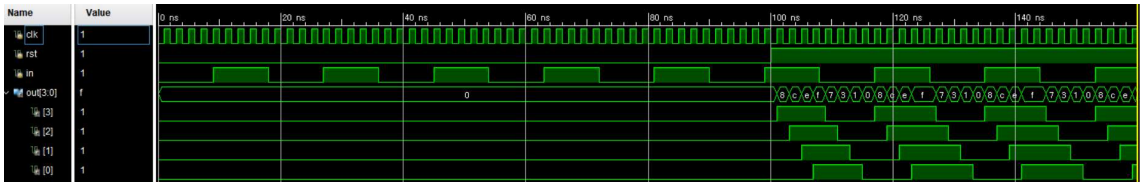
13주차 결과보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.




```
`timescale 1ns / 1ps

module shift(
input clk, rst, in,
output reg [3:0] out
);

initial begin
out = 4'b0;
end

always@(posedge clk) begin
  if(rst == 0)
    out <= 4'b0;
  else begin
    out <= out >> 1;
    out[3] <= in;
  end
end
endmodule
```

```
`timescale 1ns / 1ps

module shift_tb;
reg clk, rst, in;
wire [3:0] out;

shift test(
  .clk(clk),
  .rst(rst),
  .in(in),

  .out(out)
);

initial begin
  clk = 1'b0;
  rst = 1'b0;
  in = 1'b0;
  #160
  $finish;
end

always@(clk or rst or in) begin
  rst <= #100 ~rst;
  clk <= #1 ~clk;
  in <= #9 ~in;
end

endmodule
```

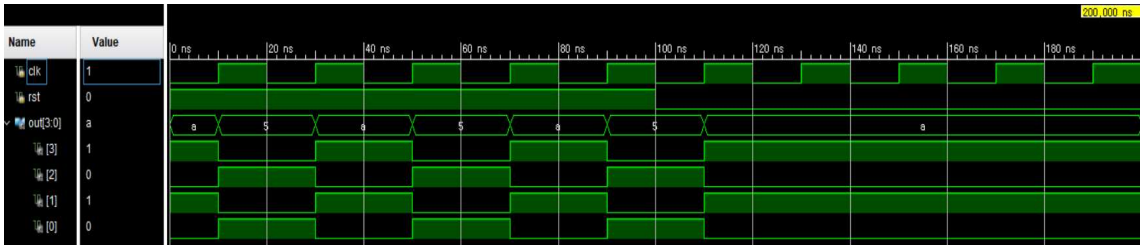
Shift Register OUTPUT Table					
Number of Clock Transitions	INPUTS	OUTPUTS			
	IN	out[3]	out[2]	out[1]	out[0]
1	0	0	0	0	0
2	1	1	0	0	0
3	0	0	1	0	0
4	1	1	0	1	0
5	0	0	1	0	1
6	1	1	0	1	0
7	1	1	1	0	1

Shift Register는 여러 플립플롭이 직렬로 연결되어 있는 형태입니다. clk의 postive edge마다 data가 전단의 ff에서 후단의 ff으로 이동합니다. input으로는 clk, rst과 별도 input인 in을 선언했습니다. output으로는 output reg형으로 out 4개를 배열로 선언하였습니다. 이전 실습에서처럼 처음 상태를 모르기 때문에 simulation 결과로 초기 결과가 x로 표현되는 것을 막기 위해 편의상 초기 output 4개를 모두 0으로 설정해줍니다.

동작으로는 clk이 positive edge일 때마다 out을 우측으로 1bit씩 shift합니다. 들어오는 별도 input인 in을 가장 처음 ff인 out[3]에 넣어줍니다. rst이 0인 경우에는(fpga보드의 rst버튼이 active Low이기 때문에) 모든 out에 0을 넣어주며 초기화합니다.

시뮬레이션 결과를 확인해보자면 초기에는 초기상태로 설정해두었던 0000이 나오고, in으로 1이 들어와 1 0 0 0인 8, 또 계속 1이 들어와 1 1 0 0인 c, 1 1 1 0인 e, 1 1 1 1인 f와 같은 순서로 clock의 positive edge마다 output이 1 bit씩 shift 되어 출력되는 모습을 확인할 수 있습니다.

