SoC설계

Lab#7

Finite State Machine

컴퓨터공학과

201402439

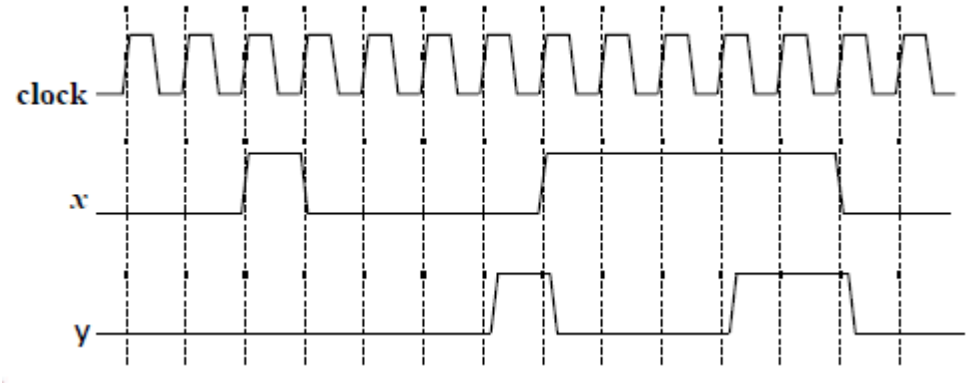
천원준

1. Purpose of the lab

이번 과제의 목표는 주어진 Timing Diagram에 대한 Mealy Machine, Moore Machine를 구현 후 제대로 된 동작을 하는지 검증하는 것입니다.

