# Układy cyfrowe i systemy wbudowane

## SPRAWOZDANIE NR 5

Układy sekwencyjne w VHDL

**PROWADZĄCY:**

DR INŻ. JACEK MAZURKIEWICZ

**AUTORZY:**

MATEUSZ GRUSZKA 249448

BARTOSZ RUDNIK 248893

**Grupa:** C

**TERMIN ZAJĘĆ:**

WT 9:15 TP

WROCŁAW, 19.01.2021 r.

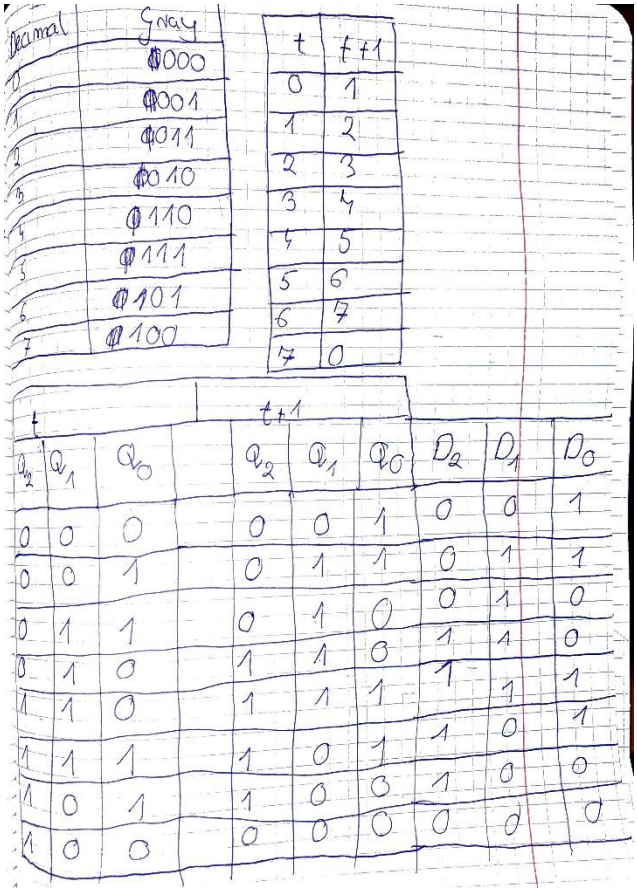
# 1. Cel ćwiczenia

**1.1** Licznik synchroniczny o zadanych parametrach funkcjonalnych.

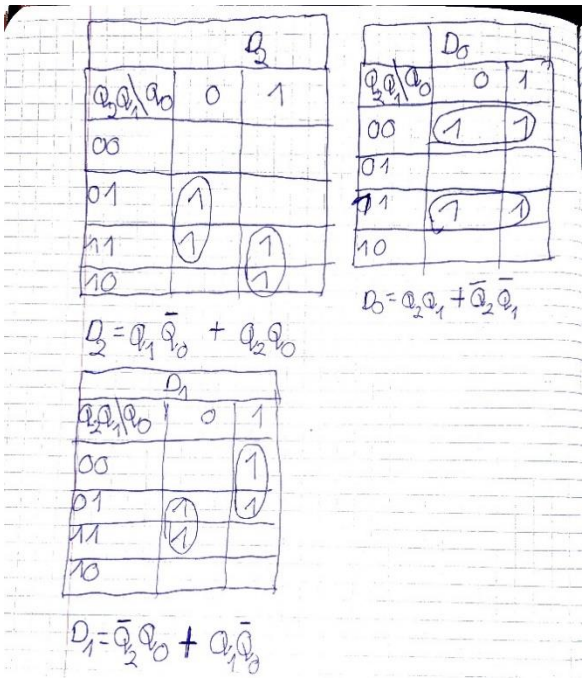
**1.2** Detektor sekwencji bitowej opisany wskazanym typem automatu, jedno wejście, jedno wyjście, brak resetu, sekwencja prawidłowa 5-bitowa. Każde zadanie realizujemy w VHDL-u: - jako maszynę stanów.