2.




```
`timescale 1ns / 1ps

module ring(
input clk, rst,
output reg [3:0] out
);

initial begin
out = 4'b1010;
end

always@(posedge clk) begin
  if (rst == 0)
    out <= 4'b1010;
  else begin
    out <= out >> 1;
    out[3] <= out[0];
  end
end
endmodule
```

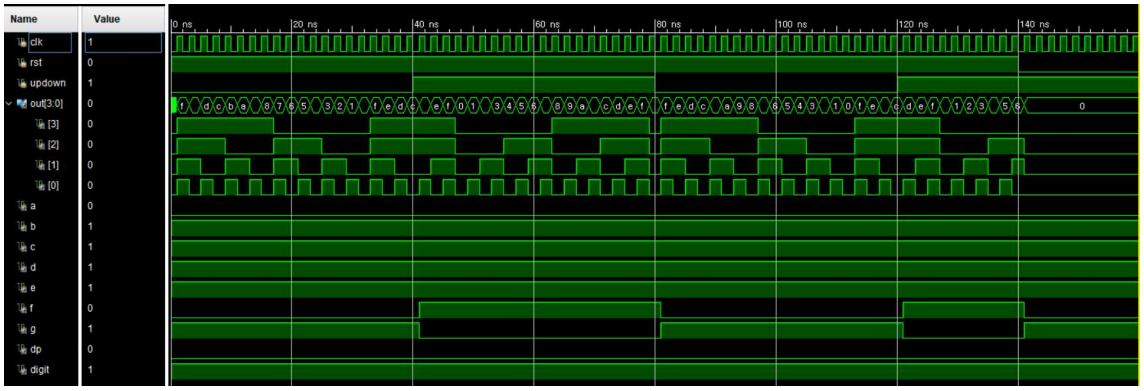
```
`timescale 1ns / 1ps

module ring_tb;
reg clk, rst;
wire [3:0]out;

ring test(
  .clk(clk),
  .rst(rst),

  .out(out)
);

initial begin
  clk = 1'b0;
  rst = 1'b1;
  #200
  $finish;
end

always@(clk or rst) begin
  rst <= #100 ~rst;
  clk <= #10 ~clk;
end
endmodule
```

Ring Counter OUTPUT Table				
Number of Clock Transitions	OUTPUTS			
	out[3]	out[2]	out[1]	out[0]
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0
4	0	1	0	1
5	1	0	1	0
6	0	1	0	1
7	1	0	1	0

Ring Counter는 여러 D flip flop이 직렬로 연결되어 있는 형태인 것은 Shift register와 비슷하지만, 가장 마지막 flip flop의 output인 가장 처음 flip flop의 input으로 들어가기 때문에 별도의 input이 필요하지 않습니다. 또한 마찬가지로 clk의 postive edge마다 data가 전단의 ff에서 후단의 ff으로 이동합니다. input으로는 clk, rst만을 선언했습니다. output으로는 output reg형으로 out 4개를 배열로 선언하였습니다. 이전 실습에서처럼 처음 상태를 모르고 별도의 input이 없어 data를 추가할 수 없습니다. 그렇기에 초기 값을 설정하지 않는다

면 제대로 동작하는지 확인할 수 없고 그렇기에 편의상 초기 output 4개를 임의로 1010으로 설정해줍니다. 정상 동작으로는 clk이 positive edge일 때마다 out을 우측으로 1bit씩 shift합니다. 또한, 가장 처음 ff인 out[3]에 가장 마지막 ff인 out[0]을 넣어줍니다. rst이 0인 경우에는(fpga보드의 rst버튼이 active Low이기 때문에) 임의로 설정했던 초기 값인 1010을 넣어주며 초기화합니다.
시뮬레이션 결과를 확인해보자면 초기에는 초기상태로 설정해두었던 1010이 나오고, clk의 positive edge마다 우측으로 1 bit씩 shift 되어 0101, 1010 이런 순서로 반복됩니다. 또한, rst 신호가 0이 되면 다시 초기값인 1010으로 초기화 되는 모습을 확인할 수 있습니다.

3 .



