Urządzenia Cyfrowe i Systemy Wbudowane
Detektor sekwencji 110011

Jędrzej Stasik, Kamil Szadkowski

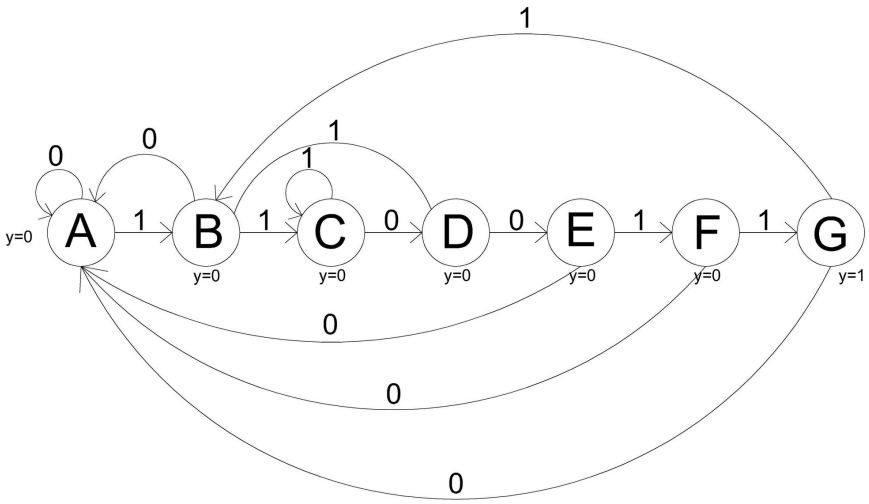
4.01.2024 r.

1 Zadanie

Zaprojektować detektor sekwencji *110011* jako automat w wersji *Moore* oraz *Mealy*. Dla obu automatów należy przeprowadzić symulację behawioralną wczytując odpowiednią 21-elementową sekwencję ($***abcdef***abcde\neg f***$), zmiana wejścia X ma nastapić 10 nanosekund przed zboczem narastającym zegara. Dodatkowym zadaniem było stworzenie projektu hierarchicznego łączącego w sobie wcześniej stworzony detektor sekwencji w wersji Moore oraz gotowy komponent *RotaryEnc*.

2 Detektor sekwencji w wersji Moore

2.1 Graf automatu



Rysunek 1: Graf detektora sekwencji jako automat moore

2.2 Plik VHDL

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

entity lab4_moore is
    Port ( X : in  STD_LOGIC;
           Y : out  STD_LOGIC;
           CLK : in  STD_LOGIC;
           CE : in  STD_LOGIC;
           RST : in  STD_LOGIC);
end lab4_moore;

architecture Behavioral of lab4_moore is

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

begin
  process1 : process(CLK)
  begin
    if rising_edge(CLK) then
      if RST = '1' then
        state <= A;
      else
        state <= next_state;
      end if;
    end if;
  end process process1;

  process2 : process (state, X , CE)
  begin
    next_state <= state;
    if CE = '1' then
      case state is

        when A =>
          if X = '1' then
            next_state <= B;
          end if;

        when B =>
          if X = '1' then
            next_state <= C;
          else
            next_state <= A;
          end if;

        when C =>
          if X = '0' then
            next_state <= D;
          else
            next_state <= C;
          end if;

        when D =>
          if X = '0' then
            next_state <= E;
          else
            next_state <= B;
          end if;

        when E =>
          if X = '1' then
            next_state <= F;
          else
            next_state <= A;
          end if;

        when F =>
          if X = '1' then
            next_state <= G;
          else
```

```
            next_state <= A;
          end if;

        When G =>
          if X = '1' then
            next_state <= B;
          else
            next_state <= A;
          end if;
      end case;
    end if;
  end process process2;

