Układy Cyfrowe i Systemy Wbudowane 1
Laboratorium 5

Rozbudowane projekty łączące maszyny stanów oraz moduły WE/WY.

Paweł Cyganiuk 263983, Patryk Jurkiewicz 263896

11.12.2023

1. Treść Zadania

Zamek szyfrowy otwierany kombinacją 4 klawiszy

(inicjały tj. PCPJ)

- symulacja behawioralna: wszystkie 13 stanów oraz zaprezentowanie ich poprzez sekwencje:

  PCPJPCPPCPJ**

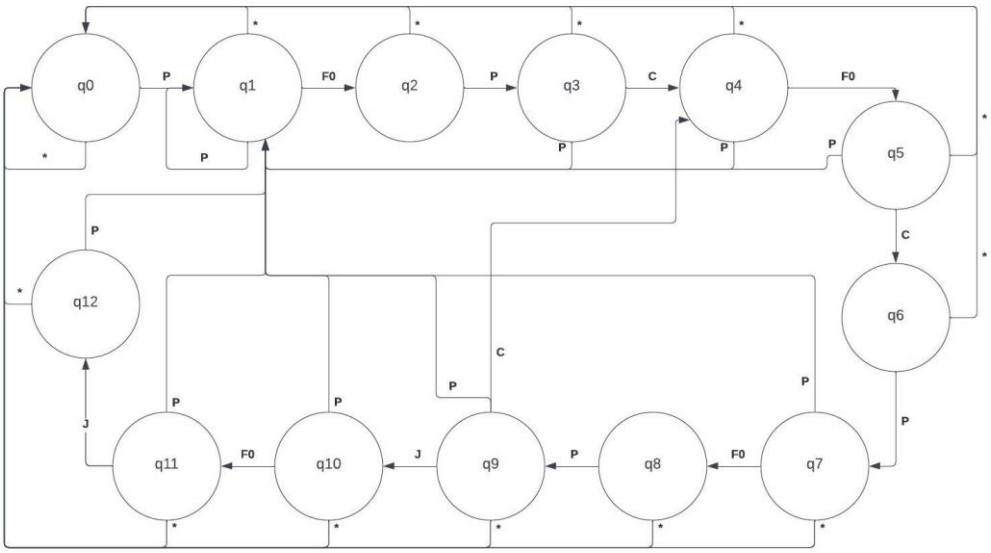
- Symulacja modułu Zamek: ciąg naciskanych klawiszy za pomocą wektora z 17 kodami klawiszy
- Śledzenie wewnętrznego sygnału „state”

