Logic Design 2nd Term Project Report

-Waiting for the ride in an amusement park-

2018320205 컴퓨터학과 신대성

# Project Summary

OS: Windows 10  
Tool: Quartus II 9.1sp2, FPGA Board

놀이기구를 타기 위해 줄을 서는 것을 시뮬레이트한 코드입니다.

대기 인원은 4명, 8명, 12명 단위로 증가하며, 최대 20명까지 대기할 수 있습니다. 이 대기 인원은 우측의 HEX0, HEX1에 표시됩니다.

한번 놀이기구를 타기 위해서는 8명 이상이 존재해야 하며, 8명씩만 탑승합니다. 탑승 가능한 횟수는 좌측의 HEX3에 표시됩니다.

더해서 20명이 되는 경우, 8명 미만인데 놀이기구를 타려는 경우 입력이 무시됩니다.

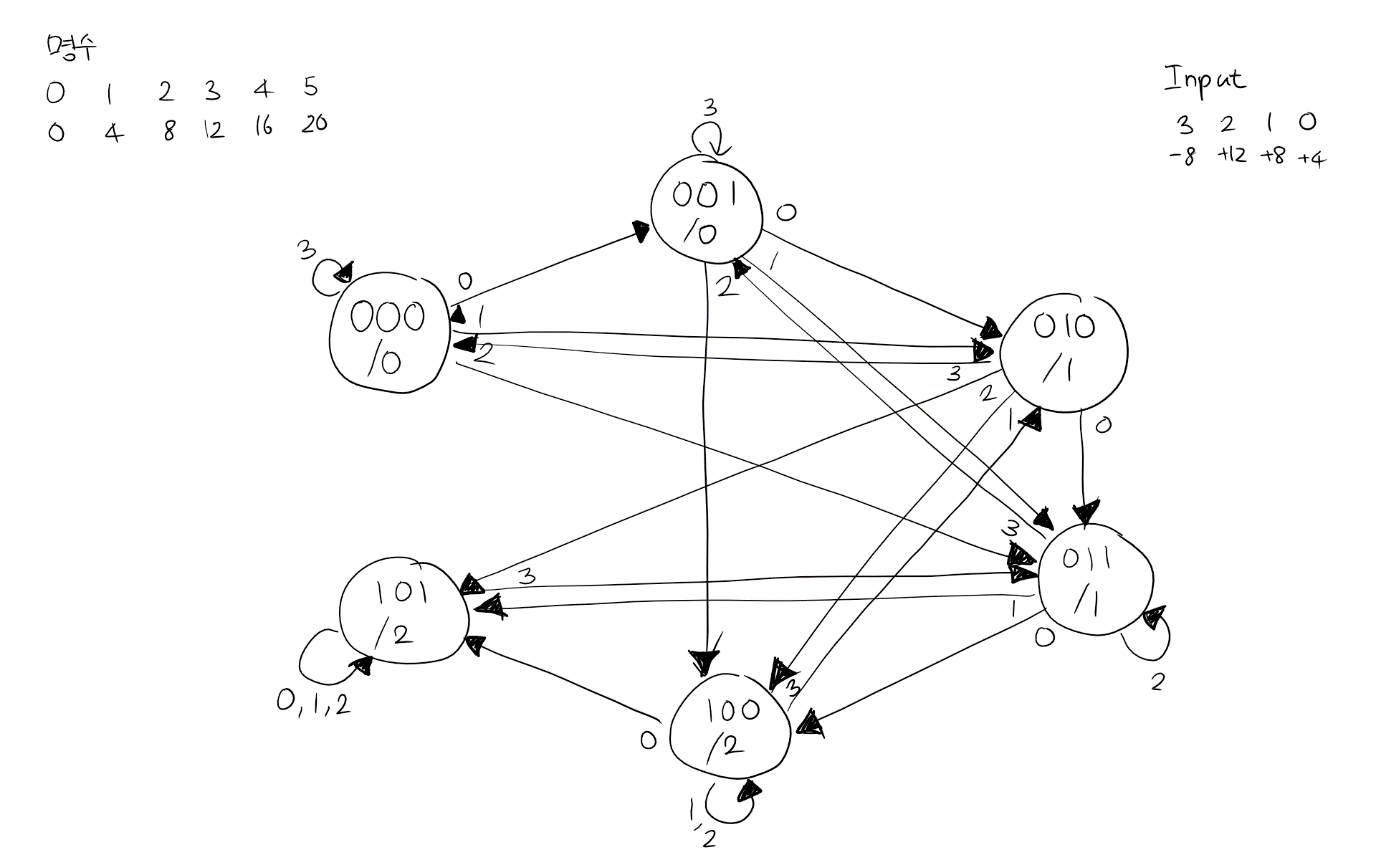
Input:

SW0: 대기인원 +4명  
 SW1: 대기인원 +8명  
 SW2: 대기인원 +12명  
 SW3: 8명 탑승(대기인원 -8명)  
 SW4: Reset

Output:

HEX0, HEX1: 대기인원 표시  
 HEX3: 탑승 가능한 횟수

# State Diagram



각 명수에 대한 state를 나열하면 총 6가지의 경우가 나오는데 이를 3bit를 가지는 state로 변환할 수 있습니다. input은 SW의 번호를 사용했습니다.