# 2. Licznik synchroniczny

Wykonywany przez nas licznik był licznikiem synchronicznym mod7, pozytywnym działającym w kodzie Graya. Licznik ten został stworzony w VHDL jako maszyna stanów. Jako, że licznik ten ma za zadanie działać jako licznik mod7 stworzonych zostało 7 stanów reprezentujących wartości w kodzie Graya jakie mogą być zwrócone po wykonaniu operacji mod7. Do każdego ze stanów przypisane zostały sygnały wyjściowe, które są zwracane w przypadku osiągnięcia odpowiadającego im stanu przez zaprojektowany układ. Oprócz pliku VHDL, w którym zrealizowaliśmy zadaną maszynę stanów stworzyliśmy także plik testowy umożliwiający przetestowanie tego układu w symulacji środowisku Xilinx ISE.




***Rysunek 1*** *Licznik synchroniczny – tabela prawdy*




***Rysunek 2*** *Licznik synchroniczny – minimalizacja*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity liczn_masz_stanow is
5      Port ( clk : in  STD_LOGIC;
6             output : out  STD_LOGIC;
7             output1 : out STD_LOGIC;
8             output2 : out STD_LOGIC);
9  end liczn_masz_stanow;
10
11 architecture ms of liczn_masz_stanow is
12
13    type state_type is (A, B, C, D, E, F, G);
14    signal state, next_state : state_type;
15    Signal W : STD_LOGIC_VECTOR (2 downto 0);
16
17 begin
18
19    process1 : process (clk)
20    begin
21       if rising_edge(clk) then
22          state <= next_state;
23       end if;
24    end process process1;
25
26    process2 : process(state)
27    begin
28       next_state <= state;
29
30       case state is
31          when A =>
32             next_state <= B;
33          when B =>
34             next_state <= C;
35          when C =>
36             next_state <= D;
37          when D =>
38             next_state <= E;
39          when E =>
40             next_state <= F;
41          when F =>
42             next_state <= G;
43          when G =>
44             next_state <= A;
45       end case;
```

***Rysunek 3*** *Licznik synchroniczny – kod źródłowy cz.1*

```
46      end process process2;
47
48      process3: process(state)
49      begin
50
51          case state is
52              when A =>
53                  W <= "000";
54              when B =>
55                  W <= "001";
56              when C =>
57                  W <= "011";
58              when D =>
59                  W <= "010";
60              when E =>
61                  W <= "110";
62              when F =>
63                  W <= "111";
64              when G =>
65                  W <= "101";
66          end case;
67      end process process3;
68
69      output <= W(2);
70      output1 <= W(1);
71      output2 <= W(0);
72
73  end ms;
74
```

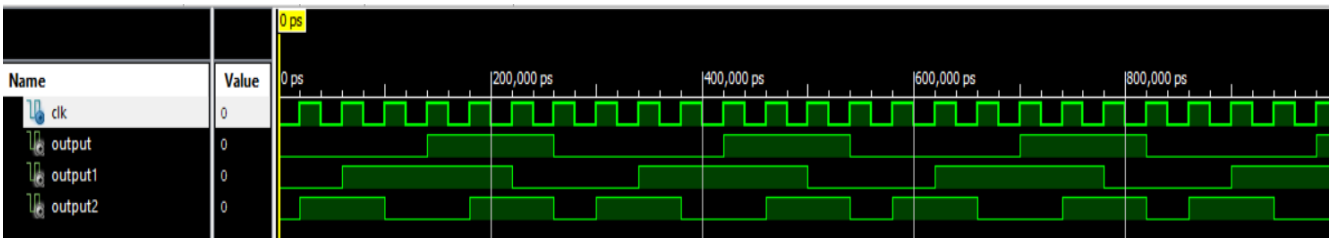
***Rysunek 4*** *Licznik synchroniczny – kod źródłowy cz.2*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY licznik_test_maszyna IS
END licznik_test_maszyna;

ARCHITECTURE behavior OF licznik_test_maszyna IS

    COMPONENT liczn_masz_stanow
    PORT(
         clk : IN  std_logic;
         output : OUT  std_logic;
         output1 : OUT  std_logic;
         output2 : OUT  std_logic
        );
    END COMPONENT;

   signal clk : std_logic := '0';

   signal output : std_logic;
   signal output1 : std_logic;
   signal output2 : std_logic;

BEGIN

   uut: liczn_masz_stanow PORT MAP (
          clk => clk,
          output => output,
          output1 => output1,
          output2 => output2
        );

   clk <= not clk after 40 ns;

END;
```