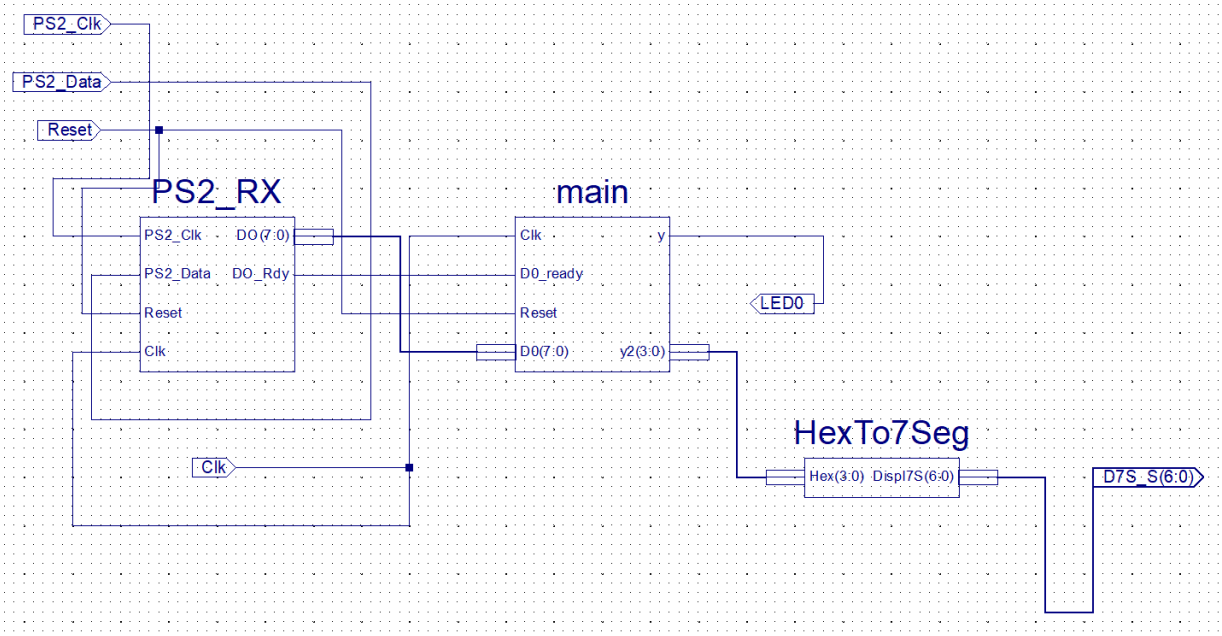
2. Założenia

Rozpoczęliśmy realizację ćwiczenia od precyzyjnego zdefiniowania 13 stanów, oznaczonych jako q0 do q12, które będą reprezentowane na automacie Moora. Kluczowym stanem sygnalizującym wyjście y=1 jest q12. W ramach przygotowań, konieczne było także zdefiniowanie kodów liter inicjałów oraz dla znaku zwolnienia klawisza jako wektorów 8-bitowych, takich jak:
P: 01001101,
C: 00100001,
J: 00111011,
F0: 11110000.

Z posiadanych teoretycznych założeń przeszliśmy do etapu implementacji głównego bloku w schemacie logicznym, który następnie zostanie połączony z pozostałymi niezbędnymi blokami w celu pełnego funkcjonowania programu. Ten etap stanowił kluczowy krok w procesie realizacji zadania, umożliwiając nam przeniesienie teorii w praktyczne działanie całego systemu.

3. Graf i schemat logiczny





4. Kod w języku VHDL

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  library UNISIM;
23  use UNISIM.VComponents.all;
24
25  use IEEE.NUMERIC_STD.ALL;
26
27  entity main is
28      Port ( Clk : in  STD_LOGIC;
29             D0 : in STD_LOGIC_VECTOR(7 downto 0);
30             D0_ready : in STD_LOGIC;
31             y : out STD_LOGIC;
32             y2 : out STD_LOGIC_VECTOR (3 downto 0);
33             Reset : in STD_LOGIC);
34  end main;
35
36  architecture Behavioral of main is
37
38  type state_type is (q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12);
39
40  constant P: std_logic_vector (7 downto 0) := "01001101";
41  constant C: std_logic_vector (7 downto 0) := "00100001";
42  constant J: std_logic_vector (7 downto 0) := "00111011";
43  constant F0: std_logic_vector (7 downto 0) := "11110000";
44
45  signal state, next_state : state_type;
46
47  begin
48
49  process1 : process( Clk )
50  begin
51     if rising_edge( Clk ) then
52     if Reset = '1' then
53        state <= q0;
54     else
55        state <= next_state;
56     end if;
57     end if;
58     end process process1;
59  process2 : process( state, D0, D0_ready )
60
61  begin
62  next_state <= state;
63
64  if D0_ready = '1' then
```

W tej części kodu zdefiniowano entity o nazwie main z różnymi portami, które są kluczowe dla funkcjonowania automatu skończonego (FSM). Porty wejściowe obejmują sygnał zegara (Clk), wektor danych wejściowych o szerokości 8 bitów (D0), sygnał gotowości danych (D0_ready), oraz sygnał resetujący (Reset). Z kolei porty wyjściowe to pojedynczy sygnał logiczny (y) oraz wektor czterobitowy (y2), który posłuży do drugiego zadania, gdzie łączymy się z blokiem HexTo7Seg.

W architekturze Behavioral zdefiniowano typ wyliczeniowy state_type, reprezentujący stany automatu od q0 do q12. Zadeklarowano również stałe (P, C, J, F0) jako wektory ośmiobitowe, które stanowią ustalone wartości w trakcie działania automatu.

W pierwszym procesie (process1), który reaguje na narastające zbocze sygnału zegarowego (Clk), określono warunki aktualizacji stanu automatu (state). Jeśli sygnał resetujący (Reset) jest aktywowany, stan ustawiany jest na q0. W przeciwnym przypadku, stan przyjmuje wartość next_state.

Drugi proces (process2) reaguje na zmiany stanu, dane wejściowe (D0) oraz sygnał gotowości danych (D0_ready). Główną funkcją tego procesu jest przechowywanie aktualnego stanu (state) i aktualizowanie stanu następnego (next_state) w zależności od warunków określonych dla poszczególnych stanów automatu. Proces ten jest istotny dla realizacji logiki automatu skończonego, który reaguje na wejściowe dane i zmienia swoje stany w zależności od warunków określonych w kodzie.

