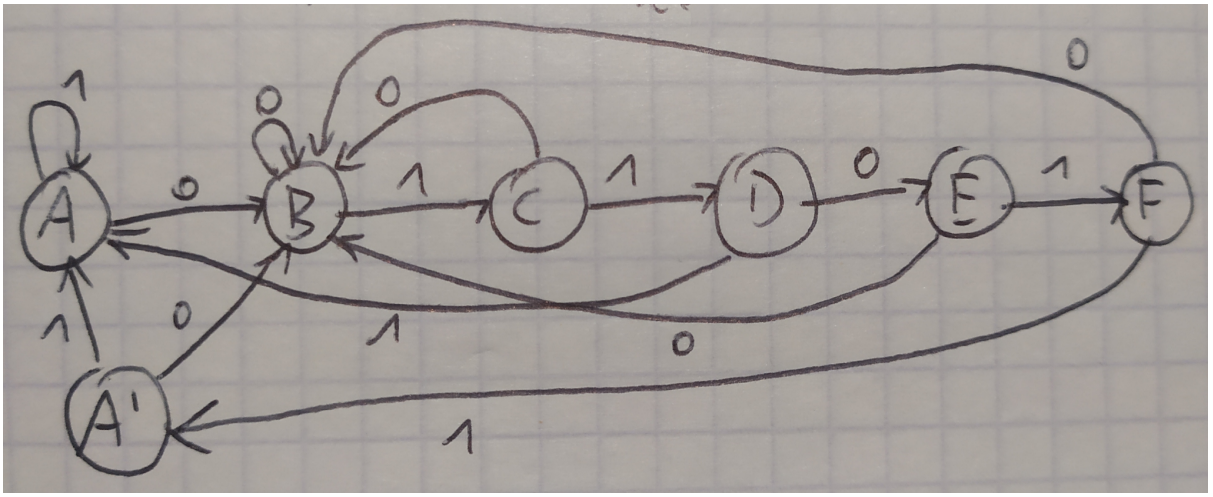
Układy cyfrowe i systemy wbudowane 1		
Prowadzący	Dr inż. Jarosław Sugier	
Temat zajęć	Układy Sekwencyjne	
Termin zajęć	Czwartek 11:15 TP	
Skład grupy	Maciej Fras	259126
	Bartosz Kloc	259175

1. Wstęp

W ramach ćwiczenia wykonano detektory sekwencji 011011 w wersjach Moore'a i Mealy. Następnie wprowadzono ciąg ***abcdef***abcde(~f)*** (gdzie * oznacza dowolną wartość) celem sprawdzenia, jak poszczególne realizacje zadania zachowują się w przypadku wykrycia prawidłowej sekwencji oraz w sytuacji, gdzie podjęto próbę ich oszukania. Głównym celem ćwiczenia było zaobserwowanie i opisanie różnicy w działaniu automatów Moore'a i Mealy, w szczególności w sytuacjach w której automaty zwracają wynik 1. Dodatkowo należało zadbać o to, by wartość wejściowa X była zmieniana 10 ns przed rosnącym zboczem zegara pobudzającym przerzutnik a częstotliwość zegara wynosiła 33⅓ MHz.

2. Graf dla detektora w wersji Moore'a



3. Realizacja detektora w wersji Moore’a

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

entity zad1 is
    Port ( X : in  STD_LOGIC;
           CE : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end zad1;

architecture Behavioral of zad1 is

           type state_type is (A, B, C, D, E, F, Ap);
           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 )
                             begin
                             if CE = '1' then
                                      next_state <= state;
                                               case state is
                                                        when A =>
                                                                 if X = '1' then
                                                                 next_state <= A;
                                                                 else
                                                                 next_state <= B;
                                                                 end if;
                                                        when B =>
                                                                 if X = '1' then
                                                                 next_state <= C;
                                                                 else
```

```
                    next_state <= B;
                    end if;
                when C =>
                    if X = '1' then
                    next_state <= D;
                    else
                    next_state <= B;
                    end if;
                when D =>
                    if X = '1' then
                    next_state <= A;
                    else
                    next_state <= E;
                    end if;
                when E =>
                    if X = '1' then
                    next_state <= F;
                    else
                    next_state <= B;
                    end if;
                when F =>
                    if X = '1' then
                    next_state <= Ap;
                    else
                    next_state <= B;
                    end if;
                when Ap =>
                    if X = '1' then
                    next_state <= A;
                    else
                    next_state <= B;
                    end if;
            end case;
        end if;
    end process process2;

