

Maciej Byczko Bartosz Matysiak	Prowadzący: dr inż. Jacek Mazurkiewicz	Numer ćwiczenia 3
PN 10:50 TP	Temat ćwiczenia: Układy Sekwencyjne	Ocena:
Grupa: B	Data wykonania: 10 Października 2021	

Spis treści

1	Zadanie 1	2
1.1	Polecenie	2
1.2	Rozwiązanie	2
1.2.1	Schemat stanów	2
1.2.2	Tabela prawdy	2
1.2.3	Siatki Karnaugh	3
1.2.4	Schemat układu	4
1.2.5	Kod VHDL	4
1.2.6	Symulacja	5
1.2.7	Plik ucf	5
2	Zadanie 2	6
2.1	Polecenie	6
2.2	Rozwiązanie	6
2.2.1	Opis symboliki	6
2.2.2	Schemat grafowy	6
2.2.3	Tabela prawdy	7
2.2.4	Siatka Karnaugh	8
2.2.5	Schemat układu	8
2.2.6	Kod VHDL	9
2.2.7	Symulacja	10
2.2.8	Plik ucf	10
3	Zadanie 2 - wersja rozszerzona	11
3.1	Polecenie	11
3.2	Rozwiązanie	11
3.2.1	Schemat układu	11
3.2.2	Kod VHDL	11
3.2.3	Plik ucf	12
4	Wnioski	13

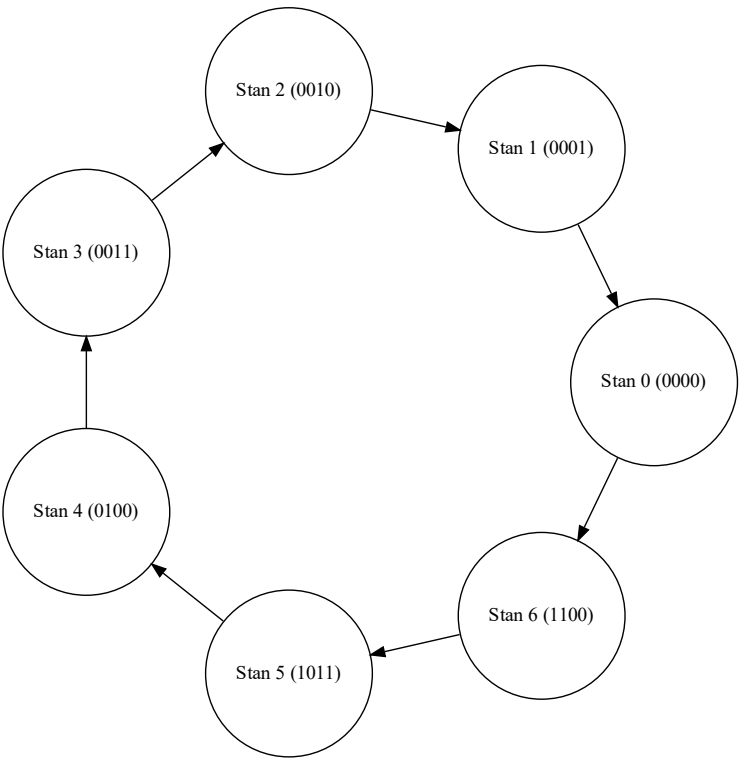
1 Zadanie 1

1.1 Polecenie

Zaprojektować licznik synchroniczny liczący w tył na bazie kodu Aikena w zakresie 0-6 (mod 7).