```
61  begin
62  next_state <= state;
63
64  if D0_ready = '1' then
65
66  case state is
67
68  when q0 =>
69     y2 <= X"0";
70     if D0 = P then
71        next_state <= q1;
72     else
73        next_state <= q0;
74        end if;
75  when q1 =>
76  y2 <= X"1";
77     if D0 = F0 then
78        next_state <= q2;
79     elsif D0 = P then
80        next_state <= q1;
81     else
82        next_state <= q0;
83        end if;
84  when q2 =>
85  y2 <= X"2";
86     if D0 = P then
87        next_state <= q3;
88     else
89        next_state <= q0;
90        end if;
91
92  when q3 =>
93  y2 <= X"3";
94     if D0 = C then
95        next_state <= q4;
96     elsif D0 = P then
97        next_state <= q1;
98     else
99        next_state <= q0;
100       end if;
101 when q4 =>
102 y2 <= X"4";
103    if D0 = F0 then
```

```
when q11 =>
y2 <= X"B";
   if D0 = J then
      next_state <= q12;
   elsif D0 = P then
      next_state <= q1;
   else
      next_state <= q0;
      end if;
when q12 =>
y2 <= X"C";
   if D0 = P then
      next_state <= q1;
   else
      next_state <= q0;
      end if;

end case;

end if;
end process process2;

y <= '1' when state = q12 else '0';

