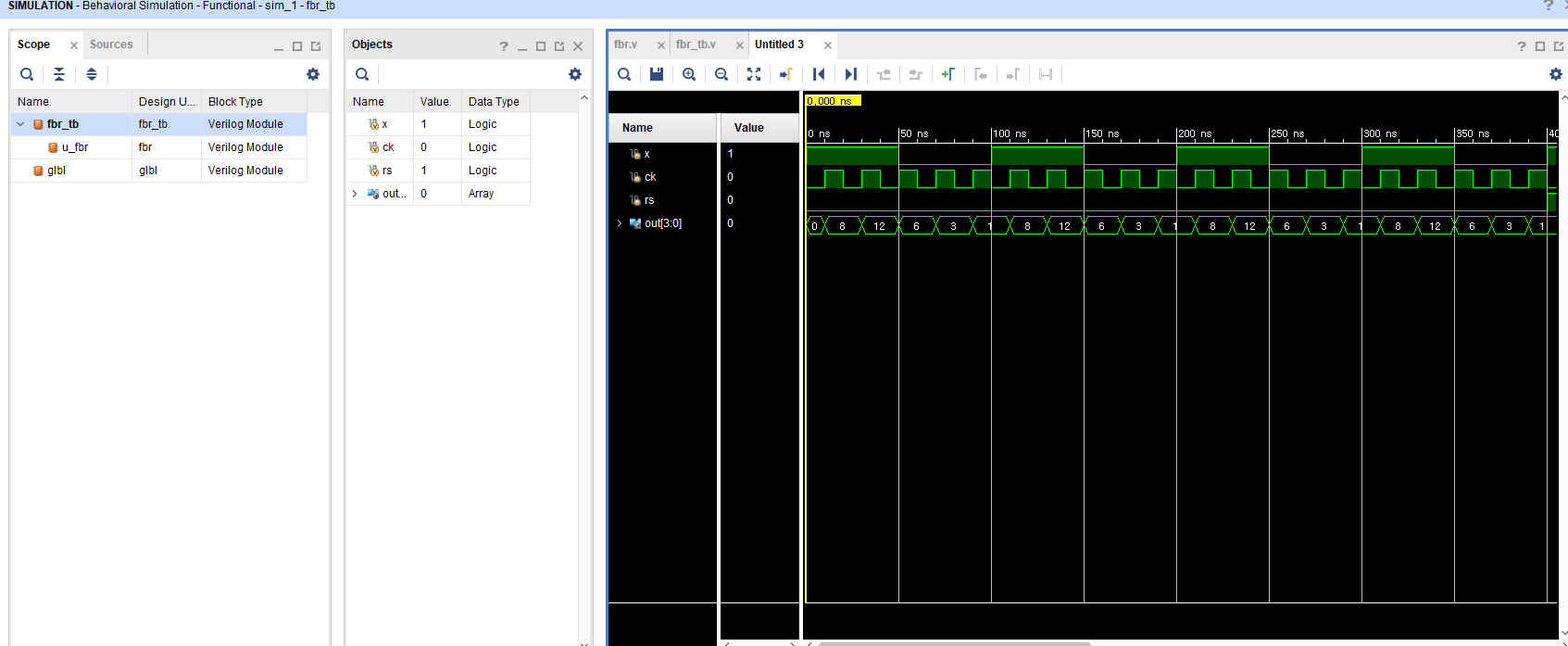
13주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20221549 이름: 김효림

**1.**

.................



4-bit shift register의 simulation 결과

|  |  |
| --- | --- |
| Source code | Simulation code |
| module fbr(  ck,rs,out,x  );    output [3:0] out;  reg [3:0] out;    input ck;  input rs;  input x;    initial out = 4'b0000;    always @(posedge ck ) begin  if(rs == 1'b1) out = 4'b0000;  else begin  out = out >> 1;  out[3] = x;  end  end      endmodule | module fbr\_tb;  reg x, ck, rs;  wire [3:0] out;  fbr u\_fbr(  .x(x),  .ck(ck),  .rs(rs),  .out(out)  );  initial begin  ck = 1'b0;  rs = 1'b0;  x = 1'b1;  end  always ck = #10~ck;  always x = #50 ~x;  initial rs <= #400 ~rs;  endmodule |