1.2 Rozwiązanie

1.2.1 Schemat stanów



1.2.2 Tabela prawdy

n	Q(t)				Q(t+1)				JK							
	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	1	0	0	1	-	1	-	0	-	0	-
1	0	0	0	1	0	0	0	0	0	-	0	-	0	-	-	1
2	0	0	1	0	0	0	0	1	0	-	0	-	-	1	1	-
3	0	0	1	1	0	0	1	0	0	-	0	-	-	0	-	1
4	0	1	0	0	0	0	1	1	0	-	-	1	1	-	1	-
5	1	0	1	1	0	1	0	0	-	1	1	-	-	1	-	1
6	1	1	0	0	1	0	1	1	-	0	-	1	1	-	1	-

1.2.3 Siatki Karnaugh

		Q_1Q_0			
		00	01	11	10
Q_3Q_2	00	1	0	0	0
	01	0	-	-	-
	11	-	-	-	-
	10	-	-	-	-

$$J_3 = \overline{Q_2} \overline{Q_1} \overline{Q_0}$$

		Q_1Q_0			
		00	01	11	10
Q_3Q_2	00	1	0	0	0
	01	-	-	-	-
	11	-	-	-	-
	10	-	-	1	-

$$J_2 = \overline{Q_1} \overline{Q_0} + Q_3$$

		00	01	11	10
Q_3Q_2	00	0	0	-	-
	01	1	-	-	-
	11	1	-	-	-
	10	-	-	-	-

$$J_1 = Q_2$$

		00	01	11	10
Q_3Q_2	00	0	-	-	1
	01	1	-	-	-
	11	1	-	-	-
	10	-	-	-	-

$$J_0 = Q_1 + Q_2$$

		00	01	11	10
Q_3Q_2	00	-	-	-	-
	01	-	-	-	-
	11	0	-	-	-
	10	-	-	1	-

$$K_3 = Q_1$$

		00	01	11	10
Q_3Q_2	00	-	-	-	-
	01	1	-	-	-
	11	1	-	-	-
	10	-	-	-	-

$$K_2 = 1$$

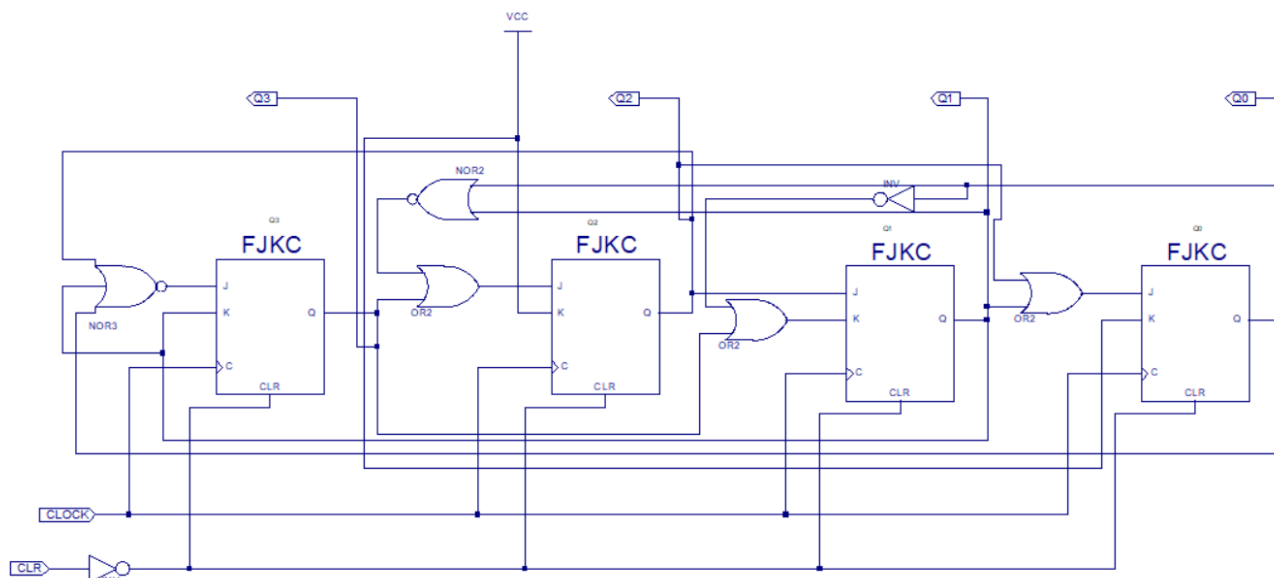
		00	01	11	10
Q_3Q_2	00	-	-	0	1
	01	-	-	-	-
	11	-	-	-	-
	10	-	-	1	-

