POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Układy cyfrowe i systemy wbudowane

Laboratorium 3

Licznik synchroniczny o zadanych parametrach funkcjonalnych
Detektor sekwencji bitowej

Termin zajęć: Czwartek TP, 7:30

Autorzy:
Daria Jeżowska, 252731
Kacper Śleziak, 252703

Prowadzący zajęcia:
dr inż. Jacek Mazurkiewicz

Zadania do wykonania

Naszym zadaniem było zbudowanie schematów dwóch układów w progamie *Xillinx*, a następnie napisanie kodu w języku *VHDL* i zasymulowanie układu. Jeśli symulacja zadziała – należało przetestować układ na prawdziwej płytce. Pierwszym zadaniem było stworzenie licznika negatywnego (zliczającego do dołu) synchronicznego modulo 7, działającego w kodzie Aikena. Kolejnym zadaniem było stworzenie detektora sekwencji **11011** na podstawie automatu Mealy'ego.

Zadanie 1

Licznik synchroniczny o zadanych parametrach funkcjonalnych:

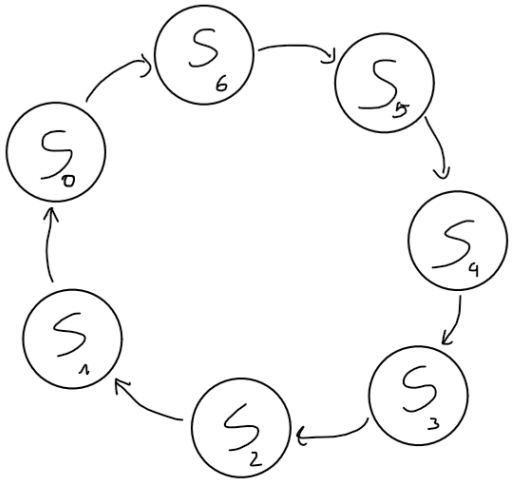
synchroniczny, mod 7, negatywny, kod Aikena

Założenia:

- konieczna prezentacja algorytmu tworzenia układu licznika, przerzutniki dowolne,
- licznik powinien zliczać impulsy zegarowe generowane w przystawce,
- bieżący stan licznika ma być widoczny na diodach przystawki,
- wolno stosować dowolne elementy dostępne w bibliotekach Xilinx-a.

Niezakodowany graf przejść

Stany reprezentują kolejno cyfry od 6 do 0 zapisane binarnie, gdzie stan S6 oznacza 6, a stan S0 oznacza 0. Pierwszym stanem od, którego zaczynamy liczenie w dół jest S6.



Zakodowana tabela prawdy

	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**
6	1	1	0	0	1	0	1	1	-	0	-	1	1	-	1	-
5	1	0	1	1	0	1	0	0	-	1	1	-	-	1	-	1
4	0	1	0	0	0	0	1	1	0	-	-	1	1	-	1	-
3	0	0	1	1	0	0	1	0	0	-	0	-	-	0	-	1
2	0	0	1	0	0	0	0	1	0	-	0	-	-	1	1	-
1	0	0	0	1	0	0	0	0	0	-	0	-	0	-	-	1
0	0	0	0	0	1	1	0	0	1	-	1	-	0	-	0	-

Minimalizacja siatkami Karnaugh

J3

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	1	0	0	0
01	0	-	-	-
11	-	-	-	-
10	-	-	-	-

$J_3 = \overline{Q_2 Q_1 Q_0}$

K3

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	-	-	-	-
01	-	-	-	-
11	0	-	-	-
10	-	-	1	-

$K_3 = Q_1$

J2

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	1	0	0	0
01	-	-	-	-
11	-	-	-	-
10	-	-	1	-

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

K2

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	-	-	-	-
01	1	-	-	-
11	1	-	-	-
10	-	-	-	-

$K_2 = 1$

J1

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	0	0	-	-
01	1	-	-	-
11	1	-	-	-
10	-	-	-	-

$J_1 = Q_2$

K1

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	-	-	0	1
01	-	-	-	-
11	-	-	-	-
10	-	-	1	-

$K_1 = Q_3 + \overline{Q_0}$

J0

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	0	-	-	1
01	1	-	-	-
11	1	-	-	-
10	-	-	-	-