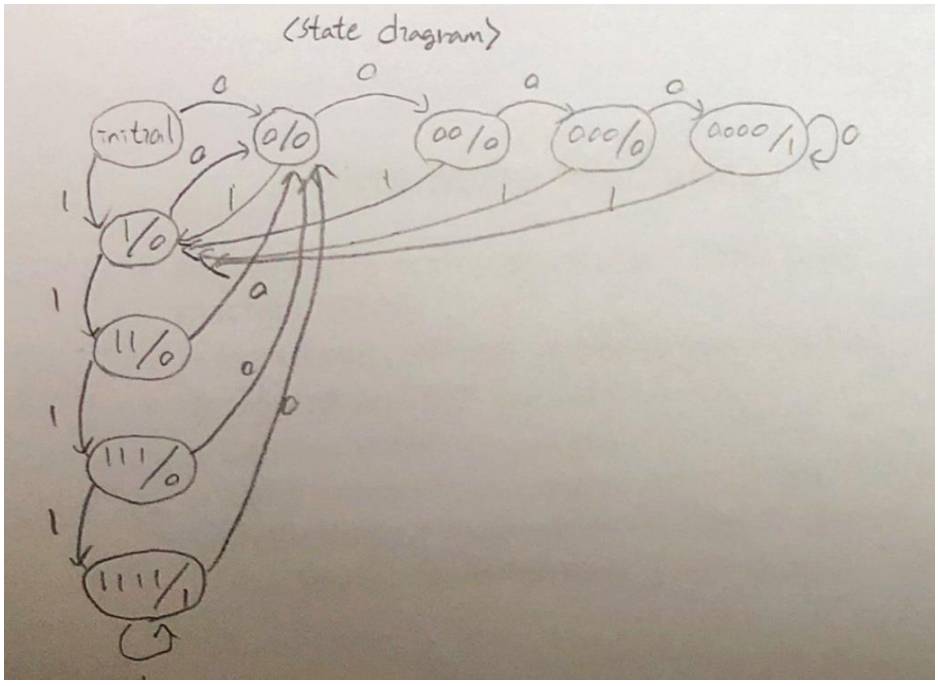
2. Design Procedure

(1) Timing diagram

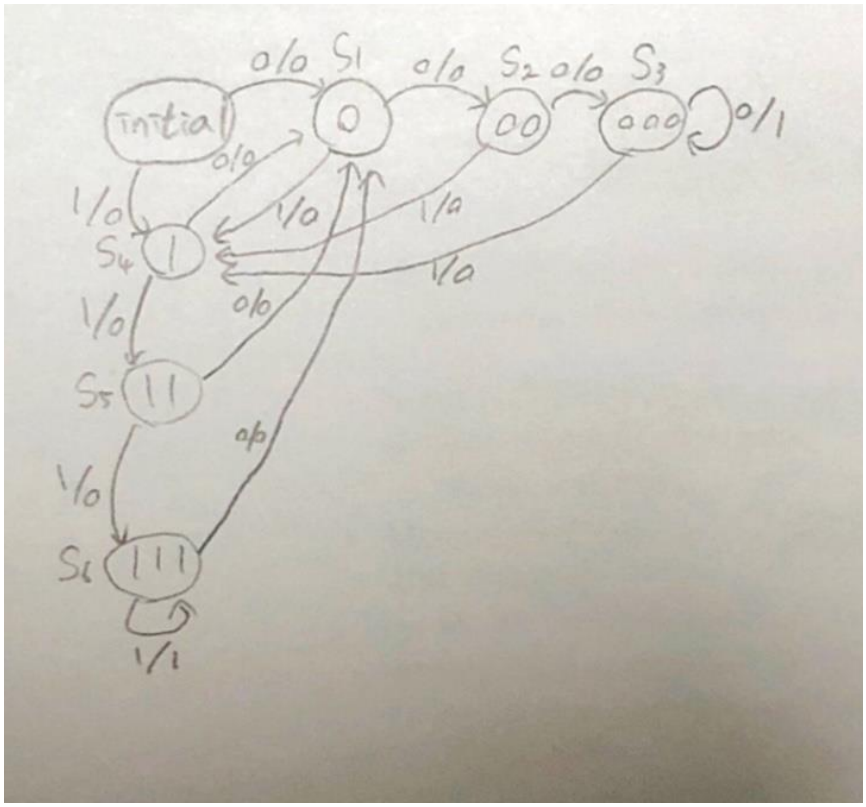


(2) State diagram

- Moore Machine

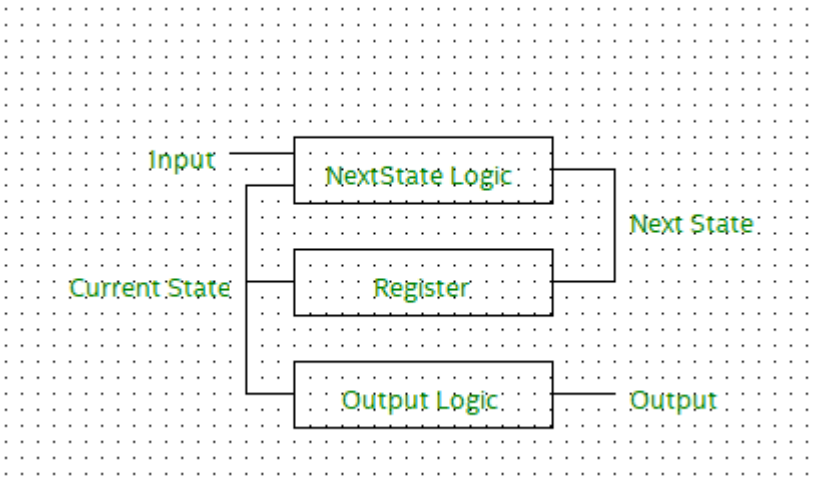


- Mealy Machine

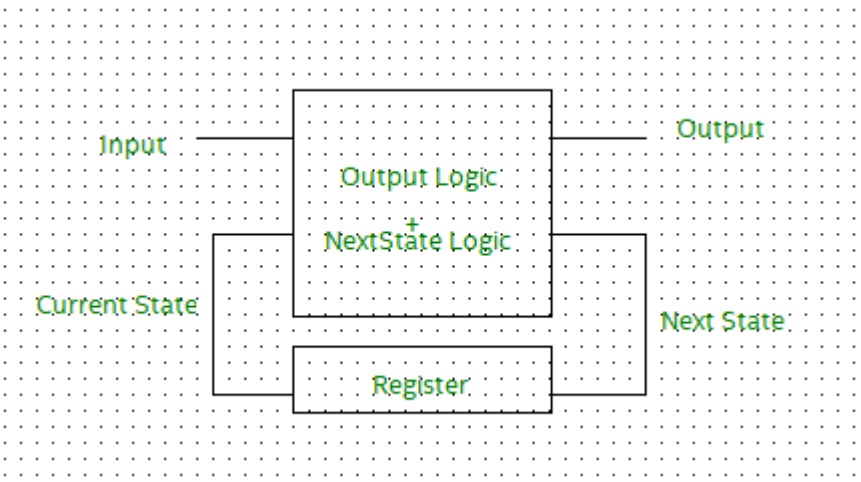


(3) Block diagram

- Moore Machine



- Mealy Machine



3. Simulation

- Moore Machine

```
//state
`define Init 4'b0000
`define S1 4'b0001
`define S2 4'b0010
`define S3 4'b0011
`define S4 4'b0100
`define S5 4'b0101
`define S6 4'b0110
`define S7 4'b0111
`define S8 4'b1000

module MooreMachine(nReset, clk, in, out);
   //input & output
   input in, nReset, clk;
   output out;

   //in case of FSM, output is reg type
   //reg CurState, NextState, out;
   reg [3:0] CurState, NextState;
   reg out;


   always @(posedge clk or negedge nReset)
      //Reset
      if(!nReset) CurState <= `Init;
      //Transition
      else CurState <= NextState;
```

```
    always @(CurState or in)
       //casex includes don't care and High-Z
       casex(CurState)
          //if CurState == `Init
          `Init: begin
             if(in==0) NextState = `S1;
             else NextState = `S5;
             out = 1'b0;
          end

          `S1: begin
             if(in==0) NextState = `S2;
             else NextState = `S5;
             out = 1'b0;
          end

          `S2: begin
             if(in==0) NextState = `S3;
             else NextState = `S5;
             out = 1'b0;
          end

          `S3: begin
             if(in==0) NextState = `S4;
             else NextState = `S5;
             out = 1'b0;
          end

          `S4: begin
             if(in==0) NextState = `S4;
             else NextState = `S5;
             out = 1'b1;
          end


          `S5: begin
             if(in==0) NextState = `S1;
             else NextState = `S6;
             out = 1'b0;
          end

          `S6: begin
             if(in==0) NextState = `S1;
             else NextState = `S7;
             out = 1'b0;
          end

          `S7: begin
             if(in==0) NextState = `S1;
             else NextState = `S8;
             out = 1'b0;
          end

          `S8: begin
             if(in==0) NextState = `S1;
             else NextState = `S8;
             out = 1'b1;
          end

          default: begin
             NextState = `Init;
             out = 1'b0;
          end

       endcase
endmodule
```

무어 머신은 파라미터로 nReset, clk, in, out을 받는데, nReset은 Reset을 할 때 (nReset이 0이면 Reset함), clk은 클럭 입력을 의미합니다.

CurState와 NextState는 상태 정보를 저장해야 하므로 register 타입으로 선언합니다. 이들은 9가지 상태가 들어가야 하기 때문에, 4비트로 선언해줍니다.

