13주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231523 이름: 김민정

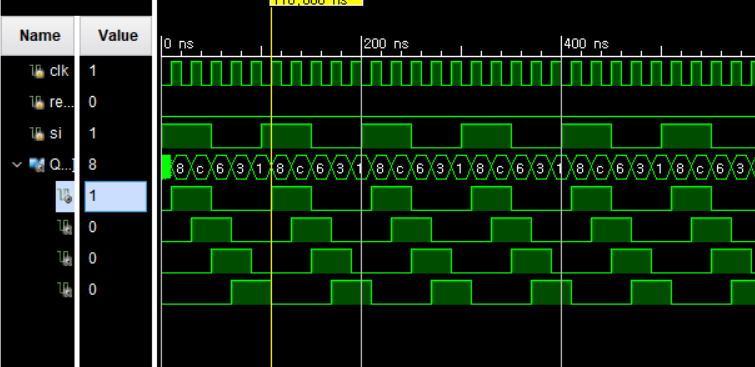
**1. 4-bit Shift Register(+ simulation)**

이전의 flip-flop(이후부터 flip-flop은 FF로 서술할 예정이다)의 value이 다음의 FF value로 전달해주는 shift register을 구현해보도록 하겠다. 4-bit이니 input은 clock과 처음 FF에 넣을 값인 S0가 필요하고 output은 4개의 FF가 필요하다. (output은 배열로 표현하였다.) 아래는 FF를 구현하기 위한 design code와 그에 따른 simulation 결과이다. 여기서 clk와 si는 앞에서 말했던 clock과 S0와 대응되고, reset은 FF의 현재 value을 모두 0으로 초기화하는 input이다. 마지막으로 [3:0] Q는 FF들의 value와 대응된다.

1) Verilog source

|  |  |
| --- | --- |
| Design code | Testbench code |
| `timescale 1ns / 1ps  module SR(  input clk, reset, si,  output [3:0] Q  );    reg [3:0] Q;  initial Q = 4'b0000;  always @(posedge clk) begin  if(reset) begin  Q = 4'b0000;  end  else begin  Q[0] <= Q[1];  Q[1] <= Q[2];  Q[2] <= Q[3];  Q[3] <= si;  end  end  endmodule | `timescale 1ns / 1ps  module SR\_tb;  reg clk, reset, si;  wire [3:0] Q;  SR u\_test(  .clk (clk ),  .reset (reset ),  .si (si ),    .Q (Q )  );    initial begin  clk = 1'b0;  reset = 1'b0;  si = 1'b1;  end  always clk = #10 ~clk;  always si = #50 ~si;  initial reset = #600 ~ reset;  endmodule |

2) Simulation



**2. 4-bit Ring counter (+ simulation)**

4-bit Ring counter는 0001->0010->0100->1000->0001->..이러한 결과값을 반복하는 장치이다. 즉, 하나의 FF에서만 state가 1이 되고 나머지 FF의 0이 되는 counter인인 것이. 이는 shift register처럼 따로 주입할 S0와 같은 input이 필요없기에 reset과 clock input만 존재하면 자연스럽게 FF 상태가 정해질 수 있다. 이도 역시 output은 FF의 state가 되고, 이러한 state들은 배열로 표현된다. 여기서 clk와 reset은 clock과 reset으로 대응되고, [0:3] Q는 FF의 state와 대응된다. 아래는 4-bit ring counter를 구현하기 위해 작성한 Verilog code이고, 그에 따른 simulation 결과이다.

1) Verilog source

|  |  |
| --- | --- |
| Design code | Testbench code |
| `timescale 1ns / 1ps  module RC(  input clk, reset,    output [3:0]Q  );    reg [3:0] Q;  initial Q = 4'b1000;  always @(posedge clk) begin  if(reset) begin  Q = 4'b1000;  end  else begin  Q[0] <= Q[1];  Q[1] <= Q[2];  Q[2] <= Q[3];  Q[3] <= Q[0];  end  end  endmodule | `timescale 1ns / 1ps  module RC\_tb;  reg clk, reset;  wire [3:0] Q;  RC u\_test(  .clk (clk ),  .reset (reset ),    .Q (Q )  );    initial begin  clk = 1'b0;  reset = 1'b0;  end  always clk = #10 ~clk;  always reset = #500 ~reset;  initial begin  #1000  $finish;  end  endmodule |