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


end Behavioral;
```

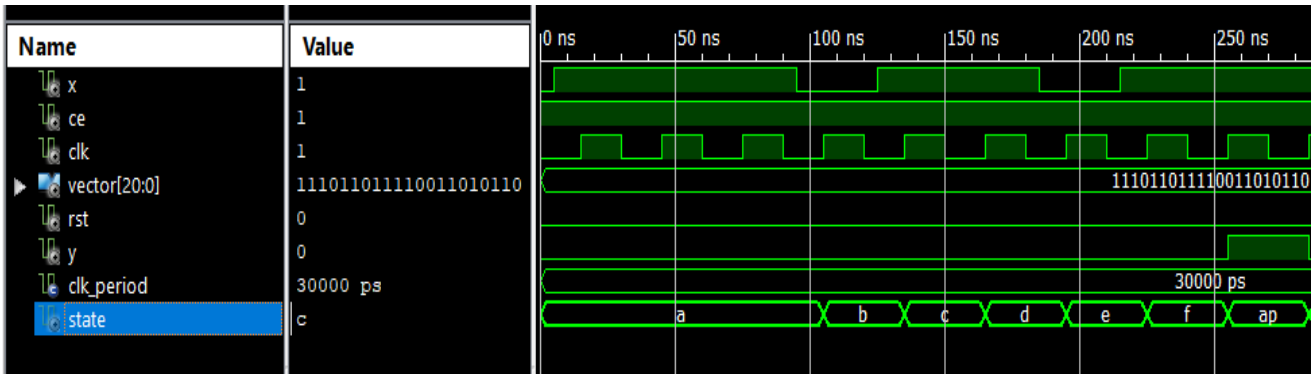
4. Symulacja

W celu symulacji działania obu automatów przygotowano po jednej symulacji behawioralnej do obu automatów. Obie symulacje zostały jednak stworzone w ten sam sposób, różnica jest jedynie w implementacji obu detektorów. Poniżej przedstawiono kod źródłowy dla detektora w wersji Moore'a, plik z symulacją drugiego układu wygląda analogicznie.

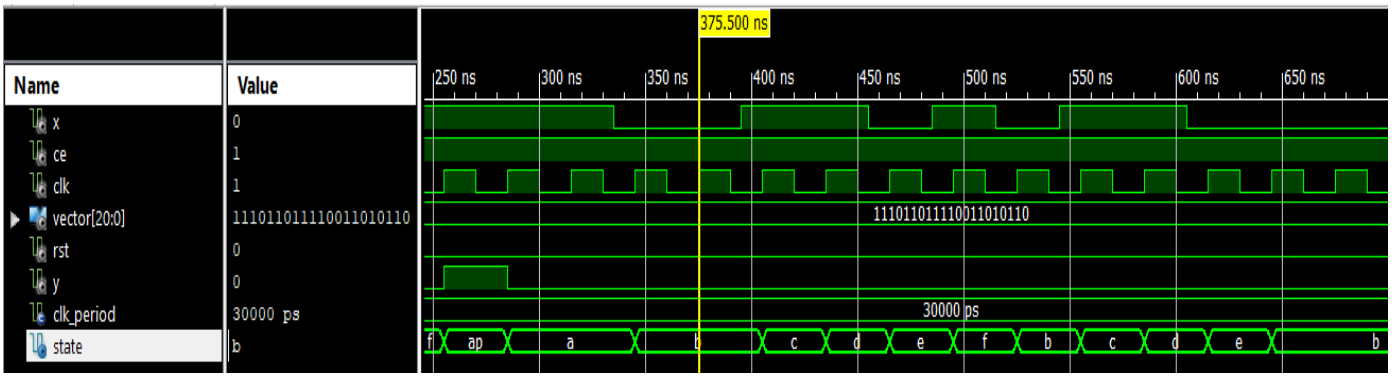
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY zad1TB IS
END zad1TB;

ARCHITECTURE behavior OF zad1TB IS

    COMPONENT zad1
    PORT(
         X : IN  std_logic;
         CE : IN  std_logic;
         CLK : IN  std_logic;
         RST : IN  std_logic;
         Y : OUT  std_logic
        );
    END COMPONENT;

   signal X : std_logic := '0';
   signal CE : std_logic := '1';
   signal CLK : std_logic := '0';
          signal Vector : std_logic_vector (20 downto 0) := "11101101111001101010110";
   signal RST : std_logic := '0';

   signal Y : std_logic;

   constant CLK_period : time := 30 ns;

BEGIN

   uut: zad1 PORT MAP (
         X => X,
         CE => CE,
         CLK => CLK,
         RST => RST,
         Y => Y
        );
```

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

stim_proc: process
begin
			wait for 5 ns;
  for i in 20 downto 0 loop
			X <= Vector(i);
			wait for CLK_period;
		end loop;
		wait;
end process;

