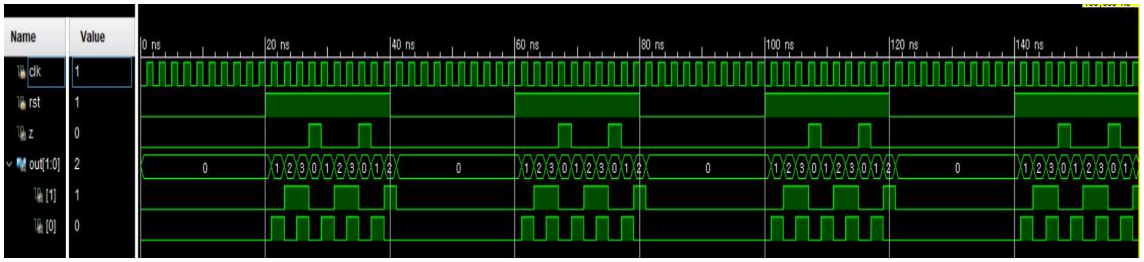
12주차 결과보고서

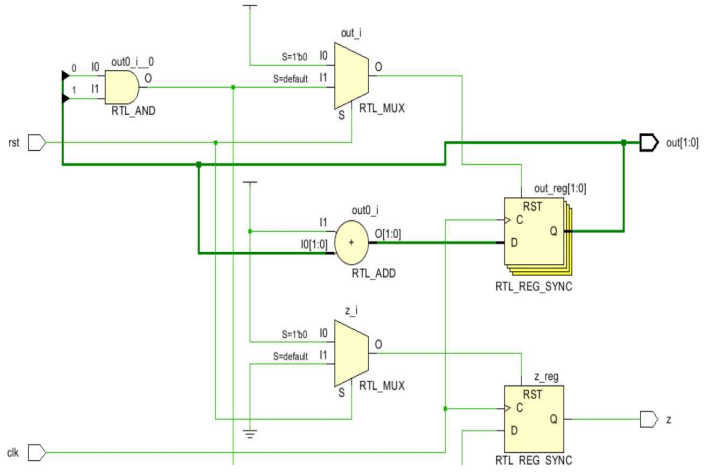
전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.



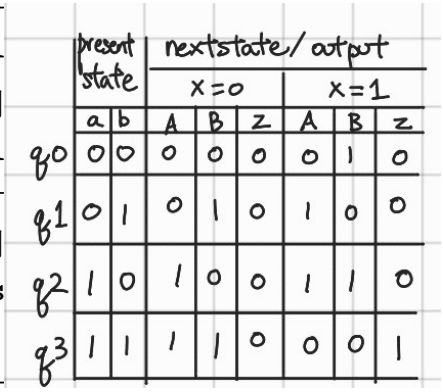

```
`timescale 1ns / 1ps

module twoBitCounter(
  input clk, rst,
  output reg z, reg [1:0] out
);

initial begin
  out = 2'b0;
  z = 1'b0;
end

always @(posedge clk) begin
  if (rst == 0) begin
    out <= 2'b0;
    z <= 1'b0;
  end

  else begin
    if((out[0] && out[1])) begin
      // output a, b가 둘 다 1, 1일 경우
      z <= 1'b1;
      out <= 2'b0;
    end

    else begin
      out <= out+1;
      z <= 1'b0;
    end
  end
end

endmodule
```

```
`timescale 1ns / 1ps

module twoBitCounter_tb;
reg clk, rst;
wire z;
wire [1:0] out;

twoBitCounter test(
  .clk(clk),
  .rst(rst),

  .z(z),
  .out(out)
);

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

always@(clk or rst) begin
  rst <= #20 ~rst;
  clk <= #1 ~clk;
end

endmodule
```



