Układy cyfrowe i systemy wbudowane - laboratorium

Karol Kulawiec 241281
Bartosz Rudnikowicz 241382

21.11.2019

1 Wstęp

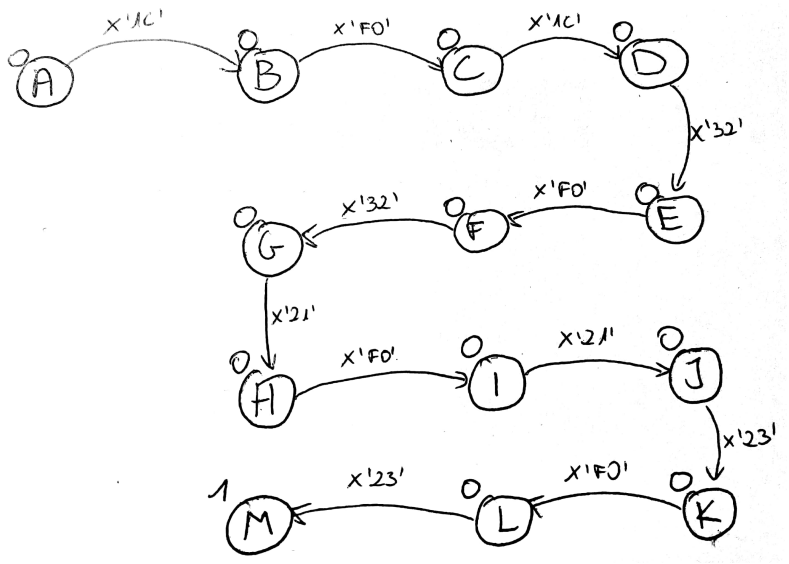
Naszym celem było stworzenie zamka szyfrowego reprezentowanego jako automat Moore'a, następnie zaimplementowanie go sprzętowo oraz zasymulowanie.
Sekwencja, którą miał wykrywać automat to 'ABCD' i była wprowadzana poprzez klawiaturę z interfejsem PS2.
Po wykryciu sekwencji zamek miał się otwierać, co było reprezentowane zapalającą się diodą, a następnie po wciśnięciu kolejnego klawisza zamek ponownie się zamykał.

2 Przebieg zajęć

Zajęcia składały się z 2 części. W przeciwieństwie do poprzednich zajęć, podczas tych, najpierw mieliśmy zaimplementować program na sprzęcie, a dopiero później wykonać jego symulację, co okazało się logiczne, gdyż symulacja przysporzyła nam więcej problemów.
Aby poprawnie obsłużyć sygnały, należało na początku sprawdzić, jak te sygnały wyglądają. W porównaniu do odczytu sygnałów z modułu RS232_RX, moduł PS2_RX nie korzystał z kodu ASCII, naciśnięcie i zwolnienie klawisza, powodowało przesłanie kodu klawisza, natomiast zwolnienie klawisza, przesyłało dodatkowo flagę zwolnienia klawisza (X'F0'). Ponieważ naszym kluczem był kod 'ABCD', musieliśmy jedynie odczytać kod tych znaków, które wyglądały następująco: A - X'1C', B - X'32', C - X'21', D - X'23'. Tak więc kod otwierający zamek wyglądał: X'C1', X'F0', X'C1', X'32', X'F0', X'32', X'21', X'F0', X'21', X'23', X'F0', X'23'. W zadaniu przyjęto, że poprawnie podany kod, wymaga kolejno naciśnięcia oraz zwolnienia klawiszy A, B, C, D. Dopiero po zwolnieniu klawisza D i podaniu poprawnej sekwencji, 'zamek otwierał się'.

