Sprawozdanie - laboratorium 5

Maksim Birszel, Piotr Karoń

12 stycznia 2020

1 Zadanie

Zaprojektować moduł, zasymulować i zaimplementować na płycie ZL-9572 zamek szyfrowy, otwierający się na podaną sekwencję podaną z podłączonej klawiatury -> PKMB.

2 Założenia

- Projekt ma zawierać tylko 1 moduł VHDL
- Symulacja dla ciągu: $x = **PKMB**PKMPKMB**$
- wektor 17-bitowy oraz 13 stanów wewnętrznych
- Sprzęt: CLK_XF, PS2_Clk, PS2_Data, RST->K7, y->LED0

3 Moduł VHDL

```
1  entity zamak_szyfrowy_modul is
2      Port (
3              DO_Rdy : in  STD_LOGIC;
4                       DO : in STD_LOGIC_VECTOR(7 downto 0);
5              CLK_XT : in  STD_LOGIC;
6              RST : in  STD_LOGIC;
7              y : out  STD_LOGIC);
8  end zamak_szyfrowy_modul;
9
10 architecture Behavioral of zamak_szyfrowy_modul is
11
12 type state_type is (q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12);
13 signal state, next_state : state_type;
14
15 begin
16
```

```
17 SYNC_PROC: process(CLK_XT)
18 begin
19         if (rising_edge(CLK_XT)) then
20          if (RST = '1') then
21             state <= q0;
22          elsif (DO_Rdy ='1') then
23             state <= next_state;
24          end if;
25    end if;
26 end process;
27
28 OUTPUT_DECODE: process (state)
29 begin
30         if state = q12 then
31                 Y <= '1';
32         else
33                 Y <= '0';
34         end if;
35 end process;
36
37 NEXT_STATE_DECODE: process (state, DO)
38 begin
39         case (state) is
40                 when q0 =>
41                         if DO = X"4D" then
42                                 next_state <= q1;
43                         else next_state <= q0;
44                         end if;
45
46                 when q1 =>
47                         if DO = X"F0" then
48                                 next_state <= q2;
49                         else next_state <= q0;
50                         end if;
51
52                 when q2 =>
53                         if DO = X"4D" then
54                                 next_state <= q3;
55                         else next_state <= q0;
56                         end if;
57
58                 when q3 =>
59                         if DO = X"42" then
60                                 next_state <= q4;
61                         elsif DO = X"4D" then
```

```
62                              next_state <= q1;
63                      else next_state <= q0;
64                      end if;
65
66              when q4 =>
67                      if DO = X"F0" then
68                              next_state <= q5;
69                      else next_state <= q0;
70                      end if;
71
72              when q5 =>
73                      if DO = X"42" then
74                              next_state <= q6;
75                      else next_state <= q0;
76                      end if;
77
78              when q6 =>
79                      if DO = X"3A" then
80                              next_state <= q7;
81                      elsif DO = X"4D" then
82                              next_state <= q1;
83                      else next_state <= q0;
84                      end if;
85
86              when q7 =>
87                      if DO = X"F0" then
88                              next_state <= q8;
89                      else next_state <= q0;
90                      end if;
91
92              when q8 =>
93                      if DO = X"3A" then
94                              next_state <= q9;
95                      else next_state <= q0;
96                      end if;
97
98              when q9 =>
99                      if DO = X"32" then
100                             next_state <= q10;
101                     elsif DO = X"4D" then
102                             next_state <= q1;
103                     else next_state <= q0;
104                     end if;
105
106             when q10 =>
```

```
107                         if DO = X"F0" then
108                                 next_state <= q11;
109                         else next_state <= q0;
110                         end if;
111
112                 when q11 =>
113                         if DO = X"32" then
114                                 next_state <= q12;
115                         else next_state <= q0;
116                         end if;
117
118                 when q12 =>
119                         if DO = X"4D" then
120                                 next_state <= q1;
121                         else next_state <= q0;
122                         end if;
123
124         end case;
125 end process;
126
127 end Behavioral;
```