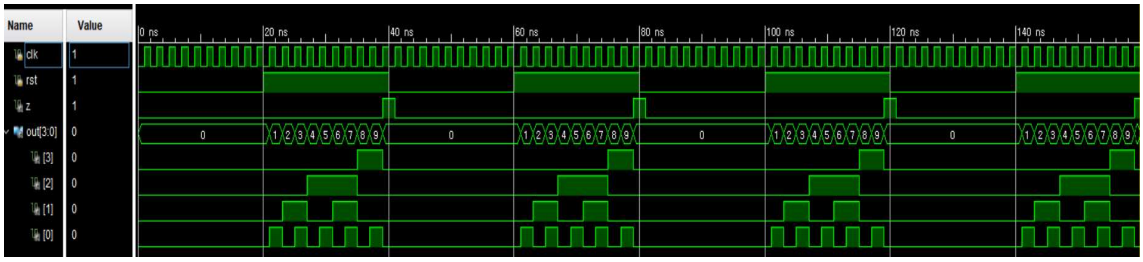
설명은 다음 장에 이어서

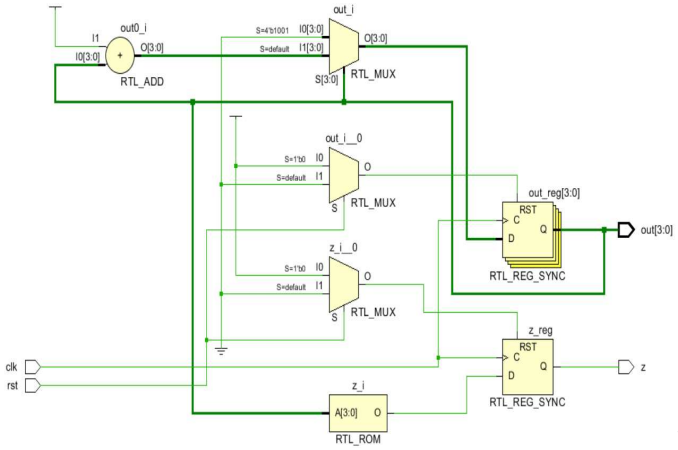
2bit counter를 구현하기 위해서 input으로 clk과 rst을 output으로 cycle을 한 번 돌 때마다 1을 출력할 z와 output 2개를 배열로 선언해줍니다. 이대로는 처음 상태를 모르기 때문에 simulation 결과로 초기 결과가 x로 표현되는 것을 막기 위해 편의상 초기 output 3개를 모두 0으로 설정해줍니다. clk에 따라 동작해야 하기 위해, 회로를 설계할 때 always 문을 사용해야 하는데 이 안에 있는 변수는 reg형이어야 합니다. 그렇기에 일반 output이 아니라 output reg로 설정해줍니다.



clk이 positive edge일 때만 동작해야 하기 때문에 always@(posedge clk)문 안에서 회로 동작을 설계해줍니다. 본래 rst이 1일 때 동작해야하지만 FPGA 보드 상에서 rst button이 active low이기 때문에 rst이 0인 경우에는 out과 z를 0으로 초기화시켜줍니다. 이제 rst 값이 0이 아닌 경우에 대해(일반적인 동작을 하는 경우) 설정을 해줘야합니다. out이 둘 다 1인 경우에는 다시 처음 상태로 돌아가야 하기 때문에 out은 0으로 초기화해주고, z는 1이 나오게 만듭니다. 그 이외에는 out에 1 증가한 값과 z에는 0을 대입해줍니다. 그 결과 rst이 1인 상태에서는 clk 신호가 계속 들어오면 00, 01, 10, 11로 증가하다가 z값으로 1이 출력이 되고 00으로 돌아가는 것을 확인할 수 있었습니다. 또한, rst이 0이면 모두 0으로 초기화됩니다.

2.





