefix("0b")

    hex_val = hex(int(door_open_b + r3_b + r2_b + r1_b + close_door_pb_b + at_stop_b + F3_b + F2_b + F1_b, 2))
                    .removeprefix("0x").rjust(8, '0')
    f.write(hex_val + "\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()
```

Po zaimportowaniu wygenerowanego pliku do generatora słów, wygląda on następująco:

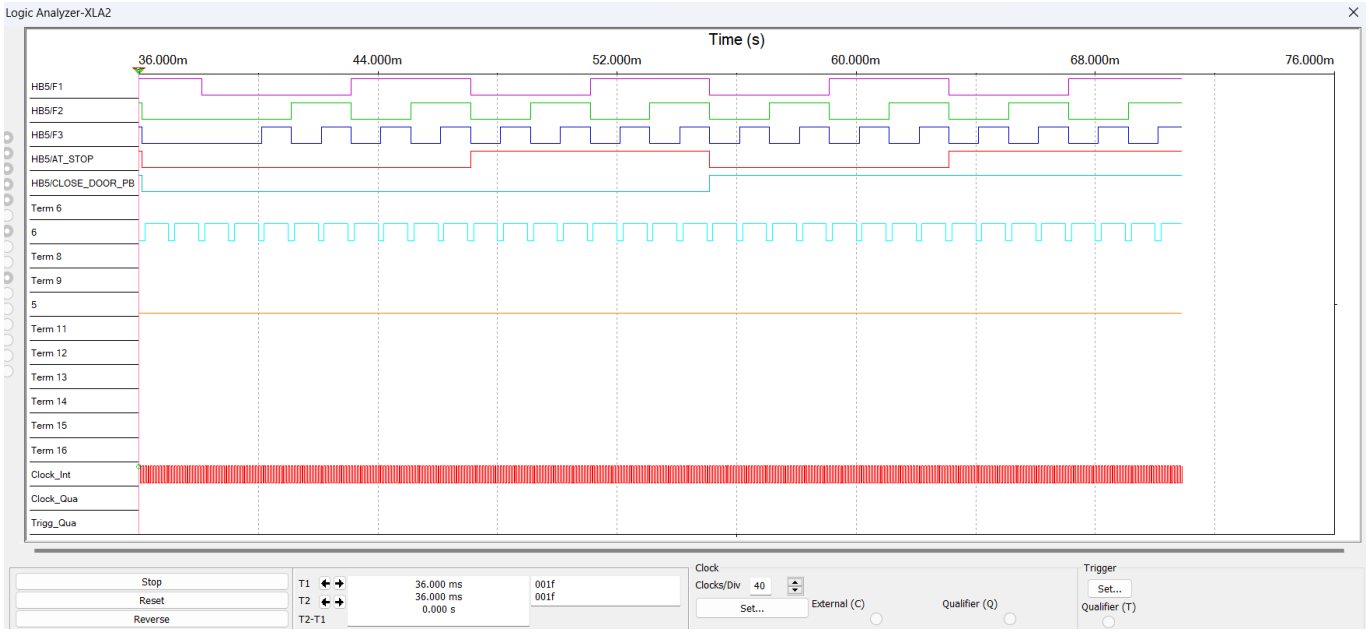


Rysunek 4.12: Dane zaimportowane do generatora słów

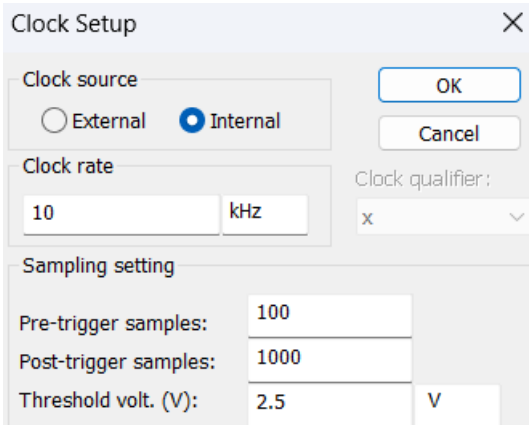
Poniżej wynik testu w postaci przebiegu analizatorów stanów logicznych:



Rysunek 4.13: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



Rysunek 4.14: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wejścia układu



Rysunek 4.15: Ustawienia analizatorów

5 Zastosowania

Winda?

6 Wnioski

Inne sposoby realizacji które rozważaliśmy: - stworzenie automatu ze wszystkimi potencjalnymi stanami - nasze pomysły były bardzo skomplikowane i trudne do realizacji poprzez ręczne układanie bramek w multisimie, w przypadku układu FPGA przy użyciu języka typu VHDL reprezentacja stanów automatu byłaby dużo prostsza

- rozbicie systemu na mniejsze bloczki stosowane na każdych piętrach które mogłyby być spinane w dowolnej ilości, nie udało nam się tego zrealizować, a dla 3 pięter nasz układ nie ma dużej złożoności