Logic Design 2nd term project

2018320192 손재민

1. Project summary

a. Overview : 이 프로그램은 최대 20명의 사람들이 줄을 설 수 있고, 한번에 반드시 8명의 사람이 탑승해야 하는 놀이기구를 구현한 프로그램이다. 보드 상에서 SW[4]부터 SW[0]까지가 입력 값에 해당하는데 SW[0]만 켜져 있는 경우 4명이, SW[1]만 켜져 있는 경우 8명이, SW[2]만 켜져 있는 경우 12명이 줄을 서는 것이고, SW[3]만 켜져 있는 경우 8명을 놀이기구에 태운 것이며, SW[4]만 켜져 있는 경우는 리셋에 해당한다. 또 보드에 내장된 CLOCK_50을 클럭으로 사용한다. HEX3은 현재 줄 서있는 사람들의 수로 총 몇 번의 놀이기구를 운영할 수 있는지, HEX1과 HEX0은 현재 줄 서있는 사람들의 수를 각각 10의 자리와 1의 자리로 출력한다.

b. Modules

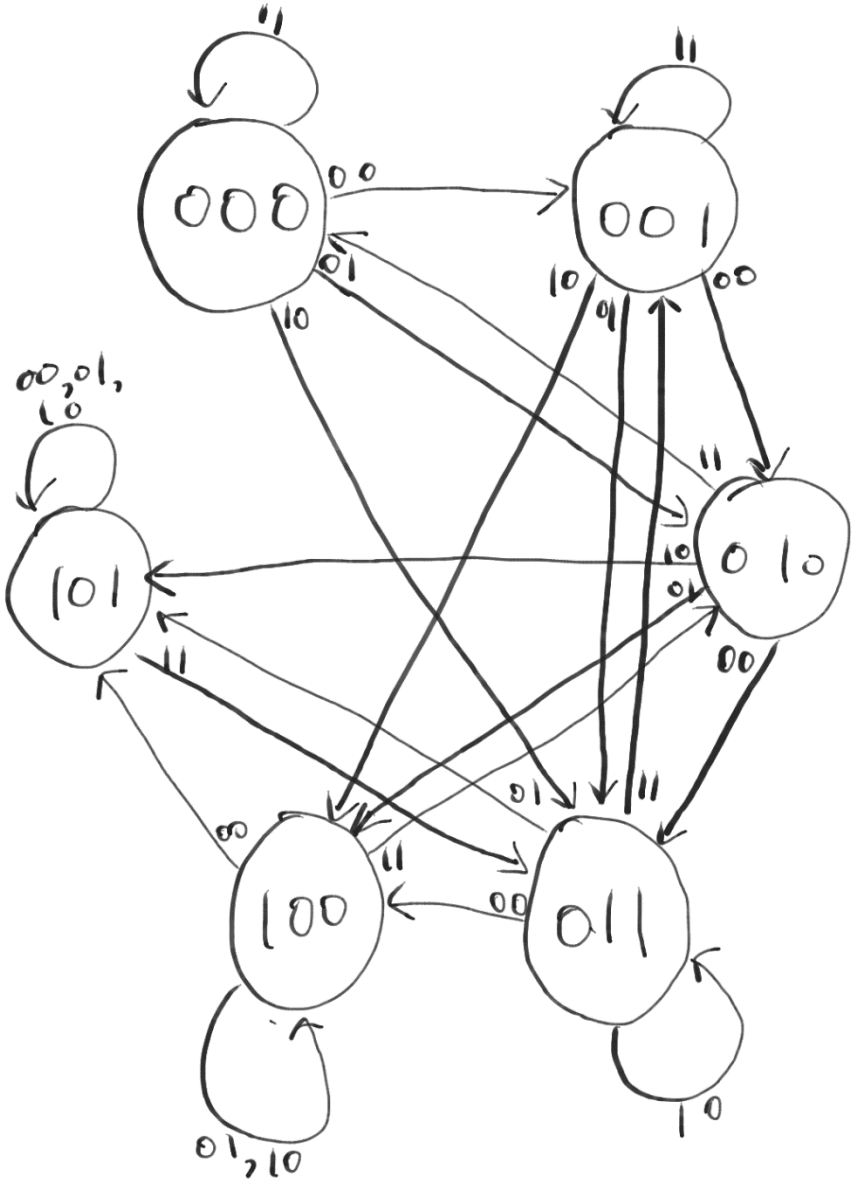
1) clock: 입력 값으로 클럭 clk를 입력 받는다. clk의 매 positive edge 마다 카운트를 세어 clk가 2^22번 positive edge가 될 때마다 0이면 1로, 1이면 0으로 값을 바꿔 newClk에 출력한다.