$J_0 = Q_2 + Q_1$

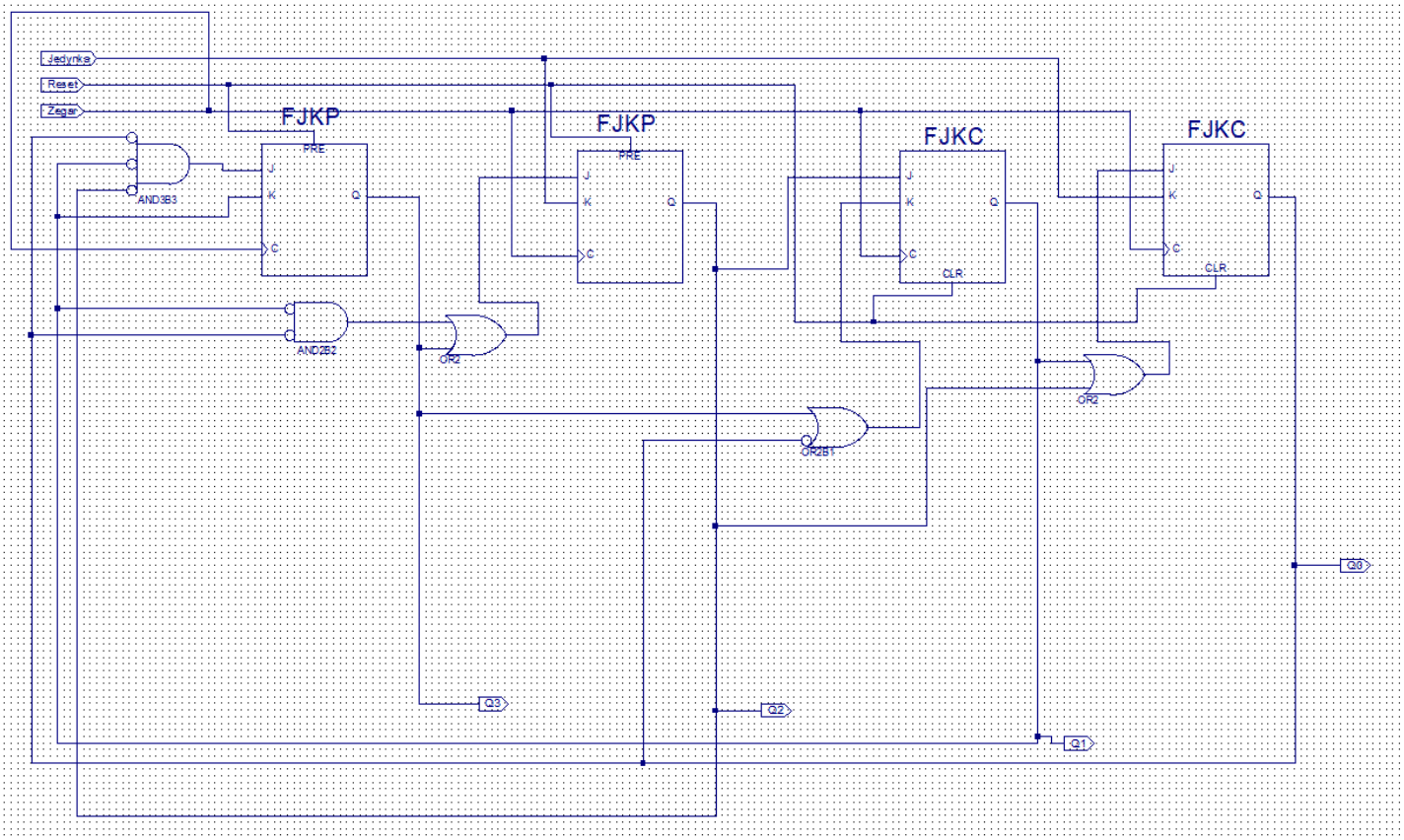
K0

$Q_3Q_2 \setminus Q_1Q_0$	00	01	11	10
00	-	1	1	-
01	-	-	-	-
11	-	-	-	-
10	-	-	1	-

$K_0 = 1$

Schemat układu

Do stworzenia układu wykorzystaliśmy 2 typy przerzutników FJKP oraz FJKC. Zdecydowaliśmy się na użycie 2 typów przerzutników o różnych stanach początkowych (0 dla FJKC oraz 1 dla FJKP) aby zgodnie z treścią zadania zacząć odliczanie od stanu 1100. Przerzutniki różnią się nieco sposobem sterowania na początku. FJKC zawiera asynchroniczny prereset, a FJKP asynchroniczny reset. Oba wyżej wymienione wejścia są stale ustawione na 0, co powoduje prawidłowe działanie przerzutników od samego początku bez potrzeby nadawania dodatkowego sygnału wejściowego.



