

Laboratorium 3

Sterownik windy

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1	Cel zadania	3
2	Rozwiązania	3
2.1	Symulator silnika windy	3
2.1.1	Black box	3
2.1.2	Wejścia	4
2.1.3	Wyjścia	4
2.1.4	Realizacja układu	4
2.2	Kontroler ruchu windy	6
2.2.1	Black box	7
2.2.2	Wejścia	7
2.2.3	Wyjścia	7
2.2.4	Realizacja układu	7
2.3	Kontroler kierunku ruchu	9
2.3.1	Black box	10
2.3.2	Wejścia	10
2.3.3	Wyjścia	10
2.3.4	Realizacja układu	11
2.4	Kontroler drzwi windy	12
2.4.1	Black box	13
2.4.2	Wejścia	13
2.4.3	Wyjścia	13
2.4.4	Realizacja układu	13
2.5	Układ zamykający drzwi	15
2.5.1	Black box	15
2.5.2	Wejścia	15
2.5.3	Wyjścia	15
2.5.4	Realizacja układu	16
3	Całość układu	18
4	Testy	19
4.1	Testy układu floor_controller	19
4.2	Testy układu direction_controller	22
4.3	Testy układu door_controller	26
5	Zastosowania	29
6	Wnioski	29

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 Rozwiązania

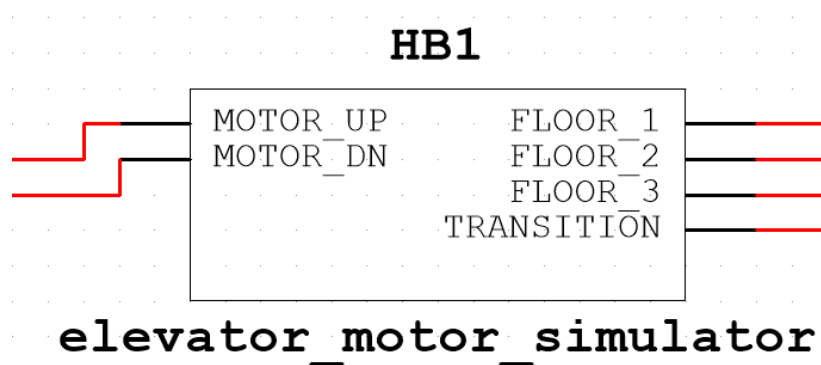
Postanowiliśmy rozbić problem na wiele mniejszych problemów i rozwiązać je osobno aby na końcu połączyć je w jeden działający system. Poniżej zamieszczamy schemat blokowy przedstawiający poszczególne systemy.

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. Przyjeliśmy, 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:

```

1  import logicmin
2  elevator_motor_mux_tt = logicmin.TT(4, 2)
3
4  for i in range(16):
5      permutation = bin(i).removeprefix("0b").rjust(4, '0')
6
7      a1 = '1'
8      b1 = '1'
9
10     if i == 0:
11         a1 = '0'
12         b1 = '0'
13     elif i == 8:
14         a1 = '1'
15         b1 = '0'
16     elif i == 15:
17         a1 = '0'
18         b1 = '1'
19
20     elevator_motor_mux_tt.add(permutation, [a1, b1])
21
22 print("-----elevator_motor_tt")
23 sols = elevator_motor_mux_tt.solve()
24 print(sols.printN(xnames=['QD', 'QC', 'QB', 'QA'], ynames=['1A', '1B']))

```

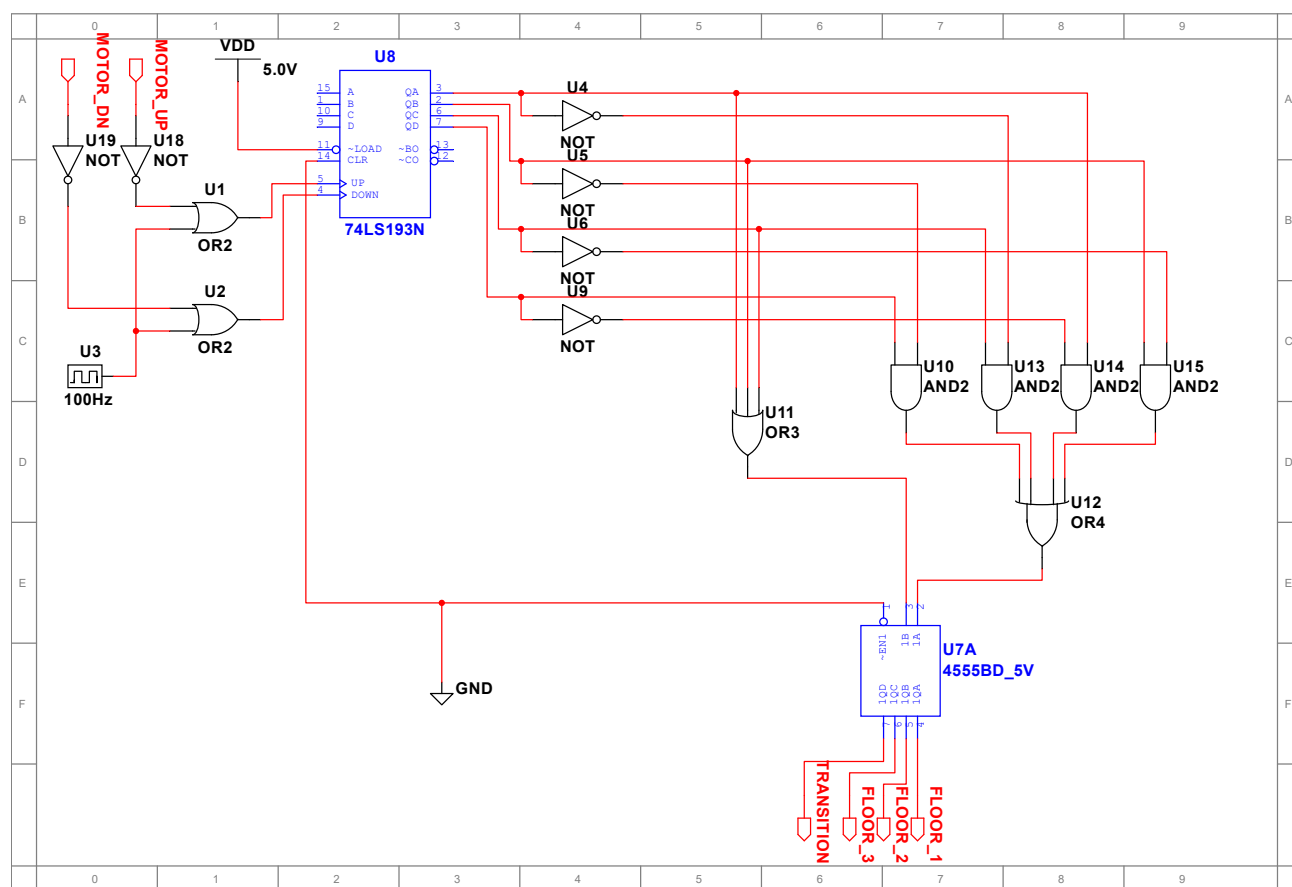
Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```

-----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:



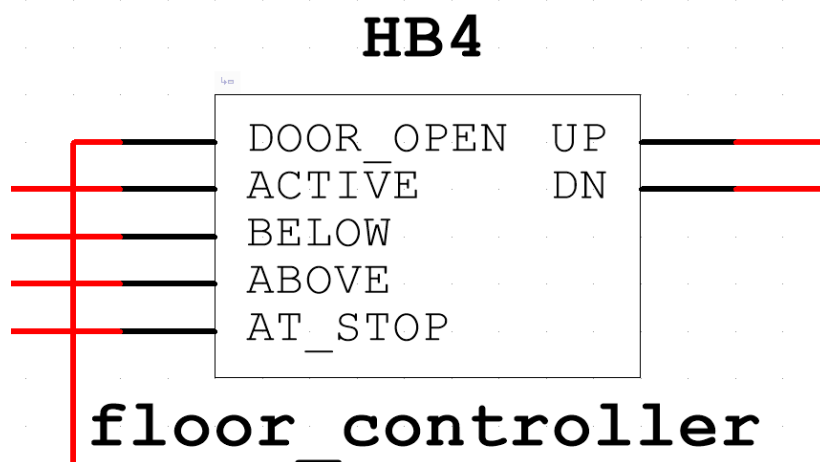
Rysunek 2.2: Schemat układu symulującego silnik windy

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 obecnej pozycji windy
- ABOVE - sygnał wejściowy informujący, że piętro docelowe znajduje się powyżej obecnej 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 gotowości do ruchu w górę
- DN - sygnał wyjściowy informujący o gotowości 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.

```

1 import logicmin
2 floor_controller_tt = logicmin.TT(5, 4)
3
4 for i in range(32):
5     permutation = bin(i).removeprefix("0b").rjust(5, '0')
6
7     variables = {
8         'DOOR_OPEN': int(permutation[0]),
9         'ACTIVE': int(permutation[1]),
10        'BELOW': int(permutation[2]),
11        'ABOVE': int(permutation[3]),
12        'AT_STOP': int(permutation[4])
13    }
14
15    set_up = '0'
16    reset_up = '0'
17    set_dn = '0'
18    reset_dn = '0'
19
20    if not variables['ABOVE'] and not variables['BELOW'] and variables['AT_STOP']:
21        set_up = '0'
22        reset_up = '1'
23        set_dn = '0'
24        reset_dn = '1'
25
26    elif variables['ACTIVE'] and variables['BELOW'] and not variables['DOOR_OPEN']:
27        set_up = '0'
28        reset_up = '0'
29        set_dn = '1'
30        reset_dn = '0'
31
32    elif variables['ACTIVE'] and variables['ABOVE'] and not variables['DOOR_OPEN']:
33        set_up = '1'
34        reset_up = '0'
35        set_dn = '0'
36        reset_dn = '0'
37
38    floor_controller_tt.add(permutation, [set_up, reset_up, set_dn, reset_dn])
39
40 print("-----floor_controller_tt")
41 sols = floor_controller_tt.solve()
42 print(sols.printN(xnames=['DOOR_OPEN', 'ACTIVE', 'BELOW', 'ABOVE', 'AT_STOP'],
43                     ynames=['SET_UP', 'RESET_UP', 'SET_DN', 'RESET_DN']))