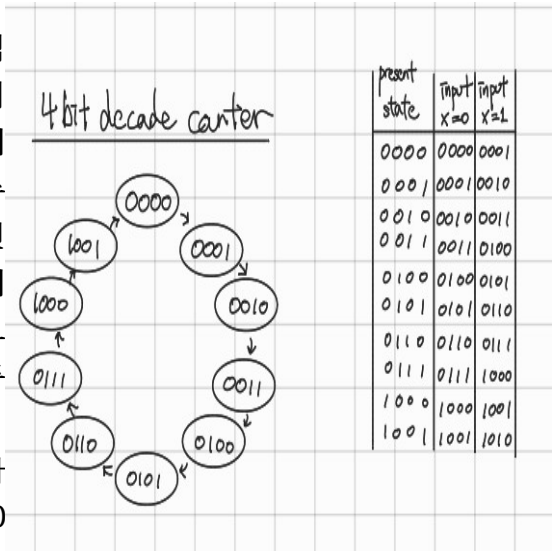
다음장에 계속

```
`timescale 1ns / 1ps

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

initial begin
  out = 4'b0;
  z = 1'b0;
end

always @(posedge clk) begin
  if (rst == 0) begin
    out <= 4'b0;
    z <= 1'b0;
  end

  else begin
    if(out == 9) begin
      z <= 1'b1;
      out <= 4'b0;
    end

    else begin
      out <= out+1;
      z <= 1'b0;
    end
  end
end

endmodule
```

```
`timescale 1ns / 1ps

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

decadeCounter test(
  .clk(clk),
  .rst(rst),

  .z(z),
  .out(out)
);

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

always@(clk or rst) begin
  rst <= #20 ~rst;
  clk <= #1 ~clk;
end

endmodule
```

4 bit decade counter를 구현하기 위해서 input으로 clk과 rst을 output으로 cycle을 한 번 돌 때마다 1을 출력할 z와 output 4개를 배열[3:0] out으로 선언해줍니다. 전에 했던 것과 마찬가지로 이대로는 처음 상태를 모르기 때문에 simulation 결과로 초기 결과가 x로 표현되는 것을 막기 위해 편의상 초기 output 5개를 모두 0으로 설정해줍니다. clk에 따라 동작해야 하기 위해, 회로를 설계할 때 always문을 사용해야 하는데 이 안에 있는 변수는 reg형이어야 합니다. 그렇기에 일반 output이 아니라 output reg로 설정해줍니다.

clk이 positive edge일 때만 동작해야 하기 때문에 always@(posedge clk)문 안에서 회로 동작을 설계해줍니다. 본래 rst이 1일 때 동작해야 하지만 FPGA 보드 상에서 rst button이 active low이기 때문에 rst이 0인 경우에는 out과 z를 0으로 초기화시켜줍니다.