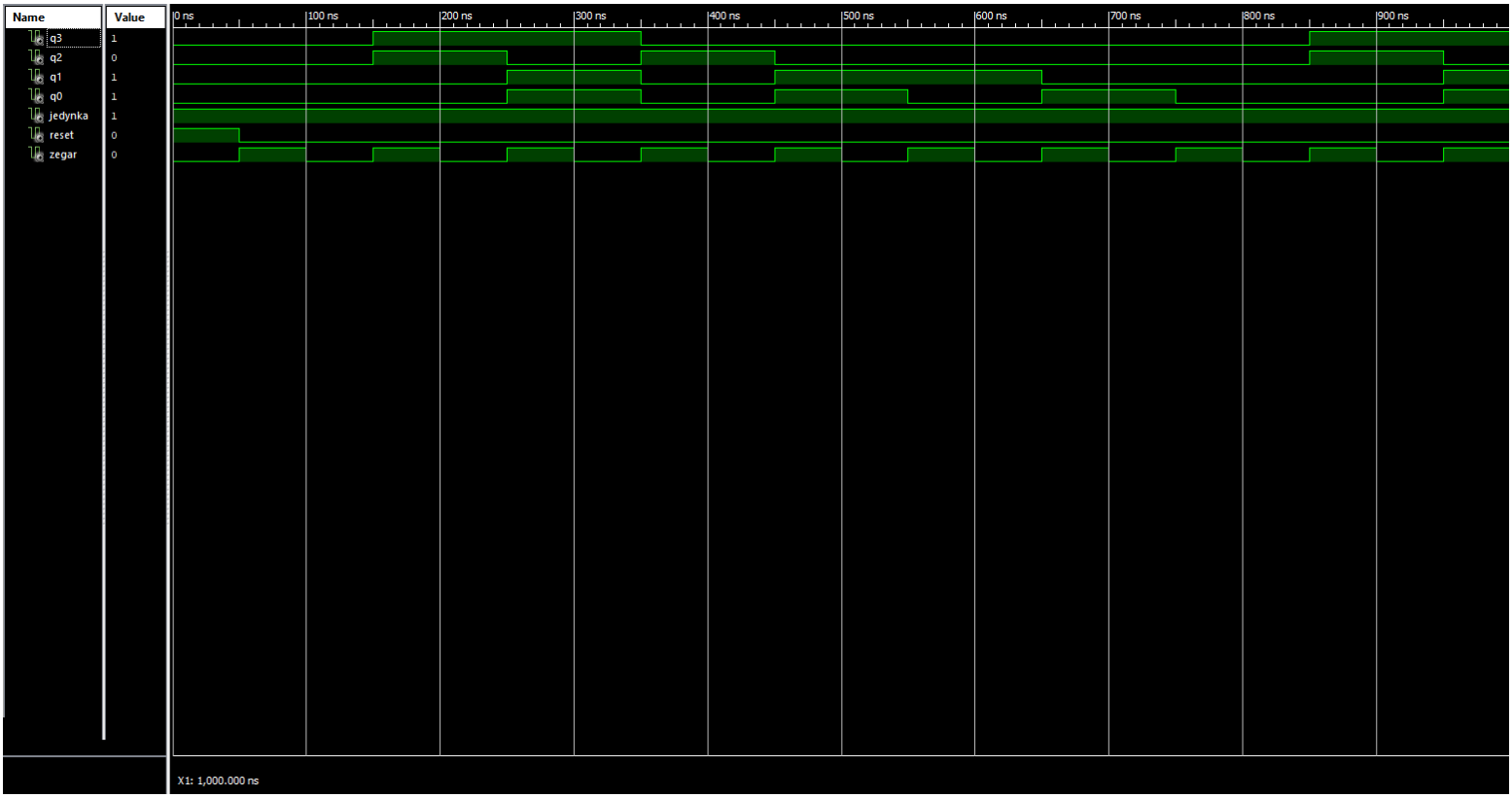
Kod w VHDL

Kod w języku *VHDL* został wygenerowany automatycznie, a jedyne zmiany, które zostały ręcznie wprowadzone do kodu dotyczyły taktowania zegara.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY counter_counter_sch_tb IS
END counter_counter_sch_tb;
ARCHITECTURE behavioral OF counter_counter_sch_tb IS

   COMPONENT counter
   PORT( Q3 :  OUT   STD_LOGIC;
         Q2   :  OUT   STD_LOGIC;
         Q1   :  OUT   STD_LOGIC;
         Q0   :  OUT   STD_LOGIC;
         Jedynka :  IN STD_LOGIC;
         Reset   :  IN STD_LOGIC;
         Zegar   :  IN STD_LOGIC);
   END COMPONENT;

   SIGNAL Q3   :  STD_LOGIC;
   SIGNAL Q2   :  STD_LOGIC;
   SIGNAL Q1   :  STD_LOGIC;
   SIGNAL Q0   :  STD_LOGIC;
   SIGNAL Jedynka :  STD_LOGIC;
   SIGNAL Reset   :  STD_LOGIC;
   SIGNAL Zegar   :  STD_LOGIC:='0';

BEGIN

   UUT: counter PORT MAP(
      Q3 => Q3,
      Q2 => Q2,
      Q1 => Q1,
      Q0 => Q0,
      Jedynka => Jedynka,
      Reset => Reset,
      Zegar => Zegar
   );
   Jedynka <= '1';
   Reset <= '0';
   Zegar <= not Zegar after 50ns;
-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Przebieg symulacji

Jak widać na symulacji licznik działa zgodnie z założeniami, pierwszym stanem od którego zaczynamy jest q3q2q1q0 = 1100 (liczba 6 w systemie dziesiętnym), a kończy się na na q3q2q1q0 = 0000 (liczba 0 w systemie dziesiętnym). Po przejściu wszystkich cyfr od 6 do 0 w systemie binarnym licznik przechodzi do stanu początkowego reprezentującego cyfrę 6 i zaczyna zliczanie w dół od początku.



