전공: 컴퓨터공학 학년: 2학년 학번: 20161603 이름: 신민준

1. 4-bit Shift Register의 결과 및 simulation 과정에 대해서 설명하시오.

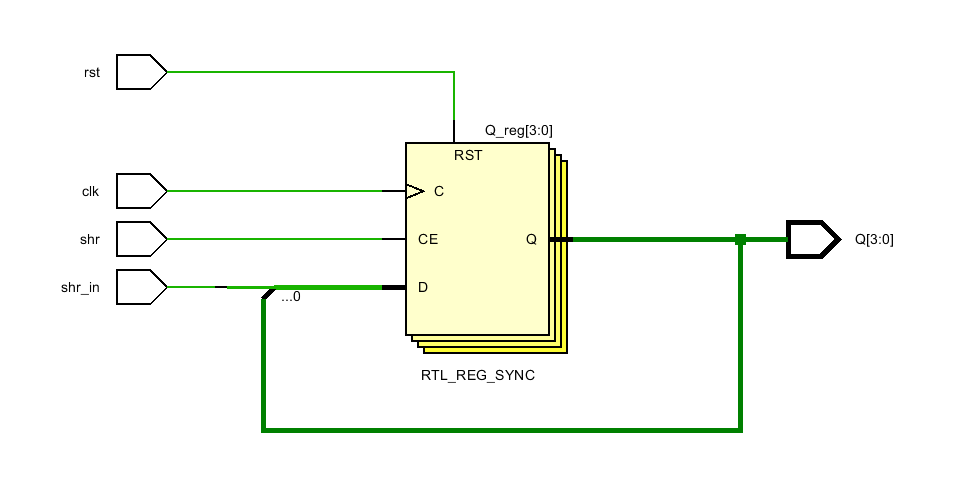
4-bit shift register는 4개의 Flip-flop이 직렬로 연결되어, 이전의 Flip-flop의 출력을 다음 Flip-flop의 입력으로 사용하는 형태를 구조했다. 이 때, 각 flip-flop의 index 번호는 순서대로 Q[3], Q[2], Q[1], Q[0]의 구조를 보이도록 구현했다.

각 Flip-flop은 leading edge에서 trigger되며, rst 신호가 들어오면 네 flip-flop들의 값이 전부 다 0으로 초기화할 수 있게 구현했다. 입력은 shr\_in wire를 통해 들어오게 되며, 입력된 값은 4개의 flip-flop을 거치면서 밀려난다.

아래는 이 shift register를 구현한 Verilog code이다.

|  |
| --- |
| `timescale 1ns / 1ps  module shift\_register(shr, rst, shr\_in, clk, Q);  input clk, rst, shr\_in, shr;  output[3:0] Q;    reg [3:0] Q;    always @(posedge clk) begin  if(rst) begin  Q <= 4'b0000;  end  else if(shr) begin  Q[0] <= Q[1];  Q[1] <= Q[2];  Q[2] <= Q[3];  Q[3] <= shr\_in;  end  end  endmodule |

위 구현한 모듈에서는 레지스터인 Q를 출력으로 사용해 현재 flip-flop의 상태를 보이도록 했다. 구현한 모듈의 Schematic을 통해 회로도를 확인한 결과는 다음 회로와 같았다.

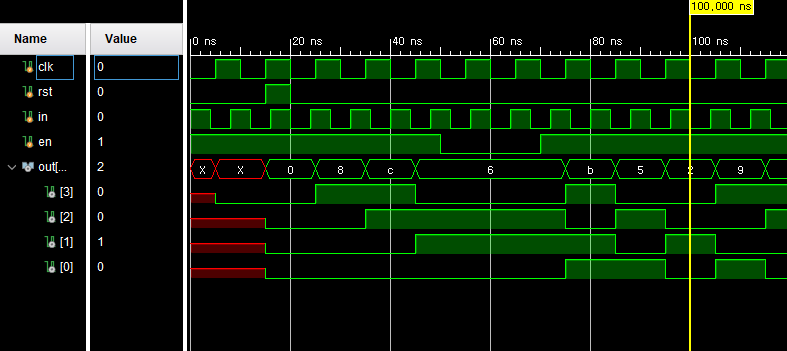


구현한 shift register가 정상적으로 작동하는지 확인할 수 있도록 다음과 같은 simulation code를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module shr\_sim;  reg clk, rst, in, en;  wire[3:0] out;    shift\_register shr(  .shr(en),  .rst(rst),  .shr\_in(in),  .clk(clk),  .Q(out)  );    initial en = 1'b1;  initial clk = 1'b0;  initial rst = 1'b0;  initial in = 1'b1;    always clk = #5 ~clk;  always in = #4 ~in;    initial begin  #15  assign rst = 1'b1;  #5  assign rst = 1'b0;  #30  assign en = 1'b0;  #20  assign en = 1'b1;    $finish;  end  endmodule |