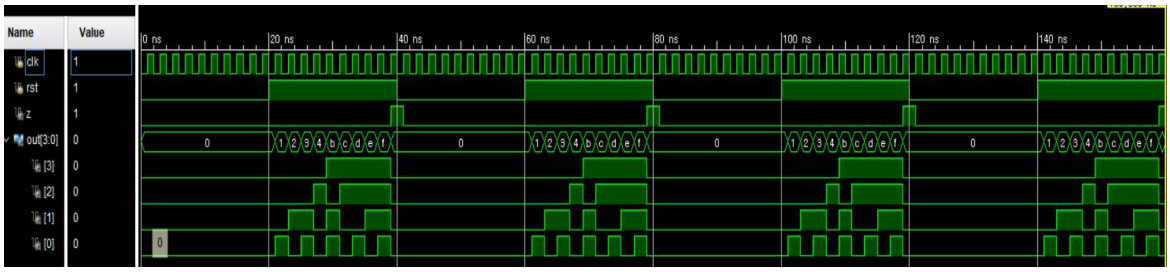


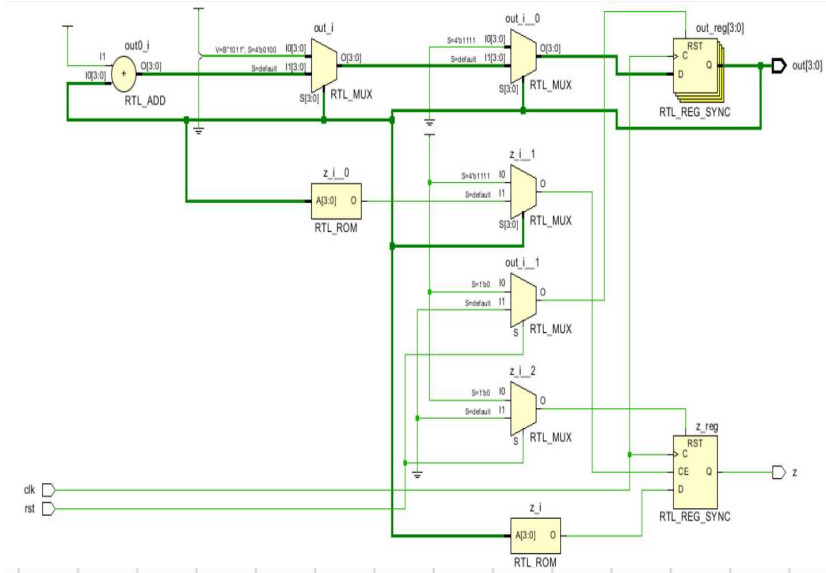
이제 rst 값이 0이 아닌 경우에 대해(일반적인 동작을 하는 경우) 설정을 해줘야 합니다. out이 1001인 경우에는 다시 처음 상태로 돌아가야 합니다. 1001인 상태는 10진수로 변환했을 때 9이기 때문에 out == 9인 경우로 분기를 하여 out은 0으로 초기화해주고, 한 사이클이 끝났으므로 z는 1이 나오게 만듭니다. 그 외의 경우에는 out에 1 증가한 값과 z에는 0을 대입해줍니다.

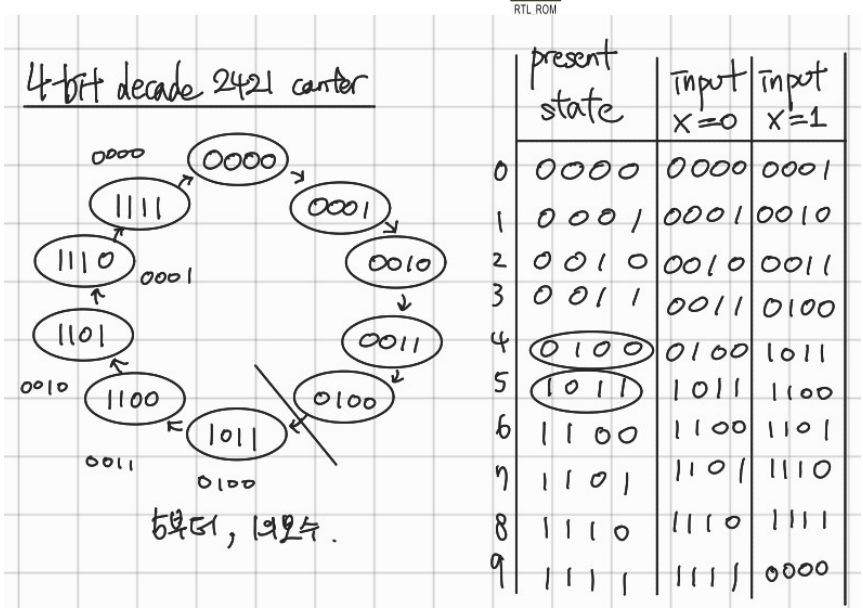
그 결과 rst이 1인 상태에서는 clk 신호가 계속 들어오면 0000, 0001, 0010, 0011,0

0100, 0101, 0110, 0111, 1000, 1001으로 증가하다가 z값으로 1이 출력이 되고 0000으로 돌아가는 것을 확인할 수 있었습니다. 또한, rst이 0이면 모두 0으로 초기화됩니다.

3.







다음 장에 계속

```
`timescale 1ns / 1ps

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

initial begin
  out = 4'b0;
  z = 1'b0;
end

always @(posedge clk) begin
  if (rst == 0) begin
    out <= 4'b0;
    z <= 1'b0;
  end

  else begin
    if(out == 15) begin
      z <= 1'b1;
      out <= 4'b0;
    end
    else if (out == 4) begin
      out <= ~out;
    end
    else begin
      out <= out+1;
      z <= 1'b0;
    end
  end
end

endmodule
```

```
`timescale 1ns / 1ps

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

decadeCounter test(
  .clk(clk),
  .rst(rst),

  .z(z),
  .out(out)
);

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

always@(clk or rst) begin
  rst <= #20 ~rst;
  clk <= #1 ~clk;
end

endmodule
```

4 bit decade 2421 counter를 구현하기 위해서 input으로 clk과 rst을 output으로 cycle을 한 번 돌 때마다 1을 출력할 z와 output 4개를 배열[3:0] out으로 선언해줍니다.