4-bit shift register의 Verilog Source code

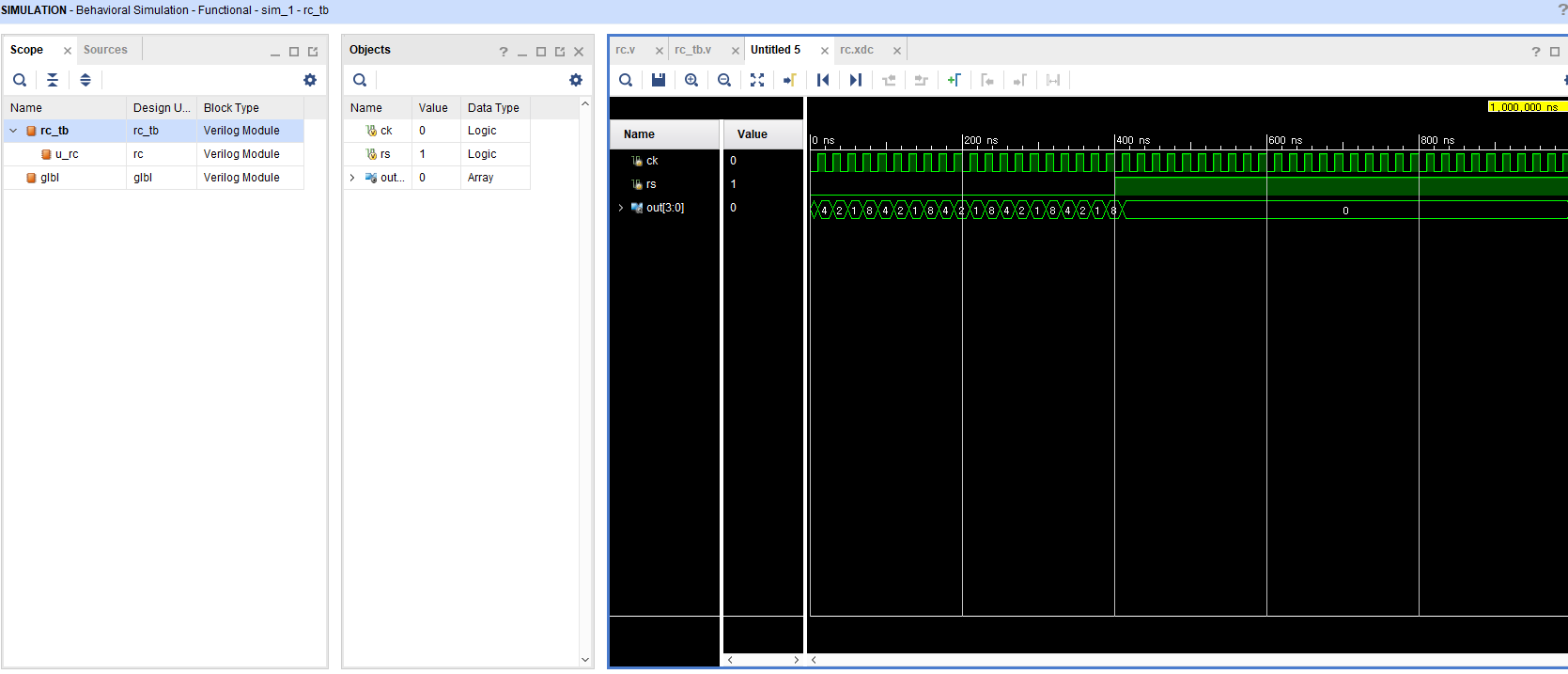
Input 값인 ck는 clock을 의미하고, rs는 리셋 버튼을 의미한다. 즉, positive edge일 때 reset 버튼이 눌렸을 경우에는 out 값을 0000으로 설정하고, 그렇지 않다면 오른쪽으로 1씩 이동하고 out[3] 자리에는 input 값을 대입해 준다. 따라서 simulation 결과를 통해 확인할 수 있듯, 0(0000, initial value) -> 8(1000) -> 12(1100) -> 6(0110) -> 3(0011) -> 1(0001) -> 8(1000) -> ⋯

로 움직이는 것을 확인할 수 있다.

................

**2.**

.......................