Testowanie na płytce

Najpierw musieliśmy odpowiednio skonfigurować plik UCF - nasz zegar ustawiliśmy na czas 5ms, abyśmy byli w stanie cokolwiek zobaczyć na płytce. Zegar ustawiliśmy na 5ms, abyśmy byli w stanie zobaczyć co się dzieje z diodami led. Te natomiast odpowiednio przypisaliśmy do ledów 0 (Q_0), 1 (Q_1), 2 (Q_2) oraz 3 (Q_3). Przyciski 0 i 1 natomiast przypisaliśmy wejściom *Jedynka* i *Reset*. Układ został przetestowany i działał poprawnie.

```
# Clocks
NET "Zegar" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "Jedynka" LOC = "P42";
NET "Reset" LOC = "P40";
#NET "Key<2>" LOC = "P43";
#NET "Key<3>" LOC = "P38";
#NET "Key<4>" LOC = "P37";
#NET "Key<5>" LOC = "P36";  # shared with ROT_A
#NET "Key<6>" LOC = "P24";  # shared with ROT_B
#NET "Key<7>" LOC = "P39";  # GSR

# LEDS
NET "Q0"  LOC = "P35";
NET "Q1"  LOC = "P29";
NET "Q2"  LOC = "P33";
NET "Q3"  LOC = "P34";
#NET "LED<4>"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
#NET "LED<7>"  LOC = "P25";
```

Zadanie 2

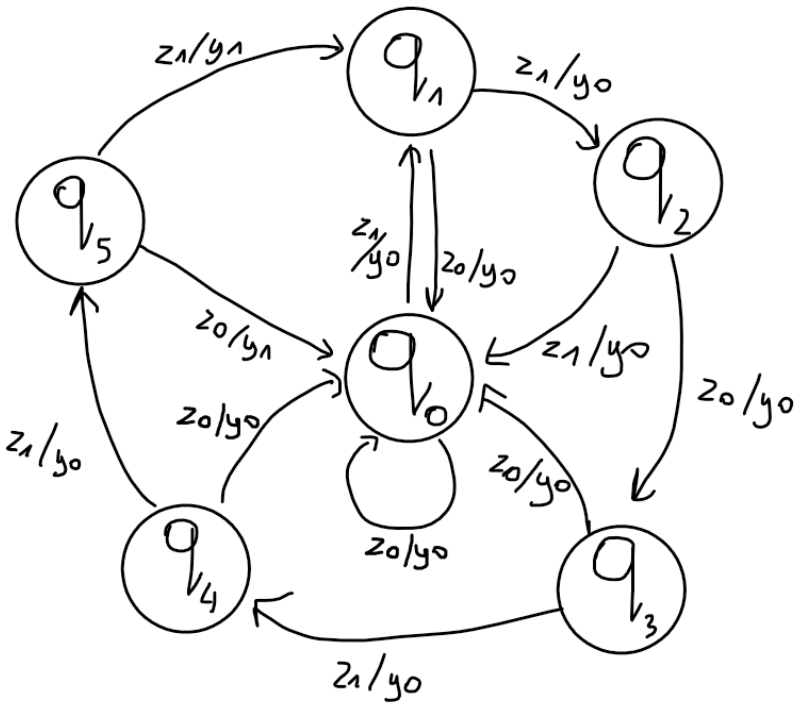
Detektor sekwencji bitowej opisany wskazanym typem automatu, jedno wejście, jedno wyjście, brak resetu, sekwencja prawidłowa 5-bitowa

sekwencja: 11011, automat Mealy-ego

Założenia:

- konieczna prezentacja algorytmu tworzenia układu detektora, przerzutniki dowolne,
- detektor jest napędzany zegarem generowanym w przystawce,
- wybrany guzik z przystawki ma służyć do wprowadzania sekwencji bitowej,
- wykrycie prawidłowej sekwencji sygnalizuje wybrana dioda z przystawki,
- w „ulepszonej" wersji układu:
    - Rotary Encoder ma służyć do wprowadzania sekwencji bitowej: pokręcanie w prawo generuje 1, pokręcanie w lewo generuje 0,
    - bieżący stan układu jest prezentowany w postaci cyfry na wyświetlaczu 7-segmentowym,
    - wolno stosować dowolne elementy dostępne w bibliotekach Xilinx-a