$$K_1 = \overline{Q_0} + Q_3$$

		00	01	11	10
Q_3Q_2	00	-	1	1	-
	01	-	-	-	-
	11	-	-	-	-
	10	-	-	1	-

$$K_0 = 1$$

1.2.4 Schemat układu



1.2.5 Kod VHDL

```

1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY AikenCounterScheme_AikenCounterScheme_sch_tb IS
7  END AikenCounterScheme_AikenCounterScheme_sch_tb;
8  ARCHITECTURE behavioral OF
    AikenCounterScheme_AikenCounterScheme_sch_tb IS
9
10     COMPONENT AikenCounterScheme
11     PORT( Q2 : OUT STD_LOGIC;
12           Q1  : OUT STD_LOGIC;
13           Q0  : OUT STD_LOGIC;
14           CLOCK : IN  STD_LOGIC;
15           Q3   : OUT STD_LOGIC;
16           CLR  : IN  STD_LOGIC);
17     END COMPONENT;
18
19     SIGNAL Q2  : STD_LOGIC;
20     SIGNAL Q1  : STD_LOGIC;
21     SIGNAL Q0  : STD_LOGIC;
22     SIGNAL CLOCK : STD_LOGIC := '0';
23     SIGNAL Q3   : STD_LOGIC;
24     SIGNAL CLR  : STD_LOGIC;
25
26 BEGIN
27
28     UUT: AikenCounterScheme PORT MAP(
29         Q2 => Q2,
30         Q1 => Q1,

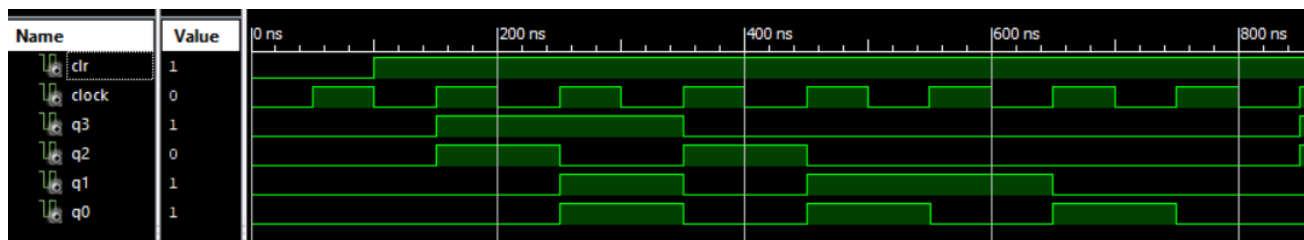
```

```

31     Q0 => Q0,
32     CLOCK => CLOCK,
33     Q3 => Q3,
34     CLR => CLR
35 );
36 CLR <= '0', '1' after 100 ns;
37 CLOCK <= not CLOCK after 50 ns;
38
39 END;

```

1.2.6 Symulacja



UWAGA: Powyższy przebieg przedstawia działanie układu licznika rozpoczęte stanem "0000".

1.2.7 Plik ucf

```

1 # Clocks
2 NET "CLOCK" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;
3
4 # Keys
5 NET "CLR" LOC = "P42";
6
7 # LEDS
8 NET "Q0" LOC = "P35";
9 NET "Q1" LOC = "P29";
10 NET "Q2" LOC = "P33";
11 NET "Q3" LOC = "P34";

```

Podczas zajęć laboratoryjnych, przy pierwszej wersji programu, aby licznik rozpoczął pracę, musieliśmy trzymać przycisk resetu, gdyż zastosowaliśmy negację na wejściu CLR.

