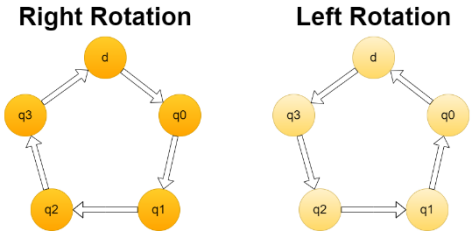
SOC Design

한양대학교 공과대학 컴퓨터소프트웨어학부

최가온(학번: 2019009261)

[Homework 1] Blocking and Non-blocking Assignments

[Github Link] https://github.com/Gaon-Choi/ITE4003/tree/main/Week06/blk



아래의 4개 구현 모두 right rotation의 성질을 띠고 있다. 그러나, blocking과 non-blocking의 차이에 따라 그것의 결과(schematic, waveform, …)가 다르게 나타난다.

- blk1 vs. blk2

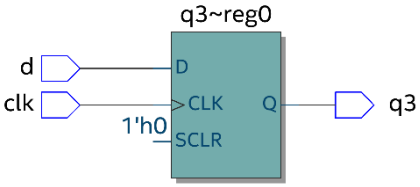


Figure 1. Result of blk1

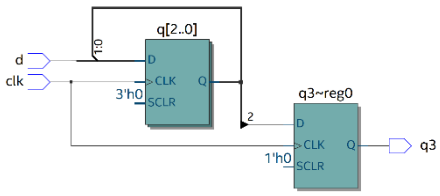


Figure 2. Result of blk2

- non_blk1 vs. non_blk2

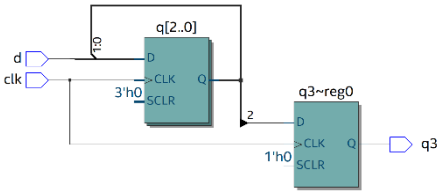


Figure 3. Result of non_blk1

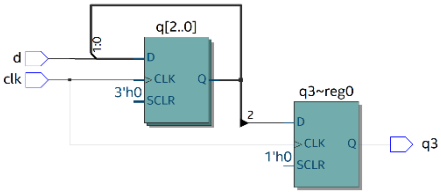


Figure 4. Result of non_blk2

blk1의 경우 윗부분부터 순차적으로 실행되기에 d의 값이 d → q0 → q1 → q2 → q3을 따라 q3에 저장된다. 반면, blk2의 경우 가장 윗부분에서 q3에 q2의 값이 저장되고 그 이후 부분은 결과에 직접적인 영향을 미치지는 않는다. non_blk1과 non_blk2는 4줄의 코드가 동시에 실행된다. 따라서 두 코드는 결론적으로 같은 코드가 된다. 위에 첨부한 Right Rotation이 한꺼번에 일어난다고 가정하면 된다. 즉, q3는 q2의 값을 받게 되므로 blk2, non_blk1, non_blk2는 같은 결과(같은 schematic)를 나타내게 된다.

[Homework 2] Moore FSM

[Github Link] https://github.com/Gaon-Choi/ITE4003/tree/main/Week06/fsm

```
module fsm(clk, rst, bypss, out);
    input clk, rst;
    input bypss;
    output reg [1:0] out;
    parameter ST0 = 2'b00, ST1 = 2'b01, ST2 = 2'b10, ST3
= 2'b11;

    reg [1:0] state, next;

    always @(posedge clk or negedge rst) begin
        // state register block
        if(!rst) state <= ST0;
        else        state <= next;
    end

    always @(state or bypss) begin
        // next state logic block
        case(state)
            ST0:  next = ST1;
            ST1:    if(bypss == 1)  next = ST3;
                    else    next = ST2;
            ST2:    next = ST3;
```

```
            ST3:    next = ST0;
        endcase
    end

    always @(posedge clk or negedge rst) begin
            // output logic block
            if(!rst) out <= 2'b00;
            else begin
                case(next)
```

```
                    ST0: out <= 2'b00;
                    ST1: out <= 2'b01;
                    ST2: out <= 2'b10;
                    ST3: out <= 2'b11;
                endcase
            end
    end
endmodule
```