Automat Mealy'ego



Alfabet wejściowy:
z_1 - na wejściu dostarczamy 1
z_0 - na wejściu dostarczamy 0

Alfabet wyjściowy:
y_1 - na wyjściu otrzymujemy 1, co oznacza, że sekwencja została wykryta
y_0 - na wyjściu otrzymujemy 0, co oznacza, że sekwencja nie została wykryta

Stany wewnętrzne
q_0 - stan początkowy
q_1 - stan do którego przechodzimy po wykryciu pierwszej w sekwencji jedynki
q_2 - stan do którego przechodzimy po wykryciu drugiej w sekwencji jedynki
q_3 - stan do którego przechodzimy po wykryciu pierwszego w sekwencji zera
q_4 - stan do którego przechodzimy po wykryciu trzeciej w sekwencji jedynki
q_5 - stan do którego przechodzimy po wykryciu ostatniej w sekwencji jedynki

Zakodowane tabele prawdy

Tabele prawdy zostały zakodowane na podstawie powyższego automatu i przedstawiają kolejne kroki w zależności od wejścia *Z*.

Q(t)				Q(t+1)			JK					
Z	**Q_2**	**Q_1**	**Q_0**	**Q_2**	**Q_1**	**Q_0**	**J_2**	**K_2**	**J_1**	**K_1**	**J_0**	**K_0**
0	0	0	0	0	0	0	0	-	0	-	0	-
1	0	0	0	0	0	1	0	-	0	-	1	-
0	0	0	1	0	0	0	0	-	0	-	-	1
1	0	0	1	0	1	0	0	-	1	-	-	1
0	0	1	0	0	1	1	0	-	-	0	1	-
1	0	1	0	0	0	0	0	-	-	1	0	-
0	0	1	1	0	0	0	0	-	-	1	-	1
1	0	1	1	1	0	0	1	-	-	1	-	1
0	1	0	0	0	0	0	-	1	0	-	0	-
1	1	0	0	1	0	1	-	0	0	-	1	-
0	1	0	1	0	0	0	-	1	0	-	-	1
1	1	0	1	0	0	1	-	1	0	-	-	0

Z	**Q2**	**Q1**	**Q0**	**y_i**	**Y**
0	0	0	0	y_0	0
1	0	0	0	y_0	0
0	0	0	1	y_0	0
1	0	0	1	y_0	0
0	0	1	0	y_0	0
1	0	1	0	y_0	0
0	0	1	1	y_0	0
1	0	1	1	y_0	0
0	1	0	0	y_0	0
1	1	0	0	y_0	0
0	1	0	1	y_1	1
1	1	0	1	y_1	1