# HDL Code

module logic\_design(SW, CLOCK\_50, HEX0, HEX1, HEX3, LEDG);

input [4:0] SW; // 4: Reset, 3: -8(drive), 2: +12, 1: +8, 0: +4

input CLOCK\_50;

output reg [0:6] HEX0, HEX1; // count of waiting people

output reg [0:6] HEX3; // count of how many amuse can be driven

wire newclock; // slowed clock

wire noInput; // if there are no input, this is 1

reg stayIn; // check if there have same input with before

wire [1:0] in; // converted Input

reg [2:0] cS; // current State

reg [2:0] nextState; // next State

// constant values

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;

parameter SegErr = 7'b111\_1111;

convert4to2(SW[3:0], in[1:0], noInput);

newClk(CLOCK\_50, newclock);

initial

begin

cS = 3'b000; // inital state

stayIn = 1'b0;

end

always@(\*)

begin // nextState made with current State and input

nextState[2] <= cS[2]&~in[1] | cS[2]&~in[0] | cS[1]&~in[1]&in[0] | cS[1]&cS[0]&~in[1] | ~cS[1]&cS[0]&in[1]&~in[0] | cS[1]&~cS[0]&in[1]&~in[0];

nextState[1] <= cS[2]&in[1]&in[0] | ~cS[2]&~cS[1]&~in[1]&in[0] | ~cS[2]&~cS[1]&cS[0]&~in[1] | cS[1]&~cS[0]&~in[1]&~in[0] | cS[1]&cS[0]&in[1]&~in[0] | ~cS[2]&~cS[1]&~cS[0]&in[1]&~in[0];

nextState[0] <= cS[0]&in[0] | cS[2]&cS[0] | ~cS[2]&~cS[0]&~in[0] | ~cS[0]&~in[1]&~in[0] | cS[1]&in[1]&~in[0];

end

always@(\*)

begin

case(cS) // make output(HEX3) to be decided by currentState.

0: begin HEX0 = Seg0; HEX1 = SegErr; HEX3 = Seg0; end // 0

1: begin HEX0 = Seg4; HEX1 = SegErr; HEX3 = Seg0; end // 4

2: begin HEX0 = Seg8; HEX1 = SegErr; HEX3 = Seg1; end // 8

3: begin HEX0 = Seg2; HEX1 = Seg1; HEX3 = Seg1; end // 12

4: begin HEX0 = Seg6; HEX1 = Seg1; HEX3 = Seg2; end // 16

5: begin HEX0 = Seg0; HEX1 = Seg2; HEX3 = Seg2; end // 20

default: begin HEX0 = SegErr; HEX1 = SegErr; HEX3 = SegErr; end // error

endcase;

end

always@(posedge newclock)

begin

if (SW[4]) begin // reset

cS <= 3'b000;

end

else if (!noInput & !stayIn) begin

cS <= nextState;

stayIn <= 1'b1;

end

else if (stayIn & noInput) stayIn <= 1'b0; // switch downed

end

endmodule

// slow clock

module newClk(in, out);

input in;

output reg out;

reg [23:0] nclk;

always@(posedge in)

begin

nclk <= nclk + 1;

out <= nclk[23];

end

endmodule

// convert 4 bit input to 2 bit

module convert4to2(in, out, err);

input [3:0] in;

output reg [1:0] out;

output reg err;

always@(\*)

begin

case(in)

4'b1000: begin err = 1'b0; out = 2'b11; end

4'b0100: begin err = 1'b0; out = 2'b10; end

4'b0010: begin err = 1'b0; out = 2'b01; end

4'b0001: begin err = 1'b0; out = 2'b00; end

default: begin err = 1'b1; out = 2'b00; end // input was 0000 or other

endcase

end

endmodule

# Troubleshooting process

처음에 구상하면서 3비트의 state와 4비트의 input을 처리하는데, input을 2비트로 만들어도 5개의 변수에 대해 k-map을 그렸어야 했다. 이부분이 좀 귀찮은 부분이었다고 생각한다.

하지만 가장 힘들었던 점은 클럭 관리였다. 처음에는 critical path에 대해 생각하지 않고 CLOCK\_50의 posedge에만 state가 변하게 만들었었다. 그런데 여러모로 이상한 반응들이 일어나서, 어디에 문제가 있는지 오랜 시간 고민했었다. slow clock을 이용해 클럭을 느리게 만들어 critical path보다 길게 만들어 해결했다.