2 Zadanie 2

2.1 Polecenie

Detektor sekwencji 11011, automat Mealy-ego, jedno wejście, jedno wyjście, brak resetu, sekwencja prawidłowa 5-bitowa.

2.2 Rozwiązanie

2.2.1 Opis symboliki

Alfabet wejściowy

- $z_0 = 0$
- $z_1 = 1$

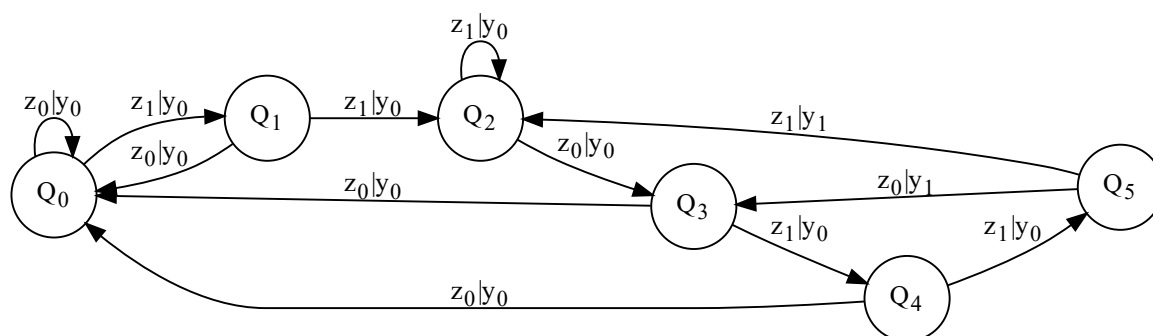
Stany wewnętrzne

- q_0 - stan początkowy | wprowadzono niepoprawny ciąg bitów
- q_1 - wprowadzono pierwszą cyfrę prawidłowego ciągu
- q_2 - wprowadzono drugą cyfrę prawidłowego ciągu
- q_3 - wprowadzono trzecią cyfrę prawidłowego ciągu
- q_4 - wprowadzono czwartą cyfrę prawidłowego ciągu
- q_5 - wprowadzono poprawną sekwencję

Alfabet wyjścia

- y_0 - Wprowadzony ciąg nadal jest niepoprawny
- y_1 - Wprowadzono poprawną sekwencję

2.2.2 Schemat grafowy



2.2.3 Tabela prawdy

S	Q(t)			Z	Q(t+1)			Y	T(t)		
	Q_2	Q_1	Q_0		Q_2	Q_1	Q_0		T_2	T_1	T_0
Q_0	0	0	0	0	0	0	0	0	0	0	0
Q_0	0	0	0	1	0	0	1	0	0	0	1
Q_1	0	0	1	0	0	0	0	0	0	0	1
Q_1	0	0	1	1	0	1	0	0	0	1	1
Q_2	0	1	0	0	0	1	1	0	0	0	1
Q_2	0	1	0	1	0	1	0	0	0	0	0
Q_3	0	1	1	0	0	0	0	0	0	1	1
Q_3	0	1	1	1	1	0	0	0	1	1	1
Q_4	1	0	0	0	0	0	0	0	1	0	0
Q_4	1	0	0	1	1	0	1	0	0	0	1
Q_5	1	0	1	0	0	1	1	1	1	1	0
Q_5	1	0	1	1	0	1	0	1	1	1	1
-	1	1	0	0	-	-	-	-	-	-	-
-	1	1	0	1	-	-	-	-	-	-	-
-	1	1	1	0	-	-	-	-	-	-	-
-	1	1	1	1	-	-	-	-	-	-	-