```

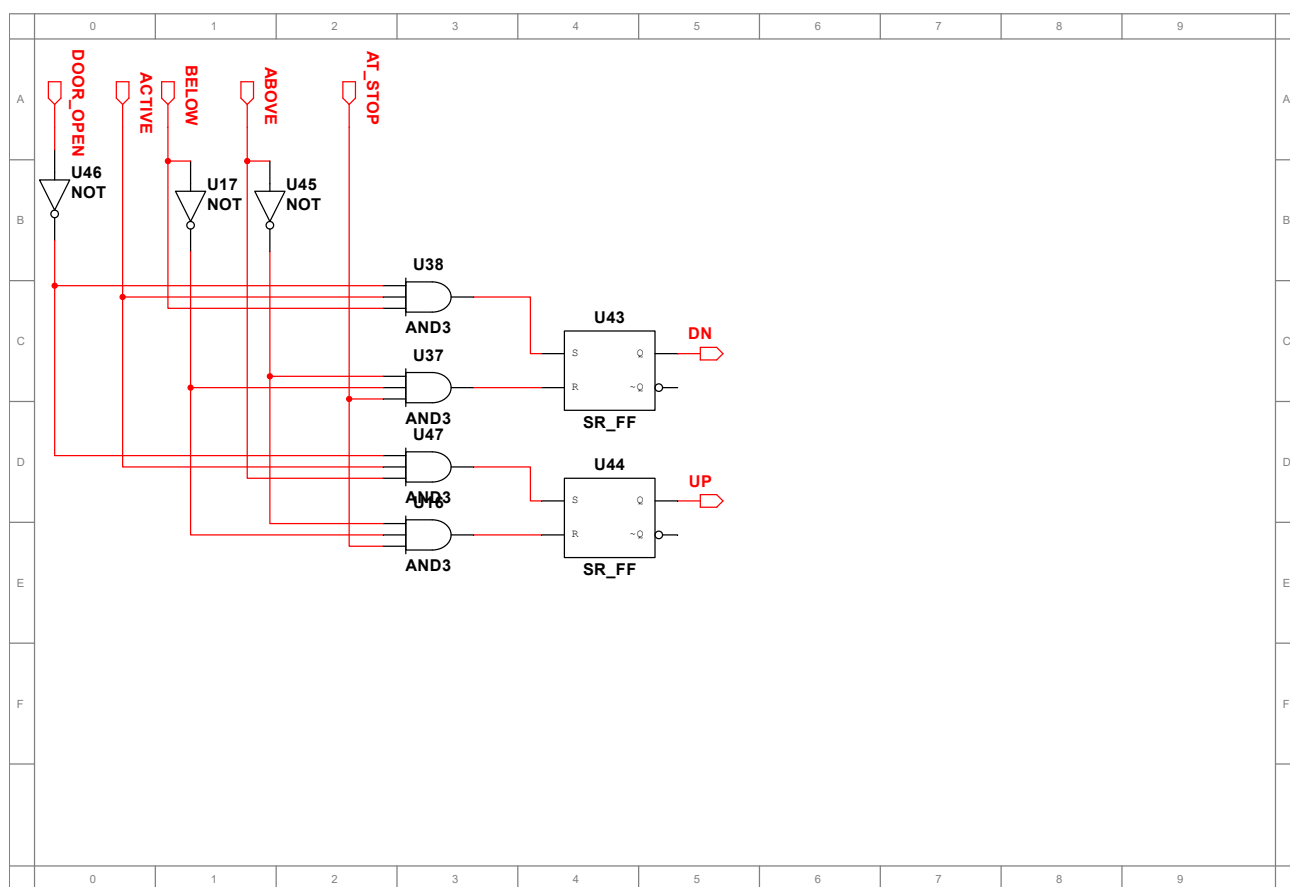
Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```

-----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.

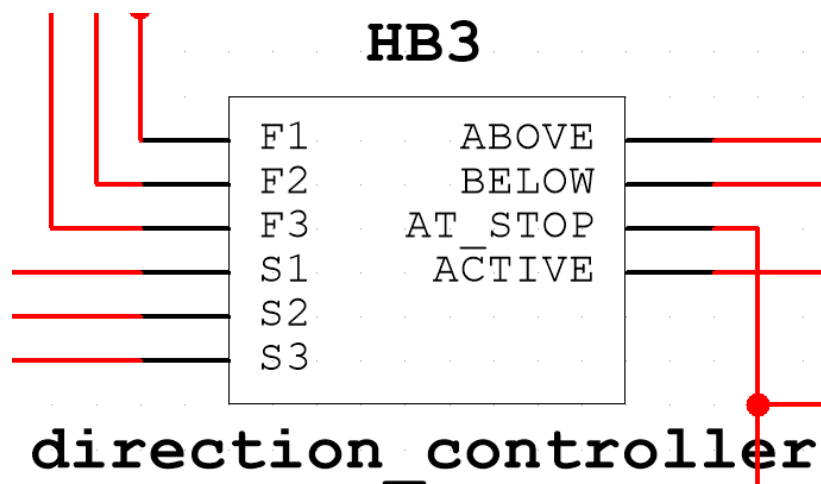


Rysunek 2.4: Schemat układu inicjującego ruch windy

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 obecnej pozycji windy
- BELOW - sygnał wyjściowy informujący, że piętro docelowe znajduje się poniżej obecnej pozycji windy
- AT_STOP - sygnał wyjściowy informujący, że winda 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.

```

1 import logicmin
2 direction_controller_tt = logicmin.TT(6, 4)
3
4 for i in range(64):
5     permutation = bin(i).removeprefix("0b").rjust(6, '0')
6
7     variables = {
8         'F1': int(permutation[0]),
9         'F2': int(permutation[1]),
10        'F3': int(permutation[2]),
11        'S1': int(permutation[3]),
12        'S2': int(permutation[4]),
13        'S3': int(permutation[5])
14    }
15
16    above = '0'
17    below = '0'
18    at_stop = '0'
19    active = '0'
20
21    if variables['F1'] and (variables['S2'] or variables['S3']):
22        above = '1'
23
24    elif variables['F2'] and variables['S1']:
25        below = '1'
26
27    elif variables['F2'] and variables['S3']:
28        above = '1'
29
30    elif variables['F3'] and (variables['S1'] or variables['S2']):
31        below = '1'
32
33    if (variables['F1'] and variables['S1']) or (variables['F2'] and variables['S2'])
34        or (variables['F3'] and variables['S3']):
35        at_stop = '1'
36
37    if variables['S1'] or variables['S2'] or variables['S3']:
38        active = '1'
39
40    direction_controller_tt.add(permutation, [above, below, at_stop, active])
41
42 print("-----direction_controller_tt")
43 sols = direction_controller_tt.solve()
44 print(sols.printN(xnames=['F1', 'F2', 'F3', 'S1', 'S2', 'S3'], ynames=['ABOVE', 'BELOW', 'AT_STOP', 'ACTIVE']))