Minimalizacja siatkami Karnaugh

J_2

Q2Q1/Q0Z	00	01	11	10
00	0	0	0	0
01	-	-	1	-
11	-	-	-	-
10	0	0	0	0

K_2

Q2Q1/Q0Z	00	01	11	10
00	-	-	-	-
01	-	-	-	-
11	-	-	-	-
10	1	0	1	1

J_1

Q2Q1/Q0Z	00	01	11	10
00	0	0	1	0
01	-	-	-	-
11	-	-	-	-
10	0	0	0	0

K_1

Q2Q1/Q0Z	00	01	11	10
00	-	-	-	-
01	0	1	1	1
11	-	-	-	-
10	-	-	-	-

J_0

Q2Q1/Q0Z	00	01	11	10
00	0	1	-	-
01	1	0	-	-
11	-	-	-	-
10	0	1	-	-

K_0

Q2Q1/Q0Z	00	01	11	10
00	-	-	1	1
01	-	-	1	1
11	-	-	-	-
10	-	-	0	1

$$J_2 = Q_1 Q_0 Z$$

$$K_2 = \bar{Z} + Q_0$$

$$J_1 = \overline{Q_2} Q_0 Z$$

$$K_1 = Z + Q_0$$

$$J_0 = Q_1 \bar{Z} + \overline{Q_1} Z$$

$$K_0 = \overline{Q_2} + \bar{Z}$$

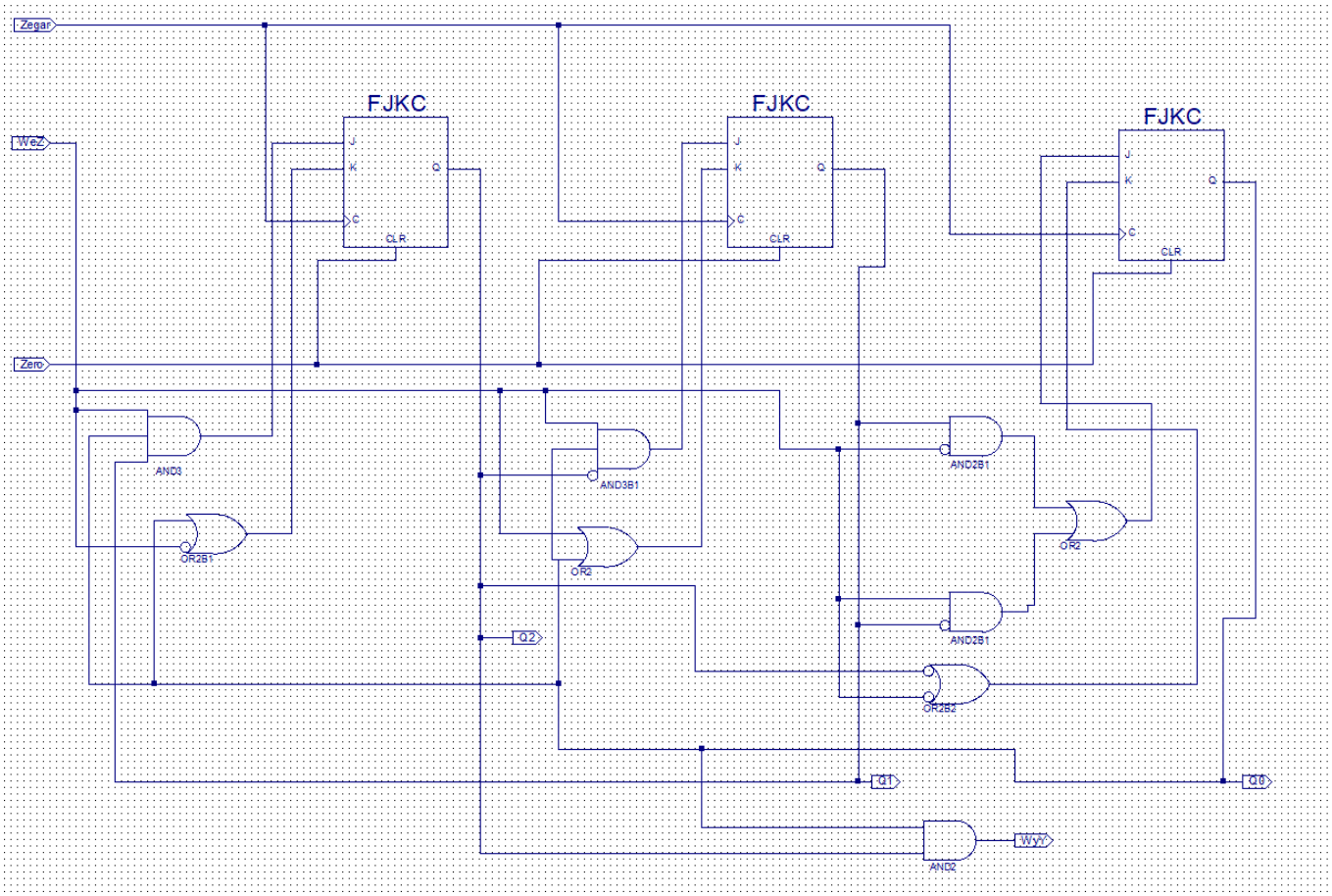
Y

Q2Q1/Q0Z	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	-	-	-	-
10	0	0	1	1

$$Y = Q_2 Q_0$$

Schemat

Na schemacie użyto trzech bramek JK, a dokładniej FJKC, które zaczynają swoje działanie od stanu 0. Są trzy wejścia – *Zegar*, *zero* i *WeZ*. Zegar jak wskazuje nazwa jest naszym zegarem, *zero* jest po prostu cały czas zerem i nie zmienia swojego stanu, natomiast *WeZ* jest naszym przyszłym przyciskiem.Wyjścia są cztery – Q_2, Q_1, Q_0 oraz Wy. Wyjścia (poza *Wy*) są stanami przełączników JK, natomiast *Wy* mówi nam czy sekwencja 11011 została wyłapana czy nie - odpowiednio stan 1 oraz 0.