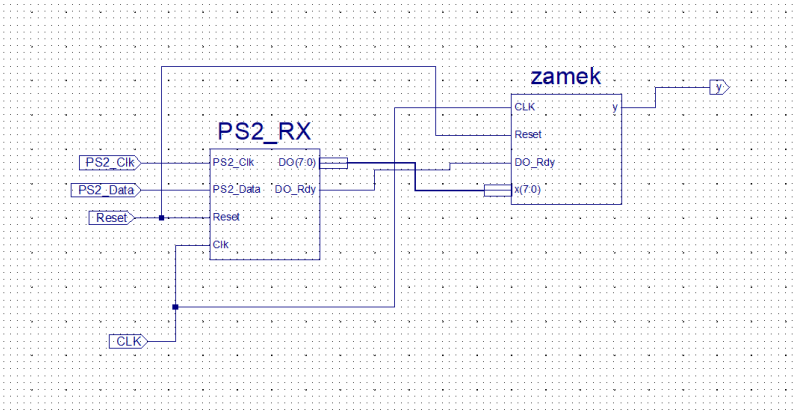
2.1 Automat Moore'a

Dla lepszej przejrzystości schematu Moore'a na schemacie zawarliśmy jedynie sygnały, które powodują przejście dalej. W każdym dowolnym momencie podanie stanu X'1C' powoduje przeskok do stanu B (poza stanem C, gdzie skacze do D), a podanie sygnału innego, niż przedstawionego na rysunku, powodowało przejście do stanu A.



Rysunek 1: Graf automatu Moore'a

Automat zamknęliśmy w module o nazwie 'zamek', połączyliśmy z modułem PS2_RX i uruchomiliśmy na sprzęcie.



Rysunek 2: Graf automatu Moore'a

2.2 Implementacja sprzętowa

Aby można było użyć modułu PS2_RX, należało go uprzednio dodać do projektu oraz odkomentować w pliku ZL-9572.ucf, ustawić przycisk Reset na Key<7>, wyjście y na diodę LED<0> oraz zegar CLK z okresem 500ns.

```
entity zamek is
    Port ( CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           x : in  STD_LOGIC_VECTOR (7 downto 0);
           y : out  STD_LOGIC;
           DO_Rdy : in  STD_LOGIC);
end zamek;
```

Następnie stworzyliśmy maszynę stanów, która zachowywała się wg wcześniej zamieszczonego schematu.

```
architecture Behavioral of zamek is

    type state_type is ( A, B, C, D, E, F, G, H, I, J, K, L, M);
    signal state, next_state : state_type;
```

```
begin

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

    process2: process(Do_Rdy, state, X)
    begin
    next_state <= state;
    if Do_Rdy = '1' then
        case state is
        when A => if x = X"1C" then next_state <= B; end if;
        when B => if x = X"F0" then next_state <= C; else next_state <=
            A; end if;
        when C => if x = X"1C" then next_state <= D; else next_state <=
            A; end if;
        when D => if x = X"32" then next_state <= E; elsif x = X"1C"
           then next_state <= B; else next_state <= A; end if;
        when E => if x = X"F0" then next_state <= F; else next_state <=
            A; end if;
        when F => if x = X"32" then next_state <= G; else next_state <=
            A; end if;
        when G => if x = X"21" then next_state <= H; elsif x = X"1C"
           then next_state <= B; else next_state <= A; end if;
        when H => if x = X"F0" then next_state <= I; else next_state <=
            A; end if;
        when I => if x = X"21" then next_state <= J; else next_state <=
            A; end if;
        when J => if x = X"23" then next_state <= K; elsif x = X"1C"
           then next_state <= B; else next_state <= A; end if;
        when K => if x = X"F0" then next_state <= L; else next_state <=
            A; end if;
        when L => if x = X"23" then next_state <= M; else next_state <=
            A; end if;
        when M => if x = X"1C" then next_state <= B; else next_state <=
            A; end if;
        end case;
    end if;
end process process2;

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

end Behavioral;
```