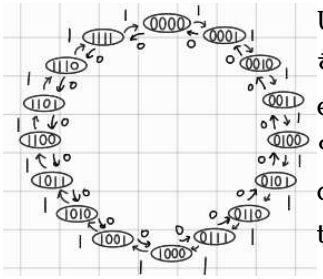

```
`timescale 1ns / 1ps
module updown(
input clk, rst, updown,
output reg [3:0] out,
output reg a, b, c, d, e, f, g, dp, digit
);
initial begin
  a = 0;
  b = 1;
  c = 1;
  d = 1;
  e = 1;
  f = 0;
  g = 1;
  dp = 0;
  digit = 1;
  out = 4'b0;
end
always@(posedge clk) begin
  if(rst == 0) begin
    out <= 0;
    f <= 0;
    g <= 1;
  end
  else if(updown) begin
    // up mode
    f <= 1;
    g <= 0;
    out <= out + 1;
  end
  else begin
    // down mode
    f <= 0;
    g <= 1;
    out <= out - 1;
  end
end
endmodule
```

```
`timescale 1ns / 1ps

module updown_tb;
reg clk, rst, updown;
wire [3:0] out;
wire a, b, c, d, e, f, g, dp, digit;

updown test(
.clk(clk), .rst(rst), .updown(updown),

.out(out), .a(a), .b(b), .c(c), .d(d),
.e(e), .f(f), .g(g), .dp(dp), .digit(digit)
);

initial begin
clk = 1'b0;
rst = 1'b1;
updown = 1'b0;
#160
$finish;
end

always@(clk or rst or updown) begin
rst <= #140 ~rst;
clk <= #1 ~clk;
updown <= #40 ~updown;
end
endmodule
```

다음 장에 계속

UP Counter OUTPUT Table					
Number of Clock Transitions	OUTPUTS				
	out[3]	out[2]	out[1]	out[0]	DISPLAY
1	0	0	0	1	U
2	0	0	1	0	U
3	0	0	1	1	U
4	0	1	0	0	U
5	0	1	0	1	U
6	0	1	1	0	U
7	0	1	1	1	U

DOWN Counter OUTPUT Table					
Number of Clock Transitions	OUTPUTS				
	out[3]	out[2]	out[1]	out[0]	DISPLAY
1	0	1	1	1	d
2	0	1	1	0	d
3	0	1	0	1	d
4	0	1	0	0	d
5	0	0	1	1	d
6	0	0	1	0	d
7	0	0	0	1	d



UP / DOWN counter는 clk과 reset 신호 값 말고도 입력 신호를 하나 더 받습니다. 해당 신호의 입력 값에 따라 clock의 postive edge마다 현재 상태에서 1을 증가시키거나 1을 감소시키는 카운터입니다. 우리가 이번 실험에서 구현하려는 4 bit UP / DOWN counter의 경우 좌측과 같은 state diagram과 상단의 output table을 가집니다.

input으로는 clk, rst, updown을 선언했습니다. output으로는 output reg형으로 out 4개를 배열로 선언하고 7 segment에 출력하기 위한 output을 선언합니다. 편의상 초기 output 4개를 임의로 0000으로 설정해주고 segment display에 d가 출력될 수 있도록 합니다.

우선 rst 상태가 아닐 시 정상 동작으로는 clk이 positive edge일 때마다 updown 신호로 1이 들어오면 up mode로 out을 1씩 추가해줍니다. 또한, segment display에 U가 출력되어야 하기 때문에 이는 기존의 d인 상태에서 segment display의 f와 g만 바꿔주면 됩니다. 반면 updown 신호가 0이면 down mode이기 때문에 out을 1씩 감소시켜줍니다. 또한 d가 출력되면 되기 때문에 f에는 0 g에는 1을 넣어주면 됩니다.

시뮬레이션 결과를 확인해보자면 초기에는 초기상태로 설정해두었던 0000이 나오고, updown 신호가 0이기 때문에 clk의 positive edge마다 1씩 감소하여 1111, 1110 ... 0000까지 감소합니다. updown 신호가 1이면 0000부터 1111까지 증가하는 양상을 보입니다. 또한, rst 신호가 0이 되면 다시 초기값인 0000으로 초기화 되는 모습을 확인할 수 있습니다.

4.

해당 실험을 통해 Shift register와 ring counter가 비슷하면서 어떻게 다른지, 그리고 Updown counter에 대해 총체적으로 이해할 수 있었습니다. 또한, verilog 코드에서 full adder 같은 것이 필요 없이 그렇나 연산을 시프트 연산자 “<<”, “>>”나 “+” 연산자를 이용해 구현할 수 있다는 것도 학습하게 되었습니다. 또한, 7 segment display를 구현할 때 dp 변수에 1을 할당해야지 사용이 가능하다는 것을 기억하게 되었습니다.

5.

Johnson Ring Counter

마지막 플립플롭의 출력을 반전시켜 가장 처음 플립플롭의 입력으로 넣어줍니다. 만약, n개의 플립플롭이 존재한다면 2 * n개의 상태 표현이 가능합니다. 주파수 분배기 혹은 디지털 시계처럼 정확한 타이밍 요구될 때 위상 교대 정사각형 파를 생성합니다.