  Y <= '1' when state = G else '0';
end Behavioral;
```

Listing 1: Implementacja detektora sekwencji 110011 jako automat Moore

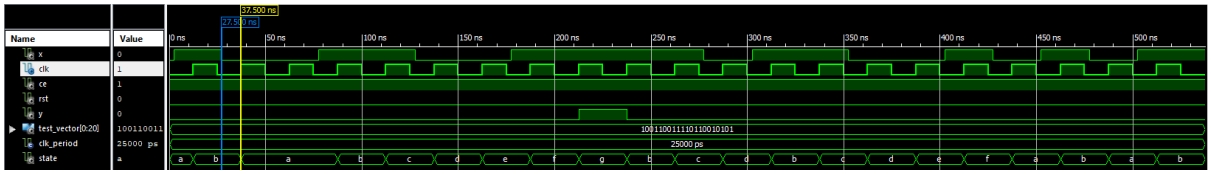
2.3 Testbench

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

ENTITY moore_test IS
END moore_test;

ARCHITECTURE behavior OF moore_test IS

    COMPONENT lab4_moore
    PORT(
         X : IN  std_logic;
         Y : OUT  std_logic;
         CLK : IN  std_logic;
         CE : IN  std_logic;
         RST : IN  std_logic
        );
    END COMPONENT;

   --Inputs
   signal X : std_logic := '0';
   signal CLK : std_logic := '0';
   signal CE : std_logic := '1';
   signal RST : std_logic := '0';

  --Outputs
   signal Y : std_logic;

   signal test_vector : STD_LOGIC_VECTOR (0 to 20) := "10011001111010110010101";

   -- Clock period definitions
   constant CLK_period : time := 25 ns;

BEGIN
  -- Instantiate the Unit Under Test (UUT)
   uut: lab4_moore PORT MAP (
          X => X,
          Y => Y,
          CLK => CLK,
          CE => CE,
          RST => RST
        );

   -- Clock process definitions
   CLK_process :process
   begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
   end process;
```

```
51
52
53    -- Stimulus process
54    stim_proc: process
55    begin
56     wait for 2.5 ns;
57     for i in 0 to 20 loop
58       X <= test_vector(i);
59       wait for CLK_period;
60     end loop;
61       wait;
62    end process;
63 END;
```

Listing 2: Testbench dla automatu Moore

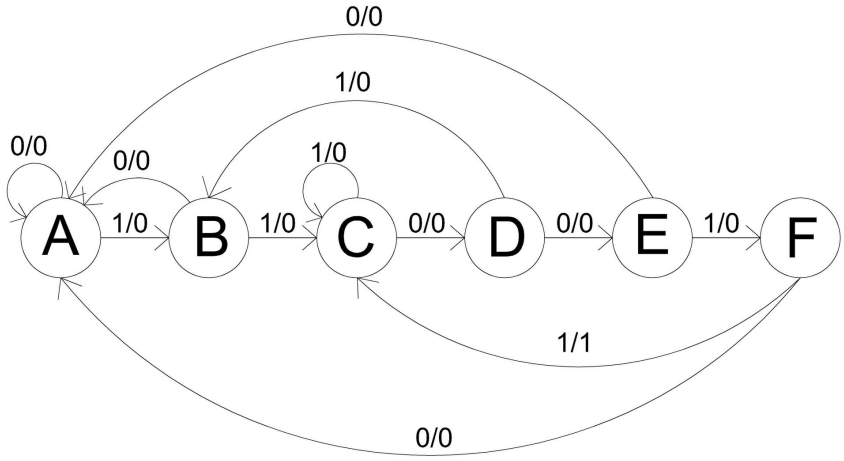
Sekwencja dla której testowaliśmy to *100 110011 110 110010 101*. Zgodnie z oczekiwaniami, po pierwszym bicie automat przechodzi do stanu B, następnie kolejne dwa bity powodują powrót i pozostanie w stanie A. Następne 6 bitów to prawidłowa sekwencja, która powoduje przejście przez wszystkie stany i zasygnalizowanie wykrycia poprawnej sekwencji. Ostatnim bitem poprawnej sekwencji jest bit 1, który powoduje przejście do stanu B, kolejna 1 do stanu C, 0 do D. Następnym wprowadzonym bitem jest 1, co skutkuje powrotem do stanu B. Jest to równocześnie pierwszy bit sekwencji różniącej się od poprawnej sekwencji tylko ostatnim bitem. Jej wprowadzenie nie spowodowało zasygnalizowania wykrycia poprawnej sekwencji, co oznacza, że zaprojektowany przez nas automat działa prawidłowo. Jej ostatnim bitem jest 0, które powoduje powrót do stanu A. Ostatnie 3 bity to losowe bity, które ostatecznie doprowadziły do stanu B.