4 Test Bench

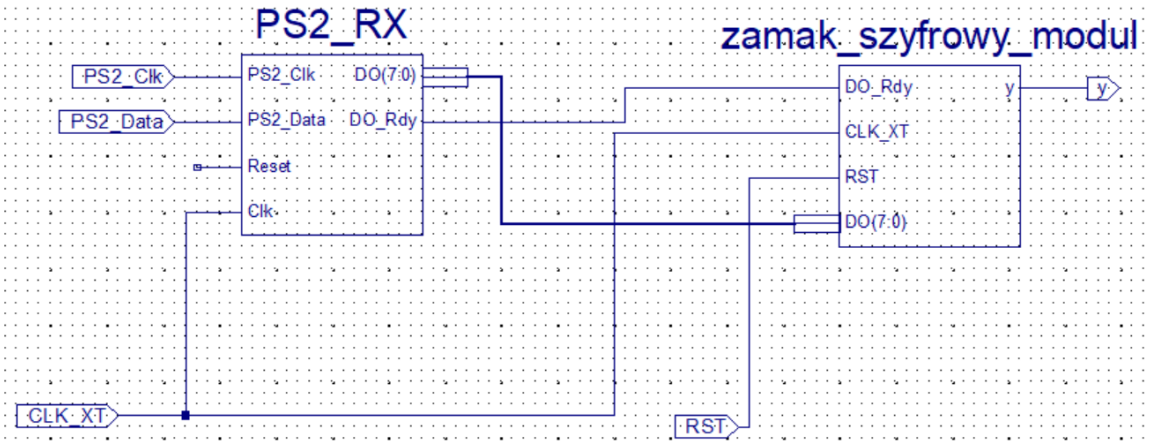
```
1  ENTITY zamek_szyfrowy_testbench IS
2  END zamek_szyfrowy_testbench;
3
4  ARCHITECTURE behavior OF zamek_szyfrowy_testbench IS
5
6      COMPONENT zamak_szyfrowy_modul
7      PORT(
8           Do_rdy : IN  std_logic;
9                        DO : in STD_LOGIC_VECTOR(7 downto 0);
10          CLK_XT : IN  std_logic;
11          RST : IN  std_logic;
12          y : OUT  std_logic
13         );
14     END COMPONENT;
15
16     signal DO : std_logic_vector(7 downto 0);
17     signal Do_rdy : std_logic := '0';
18     signal CLK_XT : std_logic := '0';
19     signal RST : std_logic := '0';
20
21     signal y : std_logic;
22
```

```
23     constant CLK_XT_period : time := 10 ns;
24
25 BEGIN
26
27     uut: zamak_szyfrowy_modul PORT MAP (
28             DO => DO,
29             Do_rdy => Do_rdy,
30             CLK_XT => CLK_XT,
31             RST => RST,
32             y => y
33           );
34
35     CLK_XT_process : process
36     begin
37                     CLK_XT <= '0';
38                     wait for CLK_XT_period/2;
39                     CLK_XT <= '1';
40                     wait for CLK_XT_period/2;
41     end process;
42
43           symulacja : process
44
45 type typeByteArray is array (NATURAL range <>) of std_logic_vector (7 downto
46 variable wektor : typeByteArray (0 to 16) := (X"00",X"00",X"4D",X"42",X"3A",
47 X"32",X"00",X"00",X"4D",X"42",X"3A",X"4D",X"42",X"3A",X"32",X"00",X"00");
48
49           begin
50                     wait for 10 ns;
51
52                     for i in 0 to 17 loop
53                             wait for 10 ns;
54                             DO <= wektor(i);
55                             Do_rdy <= '1';
56                             wait for 10 ns;
57
58                             Do_rdy <= '0';
59                             wait for 80 ns;
60
61                             DO <= X"F0";
62                             Do_rdy <= '1';
63                             wait for 10 ns;
64
65                             Do_rdy <= '0';
66                             wait for 40 ns;
67
```

```
68                                  DO <= wektor(i);
69                                  Do_rdy <= '1';
70                                  wait for 10 ns;
71
72                                  Do_rdy <= '0';
73                                  wait for 110 ns;
74
75                          end loop;
76                  end process;
77  END;
```

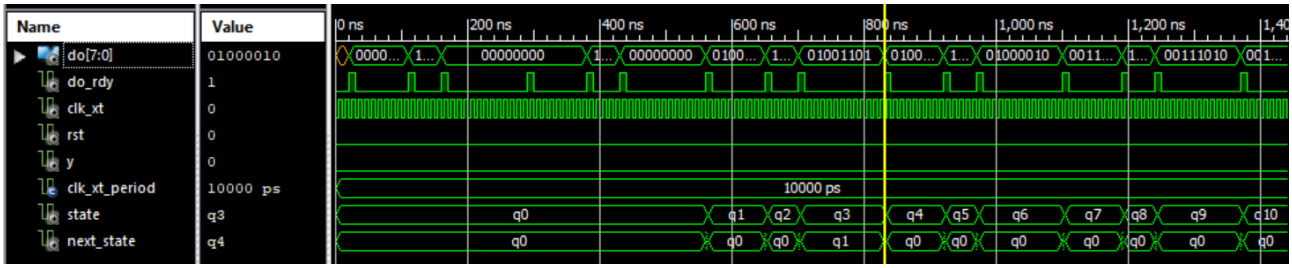
5 Schemat

Po utworzeniu zaimplementowanego modułu i połączeniu go z modułem PS2_RX pobranym ze strony całość wygląda następująco:

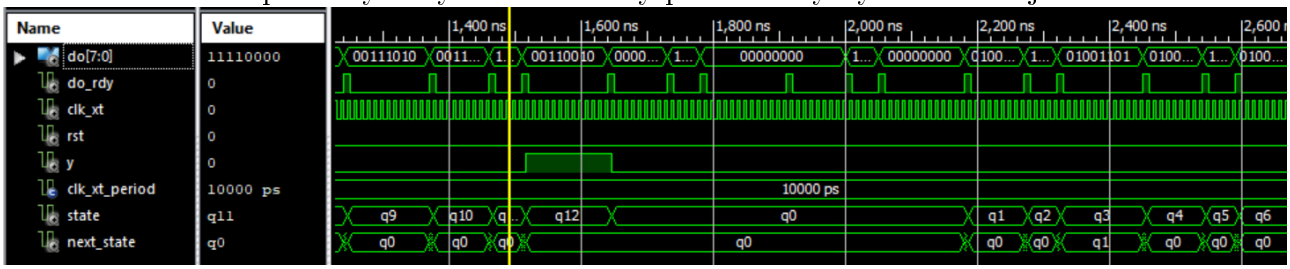


6 Symulacja behawioralna

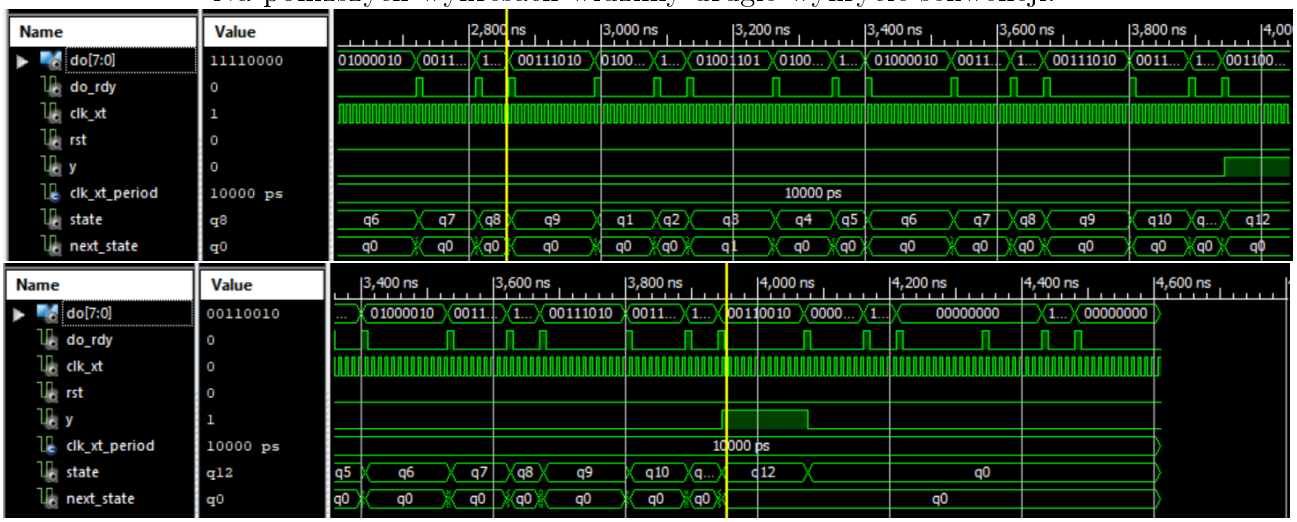
Symulacja została podzielona na cztery części, aby łatwiej było odczytać poszczególne wartości.



Na poniższym wykresie widzimy pierwsze wykrycie sekwencji.



Na poniższych wykresach widzimy drugie wykrycie sekwencji.



7 Wnioski

Moduł udało się poprawnie zaimplementować. Symulacja przebiegła według wstępnych założeń.
Najwięcej problemów sprawiło poprawne dobranie okresów czasowych w symulacji, aby stan zegara zmieniał się w odpowiednich momentach.
Początkowo moduł, nie działał poprawnie w przypadku, gdy dostawał pierwsze 3 litery wymaganej sekwencji, a 4 z nich była błędna. W takim wypadku zerował się i nie sprawdzał kolejnej podanej litery. Z tego powodu sekwencja podana w symulacji: PKMPKMB nie otwierała zamku. Aby to naprawić, musieliśmy dodać w poszczególnych stanach dodatkowe instrukcje ęlsif".
Po tych korektach moduł działał poprawnie zarówno podczas symulacji, jak i na płytce.