```

Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```

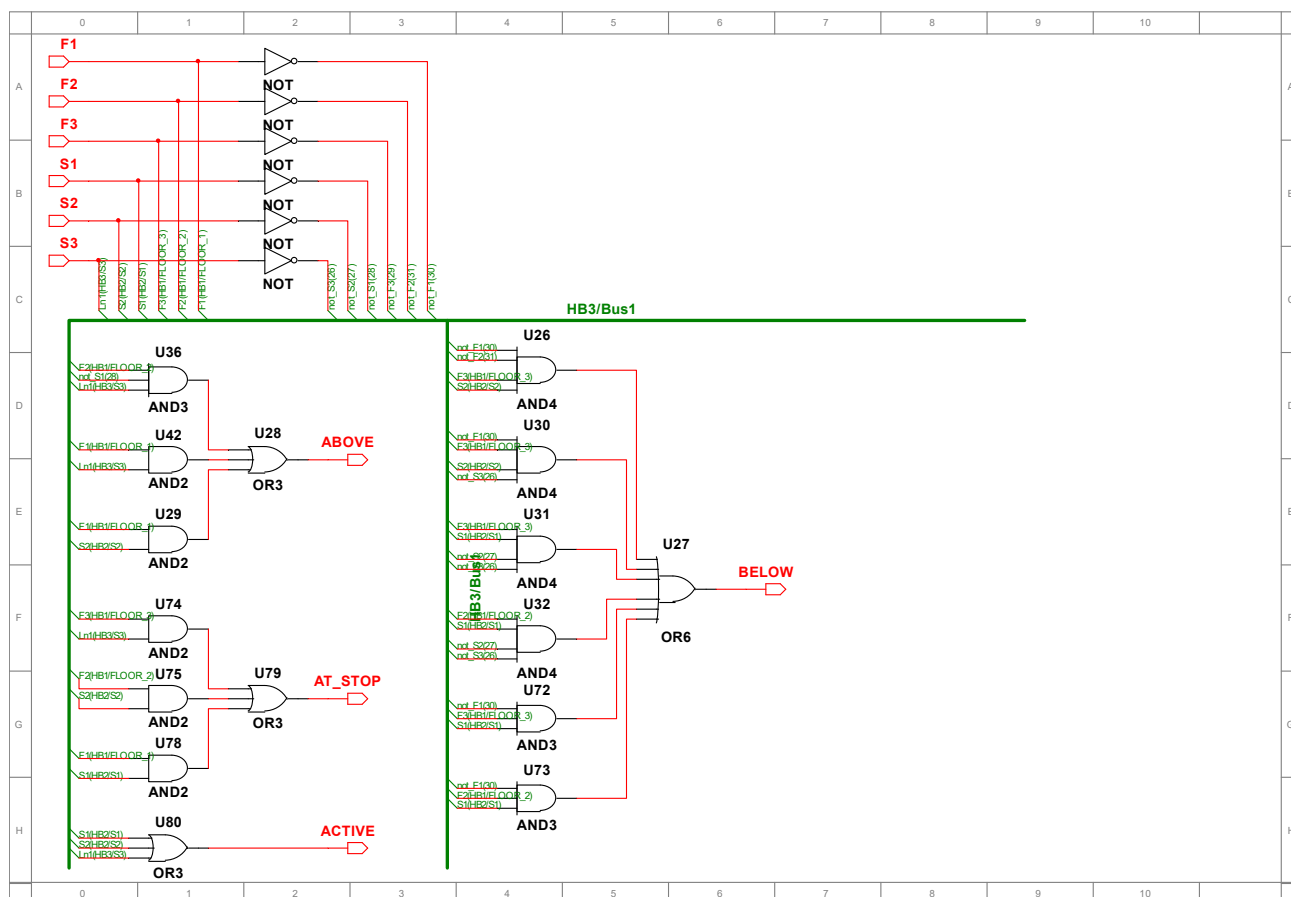
-----direction_controller_tt
ACTIVE <= S3 + S2 + S1
AT_STOP <= F3.S3 + F2.S2 + F1.S1
BELOW <= F1'.F2'.F3.S2 + F1'.F3.S2.S3' + F3.S1.S2'.S3' + F2.S1.S2'.S3' +

```

$$+ F1'.F3.S1 + F1'.F2.S1$$

$$ABOVE \leq F2.S1'.S3 + F1.S3 + F1.S2$$

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

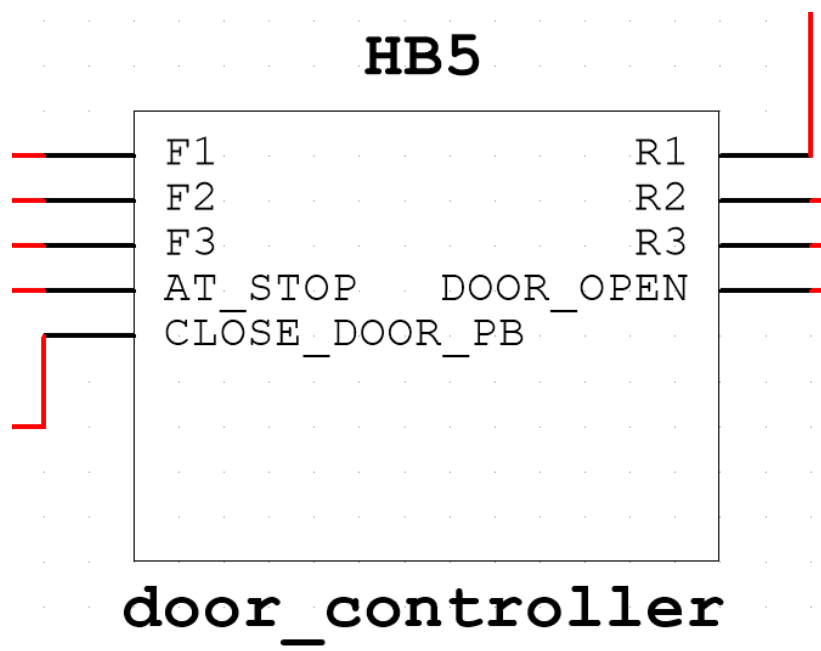


Rysunek 2.6: Schemat układu przetwarzającego wezwanie

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 winda 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.

```

1  import logicmin
2  door_controller_tt = logicmin.TT(5, 4)
3
4  for i in range(32):
5      permutation = bin(i).removeprefix("0b").rjust(5, '0')
6
7      variables = {
8          'CLOSE_DOOR_PB': int(permutation[0]),
9          'AT_STOP': int(permutation[1]),
10         'F1': int(permutation[2]),
11         'F2': int(permutation[3]),
12         'F3': int(permutation[4])
13     }
14
15     r1 = '0'
16     r2 = '0'
17     r3 = '0'
18     door_open = '0'
19
20     if variables['AT_STOP']: door_open = '1'
21     if variables['AT_STOP'] and variables['CLOSE_DOOR_PB']:
22         if variables['F1']: r1 = '1'
23         if variables['F2']: r2 = '1'
24         if variables['F3']: r3 = '1'
25
26     door_controller_tt.add(permutation, [r1, r2, r3, door_open])
27
28 print("-----door_controller_tt")
29 sols = door_controller_tt.solve()
30 print(sols.printN(xnames=['CLOSE_DOOR_PB', 'AT_STOP', 'F1', 'F2', 'F3'], ynames=['R1', 'R2', 'R3', 'DOOR_OPEN']))

```

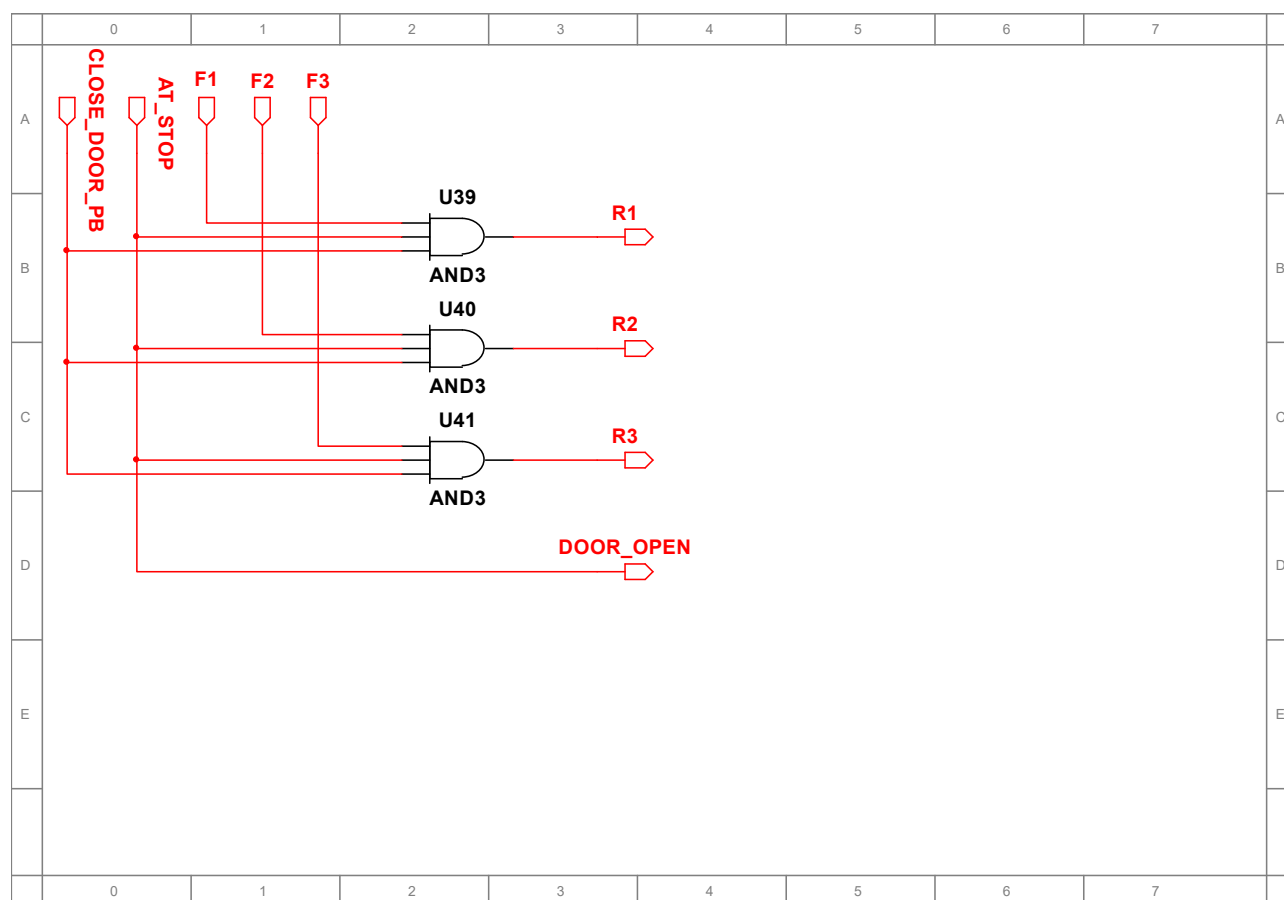
Wynikiem działania skryptu są zminimalizowane formuły logiczne:

```

-----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 Układ zamykający drzwi