Input 값으로 넘겨 주어야 하는 clk, rst, en, in 레지스터의 값을 바꿔가면서 출력 값으로 설정한 out의 값을 확인하고자 했다. Clock cycle time은 10ns로 설정했으며, in 값은 clock time과 어긋나게 바뀌도록 설정했다. 또한, 시뮬레이션 시작 후 약간의 시간 뒤에 rst 신호를 주어, rst이 들어오기 전과 후의 상태를 확인하고자 했고, 중간에 en 신호를 잠시 끊어 enable 입력이 정상적으로 작동하는지 테스트했다.

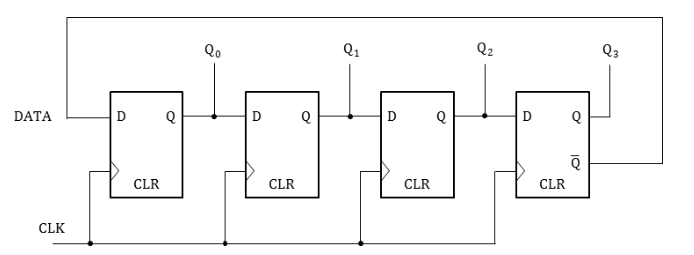
위의 simulation을 수행한 결과는 다음과 같다.



결과로부터, rst 신호가 들어오지 않은 상태의 출력 값은 정의되어지지 않은 것을 확인할 수 있는데, 이는 구현 과정에서 의도했던 바와 정확하게 맞는다. 그 외에도, in의 값이 4-bit register에 입력되고, 입력된 값이 clock이 지남에 따라 다음 flip-flop으로 넘어가는 것을 확인할 수 있었으며, en 신호가 low값을 보일 때에는 flip-flop의 변화가 없는 것을 볼 수 있다. 결과적으로 구현하려고 했던 기능들이 모두 정확하게 구현된 모습이다.

1. 4-bit Ring counter의 결과 및 simulation 과정에 대해서 설명하시오.

4-bit Ring counter는 다음과 같이 1.에서 구현한 shift counter의 마지막 flip-flop의 출력을 첫 flip-flop의 입력으로 연결해주고, 초기화 시 4개의 bit들 중 하나의 bit만 high 값을 가지도록 구현한 counter이다.



Ring counter는 앞서 구현한 shift register의 약간의 변경을 가하면 구현이 가능하다. 먼저, en 입력은 필요가 없으니 제외하고, 입력 값을 받는 부분도 필요가 없다. Input 신호로는 clk, rst만이 필요하고, 모든 Flip-flop들은 clk의 leading edge에서 trigger되도록 구현했다. 값을 초기화하는 rst의 경우, rst 값이 high 상태가 되면 Q의 값을 하나의 bit만 high상태로 초기화해야 하는데, 이 때 첫 번째 flip-flop만 1의 값을 갖도록 했다. 구현한 Verilog code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module ring\_counter(clk, rst, Q);  input clk, rst;  output [3:0] Q;    reg [3:0] Q;    always @(posedge clk) begin  if(rst)  Q = 4'b1000;  else begin  Q[0] <= Q[1];  Q[1] <= Q[2];  Q[2] <= Q[3];  Q[3] <= Q[0];  end  end  endmodule |

rst 입력이 들어온 경우, Q의 현재 값을 1000으로 덮어쓰도록 구현했고, 각 flip-flop이 지닌 값이 고리 모양을 이루며 전달되도록 코드를 작성했다. 작성한 코드의 회로도는 다음과 같이 나타났다.