2.2.4 Siatka Karnaugh

		Q_0Z			
		00	01	11	10
Q_2Q_1	00	0	0	0	0
	01	0	0	1	0
	11	-	-	-	-
	10	1	0	1	1

$$T_2 = Q_1Q_0Z + Q_2\bar{Z} + Q_2Q_0$$

		Q_0Z			
		00	01	11	10
Q_2Q_1	00	0	0	1	0
	01	0	0	1	1
	11	-	-	-	-
	10	0	0	1	1

$$T_1 = Q_0Z + Q_1Q_0 + Q_2Q_0$$

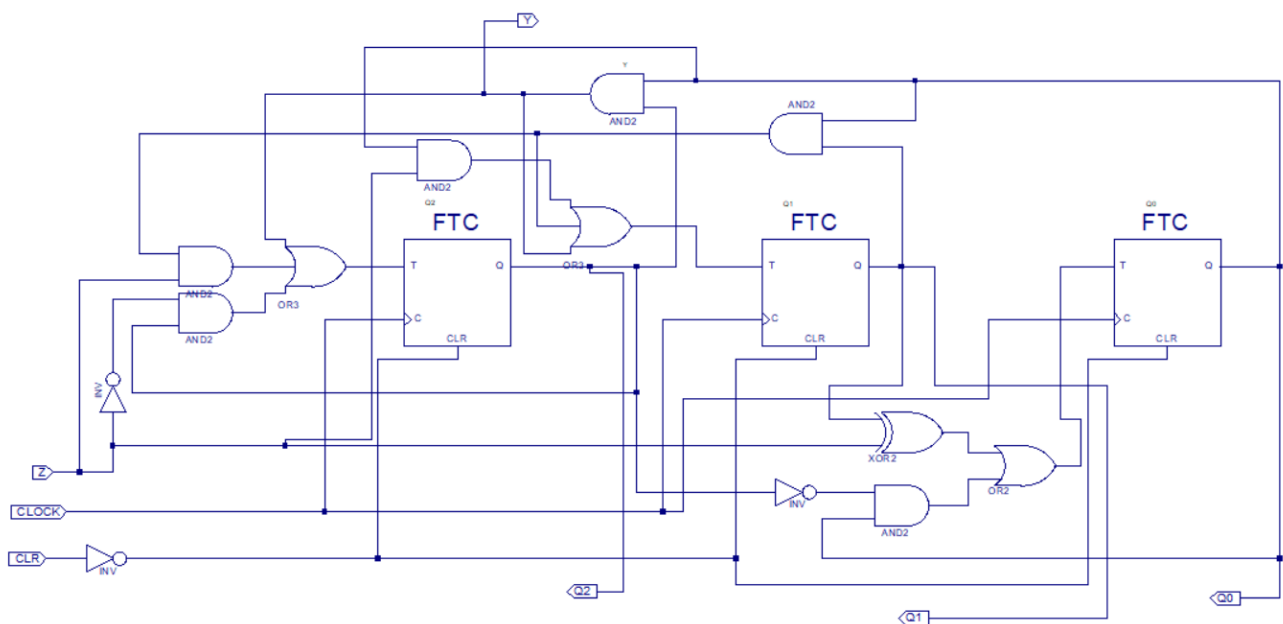
		Q_0Z			
		00	01	11	10
Q_2Q_1	00	0	1	1	1
	01	1	0	1	1
	11	-	-	-	-
	10	0	1	1	0