2.3 Test Bench

Podczas testu naszym zadaniem było sprawdzenie kombinacji XYABCDXYABCZXY, w którym XYZ to dowolne znaki (u nas X = X'22', Y = '35', Z = '1A'). Kombinacja posiada jednorazowo, poprawnie wprowadzony kod, oraz kod ze zmienioną ostatnią literą. Aby zapisać kombinację utworzyliśmy nowy typ wektora, który przechowywał 14 znaków 8 bitowych:

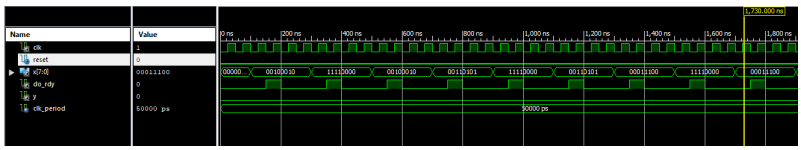
```
type typeByteArray is array (NATURAL range <>) of STD_LOGIC_VECTOR (7
    downto 0);
variable arrBytes : typeByteArray(0 to 13)
```

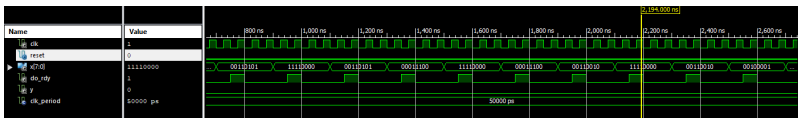
```
:= (X"22", X"35", X"1C", X"32", X"21", X"23", X"22", X"35", X"1C",
    X"32", X"21", X"1A", X"22", X"35");
```

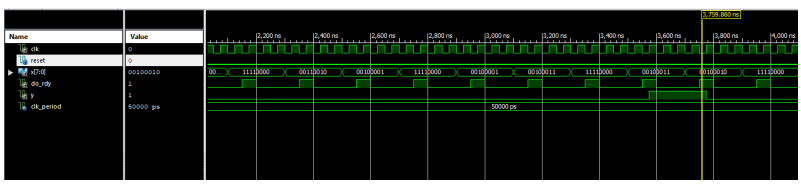
Algorytm polegał 3 krokach realizowanych dla kolejnych wartości z 'arrBytes', które były pobierane do wektora 8 bitowego 'x', odczekaniu jednego cyklu zegara, ustawieniu stanu gotowości na 1, przeczekaniu jednego cyklu zegara, następnie ustawieniu stanu gotowości na 0 i przeczekaniu 2 cykli zegara. Następnie całość była powtarzana, z tą różnicą, że za 2 razem do wektora 'x' przekazywano wartość X"F0", informującą o 'podniesieniu palca z klawisza'. Za 3 razem ponawiano krok 1.

```
begin
    wait for 100 ns;

    for i in arrBytes'RANGE loop

        x <= arrBytes(i);
        wait for CLK_period;
        DO_Rdy <= '1';
        wait for CLK_period;
        DO_Rdy <= '0';
        wait for 2*CLK_period;


        x <= X"F0";
        wait for CLK_period;
        DO_Rdy <= '1';
        wait for CLK_period;
        DO_Rdy <= '0';
        wait for 2*CLK_period;


        x <= arrBytes(i);
        wait for CLK_period;
        DO_Rdy <= '1';
        wait for CLK_period;
        DO_Rdy <= '0';
        wait for 2*CLK_period;

    end loop;

  wait;
```

Czekanie miało na celu ustabilizowanie sygnału oraz odczytaniu go w trakcie jego trwania, a nie w trakcie zmiany, aby odczytać prawidłowe wartości.
Wyniki symulacji prezentują się następująco:







Jak widać odczyt sygnałów zachodzi prawidłowo i dla podanej sekwencji 'ABCD' 'zamek poprawnie otwiera się.

3 Wnioski

Podczas zajęć zaprojektowaliśmy, zaimplementowaliśmy oraz zasymulowaliśmy zadany układ. Większych problemów z zadaniem nie było. Układ działał prawidłowo.