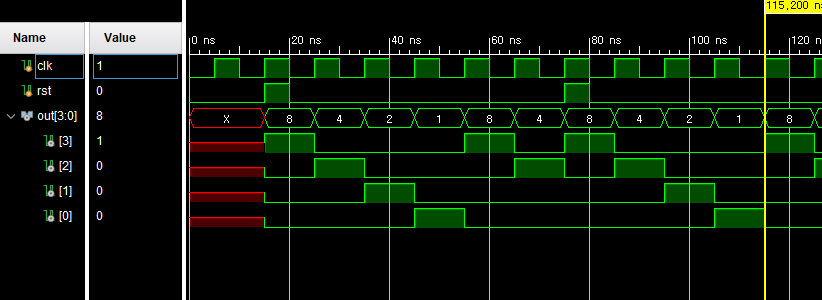
텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

위의 구현한 모듈이 정상적으로 구현되었는지 확인하기 위해, 다음과 같이 simulation code를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module rc\_sim;  reg clk, rst;  wire[3:0] out;    ring\_counter rc(  .clk(clk),  .rst(rst),  .Q(out)  );    initial clk = 1'b0;  initial rst = 1'b0;    always clk = #5 ~clk;    initial begin  #15  assign rst = 1'b1;  #5  assign rst = 1'b0;    #55  assign rst = 1'b1;  #5  assign rst = 1'b0;    #1600  $finish;  end  endmodule |

위 simulation code에서 확인하고 싶었던 것은, rst 신호가 정상적으로 값을 초기화시키는지, 그리고 각 clk의 leading edge마다 flip-flop의 데이터가 올바르게 이동하는지 여부이다. Clock cycle을 10ns로 두고, rst 값을 중간중간 변경해주면서 결과를 파악했다. Simulation의 결과는 다음과 같았다.



Simulation 결과로부터, reset 신호가 들어오기 전까지 flip-flop들이 정상적으로 undefined behaviour를 보이는 것을 볼 수 있고, 또한 레지스터의 값들이 1000->0100->0010->0001->1000->… 을 반복하고 있음을 확인했다. Reset 신호가 들어오면 레지스터 안의 있는 값이 1000으로 초기화되는 모습도 볼 수 있었다. 구현하고자 한 기능이 전부 정상적으로 작동하였기 때문에, 모듈의 구현이 잘 이루어졌다고 판단할 수 있다

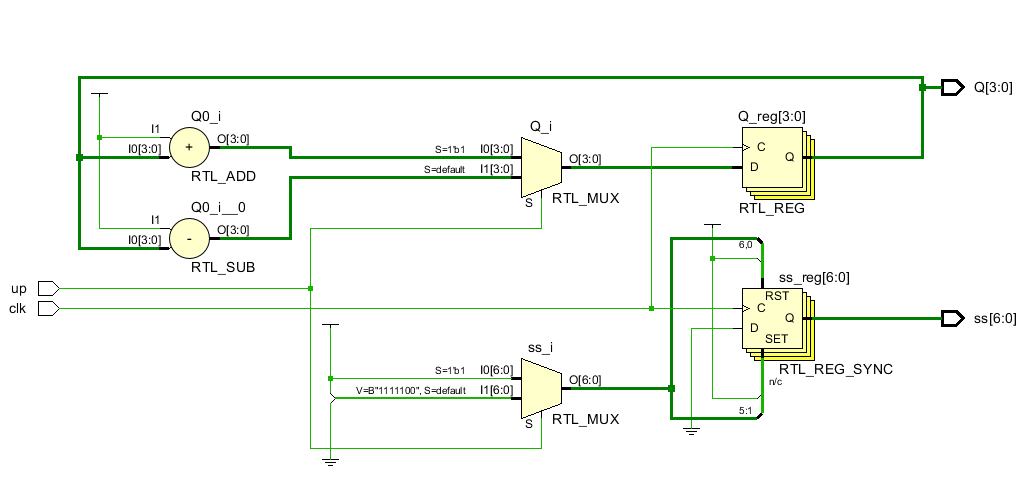
1. 4-bit Up/Down counter의 결과 및 simulation 과정에 대해서 설명하시오.

Up/Down counters는 신호를 받아, counter의 counting 방향을 지정할 수 있는 카운터이다. 이 counter를 구현하면서, clk의 positive edge에서 trigger되는 회로를 가정했고, trigger되었을 때 신호 값을 참고해 다음 4-bit Flip-flop의 증감을 결정했다. 추가적으로, 현재의 신호 값을 7-segment display로 표시해 알파벳 ‘d’와 ‘U’가 각각의 경우에 따라 표시될 수 있도록 구현했다. Up/down counter의 Verilog code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module up\_down\_counter(up, clk, Q, ss, digit);  input up, clk;  output [3:0] Q;  output [6:0] ss;  output digit;    assign digit = 1'b1;    reg[3:0] Q;  reg[6:0] ss;    initial Q <= 4'b0000;  always @(posedge clk) begin  if(up) begin  Q <= Q + 4'b0001;  ss <= 7'b0111110;  end  else begin  Q <= Q - 4'b0001;  ss <= 7'b0111101;  end  end  endmodule |

위 module로부터, 7-segment display를 출력하기 위해 output에 ss와 digit 값이 연결되어있는 것을 확인할 수 있다. Clk의 leading edge에 도달했을 때, up 값의 내용을 참고해 high 값이면 기존의 Q 레지스터 값에 1을 더하고, low 값이면 기존 Q 레지스터 값에 1을 뺀 값을 넣었다. 이 때, Q 안의 값은 자연스럽게 overflow/underflow되므로, 추가적인 예외 처리는 생략했다.