END;
```

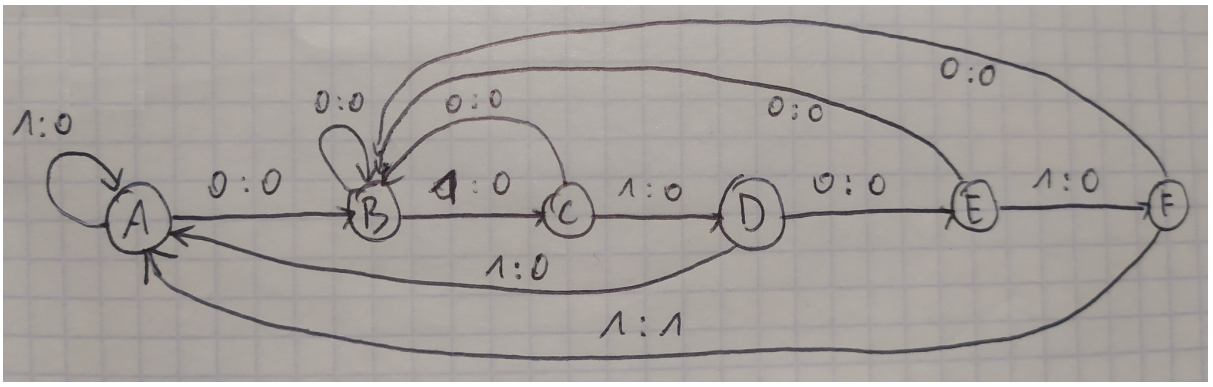
5. Wynik symulacji automatu Moore'a



Pierwsze trzy wartości były losowe, w tym przypadku były to trzy jedynki więc automat tkwił w stanie *a*. Następnie wykryto sekwencję przechodząc przez kolejne stany z grafu ostatecznie osiągając stan *ap (a')* do którego przypisano wartość Y = 1, sekwencja została wykryta. Warto zauważyć, że sygnał oznaczający wykrycie sekwencji wystąpił już po wyjściu ze stanu *f*.



6. Graf dla detektora w wersji Mealy



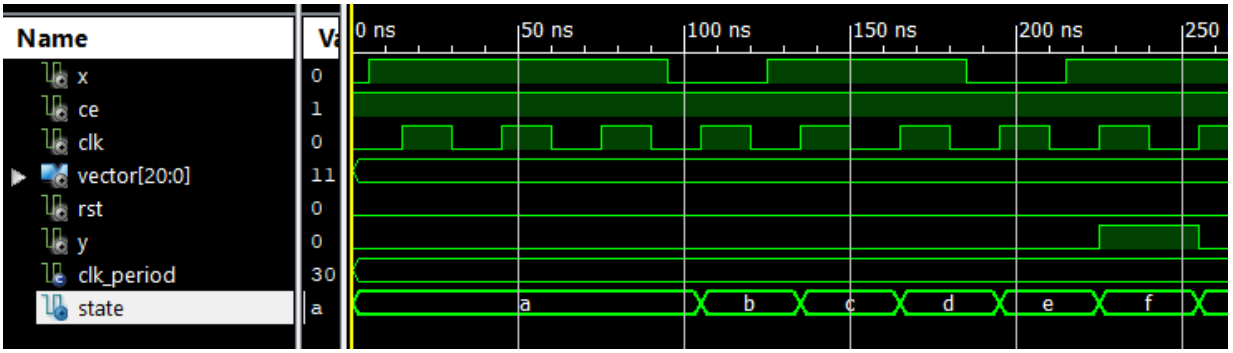
7. Realizacja detektora w wersji Mealy

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

entity zad2 is
    Port ( X : in  STD_LOGIC;
           CE : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end zad2;

architecture Behavioral of zad2 is

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

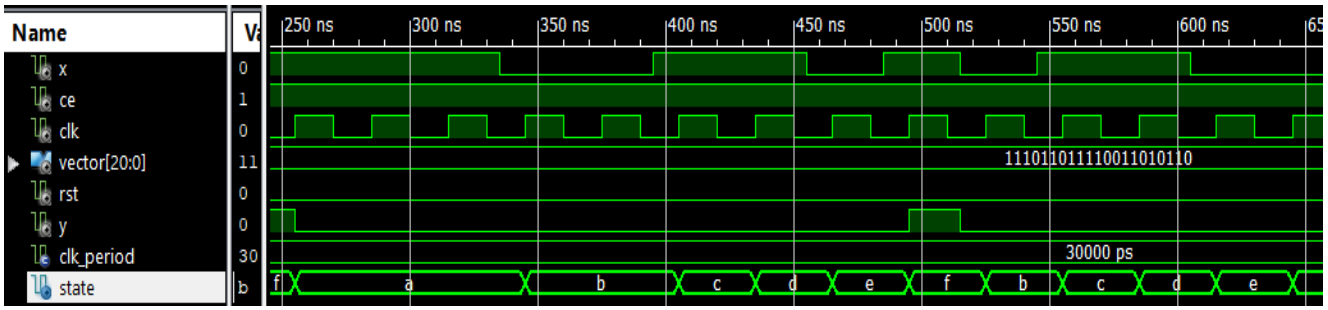
```
begin
        process1 : process ( Clk )
        begin
                if rising_edge ( Clk ) then
                        if RST = '1' then
                        state <= A;
                        else
                        state <= next_state;
                        end if;
             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