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

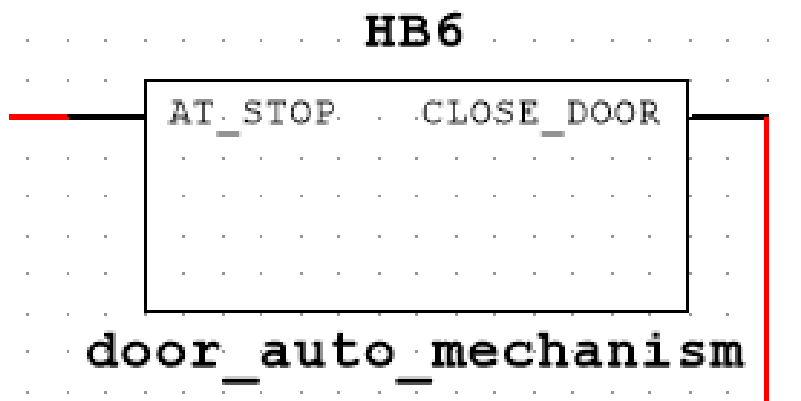
2.5.1 Black box

2.5.2 Wejścia

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

2.5.3 Wyjścia

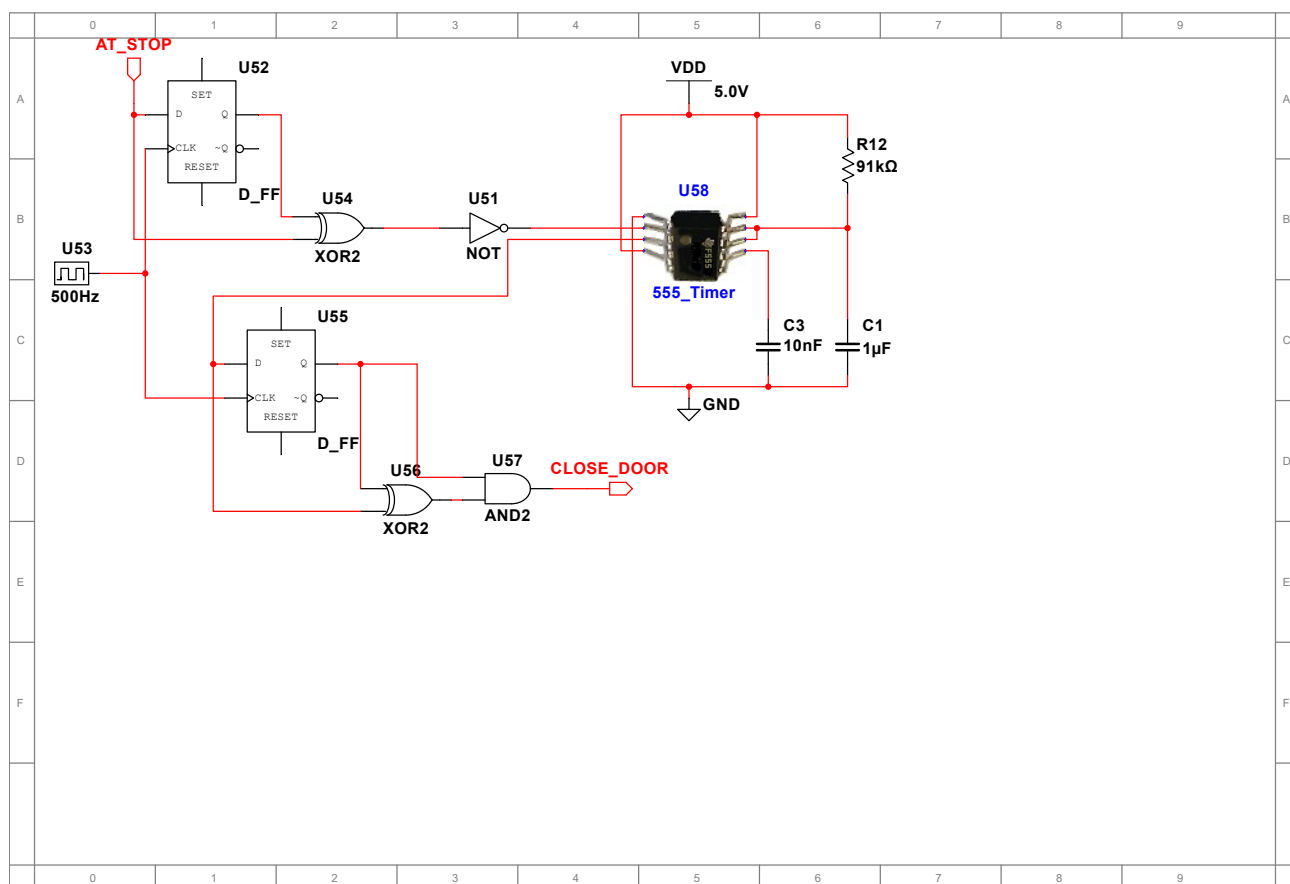
- **CLOSE_DOOR** - sygnał wyjściowy będący sygnałem dla zamknięcia drzwi, jest to krótki puls gdy licznik zakończy działanie.



Rysunek 2.9: Black box układu zamykającego drzwi automatycznie

2.5.4 Realizacja układu

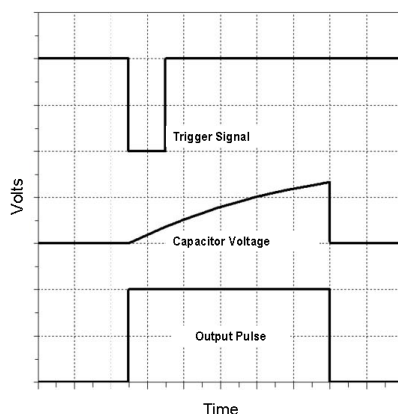
Do realizacji układu użyliśmy przerzutników D, które generują puls w momencie gdy wartość zmienia stan - wykorzystujemy je do wygenerowania pulsu startującego licznik oraz do wygenerowania sygnału wyjściowego po zakończeniu działania licznika. Jako licznik użyliśmy układu typu 555 w trybie monostabilnym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



Rysunek 2.10: Schemat układu zamykającego drzwi automatycznie

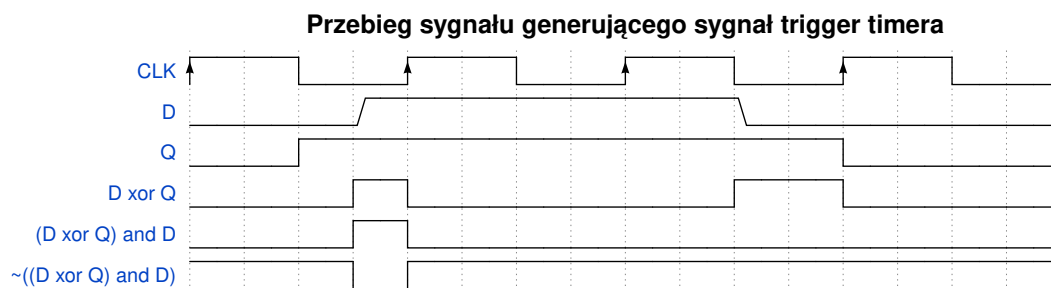
Czas odliczania licznika można regulować za pomocą pojemności kondensatora C_1 oraz rezystora R_{12} . Nasze wartości dobraliśmy z tabelki zamieszczonej na Wikipedii układu, odpowiadające czasowi odliczania 100ms - co zapewnia dobrze widoczny efekt przy tempie symulacji w programie Multisim. Czas można też obliczyć ze wzoru:

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

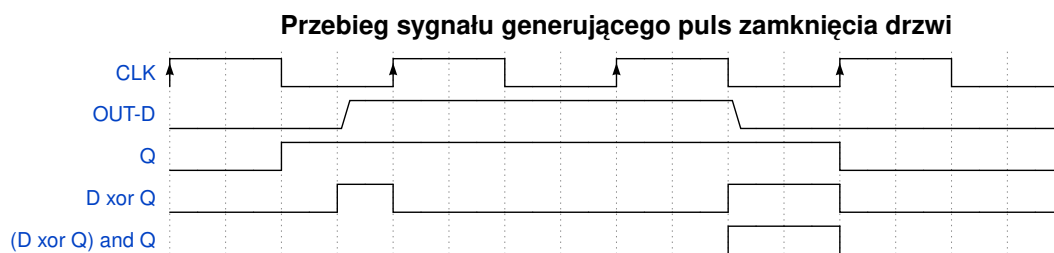


Rysunek 2.11: Wykres sygnałów w układzie monostabilnym timera 555

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



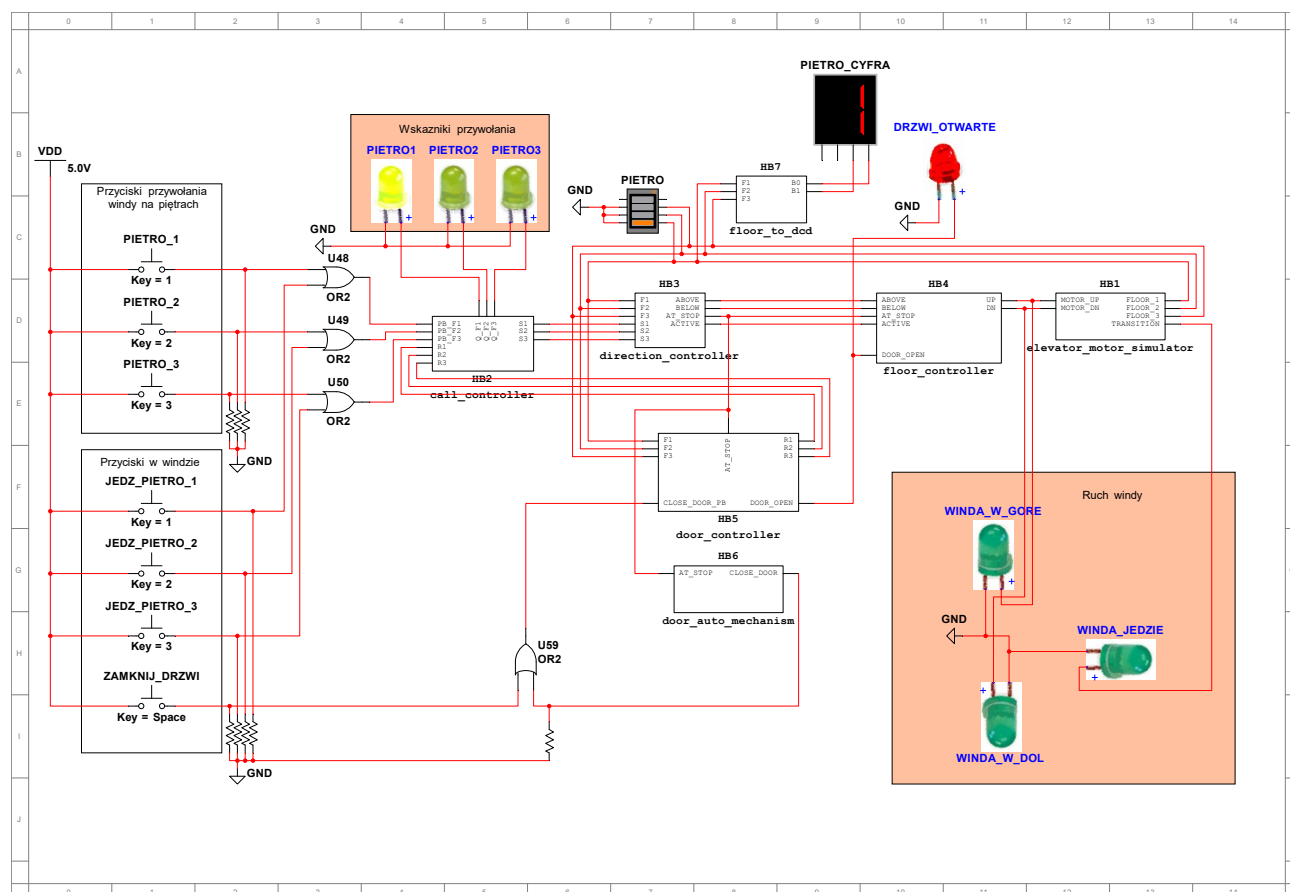
Rysunek 2.12: Przebieg sygnału ilustrujący działanie układu z przerzutnikiem D na wejściu układu



Rysunek 2.13: 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:

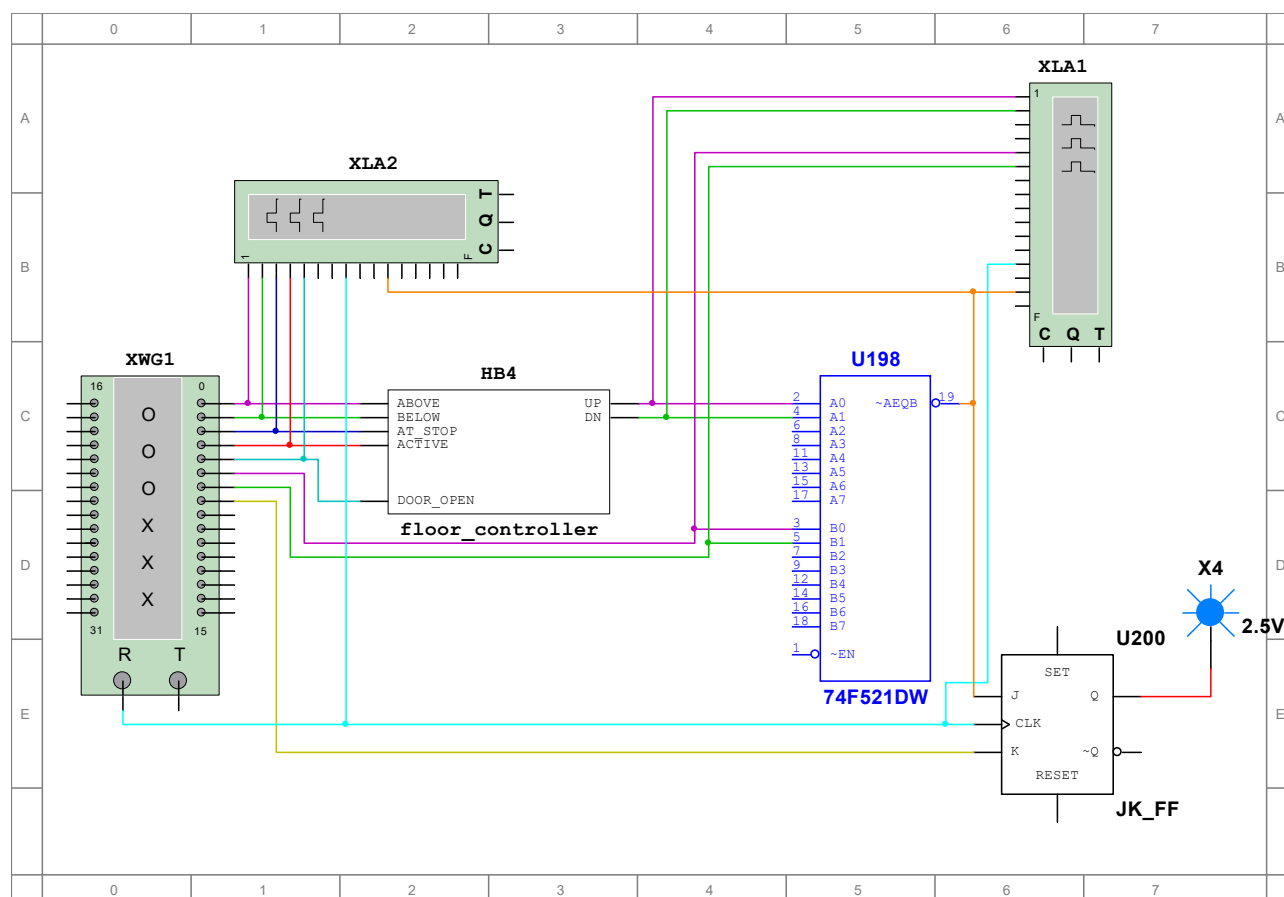
- 3 zielone ledy określające stan ruchu windy (czy jedzie, jeśli tak to w którą stronę)
- czerwonego LEDa sygnalizującego otwarcie drzwi
- 3 żółtych LED sygnalizujących na które piętro winda jest wezwana
- wyświetlacza 7 segmentowego wraz z paskiem LED pokazującym obecne piętro na którym jest winda