전에 했던 것과 마찬가지로 이대로는 처음 상태를 모르기 때문에 simulation 결과로 초기 결과가 x로 표현되는 것을 막기 위해 편의상 초기 output 5개를 모두 0으로 설정해줍니다.

clk에 따라 동작해야 하기 위해, 회로를 설계할 때 always문을 사용해야 하는데 이 안에 있는 변수는 reg형이어야 합니다. 그렇기에 일반 output이 아니라 output reg로 설정해줍니다. clk이 positive edge일 때만 동작해야 하기 때문에 always@(posedge clk)문 안에서 회로 동작을 설계해줍니다.

본래 rst이 1일 때 동작해야 하지만 FPGA 보드 상에서 rst button이 active low이기 때문에 rst이 0인 경우에는 out과 z를 0으로 초기화시켜줍니다.

이제 rst 값이 0이 아닌 경우에 대해(일반적인 동작을 하는 경우) 설정을 해줘야 합니다. out이 0100인 경우에는 out 4개가 1의 보수화되어야 합니다. 0100인 상태는 10진수로 변환했을 때 4이기 때문에 out == 4인 경우로 분기를 하여 out <=~out 문을 통해 out이 반전되게 만들어줍니다. 한편, out이 1111인 상태는 10진수로 변환했을 때 15이기 때문에 out == 15인 경우로 분기를 합니다. out은 모두 0으로 초기화해주고, 한 사이클이 끝났으므로 z는 1이 나오게 만듭니다. 그 외의 경우에는 out에 1이 증가한 값과 z에는 0을 대입해줍니다.

그 결과 rst이 1인 상태에서는 clk 신호가 계속 들어오면 0000, 0001, 0010, 0011,0

0100까지 가다가 분기문을 만나, 반전되어 1011, 1100, 1101, 1110, 1111이 출력되다가 z값으로 1이 출력이 되고 0000으로 돌아가는 것을 확인할 수 있었습니다. 또한, rst이 0이면 모두 0으로 초기화됩니다.

4.
"set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]".라는 문을 새로 추가했습니다. 정확히는 잘 모르지만 clk input 포트를 fpga보드의 clk이 아닌 일반 스위치에 할당하면서 발생하는 에러를 방지하기 위해서라고 이해했습니다. 또한, 이 줄 때문에 비바도 synthesis 및 implementation과정에서 같은 디자인 소스 파일, xdc 파일이어도 실패하는 경우가 빈번히 발생하여 비효율적인 시간 소요가 발생했습니다. 이 원인에 대해 단순히 컴퓨터 사양 혹은 비바도 버전의 오류인지 정확한 인과관계를 확인할 필요성이 있다고 느꼈습니다. 또한 이 실험을 통해 단순히 업다운 카운터가 아니라 BCD코드 등을 응용한 다양한 종류의 카운터를 만들 수 있다는 것을 깨닫게 되었습니다.

5.
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
이 문장은 우리가 실험을 할 때 일반 스위치를 클록신호로 사용하기 때문에 사용합니다. CLOCK_DEDICATED_ROUTE FALSE라는 문을 통해 이러한 라우팅 제약을 완화시킬 수 있습니다. IBUF는 input buffer의 약자로 외부 신호가 입력 버퍼를 통과할 때 프로그램으로 인해 자동으로 만들어지는 내부 net 이름입니다.

output reg[3:0] out과 output [3:0] out의 차이
전자의 경우는 포트가 레지스터 타입이라는 것을 의미합니다. Always 블록에서 직접 값을 할당 가능하고 플립플롭을 구현할 때 레지스터에 값이 저장되어야 하기 때문에 사용합니다. 후자의 경우는 포트가 와이어 타입이라는 것을 의미하고 combinational logic일 때 사용하며 즉각적인 신호 전달이 주 목적입니다.