첫번째 always문에는 clk가 상승모서리 이거나, nReset이 하강모서리 일 때 실행되는데, clk에 의한 실행일 때는 CurState를 NextState로 설정해줍니다. nReset에 의한 실행일 때는 CurState를 초기 상태로 설정해줍니다.

무어 머신은 현재 상태에 의해서만 출력이 결정되기 때문에, 각 case문에서 out 값 결정은 NextState의 결정 이후 한 번만 해줍니다.

- Mealy Machine

```
//state
`define Init 3'b000
`define S1 3'b001
`define S2 3'b010
`define S3 3'b011
`define S4 3'b100
`define S5 3'b101
`define S6 3'b110

module MealyMachine(nReset, clk, in, out);
   //input & output
   input in, nReset, clk;
   output out;

   //in case of FSM, output is reg type
   reg [2:0] CurState, NextState;
   reg out;


   always @(posedge clk or negedge nReset)
      //Reset
      if(!nReset) CurState <= `Init;
      //Transition
      else CurState <= NextState;
```

```
always @(CurState or in)
   //casex includes don't care and High-Z
   casex(CurState)
      //if CurState == `Init
      `Init: begin
         if(in==0) begin //input == 0
            NextState = `S1;
            out = 1'b0;
         end
         else begin //input == 1
            NextState = `S4;
            out = 1'b0;
         end
      end

      `S1: begin
         if(in==0) begin
            NextState = `S2;
            out = 1'b0;
         end
         else begin
            NextState = `S4;
            out = 1'b0;
         end
      end

      `S2: begin
         if(in==0) begin
            NextState = `S3;
            out = 1'b0;
         end
         else begin
            NextState = `S4;
            out = 1'b0;
         end
      end

      `S3: begin
         if(in==0) begin
            NextState = `S3;
            out = 1'b1;
         end
         else begin
            NextState = `S4;
            out = 1'b0;
         end
      end
```

```
            `S4: begin
               if(in==0) begin
                  NextState = `S1;
                  out = 1'b0;
               end
               else begin
                  NextState = `S5;
                  out = 1'b0;
               end
            end

            `S5: begin
               if(in==0) begin
                  NextState = `S1;
                  out = 1'b0;
               end
               else begin
                  NextState = `S6;
                  out = 1'b0;
               end
            end

            `S6: begin
               if(in==0) begin
                  NextState = `S1;
                  out = 1'b0;
               end
               else begin
                  NextState = `S6;
                  out = 1'b1;
               end
            end

            default: begin
               NextState = `Init;
               out = 1'b0;
            end

      endcase
endmodule
```