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.

```

1  f = open("floor_controller_test_data.dp", "w")
2  f.write("Data:\n")
3  reset_sr = 1
4  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
5  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
6  f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
7  data_count = 3
8  up = 0
9  dn = 0
10 for i in range(32):
11
12     permutation = bin(i).removeprefix("0b").rjust(5, '0')
13
14     variables = {
15         'DOOR_OPEN': int(permutation[0]),
16         'ACTIVE': int(permutation[1]),
17         'BELOW': int(permutation[2]),
18         'ABOVE': int(permutation[3]),
19         'AT_STOP': int(permutation[4])
20     }
21
22     set_up = '0'
23     reset_up = '0'
24     set_dn = '0'
25     reset_dn = '0'
26
27     if not variables['ABOVE'] and not variables['BELOW'] and variables['AT_STOP']:
28         set_up = '0'
29         reset_up = '1'
30         set_dn = '0'
31         reset_dn = '1'
32         up = dn = 0
33
34     elif variables['ACTIVE'] and variables['BELOW'] and not variables['DOOR_OPEN']:
35         set_up = '0'
36         reset_up = '0'
37         set_dn = '1'
38         reset_dn = '0'
39         dn = 1
40
41     elif variables['ACTIVE'] and variables['ABOVE'] and not variables['DOOR_OPEN']:
42         set_up = '1'
43         reset_up = '0'
44         set_dn = '0'
45         reset_dn = '0'
46         up = 1
47
48     door_open_b = str(bin(variables['DOOR_OPEN'])).removeprefix("0b")
49     active_b = str(bin(variables['ACTIVE'])).removeprefix("0b")
50     below_b = str(bin(variables['BELOW'])).removeprefix("0b")
51     above_b = str(bin(variables['ABOVE'])).removeprefix("0b")
52     at_stop_b = str(bin(variables['AT_STOP'])).removeprefix("0b")
53     up_b = str(bin(up)).removeprefix("0b")
54     dn_b = str(bin(dn)).removeprefix("0b")
55
56     hex_val = hex(int(dn_b + up_b + door_open_b + active_b + at_stop_b + below_b + above_b, 2))
57         .removeprefix("0x").rjust(8, '0')
58     f.write(hex_val + "\n")
59     data_count += 1
60

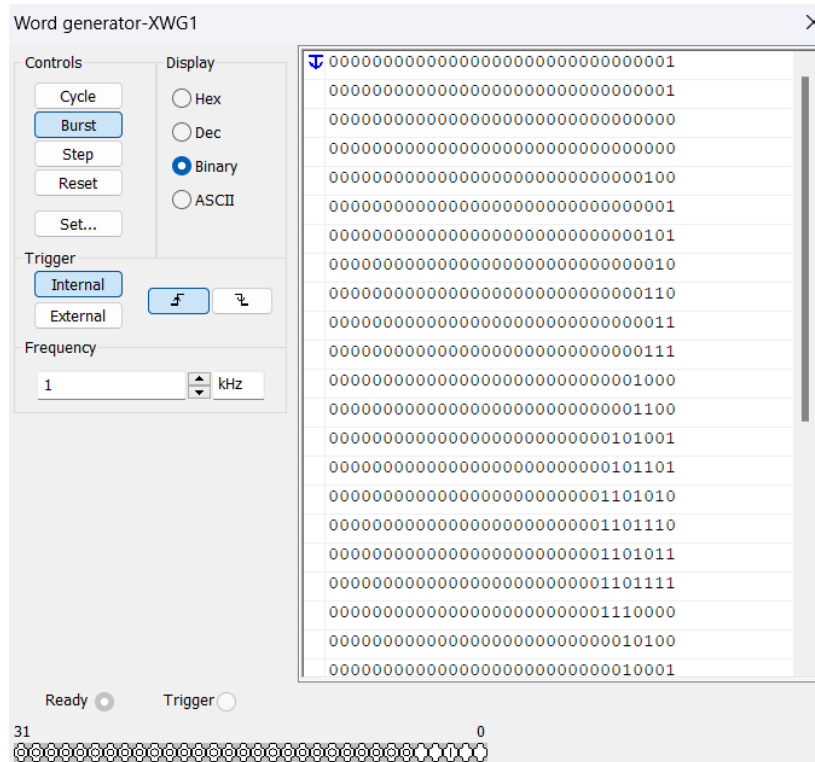
```

```

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()

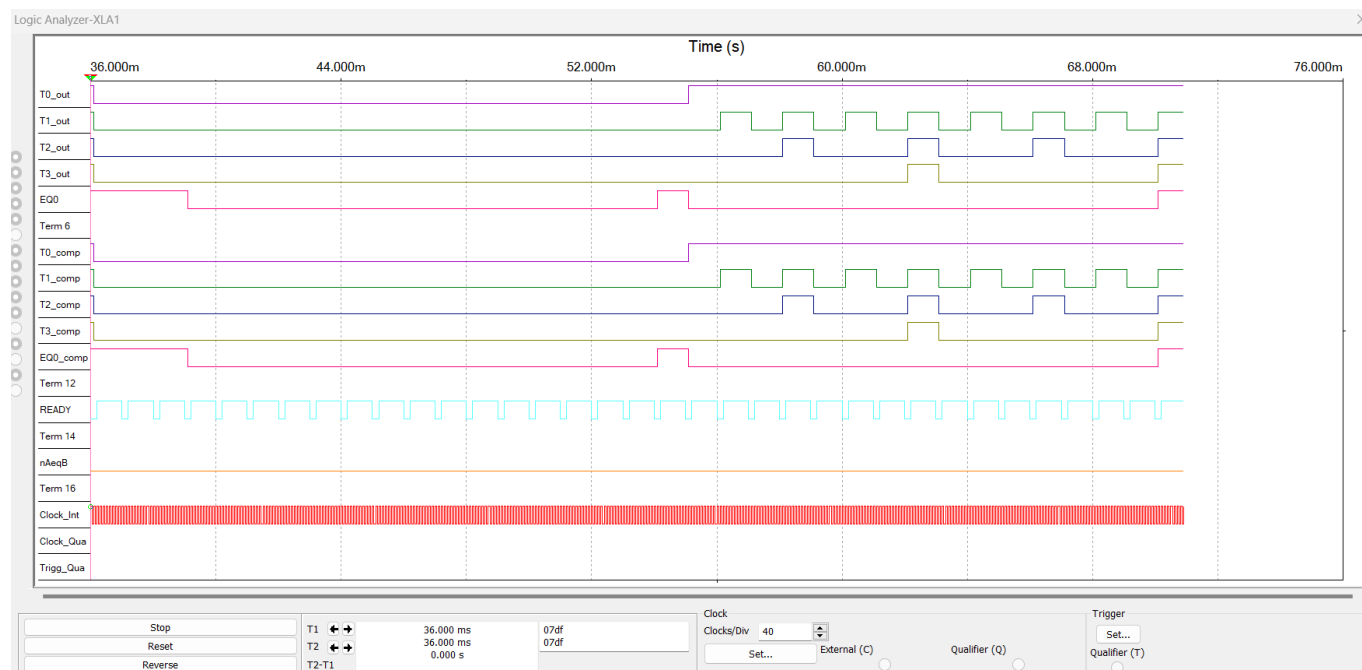
```

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:



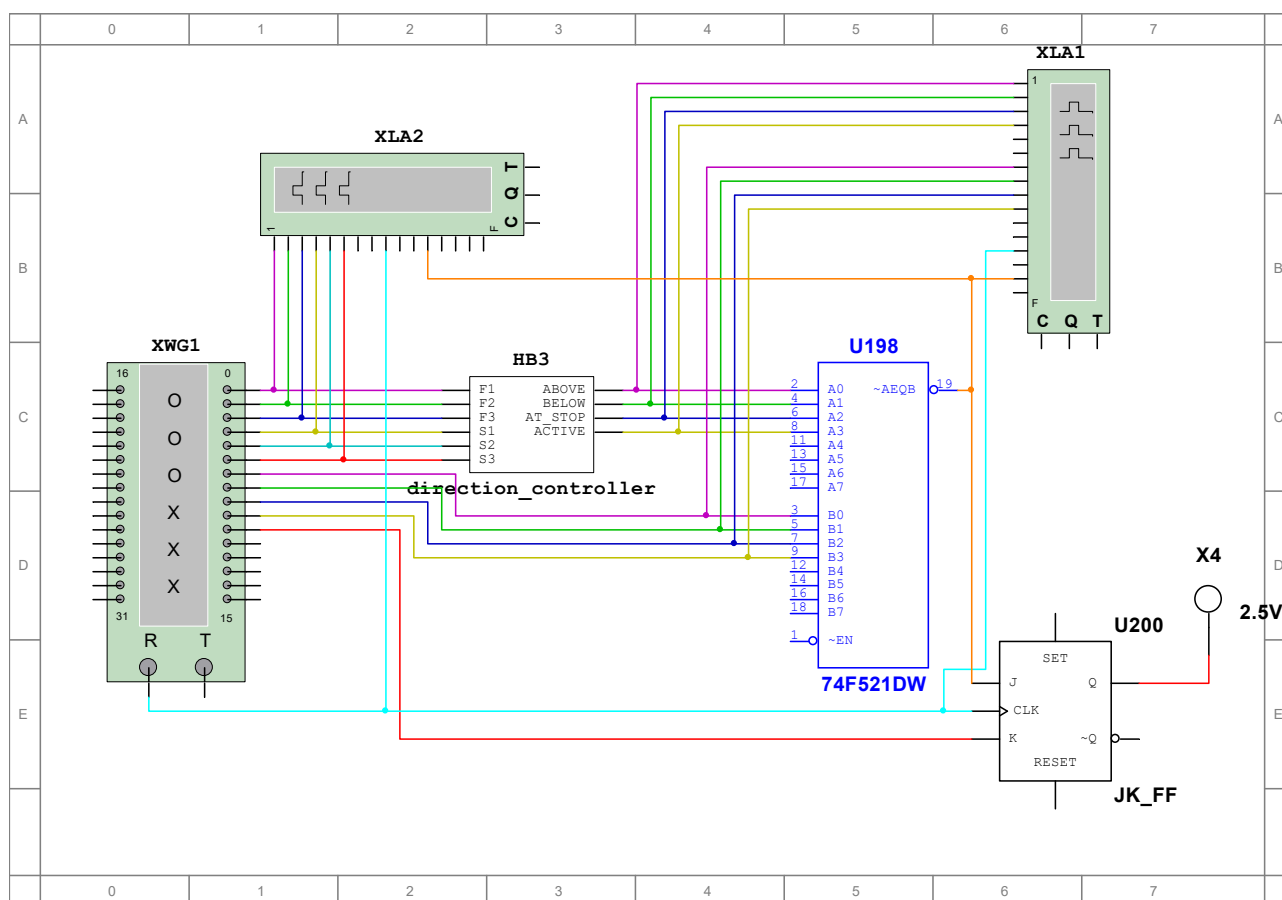
Rysunek 4.3: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