2) finalterm : 입력 값으로 각각 SW[4]부터 SW[0]까지 각각 놀이기구에 대한 입력을 받는다. 이를 각각 input_sw로 변환하는데, input_sw의 값은 각각 SW[0]이 켜져 있으면 000, SW[1]이 켜져 있으면 001, SW[2]가 켜져 있으면 010, SW[3]이 켜져 있으면 011, SW[4]가 켜져 있으면 100이 된다. SW 중에서 하나의 값만 켜져 있으면 isInput을 1로, 이외의 모든 경우에는 0으로 만든다. 또, 클럭 CLOCK_50과 모듈 clock을 이용해 newClk을 구한다. isInput의 positive edge마다, 만약 input_sw[2]가 1이면, 즉 SW[4]만 켜져 있으면 next_a와 next_b, next_c를 0으로 만든다. 만약 input_sw[2]가 0이면, next_a, next_b, next_c를 D-flip flop을 이용하여 state diagram과 k map을 통해 구한 식을 이용해 값을 저장한다. newClk의 positive edge 마다 a와 b, c에 각각 next_a, next_b, next_c에 저장된 값을 저장한다. 출력 부분에 대해서는, {a, b, c}를 확인하여 각 state에 맞게 HEX1과 HEX0을 출력하며, 만약 state가 0 또는 4라면 HEX3을 0으로, 8 또는 12라면 HEX3을 1으로, 16 또는 20이라면 HEX3을 2로 출력한다.

2. State Diagram

*각 State는 현재 줄 서고 있는 사람의 수를 의미한다. 각각 000이 0명, 001이 4명, 010이 8명, 011이 12명, 100이 16명, 101이 20명에 해당된다.

*각 input은 각각 00이 4명, 01이 8명, 10이 12명이 줄을 서기 위해 오는 것에, 11이 명이 놀이기구에 탑승한 것에 해당된다.