                process2 : process ( state, X )
                        begin
                        if CE = '1' then
                                next_state <= state;
                                        case state is
                                                when A =>
                                                        if X = '1' then
                                                        next_state <= A;
                                                        else
                                                        next_state <= B;
                                                        end if;
                                                when B =>
                                                        if X = '1' then
                                                        next_state <= C;
                                                        else
                                                        next_state <= B;
                                                        end if;
                                                when C =>
                                                        if X = '1' then
                                                        next_state <= D;
                                                        else
                                                        next_state <= B;
                                                        end if;
                                                when D =>
                                                        if X = '1' then
                                                        next_state <= A;
                                                        else
                                                        next_state <= E;
                                                        end if;
                                                when E =>
                                                        if X = '1' then
                                                        next_state <= F;
                                                        else
                                                        next_state <= B;
                                                        end if;
                                                when F =>
```

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

        y <= '1' when state = F and X = '1' else '0';


end Behavioral;
```

8. Wynik symulacji automatu w wersji Mealy



Podobnie jak podczas symulacji poprzedniej implementacji układ na początku symulacji tkwił w stanie *a* gdyż na wejściu dostał najpierw 3 jedynki. Następnie wprowadzona została sekwencja 011011 a więc wprowadzona została poprawna sekwencja już w momencie wejścia automatu w stan *f* a więc szybciej niż w automacie Moore'a co wynika z ogólnej specyfiki automatów Mealy gdzie wyjście Y zależy od aktualnego stanu oraz aktualnie wybranej wartości X. Należy zauważyć, że sygnał Y=1 utrzymywał się przez pełny cykl zegarowy i zmienił się dopiero razem ze zmianą stanu automatu.



W dalszej części symulacji po trzech kolejnych losowych wartościach wprowadzono sekwencję z zanegowanym ostatnim bitem. W efekcie zanim wprowadzono ostatni bit (0) automat przeszedł przez wszystkie kolejne stany zgodnie z kolejnością, w tym momencie wejścia w stan *f* wartość X wciąż wynosi 1 a więc zgodnie z grafem stanów sygnał Y przyjmuje wartość 1. Nie jest to jednak jednoznaczne z wykryciem prawidłowej sekwencji, sygnał Y=1 nie trwa pełnego cyklu, trwa tyle czasu, ile utrzymuje się jedynka na wejściu i zmienia się na 0 w momencie gdy zmieniony zostaje sygnał wejściowy X zanim jeszcze nastąpi pobudzenie zegarowe i automat zmieni stan. Po wyzwoleniu zboczem zegarowym przechodzimy ze stanu *f* do *b* gdyż w momencie pobudzenia wartość X była już równa 0.

9. Wnioski i podsumowanie

Ćwiczenie pokazało różnice między automatami Moore'a i Mealy. Realizacje zgodne z założeniami Moore'a mają zazwyczaj jeden stan więcej który automat osiąga w momencie wykrycia prawidłowej sekwencji i to właśnie do tego stanu przypisany jest sygnał wyjściowy równy 1. Jako że w tego typu realizacjach wyjście układu zależy jedynie od stanu w którym automat się znajduje to Y=1 możliwy do uzyskania jest jedynie wtedy, gdy automat wejdzie w te określone stany. W przypadku automatów Mealy wyjście zależy od stanu oraz aktualnej wartości wejściowej w związku z czym nie mamy dodatkowego stanu informującego o wykrytej sekwencji. Jednak jak pokazała symulacja, w tego typu automatach może dojść do sytuacji podobnych jak ta opisana w drugim akapicie punktu 8 w której mimo niewykrycia sekwencji na wyjściu zobaczymy wartość 1. Taki "przekłamany" sygnał nie trwa jednak pełnego cyklu zegarowego i zmienia się wraz ze zmianą wartości na wejściu. Należy mieć to na uwadze przy automatach opierających się na tym sposobie działania.