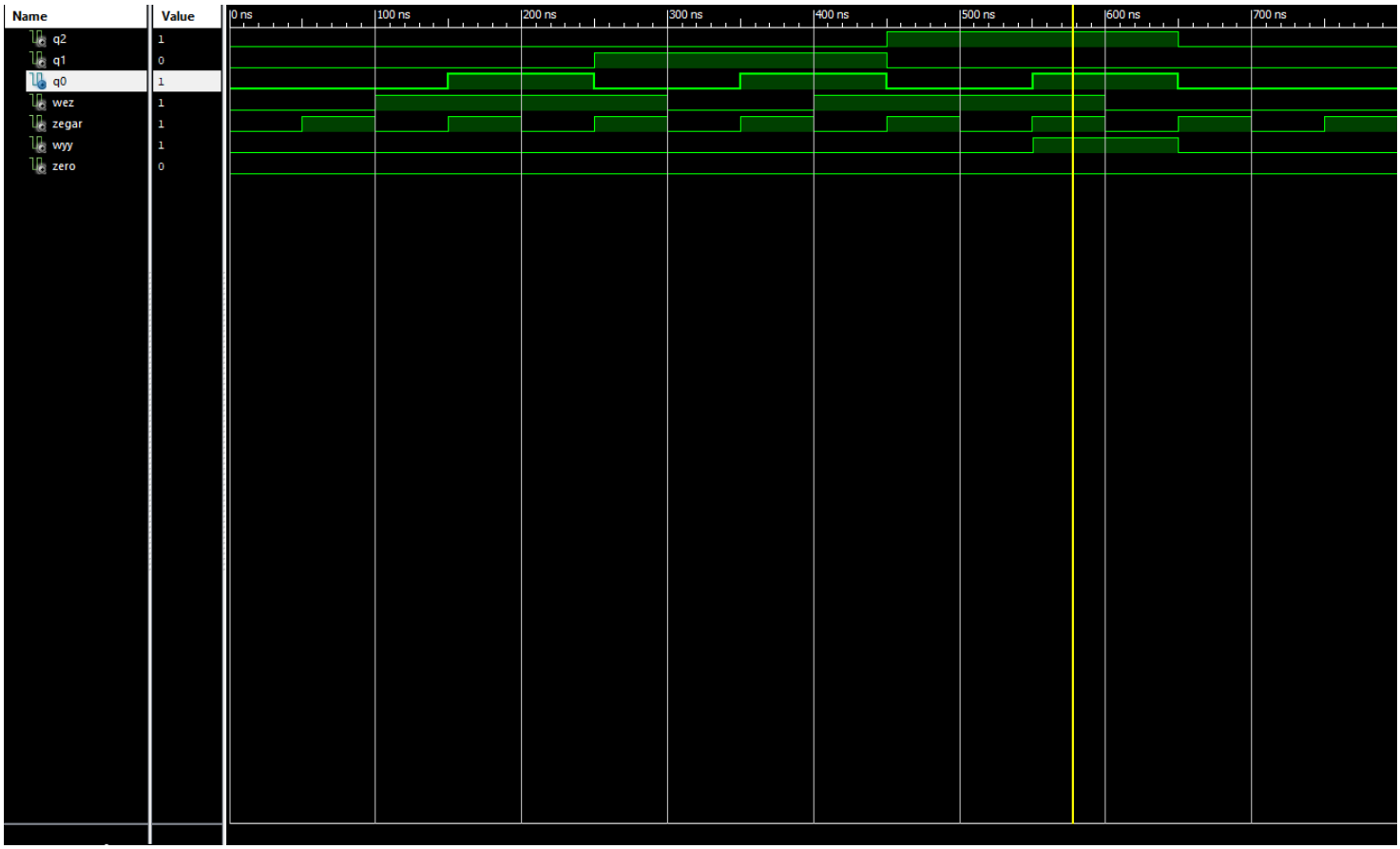
Kod VHDL

Aby poprawnie zasymulować działanie detektora sekwencji trzeba było odpowiednio zmieniać stany naszego wejścia oraz ustawić zegar, aby odpowiednio odmierzał impulsy. Nasze wejście *WeZ* zakodowaliśmy, aby co 100ns zmieniało swój stan w taki sposób, by powstała nasza sekwencja, która ma zostać wykryta. Jest to 100ns, ponieważ pełny takt zegarowy trwa właśnie 100ns i przy mniejszym czasie symulacja nie odbyłaby się prawidłowo. Screenshot kodu został wykonany już w *Visual Studio Code* zamiast w *Xillinxie*, ponieważ na zajęciach z pośpiechu go nie zrobiliśmy, a na swoich sprzętach mieliśmy problemy z instalacją *Xillinxa*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY Schem_Schem_sch_tb IS
END Schem_Schem_sch_tb;
ARCHITECTURE behavioral OF Schem_Schem_sch_tb IS

   COMPONENT Schem
   PORT( Q2 :  OUT   STD_LOGIC;
         Q1   :  OUT   STD_LOGIC;
         Q0   :  OUT   STD_LOGIC;
         Zero :  IN STD_LOGIC;
         Zegar   :  IN STD_LOGIC;
         WeZ  :  IN STD_LOGIC);
   END COMPONENT;

   SIGNAL Q2   :  STD_LOGIC;
   SIGNAL Q1   :  STD_LOGIC;
   SIGNAL Q0   :  STD_LOGIC;
   SIGNAL Zero :  STD_LOGIC;
   SIGNAL Zegar   :  STD_LOGIC:='0';
   SIGNAL WeZ  :  STD_LOGIC;