end Behavioral;
```

W powyższej części kodu zastosowano konstrukcję case do definiowania logiki przejścia stanów w ramach automatu skończonego (FSM) zgodnie z przedstawionym grafem. Działanie automatu opiera się na analizie aktualnego stanu (state) i wektora danych wejściowych (D0). Poniżej przedstawiono zwięzłe opisy dla poszczególnych przypadków w konstrukcji case:

- Stan q0:

Sygnał wyjściowy y2 ustawiany jest na wartość X"0", co oznacza szesnastkową reprezentacje liczby ‘0’.

Jeśli dane wejściowe (D0) są równe stałej P, następuje przejście do stanu q1; w przeciwnym razie pozostaje w stanie q0.

- Stan q1:

Sygnał wyjściowy y2 ustawiany jest na wartość X"1".

Jeśli dane wejściowe (D0) równają się stałej F0, następuje przejście do stanu q2; jeśli są równe stałej P, pozostaje w stanie q1; w przeciwnym razie wraca do stanu q0.

- Stan q2:

Sygnał wyjściowy y2 ustawiany jest na wartość X"2".

Jeśli dane wejściowe (D0) są równe stałej P, następuje przejście do stanu q3; w przeciwnym razie pozostaje w stanie q0.

... i tak dalej, podobnie dla pozostałych stanów.

- Stan q12:

Sygnał wyjściowy y2 ustawiany jest na wartość X"C".

Jeśli dane wejściowe (D0) są równe stałej P, następuje przejście do stanu q1; w przeciwnym razie pozostaje w stanie q0.

Po zakończeniu analizy stanu i danych wejściowych, aktualizowany jest sygnał next_state w zależności od warunków zdefiniowanych dla każdego stanu. Dodatkowo, sygnał wyjściowy y jest ustawiany na '1' w przypadku, gdy aktualny stan jest równy q12, w przeciwnym razie przyjmuje wartość '0'. Ta część kodu implementuje kluczową logikę automatu skończonego, opisującą jego zachowanie w odpowiedzi na wejściowe dane i przejścia między różnymi stanami.

5. Symulator ISim

W celu utworzenia testbencha wykorzystaliśmy wbudowaną funkcję programu ISE Project Navigator. Dzięki wcześniejszej deklaracji sygnału Clk, testbench automatycznie wygenerował wszystkie niezbędne elementy do sprawdzenia poprawności kodu. Naszym zadaniem było zdefiniowanie odpowiedniego opóźnienia zegara, który będzie odpowiadał tym zadanym w ćwiczeniu. Ten proces pozwolił nam skoncentrować się na istotnych aspektach testowania, przy jednoczesnym skorzystaniu z efektywności generowanego automatycznie testbencha.

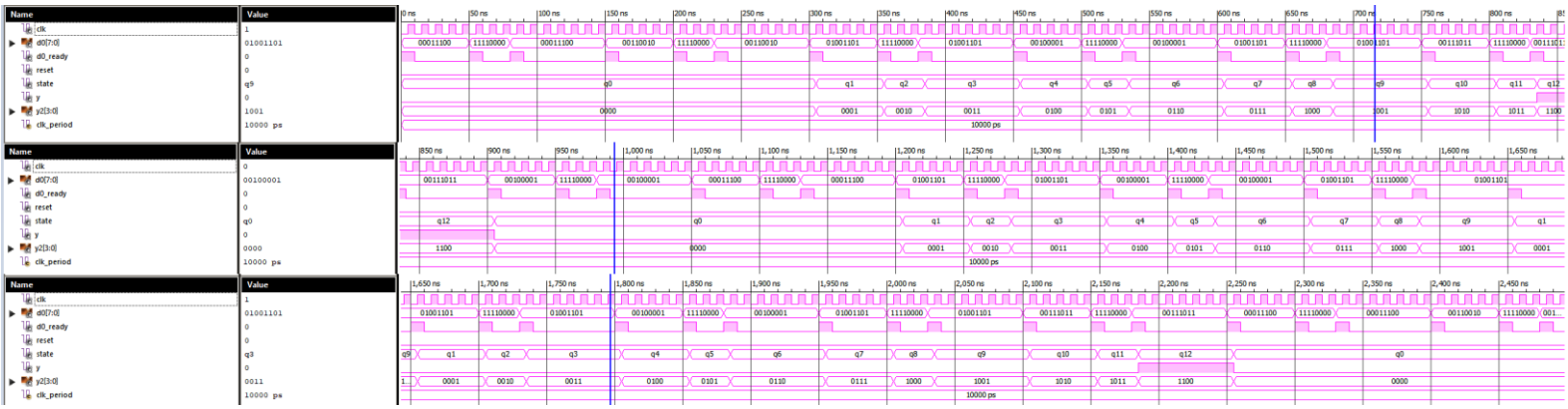
```
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY test2 IS
36  END test2;
37
38  ARCHITECTURE behavior OF test2 IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT main
43      PORT(
44           Clk : IN  std_logic;
45           D0 : IN  std_logic_vector(7 downto 0);
46           D0_ready : IN  std_logic;
47           y : OUT  std_logic;
48           Reset : IN  std_logic;
49           y2 : OUT STD_LOGIC_VECTOR(3 downto 0)
50          );
51      END COMPONENT;
52
53
54     --Inputs
55     signal Clk : std_logic := '0';
56     signal D0 : std_logic_vector(7 downto 0) := (others => '0');
57     signal D0_ready : std_logic := '0';
58     signal Reset : std_logic := '0';
59
60     --Outputs
61     signal y : std_logic;
62     signal y2 : std_logic_vector(3 downto 0);
63
64     -- Clock period definitions
65     constant Clk_period : time := 10 ns;
66
67     type ByteArray is array(16 downto 0) of std_logic_vector (7 downto 0);
68
69     -- A = 00011100
70     -- B = 00110010
71     -- P = 01001101
72     -- C = 00100001
73     -- J = 00111011
74     constant testArray: ByteArray := ("00011100", "00110010",
75     "01001101", "00100001", "01001101", "00111011",
76     "00100001", "00011100",
77     "01001101", "00100001", "01001101", "01001101", "00100001", "01001101", "00111011",
78     "00011100", "00110010");
```

W tej części kodu definiowana jest testbench dla modułu main. W sekcji ARCHITECTURE o nazwie behavior, deklarowany jest komponent (main), który reprezentuje jednostkę testowaną. Jest to odniesienie do modułu, który ma zostać przetestowany. Komponent ten posiada porty zgodne z interfejsem modułu main, tj. sygnały Clk, D0, D0_ready, y, Reset, i y2. Następnie definiowane są sygnały wejściowe i wyjściowe testbenchu, takie jak Clk, D0, D0_ready, y, Reset, i y2. Sygnały te są używane do dostarczania danych wejściowych do modułu main oraz do odbierania danych wyjściowych z tego modułu podczas symulacji. Warto zauważyć, że została zdefiniowana stała Clk_period określająca okres sygnału zegara, a także utworzony został typ ByteArray, będący tablicą zawierającą 17 wektorów ośmiobitowych, reprezentujących konkretne wartości. Ostatnia część tego bloku kodu to definicja tablicy testArray, która jest zainicjowana zestawem 17 wartości (reprezentujących sekwencję testową dla sygnału D0). Podsumowując, ta część kodu stanowi strukturę testbenchu, zawierającą definicje sygnałów, stałych, komponentu main, oraz przykładowych danych wejściowych do testowania modułu main w środowisku symulacyjnym.