4-bit ring counter의 simulation 결과(제출 화면은 rs 시 0000으로 set했을 때)

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

아래 코드에 따라 rs 시 1000로 set 하도록 simulation 재출력

|  |  |
| --- | --- |
| Source code | Simulation code |
| module rc(  ck, rs, out  );    input ck;  input rs;  output [3:0] out;  reg [3:0] out;    initial out = 4'b1000;    always @(posedge ck) begin  if (rs == 1'b1)  out <= 4'b1000;  else begin  out[3] <= out[0];  out[2] <= out[3];  out[1] <= out[2];  out[0] <= out[1];  end  end  endmodule | module rc\_tb;  reg ck, rs;  wire [3:0] out;  rc u\_rc(  .ck(ck),  .rs(rs),  .out(out)  );  initial begin  ck = 1'b0;  rs = 1'b0;  end  always ck = #10~ck;  initial rs <= #400 ~rs;  endmodule |

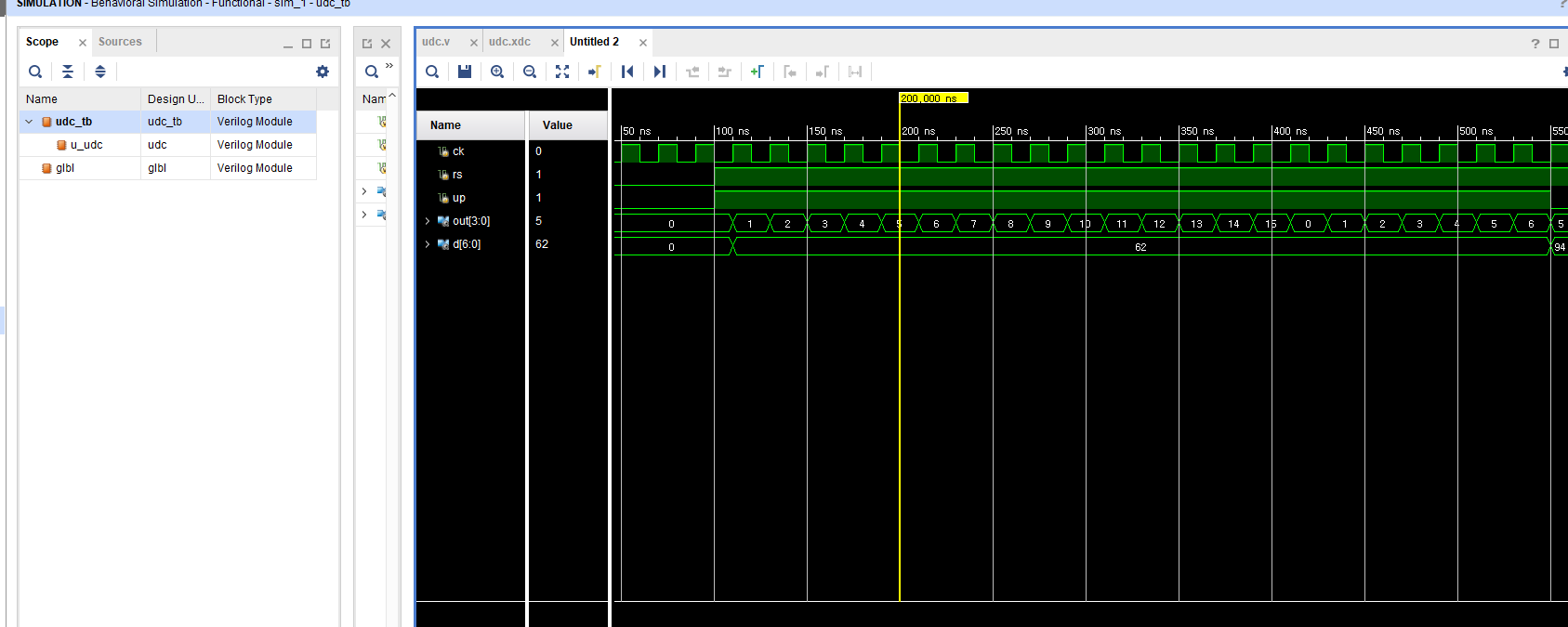
4-bit ring counter의 Verilog Source code

Input 값인 ck는 clock을, rs는 reset을 의미한다. 즉 positive edge일 때 rs 버튼이 눌리면 output 값에 다시 1000을 대입하고, 그 외에는 out[3]에 out[0](비트 수)를, out[2]에 out[3]을, out[1]에 out[2]를 out[0]에 out[1] 값을 대입한다. 즉, 오른쪽으로 자리를 한 칸 씩 옮겨주고, LSB는 다시 MSB 위치로 옮겨주어 순환하는 형태로(링 카운터) 만들어 준다. Simulation 코드를 확인하면, 처음 초기값이 1000에서 positive edge를 만날 때마다 8(1000) -> 4(0100) -> 2(0010) -> 1(0001) -> 8(1000) -> ⋯ 로 움직이는 걸 확인할 수 있다. 또한 reset 버튼이 눌리면 설정한 값으로 초기화되는 것을 확인할 수 있다.

.........................

**3.**

.......................



4-bit up/down counter의 simulation 결과

|  |  |
| --- | --- |
| Source code | Simulation code |
| module udc(  ck,rs,up,out,d,ds  );    input ck;  input rs;  input up;  output [3:0] out;  output [7:0] d;  reg[3:0] out;  reg[7:0] d;  output ds;    always@(posedge(ck)) begin  if(!rs) begin  out <= 0;  d<=0;  end  else begin  if(up==1) begin  d[0] = 1'b0;  d[1] = 1'b1;  d[2] = 