$$T_0 = \bar{Q}_2Q_0 + Q_1\bar{Z} + \bar{Q}_1Z$$

		Q_0Z			
		00	01	11	10
Q_2Q_1	00	0	0	0	0
	01	0	0	0	0
	11	-	-	-	-
	10	0	0	1	1

$$Y = Q_2Q_0$$

2.2.5 Schemat układu

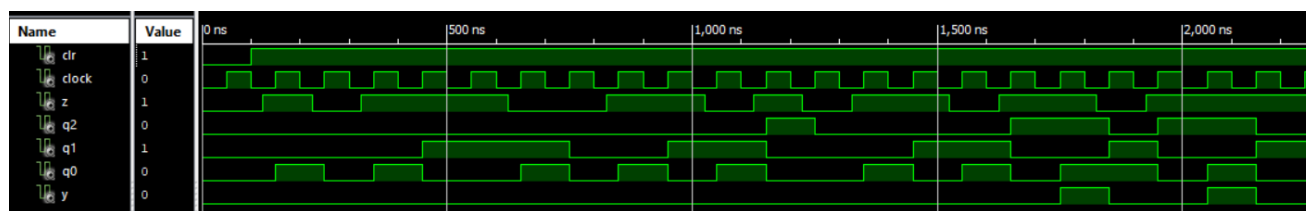


2.2.6 Kod VHDL

```

1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY MealyDetectorScheme_MealyDetectorScheme_sch_tb IS
7  END MealyDetectorScheme_MealyDetectorScheme_sch_tb;
8  ARCHITECTURE behavioral OF
    MealyDetectorScheme_MealyDetectorScheme_sch_tb IS
9
10     COMPONENT MealyDetectorScheme
11     PORT( Y   : OUT STD_LOGIC;
12           Q0  : OUT STD_LOGIC;
13           Q2  : OUT STD_LOGIC;
14           Z   : IN  STD_LOGIC;
15           CLOCK : IN  STD_LOGIC;
16           Q1  : OUT STD_LOGIC;
17           CLR  : IN  STD_LOGIC);
18     END COMPONENT;
19
20     SIGNAL Y : STD_LOGIC;
21     SIGNAL Q0 : STD_LOGIC;
22     SIGNAL Q2 : STD_LOGIC;
23     SIGNAL Z : STD_LOGIC;
24     SIGNAL CLOCK : STD_LOGIC := '0';
25     SIGNAL Q1 : STD_LOGIC;
26     SIGNAL CLR : STD_LOGIC ;
27
28 BEGIN
29
30     UUT: MealyDetectorScheme PORT MAP(
31         Y => Y,
32         Q0 => Q0,
33         Q2 => Q2,
34         Z => Z,
35         CLOCK => CLOCK,
36         Q1 => Q1,
37         CLR => CLR
38     );
39     CLR <= '0', '1' after 100 ns;
40     CLOCK <= not CLOCK after 50 ns;
41
42     Z <= '0', '1' after 125 ns, '0' after 225 ns, '1' after 325 ns,
43         '0' after 625 ns, '1' after 825 ns, '0' after 1025 ns, '1' after
44         1125 ns, '0' after 1225 ns, '1' after 1325 ns, '0' after 1525
45         ns, '1' after 1625 ns, '0' after 1825 ns, '1' after 1925 ns;
46
47 END;
```

2.2.7 Symulacja



UWAGA: Powyższy przebieg obrazuje przejście po każdej krawędzi rozpatrywanego grafu automatu Mealy'ego. Przejście do nowego stanu automatu realizuje się w momencie wystąpienia zbocza wznoszącego na wejściu CLOCK. Z przebiegu można odczytać m.in. to, że sekwencja prawidłowa "11011" została w przebiegu wykryta dwukrotnie, a także, że w chwili $t = 475ns$ układ znajdował się w stanie q_2 , i nie akceptował bieżącej sekwencji.

2.2.8 Plik ucf

```

1 # Clocks
2 NET "CLOCK" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;
3
4 # Keys
5 NET "Z" LOC = "P42";
6 NET "CLR" LOC = "P40";
7
8 # LEDS
9 NET "Q0" LOC = "P35";
10 NET "Q1" LOC = "P29";
11 NET "Q2" LOC = "P33";
12 NET "Y" LOC = "P34";

```

Podczas zajęć laboratoryjnych, przy pierwszej wersji programu, aby wprowadzić kombinację, musieliśmy trzymać przycisk resetu, gdyż zastosowaliśmy negację na wejściu CLR.