밀리 머신은 총 7가지 상태를 갖기 때문에, CurState와 NextState를 3비트로 선언해줍니다.

밀리 머신은 현재 상태와 현재 입력값에 의해 출력값이 결정되기 때문에, 각 case문에서 입력값에 따라 출력값 설정을 따로 해줍니다.

- FSM top module

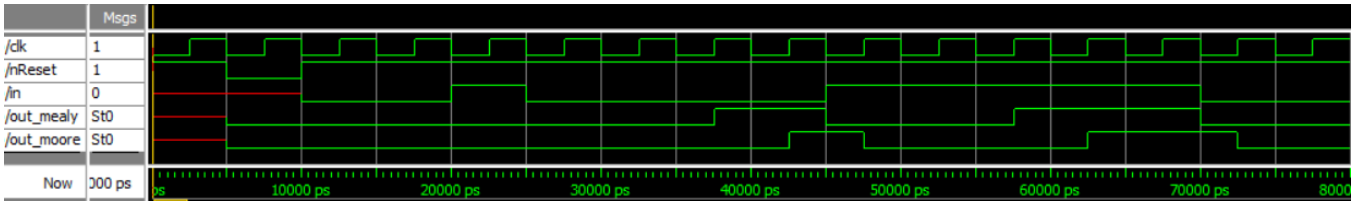
```
module FSM(nReset, clk, in, out_mealy, out_moore);

   input clk, nReset, in;
   output out_mealy, out_moore;

   MealyMachine mealy(.out(out_mealy), .nReset(nReset), .clk(clk), .in(in));
   MooreMachine moore(.out(out_moore), .nReset(nReset), .clk(clk), .in(in));


endmodule
```

FSM Top module에서는 앞서 구현한 밀리 머신과 무어 머신을 호출하여, 각 입력에 따라 밀리머신의 출력과 무어머신의 출력이 어떻게 다른지 비교할 수 있도록 합니다.

- FSM testbench

```
`timescale 1ns/100ps

module FSM_tb;
   reg clk, nReset, in;
   wire out_mealy, out_moore;

   always
      #2.5 clk = ~clk;

   FSM myFSM(.nReset(nReset), .clk(clk), .in(in), .out_mealy(out_mealy), .out_moore(out_moore));

   initial begin
      clk = 1'b0;
      nReset = 1'b1;
      #5 nReset = 1'b0;
      #5 nReset = 1'b1;

      in = 1'b0;
      #10 in = 1'b1;
      #5 in = 1'b0;
      #20 in = 1'b1;
      #25 in = 1'b0;
      #10
      $finish;

   end
endmodule
```

Clk 신호는 매 2.5ns 간격마다 값이 변화하도록 만들어줍니다. 과제에 주어진 Timing diagram의 입력을 그대로 넣어줍니다.



시뮬레이션 실행 시 다음과 같은 결과가 나옵니다. 무어 머신은 밀리 머신에 비해서 출력이 한 cycle 느린 것을 확인할 수 있습니다.

4. Evaluation

예상한 Timing diagram과 동일한 결과가 나왔으므로, 올바른 설계를 하였다고 판단했습니다.

5. Discussion

이번 실습의 key는 밀리 머신과 무어 머신의 상태전이도를 올바르게 설계하는 것이라고 생각합니다.

무어 머신이 밀리 머신보다 출력이 한 cycle 느리다는 점이 흥미로웠습니다.