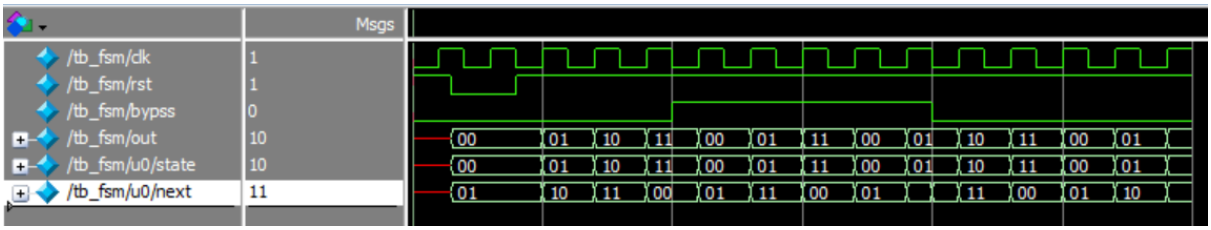


Figure 5. Waveform of Moore Machine

[Homework 3] Mealy FSM

[Github Link] https://github.com/Gaon-Choi/ITE4003/tree/main/Week06/mealy_fsm

```
module mealy_fsm(clk, rst, din_bit, dout_bit);
    input clk, rst, din_bit;
    output dout_bit;

    reg [2:0] state, next;

    parameter start = 3'b000, st1 = 3'b001, st2 = 3'b010, st3 = 3'b011, st4 = 3'b100;

    always @(posedge clk or negedge rst) begin
        // state register block
        $display (state, dout_bit);
        if(!rst) state <= start;
        else state <= next;
    end

    always @(state or din_bit) begin
        // next state logic block
        case(state)
            start:
                if (din_bit == 0)           next <= st1;
                else if(din_bit == 1)   next <= start;
                else                            next <= start;
```

```
            st1:
                if (din_bit == 0)           next <= st1;
                else if(din_bit == 1)   next <= st2;
                else                            next <= start;
            st2:
                if (din_bit == 0)           next <= st1;
                else if(din_bit == 1)   next <= st3;
                else                            next <= start;
            st3:
                if (din_bit == 0)           next <= st4;
                else if(din_bit == 1)   next <= start;
                else                            next <= start;
            st4:
                if (din_bit == 0)           next <= st1;
                else if(din_bit == 1)   next <= st2;
                else                            next <= start;
            default:                        next <= start;
        endcase
    end

    // output logic block
    assign dout_bit = (state == st3 && din_bit == 0) ? 1 : 0;
endmodule
```

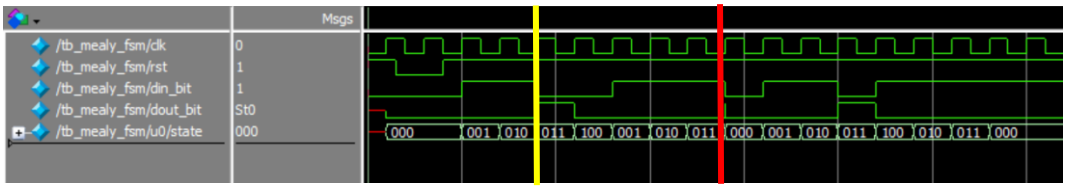


Figure 6. Waveform of Mealy Machine

무어 머신은 현재의 state에만 의존하며, 밀리 머신은 현재의 state뿐만 아니라 현재의 입력에도 의존한다. 노랑, 빨강으로 표시된 부분을 각각 A, B(Glitch)라 하자. A와 B에서 clock은 모두 posedge이다. 또한 rst의 값도 모두 1이며 din_bit도 negedge이다. A 부분 이전에는 state=2이고 din_bit=1인 상황이었다. 이전과 이후를 나누어 생각해보면 din_bit은 1→0이므로 assign문 조건의 절반은 만족한다. 또 state=2→3이므로 이 또한 assign문 조건이 참이 되어 결국 두 요인 모두 assign문이 참이 되게끔 만든다.

반면, B 부분의 경우 din_bit이 1→0으로 가는 것은 assign문 조건을 참이 되게 만드나, state=3→0이므로 이는 assign문이 거짓이 되는 요인이다. 따라서 AND 조건으로 묶여있는 assign문의 참/거짓은 달라지는 것이다. Glitch는 이처럼 여러 입력요인이 작용할 때, 입력이 직접적으로 출력에 영향을 주는 구조를 갖기에 발생하는 현상이다.