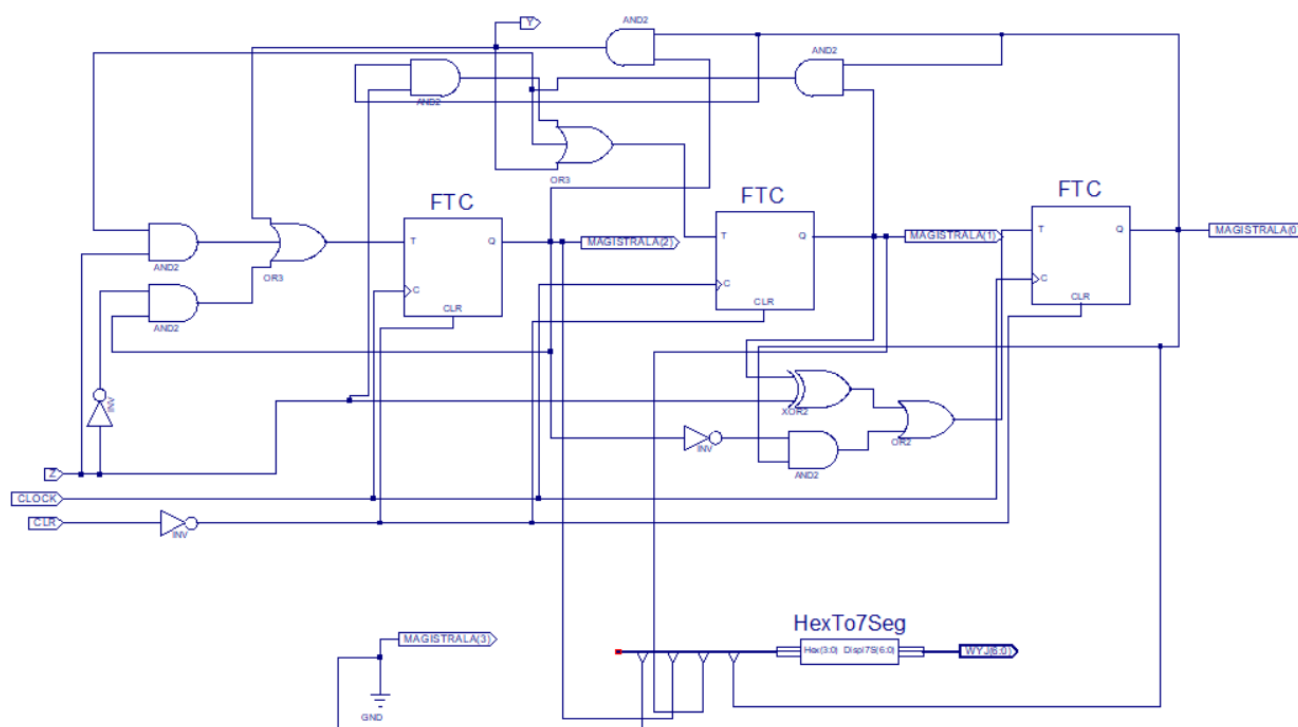
3 Zadanie 2 - wersja rozszerzona

3.1 Polecenie

Rozszerzenie podstawowej wersji detektora o obsługę wyświetlacza siedmiosegmentowego.

3.2 Rozwiązanie

3.2.1 Schemat układu



3.2.2 Kod VHDL

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY scheme_scheme_sch_tb IS
7  END scheme_scheme_sch_tb;
8  ARCHITECTURE behavioral OF scheme_scheme_sch_tb IS
9
10     COMPONENT scheme
11     PORT( Y : OUT STD_LOGIC;
12           MAGISTRALA : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
13           Z : IN STD_LOGIC;
14           CLR : IN STD_LOGIC;
15           CLOCK : IN STD_LOGIC);
16     END COMPONENT;
17
18     SIGNAL Y : STD_LOGIC;
19     SIGNAL MAGISTRALA : STD_LOGIC_VECTOR (3 DOWNTO 0);

```