Rysunek 2: Wynik symulacji behawioralnej automatu Moore

3 Detektor sekwencji w wersji Mealy

3.1 Graf automatu



Rysunek 3: Graf detektora sekwencji jako automat mealy

3.2 Plik VHDL

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

library UNISIM;
use UNISIM.VComponents.all;

entity lab4_mealy is
    Port ( X : in  STD_LOGIC;
           Y : out  STD_LOGIC;
           CLK : in  STD_LOGIC;
           CE : in  STD_LOGIC;
           RST : in  STD_LOGIC);
end lab4_mealy;

architecture Behavioral of lab4_mealy 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
      if RST = '1' then
        state <= A;
      else
        state <= next_state;
      end if;
    end if;
  end process process1;

  process2 : process (state, X , CE)
```

```
  begin
    next_state <= state;
    if CE = '1' then
      case state is

        when A =>
          if X = '1' then
            next_state <= B;
          end if;

        when B =>
          if X = '1' then
            next_state <= C;
          else
            next_state <= A;
          end if;

        when C =>
          if X = '0' then
            next_state <= D;
          end if;

        when D =>
          if X = '0' then
            next_state <= E;
          else
            next_state <= B;
          end if;

        when E =>
          if X = '1' then
            next_state <= F;
          else
            next_state <= A;
          end if;

        when F =>
          if X = '1' then
            next_state <= C;
          else
            next_state <= A;
          end if;
      end case;
    end if;
  end process process2;

  Y <= '1' when state = F and X = '1' else '0';
end Behavioral;
```

Listing 3: Implementacja detektora sekwencji 110011 jako automat Mealy

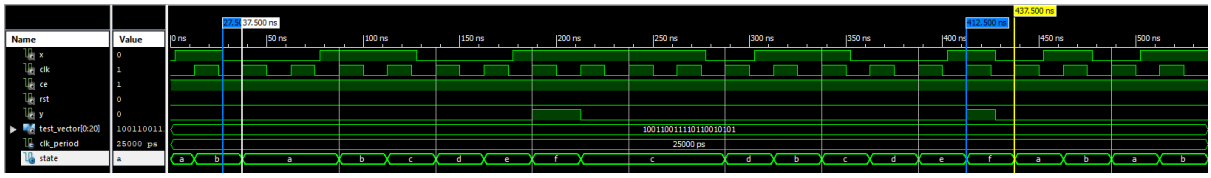
3.3 Testbench

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

ENTITY mealy_test IS
END mealy_test;

ARCHITECTURE behavior OF mealy_test IS

    COMPONENT lab4_mealy
    PORT(
         X : IN  std_logic;
         Y : OUT  std_logic;
         CLK : IN  std_logic;
         CE : IN  std_logic;
         RST : IN  std_logic
        );
    END COMPONENT;
```

```
18
19
20    --Inputs
21    signal X : std_logic := '0';
22    signal CLK : std_logic := '0';
23    signal CE : std_logic := '1';
24    signal RST : std_logic := '0';
25
26   --Outputs
27    signal Y : std_logic;
28
29   signal test_vector : STD_LOGIC_VECTOR (0 to 20) := "10011001111010010101";
30
31    -- Clock period definitions
32    constant CLK_period : time := 25 ns;
33
34 BEGIN
35   -- Instantiate the Unit Under Test (UUT)
36    uut: lab4_mealy PORT MAP (
37           X => X,
38           Y => Y,
39           CLK => CLK,
40           CE => CE,
41           RST => RST
42         );
43
44    -- Clock process definitions
45    CLK_process :process
46    begin
47     CLK <= '0';
48     wait for CLK_period/2;
49     CLK <= '1';
50     wait for CLK_period/2;
51    end process;
52
53    -- Stimulus process
54    stim_proc: process
55    begin
56     wait for 2.5 ns;
57     for i in 0 to 20 loop
58       X <= test_vector(i);
59       wait for CLK_period;
60     end loop;
61       wait;
62    end process;
63 END;
```