1'b1;  d[3] = 1'b1;  d[4] = 1'b1;  d[5] = 1'b1;  d[6] = 1'b0;  d[7] = 1'b0;      if(out == 15)  out<=0;  else  out <= out+1;  end    else begin  d[0] = 1'b0;  d[1] = 1'b1;  d[2] = 1'b1;  d[3] = 1'b1;  d[4] = 1'b1;  d[5] = 1'b0;  d[6] = 1'b1;  d[7] = 1'b0;    if(out == 0)  out <=15;  else  out <= out-1;  end  end  end    assign ds = d[0] | d[1] | d[2] | d[3] | d[4] | d[5] | d[6] | d[7];    endmodule | module udc\_tb;  reg ck, rs, up;  wire [3:0] out;  wire [6:0] d;  udc u\_udc(  .ck(ck),  .rs(rs),  .up(up),  .out(out),  .d(d)  );  initial ck = 0;  always #10 ck = ~ck;  initial begin  ck = 0;  rs = 0;  up = 0;    #100;  rs = 1;  up = 1;    #450  rs =1;  up = 0;  end  endmodule |

4-bit up/down counter의 Verilog Source code

Input에서 ck는 clock을, rs는 reset을 의미하며, up은 up/down 상태를 나타내기 위한 입력 값이다. D는 7 segment에 U와 d를 표기하기 위해 설정한 output 배열값으로, up이 1인 상태인 경우 u가 display 되고, up이 1이 아닌 경우 down counter가 되어 d가 display되도록 값을 부여했다. Out 값은 up counter일 경우 15 전까지 증가하고 15가 되면 0을 대입하도록, down counter는 감소하고 0이 되면 다시 15가 되도록 설계했다. Simulation 결과를 확인하면 설계한 대로 작동하는 것을 확인할 수 있었다.

.........................

**4.**

.......................

4-bit shift register, 4-bit ring counter, 4-bit up/down counter를 Verilog 코드로 작성하고 simulation 및 FPGA 보드를 통해 동작을 확인하며 작성한 코드가 의도대로 작동함을 확인할 수 있었다. 조건문이 아닌 flip flop으로도 구현할 수 있을지 생각해 보아야겠다.

.........................

**5.**

.......................

실습 시간에 다룬 카운터 외에도 그레이 코드 카운터라는 게 존재한다. 그레이 코드 카운터란 동시에 하나의 비트만 변하지 않게 코드를 할당하여 디코드 시의 스태틱 해저드(단시간의 스파이크장 잡음)을 방지하여 만든 것이다. 외에도 논 웨이팅 카운터가 있는데, 이는 각비트에 수치화 평가가 없는 것이다. 선형 피드백 시프트 레지스터라고 불리는 임의의 값을 적재한 플립 플롭의 비트열을 클럭 펄스마다 이웃의 플립플럽으로 이동해 가는 회로가 기본이 된다.

.........................