```

20  SIGNAL Z : STD_LOGIC;
21  SIGNAL CLR : STD_LOGIC;
22  SIGNAL CLOCK : STD_LOGIC := '0';
23
24  BEGIN
25
26      UUT: scheme PORT MAP(
27          Y => Y,
28          MAGISTRALA => MAGISTRALA,
29          Z => Z,
30          CLR => CLR,
31          CLOCK => CLOCK
32      );
33
34      CLR <= '0', '1' after 100 ns;
35      CLOCK <= not CLOCK after 50 ns;
36
37      Z <= '0', '1' after 125 ns, '0' after 225 ns, '1' after 325 ns,
          '0' after 625 ns, '1' after 825 ns, '0' after 1025 ns, '1' after
          1125 ns, '0' after 1225 ns, '1' after 1325 ns, '0' after 1525
          ns, '1' after 1625 ns, '0' after 1825 ns, '1' after 1925 ns;
38
39
40  END;
```

3.2.3 Plik ucf

```

1  # Clocks
2  NET "CLOCK" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;
3
4  # Keys
5  NET "Z" LOC = "P42";
6  NET "CLR" LOC = "P40";
7
8  # LEDS
9  NET "Y" LOC = "P35";
10
11 # DISPL. 7-SEG
12 NET "WYJ(0)" LOC = "P12"; # Seg. A; shared with LED<10>
13 NET "WYJ(1)" LOC = "P13"; # Seg. B; shared with LED<8>
14 NET "WYJ(2)" LOC = "P22"; # Seg. C; shared with LED<12>
15 NET "WYJ(3)" LOC = "P19"; # Seg. D; shared with LED<14>
16 NET "WYJ(4)" LOC = "P14"; # Seg. E; shared with LED<15>
17 NET "WYJ(5)" LOC = "P11"; # Seg. F; shared with LED<9>
18 NET "WYJ(6)" LOC = "P20"; # Seg. G; shared with LED<13>
```

4 Wnioski

Przy wykonywaniu zadań niezbędna okazała się wiedza na temat typów przerzutników i ich działania. Oprócz tego, ćwiczenia wymagały także zapoznania się z opracowaniem dotyczącym zestawu ZL-9572, a konkretniej - generatorów sygnałów prostokątnych *Clk_XT* i *Clk_LF* oraz modułu czterocyfrowego wyświetlacza siedmiosegmentowego.

Pierwsze zadanie dotyczyło stworzenia licznika negatywnego modulo 7 zliczającego w kodzie Aikena. Wykorzystaliśmy w jego rozwiązaniu przerzutniki JK, a konkretnie komponenty FJKC z biblioteki programu Xilinx, przyjmując przy okazji założenie, że praca licznika rozpoczyna się od jego resetu (sprowadzenia do stanu "0000"), który należało zrealizować, czy to poprzez uaktywnienie sygnału CLR na etapie symulacji, czy naciśnięcie odpowiednio przypisanego przycisku w implementacji sprzętowej. Alternatywnie możliwe byłoby także wykorzystanie komponentów FJKP bądź FJKCP, gdyby zachodziła konieczność stworzenia układu rozpoczynającego pracę w innym stanie początkowym.

Podobne założenie dotyczące ustalenia stanu początkowego poprzez reset na początku pracy układu przyjęliśmy także w zadaniu 2. Należało w nim stworzyć układ detektora sekwencji "11011" oparty na automacie Mealy'ego. Przyjęliśmy, że układ prawidłowo reaguje na dowolnie długą sekwencję, o ile kończy się ona wcześniej wspomnianą, 5-bitową; możliwe jest w szczególności kilkukrotne wykrycie poprawnej sekwencji w trakcie pracy układu.

W rozszerzonej wersji, zadanie 2 należało rozszerzyć o prezentację aktualnego stanu układu przy pomocy wyświetlacza siedmiosegmentowego. Jego obsługę zrealizowano przy pomocy układu transkodera HexTo7Seg, który, jak sugeruje nazwa owego modułu, przekształca 4-bitową reprezentację cyfry heksadecymalnej na odpowiednią sekwencję 7 bitów do zapalenia odpowiednich segmentów wyświetlacza.