***Rysunek 5*** *Licznik synchroniczny – kod źródłowy cz.3*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity new_funckja_tablica is
5      Port ( w, x, y, z : in  STD_LOGIC;
6             f : out  STD_LOGIC);
7  end new_funckja_tablica;
8
9  architecture Behavioral of new_funckja_tablica is
10
11 signal S : STD_LOGIC_VECTOR (3 downto 0);
12
13 begin
14
15 S <= w & x & y & z;
16
17 with S Select
18    f <= '1' when "0000",
19         '1' when "0001",
20         '0' when "0010",
21         '1' when "0011",
22         '0' when "0100",
23         '1' when "0101",
24         '0' when "0110",
25         '1' when "0111",
26         '1' when "1000",
27         '0' when "1001",
28         '1' when "1010",
29         '0' when "1011",
30         '1' when "1100",
31         '1' when "1101",
32         '1' when "1110",
33         '1' when "1111",
34         'X' when others;
35 end Behavioral;
```

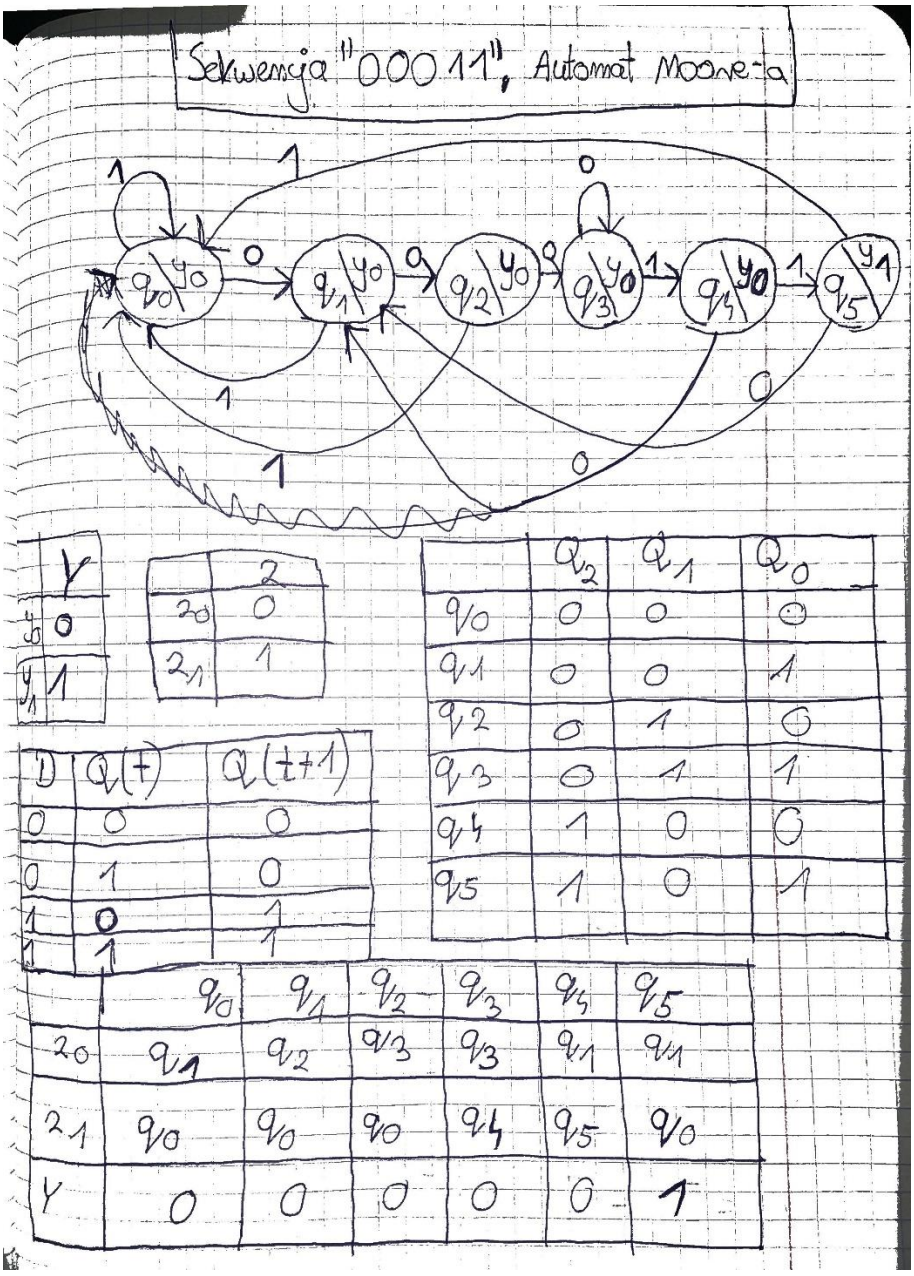
***Rysunek 6*** *Licznik synchroniczny – kod źródłowy cz.4*



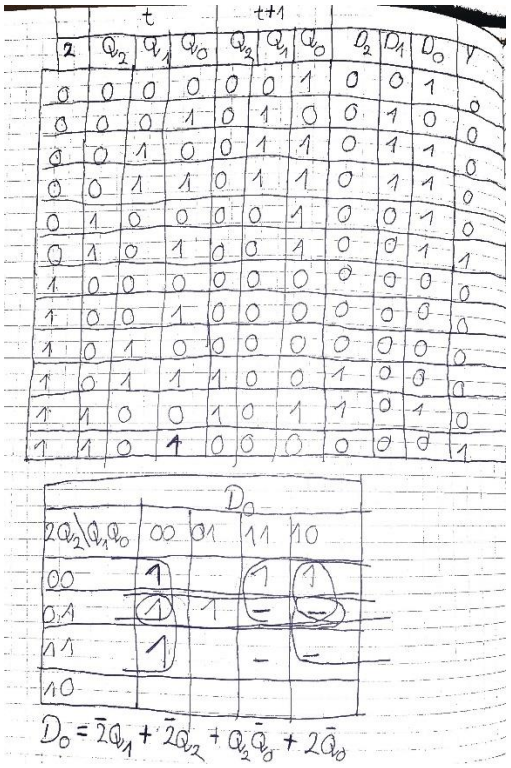

**Rysunek 7** *Licznik synchroniczny - symulacja*

# 3. Detektor sekwencji bitowej

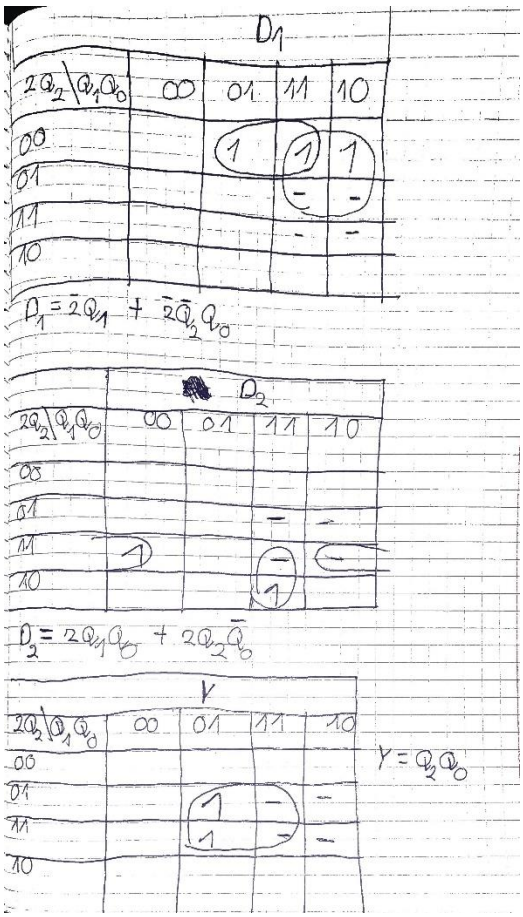
Drugim wykonanym przez nas zadaniem podczas zajęć laboratoryjnych był detektor sekwencji bitowej. W naszym przypadku detektor miał za zadanie wykryć sekwencję 00011 i być zrealizowany jako automat Moore'a. Detektor podobnie jak w przypadku poprzedniego zadania zrealizowany miał być w postaci maszyny stanów. W tym celu zdefiniowaliśmy 6 stanów odpowiadających kolejnym przejściom jakie mogły zostać wykonane przez detektor. Każdy ze zdefiniowanych przez nas stanów miał przypisany odpowiadający mu sygnał wyjściowy. Dla stanu opisującego wykrycie zadanej sekwencji bitowej zwracany był sygnał wysoki, dla pozostałych stanów opisujących sytuację nie wykrycia zadanego ciągu bitowego był to sygnał niski. Ponieważ stworzony przez nas układ miał działać jako automat Moore'a to sygnał wyjściowy zależny był wyłącznie od stanu, w którym znajdował się układ.




***Rysunek 8*** *Detektor sekwencji bitowej – tabele prawdy cz.1*



***Rysunek 9*** *Detektor sekwencji bitowej – tabele prawdy cz.2*



***Rysunek 10*** *Detektor sekwencji bitowej – minimalizacja*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vhdl_detektor is
    Port ( z : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           y : out  STD_LOGIC);
end vhdl_detektor;

architecture Behavioral of vhdl_detektor is

type state_type is (A, B, C, D, E, F);
signal state, next_state : state_type;

begin

   process1 : process(clk)
   begin
      if rising_edge(clk) then
         state <= next_state;
      end if;
   end process process1;

   process2 : process(state, z)
   begin
      next_state <= state;

      case state is
         when A =>
            if z = '0' then
               next_state <= B;
            end if;
         when B =>
            if z = '0' then
               next_state <= C;
            else
               next_state <= A;
            end if;
         when C =>
            if z = '0' then
               next_state <= D;
            else
               next_state <= A;
            end if;
         when D =>
```