Listing 4: Testbench dla automatu Mealy

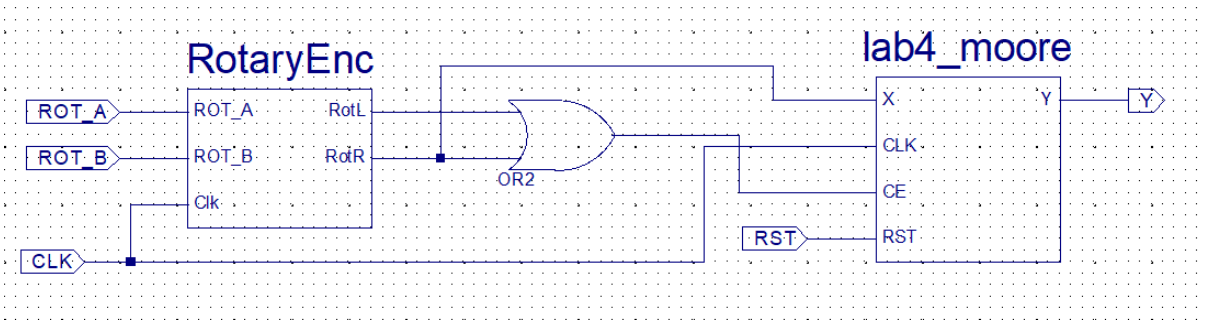
Automat Mealy'ego był testowany za pomocą tej samej sekwencji co automat Moore'a. Wykrycie poprawnej sekwencji nastąpiło w nim szybciej niż w automacie Moore'a, już w momencie wejścia w stan F, a nie dopiero po wyjściu z niego jak w automacie Moore'a. Wynika to z ogólnej specyfiki automatów Mealy'ego. Warto zauważyć, że gdy wykryto poprawną sekwencję sygnał Y=1 utrzymywał się przez pełny cykl zegarowy. W dalszej części symulacji po trzech kolejnych losowych bitach, wprowadzona została sekwencji z zanegowanym ostatnim bitem. Poskutkowało to przejście przez wszystkie stany zgodnie z kolejnością i zmianą stanu wyjścia Y na 1 już w momencie wejścia w stan F. Stan Y=1 nie utrzymywał się jednak przez pełen cykl zegarowy, tylko do momentu zmiany sygnału wejściowego X. Dzięki temu, mimo chwilowego pojawienia się stanu 1 na wyjściu Y, przejście które nastąpiło po wyzwoleniu zboczem zegarowym, było prawidłowe.



Rysunek 4: Wynik symulacji behawioralnej automatu Mealy

4 Projekt hierarchiczny

Kolejnym zadaniem było zaimplementowanie sterowania wprowadzaną sekwencją za pomocą bloku RotaryEnc i wykrywanie jej za pomocą wygenerowanego na podstawie poprzednich zadań bloku wykrywacza sekwencji. Przekręcenie pokrętła w prawo powodowało podanie na wejście X wartości 1, natomiast obrót w prawo 0. Po wgraniu programu na płytkę, wszystko działało zgodnie z założeniami zadania.



Rysunek 5: Schemat projektu hierarchicznego

5 Przypisanie wejść i wyjść

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

# Keys
#NET "CE" LOC = "P42";
#NET "X" LOC = "P40";
NET "RST" LOC = "P39";

# LEDS
NET "Y"  LOC = "P35";

# Rotary encoder
NET "ROT_A" LOC = "P36";
NET "ROT_B" LOC = "P24";
```

Listing 5: Plik .ucf

6 Wnioski

Oba automaty zostały zaprojektowane prawidłowo i działały zgodnie z założeniami zadania. Zadanie pozwoliło na zaobserwowanie praktycznych różnic w działaniu między automatem Mealy'ego i Moore'a, takich jak szybkość działania. Zgodnie z oczekiwaniami automat Moore'a miał jeden dodatkowy stan, który automat osiągał wyłącznie w sytuacji wykrycia prawidłowej sekwencji. W przypadku automatu Mealy'ego, ze względu na to, że wyjście zależy od stanu oraz aktualnej wartości wejściowej, liczba stanów była o 1 mniejsza. Z tego względu w przypadku automatu Mealy'ego może dojść do sytuacji, która była widoczna w symulacji behawioralnej wykrywacza sekwencji w tej wersji. Warto zauważyć, że sygnał na wyjściu po błędnym zasygnalizowaniu wykrycia sekwencji trwał znacznie krócej niż w przypadku poprawnego wykrycia sekwencji i zmienił się wraz ze zmianą wartości na wejściu. Niemniej jednak warto mieć na uwadze możliwość wystąpienia takich przekłamań w przypadku projektowania wykrywacza w oparciu o automat Mealy'ego.