2) Simulation

스크린샷이(가) 표시된 사진

자동 생성된 설명

**3. 4-bit Up/Down counter (+ simulation)**

4-bit Up/Down counter은 0~15까지 Up, Down input에 따라 숫자 1을 변화시키는 counter이다. 만약 up input이 1이면 숫자 1을 증가시키고(down input은 0), down input이 1이면 숫자 1을 감소시킨다. 만약 up, down input이 모두 0이라면 현재 숫자를 유지시킨다. (up과 down input이 모두 1인 경우는 존재할 수 없다. 오류처리나 don’t care 처리를 함) 아래는 4-bit up/down counter을 제작하기 위해 작성한 Verilog code와 그에 따른 simulation 결과이다. 여기서 up=1일 때, up input이 활성화하고, up=0일 때 down input을 활성화하는 형식으로 up, down 방식을 구현했다. 나머지 clk과 reset, [3:0]Q는 위 1,2번 항목과 동일한 역할을 수행한다. 여기서 새로 보이는 것은 ss, d일 것이다. 이들은 단순히 up, down state임을 알려주기 위해 화면을 표시하는 수단이다.

1) Verilog source

|  |  |
| --- | --- |
| Design code | Testbench code |
| `timescale 1ns / 1ps  module UD(  input up,clk,reset,  output [3:0] Q,  output [6:0] ss,  output d  );  assign d = 1'b1;  reg [3:0] Q;  reg [6:0] ss;  initial Q = 4'b0000;  always @(posedge clk) begin  if(up) begin  if(reset)  Q =4'b0000;  else  Q <= Q + 1;  ss <= 7'b0111110;  end  else begin  if(reset)  Q =4'b0000;  else  Q <= Q - 1;  ss <= 7'b0111101;  end  end  endmodule | `timescale 1ns / 1ps  module UD\_tb;  reg clk, up,reset;  wire[3:0] Q;  wire[6:0] ss;    UD u\_test(  .clk (clk ),  .up (up ),  .reset (reset ),    .Q (Q ),  .ss (ss )  );    initial clk = 1'b0;  initial up = 1'b1;  initial reset = 1'b0;    always clk = #10 ~clk;  always up = #50 ~up;  always reset = #400 ~reset;    initial begin  #1000  $finish;  end  endmodule |

2) Simulation

**스크린샷, 회로이(가) 표시된 사진

자동 생성된 설명**

**4. 결과검토 및 논의사항**

4-bit shift register, 4-bit ring counter, 그리고 4-bit up, down counter을 제작하며 상승 에지에서 특정 역할을 수행하는 장치를 제작할 수 있었다. 위의 세 장치는 대부분의 경우에서 상승엣지 한번에 한 번의 state 변화가 일어났지만, 가끔 기기의 문제로 한번의 엣지 상승에 여러 번의 state 변화가 일어나는 문제가 일어났다. 하지만 이러한 문제는 여러 번의 epoch를 통해 예상 결과를 얻을 수 있었다.

**5. 기타이론**

위의 4-bit up/down counter을 제작할 때에는 up, down이 모두 0일 때 값을 유지하는 기능을 추가하지 못했다. 이를 코드를 추가해 기능을 추가해보도록 하겠다. 모두 1일때도 값을 유지하도록 설정하였다. 아래는 3번 기존 코드에 down 조건을 추가하여 다시 쓴 코드이며, 그 아래 사진은 변환한 코드의 simulation 사진이다. Down과 up이 동일한 value를 가지면 이전의 값이 유지됨을 확인할 수 있다.

1) Verilog source

|  |  |
| --- | --- |
| Design code | Testbench code |
| `timescale 1ns / 1ps  module UD(  input up,down,clk,reset,  output [3:0] Q,  output [6:0] ss,  output d  );  assign d = 1'b1;  reg [3:0] Q;  reg [6:0] ss;  initial Q = 4'b0000;  always @(posedge clk) begin  if(up&(!down)) begin  if(reset)  Q =4'b0000;  else  Q <= Q + 1;  ss <= 7'b0111110;  end  else if(down&(!up)) begin  if(reset)  Q =4'b0000;  else  Q <= Q - 1;  ss <= 7'b0111101;  end  else begin  if(reset)  Q =4'b0000;  else  ss <= 7'b1111110;  end  end  endmodule | `timescale 1ns / 1ps  module UD\_tb;  reg clk, up,reset,down;  wire[3:0] Q;  wire[6:0] ss;    UD u\_test(  .clk (clk ),  .up (up ),  .reset (reset ),  .down (down ),    .Q (Q ),  .ss (ss )  );    initial clk = 1'b0;  initial up = 1'b1;  initial down = 1'b0;  initial reset = 1'b0;    always clk = #10 ~clk;  always up = #50 ~up;  always down = #70 ~down;  always reset = #400 ~reset;    initial begin  #1000  $finish;  end  endmodule |

2) Simulation

**스크린샷, 회로이(가) 표시된 사진

자동 생성된 설명**