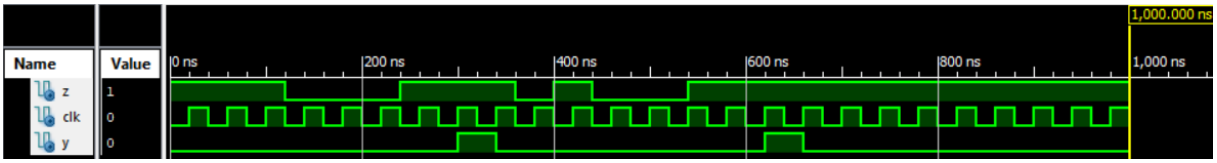
***Rysunek 11*** *Detektor sekwencji bitowej – kod źródłowy cz.1*

```
45          when D =>
46             if z = '1' then
47                next_state <= E;
48             end if;
49          when E =>
50             if z = '1' then
51                next_state <= F;
52             else
53                next_state <= B;
54             end if;
55          when F =>
56             if z = '1' then
57                next_state <= A;
58             else
59                next_state <= B;
60             end if;
61       end case;
62    end process process2;
63
64    process3: process(state)
65    begin
66
67       case state is
68          when A =>
69             y <= '0';
70          when B =>
71             y <= '0';
72          when C =>
73             y <= '0';
74          when D =>
75             y <= '0';
76          when E =>
77             y <= '0';
78          when F =>
79             y <= '1';
80       end case;
81    end process process3;
82
83 end Behavioral;
84
```