BEGIN

   UUT: Schem PORT MAP(
      Q2 => Q2,
      Q1 => Q1,
      Q0 => Q0,
      Zero => Zero,
      Zegar => Zegar,
      WeZ => WeZ
   );

   Zero <= '0';
   WeZ <= '0', '1' after 100ns, '0' after 300ns, '1' after 400ns, '0' after 600ns;
   Zegar <= not Zegar after 50ns;

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Przebieg symulacji

Dzięki użytym przerzutnikom FJKC nasze stany Q zaczynają się od 0. Następnie zegar zaczyna działać i przez kolejne 200ns na wejściu *WeZ* jest stan 1 - ponieważ stany mają się zmieniać co 100ns, a są pod rząd dwa razy stany 1. Po powstałej sekwencji widać, że nasze wyjście *WyY* zmieniło się na 1 (wskazuje na to miejsce żółta linia), co oznacza, że detektor zadziałał prawidłowo. Po zakończonej sekwencji wyjście *WyY* znów zmienia się na zero.



Testowanie na płytce

Układ udało nam się przetestować na płytce. Najpierw musieliśmy odpowiednio skonfigurować plik UCF - nasz zegar ustawiliśmy na czas 5ms, abyśmy byli w stanie cokolwiek zobaczyć na płytce. Nasze wejścia zostały zakodowane na przyciskach 0 (*WeZ)* oraz 1 (*Zero*), przy czym naszego 1 przycisku nie używaliśmy wcale, ponieważ potrzebny był nam stan zero tylko i wyłącznie do odpowiedniego połączenia przerzutników. Nasz wyjścia zostały odpowiednio przypisane do ledów 0 (Q_0), 1 (Q_1), 2 (Q_2) oraz 7 (*WyY*). Wyjście *WyY* oddzieliliśmy od wyjść z rodziny Q, aby było czytelniej. Nasz układ działał zgodnie z oczekiwaniami - po wprowadzeniu sekwencji 11011 nasza lampka *WyY* zgasła (na płytce diody led się świecą przy stanie 0 i gasną przy stanie 1).

```
# Clocks
NET "Zegar" LOC = "P7" | BUFG = CLK | PERIOD = 5ms HIGH 50%;

#NET "Clk_XT" LOC = "P5" | BUFG = CLK | PERIOD = 500ns HIGH 50%;

# Keys
NET "WeZ" LOC = "P42";
NET "Zero" LOC = "P40";
#NET "Key<2>" LOC = "P43";
#NET "Key<3>" LOC = "P38";
#NET "Key<4>" LOC = "P37";
#NET "Key<5>" LOC = "P36";  # shared with ROT_A
#NET "Key<6>" LOC = "P24";  # shared with ROT_B
#NET "Key<7>" LOC = "P39";  # GSR

# LEDS
NET "Q0"  LOC = "P35";
NET "Q1"  LOC = "P29";
NET "Q2"  LOC = "P33";
#NET "LED<3>"  LOC = "P34";
#NET "LED<4>"  LOC = "P28";
#NET "LED<5>"  LOC = "P27";
#NET "LED<6>"  LOC = "P26";
NET "WyY"  LOC = "P25";

#NET "LED<8>"  LOC = "P13"; # shared with seg. B
#NET "LED<9>"  LOC = "P11"; # shared with seg. F
#NET "LED<10>" LOC = "P12"; # shared with seg. A
#NET "LED<11>" LOC = "P18"; # shared with seg. DP
#NET "LED<12>" LOC = "P22"; # shared with seg. C
#NET "LED<13>" LOC = "P20"; # shared with seg. G
#NET "LED<14>" LOC = "P19"; # shared with seg. D
#NET "LED<15>" LOC = "P14"; # shared with seg. E
```

Wnioski

Zadania nie sprawiły większych problemów oraz zostały w pełni wykonane i przetestowane na zajęciach. Po raz kolejny wiedza zdobyta na kursie *Logika układów cyfrowych* okazała się niezbędna do ich wykonania. Napotkaliśmy małe problemy związane z minimalizacją układów - w dwóch miejscach błędnie zapisaliśmy wzory, przez co musieliśmy po raz kolejny dokładnie wszystko przeanalizować - błąd się znalazł i mogliśmy ruszyć do przodu. Przy pierwszym zadaniu trudności nie napotkaliśmy - na poprzednie zajęcia musieliśmy przygotować się z kodu Aikena, więc licznik w tym kodzie nie sprawił nam trudności. Przy zadaniu numer dwa podczas testowania mieliśmy na początku trudność, aby ,,wbić się" w rytm zegara i odpowiednio przetestować detektor, co zajęło nam sporo czasu, ale ostatecznie nauczyliśmy się jak to robić odpowiednio i wtedy nasz układ działał zgodnie z oczekiwaniami.