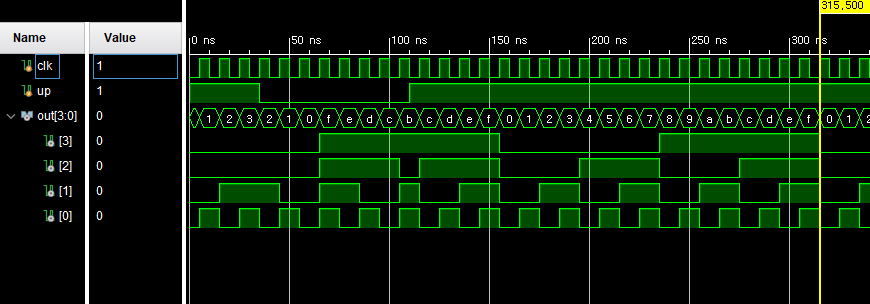
Up의 상태를 나타내기 위한 seven-segment display용 출력 ss에는 해당하는 문자를 표시할 수 있는 7-bit sequence를 저장한다. Digit 출력에는 1을 출력해 원하는 digit에 연결되도록 구현했다. 위 코드의 회로도는 다음과 같이 나타난다.



구현한 모듈이 정상적으로 올바르게 의도한 작업을 수행하는지 알아보기 위해 simulation code를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  udc\_sim;  reg clk, up;  wire[3:0] out;  wire[6:0] ss;    up\_down\_counter udc(  .clk(clk),  .up(up),  .Q(out),  .ss(ss)  );    initial clk = 1'b0;  initial up = 1'b1;    always clk = #5 ~clk;    initial begin  #35  assign up = 1'b0;    #75  assign up = 1'b1;    #1600  $finish;  end  endmodule |

Clock cycle을 10ns로 설정한 위 simulation에서, counter의 동작 중간에 up 값을 변화시켜 값이 정상적으로 의도한 바와 일치하게 변화하는지 확인해보고자 했다. 초기값은 high이므로, 0000, 0001, … 과 같이 증가하는 방향으로 counting이 진행되어야 하고, 중간에 up 값이 low가 되었을 때 1111, 1110, … 처럼 값이 감소하는 방향으로 counting 되어야 한다. 위 simulation의 결과는 다음과 같았다.



이 simulation 결과로부터, 모듈의 clk 신호가 positive edge에 상태가 될 때 마다 up 값을 참고해 출력 register의 값을 증가하거나 감소하는 모습을 확인할 수 있다. 자연스럽게 값이 overflow되는 모습도 확인할 수 있었다. 의도한 기능이 모두 정상적으로 동작하므로 모듈이 정확하게 구현되었다고 판단할 수 있다.

1. 결과 검토 및 논의 사항

Shift register, ring counter, 그리고 up/down counter의 세 모듈의 시뮬레이션 결과 의도했던 결과를 확인할 수 있었기에 각 모듈의 구현이 잘 되었다고 판단할 수 있다. 기본적으로 모든 모듈을 clocked 되어있는 디자인으로 만들었기 때문에, 각 타이밍에 따른 모듈의 behaviour를 다 확인할 수 있는 simulation을 작성해야 하는 부분이 까다로웠다.

1. 추가 이론 조사 및 작성

* Shift register들은 크게 네 가지의 카테고리로 나뉘는데, SISO(Serial In Serial Out), SIPO(Serial In Parallel Out), PISO(Parallel In Serial Out), PIPO(Parallel In Parallel Out)이 그것이다. 이번 실험에서 작성한 shift register는 SISO의 유형에 속한다.
* 만약 구현한 ring counter에서 Q[0]의 출력을 그대로 Q[3] flip-flop의 입력 값으로 넘겨주지 않고, 대신 invert된 상태로 넘겨주면, twisted counter, 혹은 Johnson counter라고 불리는 형태의 counter가 구현된다. 이 counter는 앞서 구현한 ‘straight’ ring counter에 비해 두 배 많은 state의 상태를 저장할 수 있는 특성을 지닌다.