3. HDL code

```
module logic_design(input [4:0] SW, input CLOCK_50, output reg [0:6] HEX0,HEX1,HEX3);
//newClk is for new, slow clock.
//isInput is to check if the input is wrong or not.
wire newClk, isInput;
//input_sw is a 3 bit conversion of input SW.
wire [2:0] input_sw;
//a, b, c is current state, next_, next_b, next_c is next state.
reg a, b, c, next_a, next_b, next_c;

parameter state_0 = 3'b000; parameter state_4 = 3'b001; parameter state_8 = 3'b010;
parameter state_12 = 3'b011; parameter state_16 = 3'b100; parameter state_20 = 3'b101;
parameter Seg9 = 7'b000_1100; parameter Seg8 = 7'b000_0000; parameter Seg7 = 7'b000_1111;
parameter Seg6 = 7'b010_0000; parameter Seg5 = 7'b010_0100; parameter Seg4 = 7'b100_1100;
parameter Seg3 = 7'b000_0110; parameter Seg2 = 7'b001_0010; parameter Seg1 = 7'b100_1111; parameter Seg0
= 7'b000_0001;

//Make slow clock.
clock(CLOCK_50, newClk);
```

```
//Convert SW to 3 bit input_sw.
assign input_sw[0] = SW[3]|SW[1];
assign input_sw[1] = SW[3]|SW[2];
assign input_sw[2] = SW[4];
//Check if only one of SW is on at the same time.
assign                                          isInput                                                 =
(~SW[4]&~SW[3]&~SW[2]&~SW[1]&SW[0])|(~SW[4]&~SW[3]&~SW[2]&SW[1]&~SW[0])|(~SW[4]&~SW[3]&SW[2]&~SW[1]&~S
W[0])|(~SW[4]&SW[3]&~SW[2]&~SW[1]&~SW[0])|(SW[4]&~SW[3]&~SW[2]&~SW[1]&~SW[0]);

//Set initial next_a, next_b, next_c to 0.
initial
begin
        next_a = 1'b0;
        next_b = 1'b0;
        next_c = 1'b0;
end

//When positive edge of newClk, set a, b, c to new_a, new_b, new_c.
always @(posedge newClk)
begin
        a <= next_a; b <= next_b; c <= next_c;
end

//When positive edge of isInput
always @(posedge isInput)
begin
        //If input_sw[2] is on, then reset to 0.
        if(input_sw[2])
        begin
                next_a <= 1'b0; next_b <= 1'b0; next_c <= 1'b0;
        end
        //If input_sw[2] is off, then set the value using k-map.
        else
        begin
                next_a                                                                                  <=
(a&~input_sw[1])|(a&~input_sw[0])|(b&~input_sw[1]&input_sw[0])|(b&c&~input_sw[1])|(~b&c&input_sw[1]&~i
nput_sw[0])|(b&~c&input_sw[1]&~input_sw[0]);
                next_b                                                                                  <=
(a&input_sw[1]&input_sw[0])|(~a&~b&~input_sw[1]&input_sw[0])|(~a&~b&c&~input_sw[1])|(b&~c&~input_sw[1]
&~input_sw[0])|(b&c&input_sw[1]&~input_sw[0])|(~a&~b&~c&input_sw[1]&~input_sw[0]);
                next_c                                                                                  <=
(c&input_sw[0])|(a&c)|(~a&~c&~input_sw[0])|(~c&~input_sw[1]&~input_sw[0])|(b&input_sw[1]&~input_sw[0]);
        end
end

always @(*)
begin
        //Print HEX0, HEX1, and HEX3 by {a, b, c}.
        case({a, b, c})
                state_0: begin HEX3 = Seg0; HEX1 = Seg0; HEX0 = Seg0; end
                state_4: begin HEX3 = Seg0; HEX1 = Seg0; HEX0 = Seg4; end
                state_8: begin HEX3 = Seg1; HEX1 = Seg0; HEX0 = Seg8; end
                state_12: begin HEX3 = Seg1; HEX1 = Seg1; HEX0 = Seg2; end
                state_16: begin HEX3 = Seg2; HEX1 = Seg1; HEX0 = Seg6; end
                state_20: begin HEX3 = Seg2; HEX1 = Seg2; HEX0 = Seg0; end
        endcase
end
endmodule

//Make slow clock using clk.
module clock(input clk, output newclk);
        reg [24:0]cnt;

        //When positive edge of clk, add cnt 1.
        always@(posedge clk)
        begin
                cnt <= cnt + 1'b1;
        end
        //For every 2^22 counts, invert the output.
        assign newclk = cnt[22];
endmodule
```

4. Troubleshooting process

a) 본 프로젝트를 진행하면서 겪었던 어려움으로는, 이전에 프로그램을 작성할 때는 크게 딜레이를 생각할 필요가 없었으나, 본 프로젝트는 클럭에 따라 작동하는 순서가 중요해서 딜레이를 고려하지 않고 작성했더니 경우에 따라 게이트 딜레이 때문에 원하는 순간에 값들이 시간차를 갖고 설정이 되어 오류가 계속 발생했다. 무작정 reg를 사용하면 딜레이가 커지므로 가급적 wire를 이용하여 프로그램을 작성하는 것이 딜레이를 줄이는 것에 효과적이라는 사실을 배울 수 있었다.