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

4.2 Testy układu direction_controller

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



Rysunek 4.5: Schemat układu testującego

Podobnie jak wcześniej, wykorzystaliśmy zmodyfikowany kod do wyprowadzania formuł.

```

1  f = open("direction_controller_test_data.dp", "w")
2  f.write("Data:\n")
3  reset_sr = 1
4  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
5  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
6  f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
7  data_count = 3
8  for i in range(64):
9      permutation = bin(i).removeprefix("0b").rjust(6, '0')
10
11     variables = {
12         'F1': int(permutation[0]),
13         'F2': int(permutation[1]),
14         'F3': int(permutation[2]),
15         'S1': int(permutation[3]),
16         'S2': int(permutation[4]),
17         'S3': int(permutation[5])
18     }
19
20     above = '0'
21     below = '0'

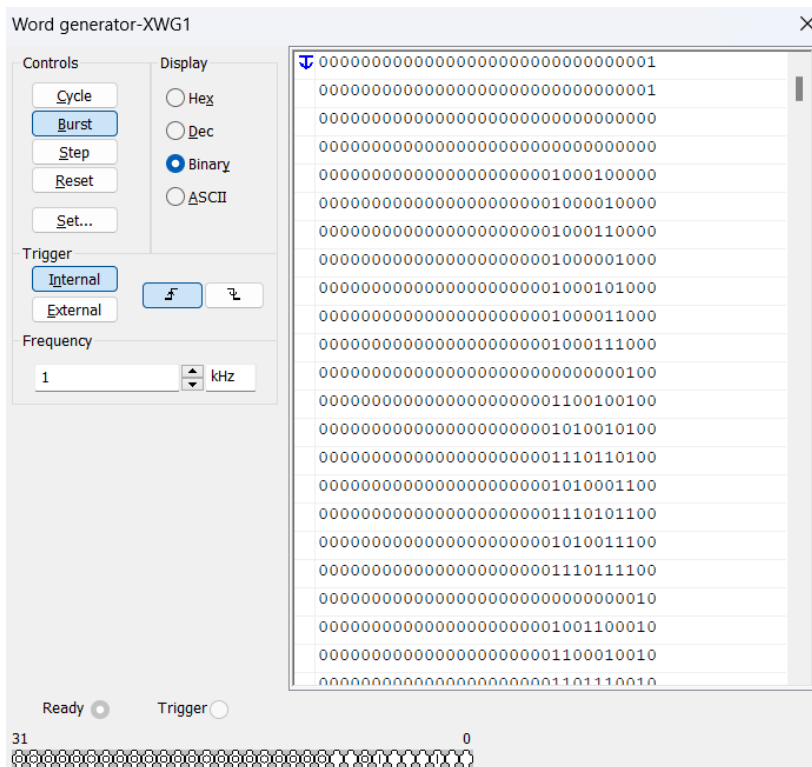
```

```

22     at_stop = '0'
23     active = '0'
24
25     if variables['F1'] and (variables['S2'] or variables['S3']):
26         above = '1'
27
28     elif variables['F2'] and variables['S1']:
29         below = '1'
30
31     elif variables['F2'] and variables['S3']:
32         above = '1'
33
34     elif variables['F3'] and (variables['S1'] or variables['S2']):
35         below = '1'
36
37     if (variables['F1'] and variables['S1']) or (variables['F2'] and variables['S2'])
38         or (variables['F3'] and variables['S3']):
39         at_stop = '1'
40
41     if variables['S1'] or variables['S2'] or variables['S3']:
42         active = '1'
43
44     F1_b = str(bin(variables['F1'])).removeprefix("0b")
45     F2_b = str(bin(variables['F2'])).removeprefix("0b")
46     F3_b = str(bin(variables['F3'])).removeprefix("0b")
47     S1_b = str(bin(variables['S1'])).removeprefix("0b")
48     S2_b = str(bin(variables['S2'])).removeprefix("0b")
49     S3_b = str(bin(variables['S3'])).removeprefix("0b")
50     above_b = str(bin(int(above))).removeprefix("0b")
51     below_b = str(bin(int(below))).removeprefix("0b")
52     at_stop_b = str(bin(int(at_stop))).removeprefix("0b")
53     active_b = str(bin(int(active))).removeprefix("0b")
54
55     hex_val = hex(int(active_b + at_stop_b + below_b + above_b + S3_b + S2_b + S1_b + F3_b + F2_b + F1_b, 2))
56                 .removeprefix("0x").rjust(8, '0')
57     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()

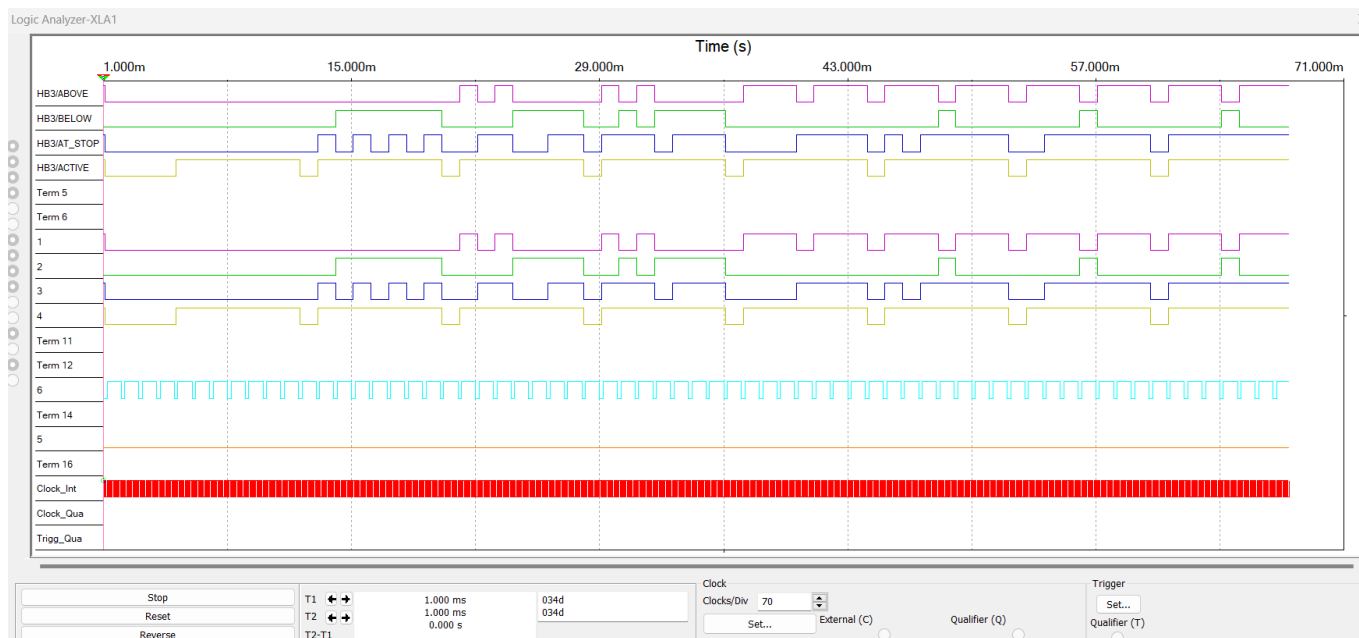
```

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

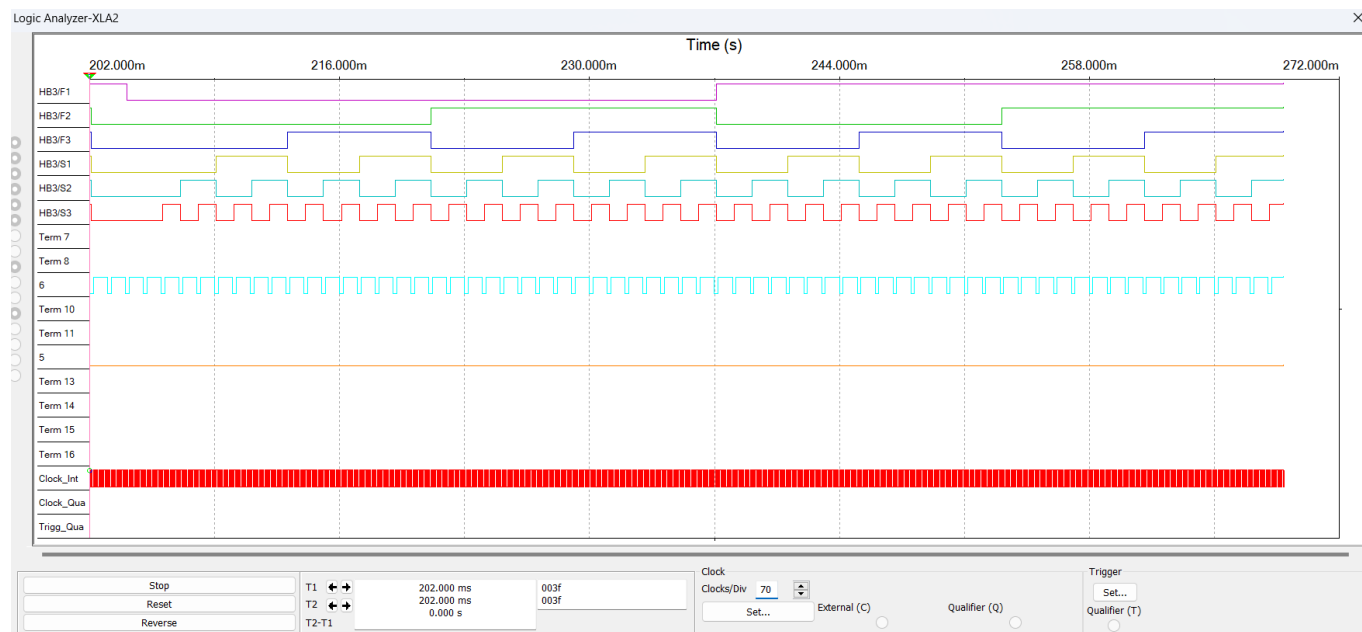


Rysunek 4.6: Dane zaimportowane do generatora słów

Poniżej wynik testu w postaci przebiegu analizatorów stanów logicznych:



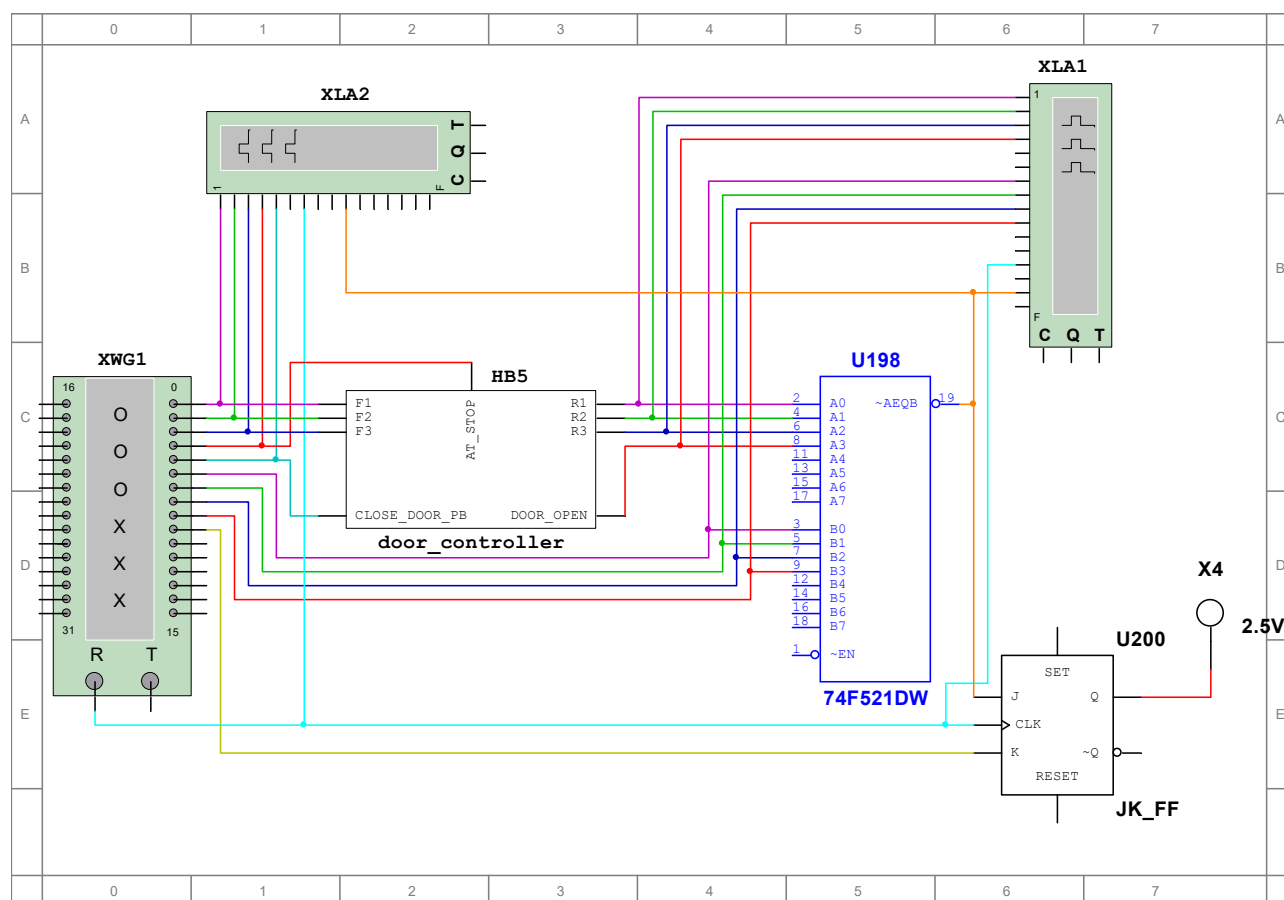
Rysunek 4.7: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



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

4.3 Testy układu door_controller

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



Rysunek 4.9: Schemat układu testującego

Podobnie jak wcześniej, wykorzystaliśmy zmodyfikowany kod do wyprowadzania formuł.

```

1  f = open("door_controller_test_data.dp", "w")
2  f.write("Data:\n")
3  reset_sr = 1
4  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
5  f.write(hex(int(str(bin(1)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
6  f.write(hex(int(str(bin(0)).removeprefix("0b").rjust(4, '0'), 2)).removeprefix("0x").rjust(8, '0') + "\n")
7  data_count = 3
8  for i in range(32):
9      permutation = bin(i).removeprefix("0b").rjust(5, '0')
10
11     variables = {
12         'CLOSE_DOOR_PB': int(permutation[0]),
13         'AT_STOP': int(permutation[1]),
14         'F1': int(permutation[2]),
15         'F2': int(permutation[3]),
16         'F3': int(permutation[4])
17     }
18
19     r1 = '0'
20     r2 = '0'
21     r3 = '0'

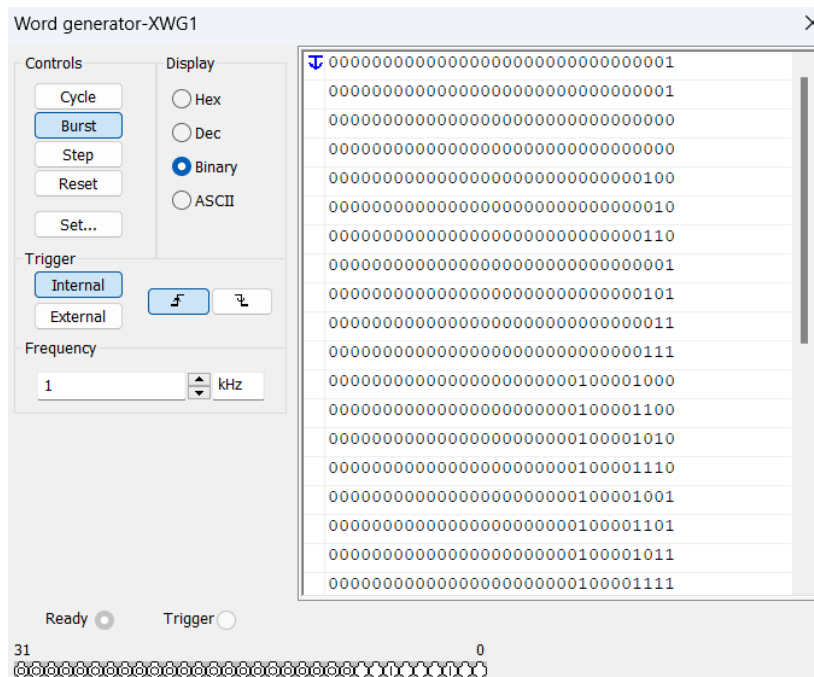
```

```

22     door_open = '0'
23
24     if variables['AT_STOP']: door_open = '1'
25     if variables['AT_STOP'] and variables['CLOSE_DOOR_PB']:
26         if variables['F1']: r1 = '1'
27         if variables['F2']: r2 = '1'
28         if variables['F3']: r3 = '1'
29
30     close_door_pb_b = str(bin(variables['CLOSE_DOOR_PB'])).removeprefix("0b")
31     at_stop_b = str(bin(variables['AT_STOP'])).removeprefix("0b")
32     F1_b = str(bin(variables['F1'])).removeprefix("0b")
33     F2_b = str(bin(variables['F2'])).removeprefix("0b")
34     F3_b = str(bin(variables['F3'])).removeprefix("0b")
35     r1_b = str(bin(int(r1))).removeprefix("0b")
36     r2_b = str(bin(int(r2))).removeprefix("0b")
37     r3_b = str(bin(int(r3))).removeprefix("0b")
38     door_open_b = str(bin(int(door_open))).removeprefix("0b")
39
40     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))
41                 .removeprefix("0x").rjust(8, '0')
42     f.write(hex_val + "\n")
43     data_count += 1
44
45 f.write("Initial:\n")
46 f.write("0000\n")
47 f.write("Final:\n")
48 f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
49
50 f.close()

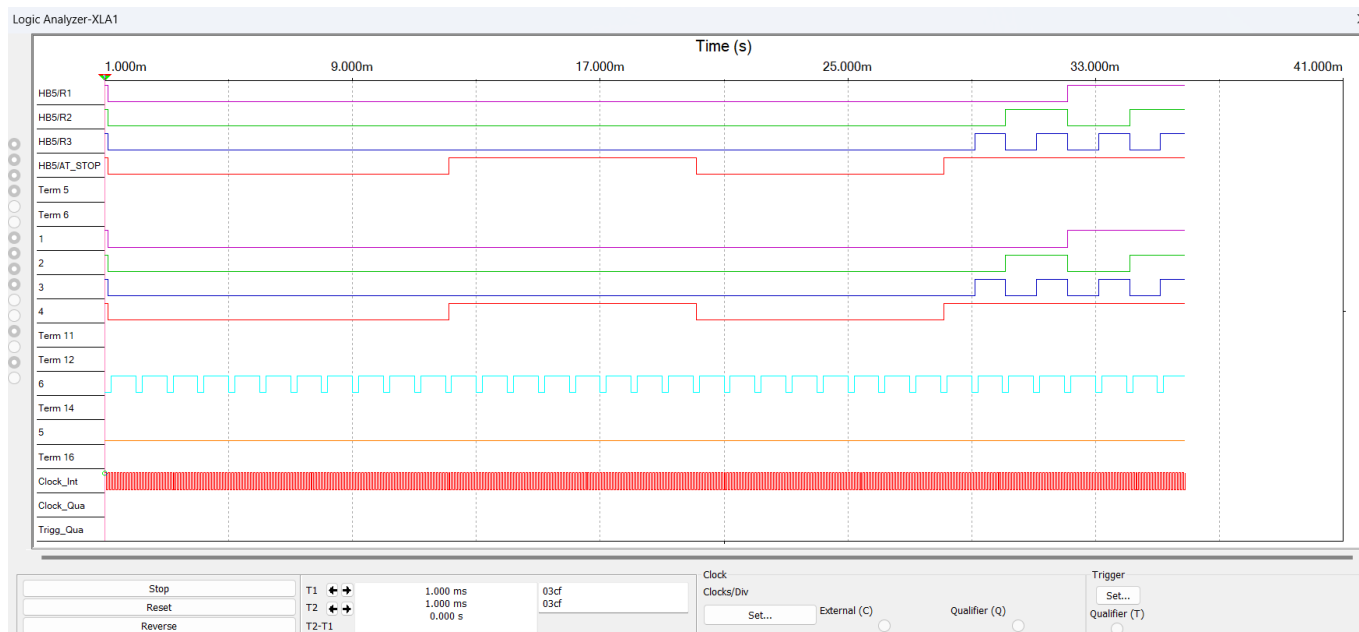
```

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



Rysunek 4.10: Dane zaimportowane do generatora słów

Poniżej wynik testu w postaci przebiegu analizatorów stanów logicznych:



Rysunek 4.11: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wyjścia układu



Rysunek 4.12: Przebieg sygnałów w analizatorze stanów logicznych podpiętym do wejścia układu

5 Zastosowania

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 języku typu VHDL reprezentacja stanów automatu byłaby dużo prostsza

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