***Rysunek 12*** *Detektor sekwencji bitowej – kod źródłowy cz.2*

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY testBench_sekwencja_maszyna IS
END testBench_sekwencja_maszyna;

ARCHITECTURE behavior OF testBench_sekwencja_maszyna IS

    COMPONENT vhdl_detektor
    PORT(
         z : IN  std_logic;
         clk : IN  std_logic;
         y : OUT  std_logic
        );
    END COMPONENT;

   signal z : std_logic := '0';
   signal clk : std_logic := '0';

   signal y : std_logic;

BEGIN

   uut: vhdl_detektor PORT MAP (
          z => z,
          clk => clk,
          y => y
        );

   clk <= not clk after 20 ns;
   z <= '1', '0' after 120 ns, '1' after 240 ns, '0' after 360 ns, '1' after 400 ns, '0' after 440 ns, '1

END;

```

***Rysunek 13*** *Detektor sekwencji bitowej – kod źródłowy cz.3*




***Rysunek 14*** *Detektor sekwencji bitowej – symulacja*

# 4. Wnioski

Zrealizowanie powyższych zadań umożliwiło nam lepsze zrozumienie języka VHDL, jego zawiłości oraz możliwości kryjących się w tej technologii. Realizowaliśmy zadania, których stronę techniczną poznaliśmy na wcześniejszych zajęciach. Podczas realizacji tego ćwiczenia mogliśmy wykonać przedstawione nam zadania w postaci maszyny stanów będącej bardzo sprawnym i wygodnym sposobem dla zaimplementowania układów sekwencyjnych. Wraz z kolejnymi laboratoriami nabywamy obycie ze środowiskiem Xilinx ISE oraz nabieramy automatyzmów w podstawowych czynnościach związanych z testowaniem zawartego kodu, czy wyszukiwaniem błędów powstałych w skutek naszych pomyłek. Każde laboratorium przybliża nas do sprawnego używania nauczanej technologii oraz umożliwia rozwijanie umiejętności poprzez praktykę.