```
80  BEGIN
81
82     -- Instantiate the Unit Under Test (UUT)
83     uut: main PORT MAP (
84            Clk => Clk,
85            D0 => D0,
86            D0_ready => D0_ready,
87            y => y,
88            y2 => y2,
89            Reset => Reset
90          );
91
92     -- Clock process definitions
93     Clk_process :process
94     begin
95        Clk <= '0';
96        wait for Clk_period/2;
97        Clk <= '1';
98        wait for Clk_period/2;
99     end process;
100
101
102    -- Stimulus process
103    stim_proc: process
104    begin
105
106       -- insert stimulus here
107       for i in 16 downto 0 loop
108
109          D0 <= testArray(i);
110          D0_ready <= '1';
111          wait for Clk_period;
112
113          D0_ready <= '0';
114          wait for 4*Clk_period;
115
116          D0 <= "11110000";
117          D0_ready <= '1';
118          wait for Clk_period;
119
120          D0_ready <= '0';
121          wait for 2*Clk_period;
122
123          D0 <= testArray(i);
124          D0_ready <= '1';
125          wait for Clk_period;
126
127          D0_ready <= '0';
128          wait for 6*Clk_period;
129       end loop;
130       wait;
131    end process;
132
133 END;
```

W tej części kodu definiowana jest sekcja BEGIN testbenchu, która zawiera instancję testowanego modułu main oraz procesy związane z generowaniem sygnałów zegarowych i symulacją stymulacji. Pierwszą czynnością jest instancjonowanie testowanego modułu poprzez utworzenie egzemplarza o nazwie uut typu main za pomocą polecenia PORT MAP. W ramach tego polecenia przypisywane są odpowiednie sygnały i porty: Clk zostaje połączony z zegarem Clk, D0 z wejściowym sygnałem danych D0, D0_ready z sygnałem gotowości danych D0_ready, y z głównym wyjściem y, y2 z wyjściem

pomocniczym y2, a Reset z sygnałem resetowania Reset. Następnie definiowane są procesy generujące sygnał zegarowy oraz stymulację. Proces Clk_process odpowiada za generowanie sygnału zegarowego. Sygnał ten jest cyklicznie ustawiany na '0' i '1' w określonym interwale, zgodnie z wartością Clk_period, określającą czas trwania jednego cyklu zegarowego. Proces stim_proc to sekcja symulacji stymulacji. W pętli for generowane są kolejne stymulacje dla sygnału D0 i D0_ready. Wykorzystywana jest tablica testArray zdefiniowana wcześniej, aby dostarczyć różne wartości testowe dla sygnału D0. Określone są czasy oczekiwania, takie jak okresy gotowości danych (D0_ready), aby odpowiednio symulować interakcję z modułem main w warunkach rzeczywistych. Podsumowując, ta część kodu odpowiada za przygotowanie i uruchomienie symulacji testów dla modułu main w środowisku testbenchu. Procesy generują sygnał zegarowy oraz sekwencję testową dla sygnału D0 i D0_ready, co umożliwia analizę zachowania modułu podczas symulacji.



6. Wnioski

Udało nam się pomyślnie zrealizować główne zadanie laboratorium, tj. implementację zamka szyfrowego otwieranego kombinacją 4 klawiszy (inicjały PCPJ) przy użyciu maszyny stanów. Symulacja behawioralna obejmująca wszystkie 13 stanów oraz prezentacja ich poprzez sekwencję **PCPJ**PCPPCPJ** została skutecznie zaimplementowana. Dodatkowo, przeprowadziliśmy symulację modułu Zamka, gdzie testowaliśmy ciąg naciskanych klawiszy za pomocą wektora z 17 kodami klawiszy. Śledzenie wewnętrznego sygnału "state" zostało również pomyślnie zrealizowane. W przypadku drugiego zadania, związanego z połączeniem maszyny stanów z blokiem HexTo7Seg, udało nam się pomyślnie zaprogramować kod, jednak ze względu na ograniczenia czasowe nie udało się go wgrać na płytkę. W trakcie tego laboratorium zdobyliśmy praktyczne doświadczenie w implementacji maszyn stanów oraz integracji modułów wejścia/wyjścia. Zastosowanie automatu skończonego w projektach cyfrowych stało się dla nas bardziej zrozumiałe i umożliwiło lepsze zrozumienie zasad ich działania. Podsumowując, laboratorium przyniosło nam praktyczną wiedzę, a także umożliwiło rozwijanie umiejętności programowania w języku VHDL i pracy z układami cyfrowymi.