컴퓨터공학실험2 13주차 결과 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 4-bit shift register의 결과 및 simulation 과정 설명

2. 4-bit ring counter의 결과 및 simulation 과정 설명

3. 4-bit Up/Down counter의 결과 및 simulation 과정 설명

4. 결과 검토 및 논의 사항

5. 추가 이론 조사 및 작성

**1. 4-bit shift register의 결과 및 simulation 과정**

4-bit shift register은 4개의 D flip-flop로 구성되며, clock pulse가 들어올 때마다, 왼쪽 flip-flop의 output Q가 다음(오른쪽) flip-flop의 input로 들어가는 형태이다. 설계한 Verilog code는 아래와 같다.

`timescale 1ns / 1ps

module shift\_register(

input A, clock, clr,

output reg Q0, reg Q1, reg Q2, reg Q3

);

always @(posedge clock)

begin

if(clr==1) begin

Q0 = 0;

Q1 = 0;

Q2 = 0;

Q3 = 0;

end

else begin

Q3 = Q2;

Q2 = Q1;

Q1 = Q0;

Q0 = A;

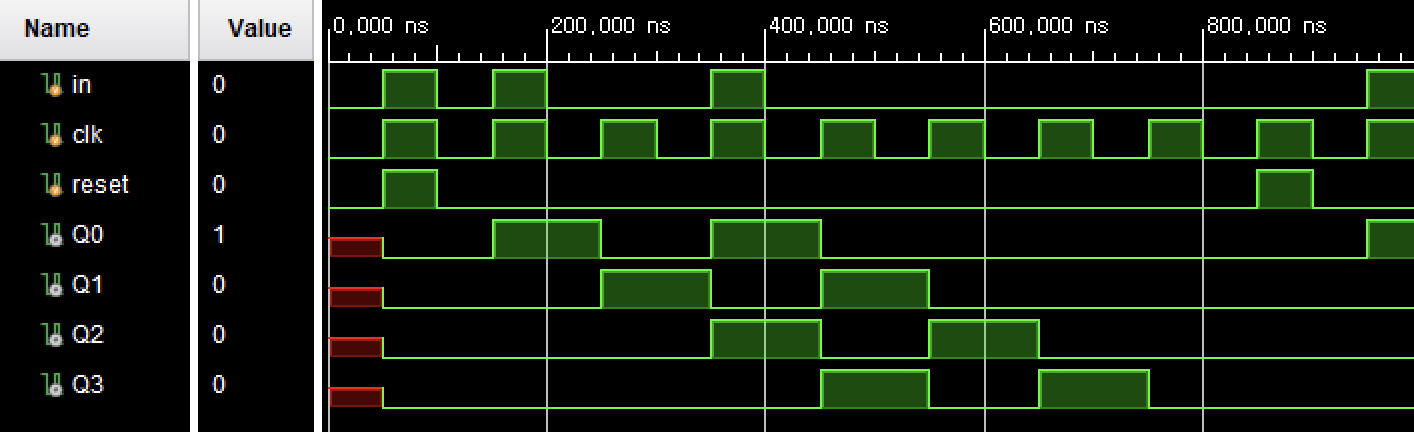
end

end

endmodule

<4-bit shift register의 Verilog code>

여기에서 shift 동작을 수행할 때, Q0부터 먼저 input을 할당하면, 뒤의 Q1, Q2, ,,, 에서 비동기적으로 Q0가 input에 의해서 바뀐 이후 shift가 일어날 수 있기 때문에, Q3 = Q2, Q2 = 1, 와 같은 형식으로 코드를 작성했다. 이에 대한 simulation 결과는 아래와 같다.



<4-bit shift register의 simulation 결과>

simulation에서 reset = 1인 경우, Q0~Q3을 0으로 clear 하며, input과 clk(clock)에 따라서 shift가 되는 형태를 파악할 수 있다.

**2. 4-bit ring counter의 결과 및 simulation 과정**

shift register은 다양한 방법으로 활용되며, counter 형태를 구현해낼 수도 있다. shift register로 구현할 수 있는 counter은 대표적으로 Johnson counter, ring counter 이 존재하는데, ring counter은 decoding을 수행하지 않고 직관적으로 count를 알 수 있다는 장점이 존재한다. 이러한 ring counter은 마지막(가장 오른쪽의) flip-flop의 output을 맨 처음(가장 왼쪽의) flip-flop의 input에 연결한 shift register의 형태이며, Verilog code는 아래와 같다.

`timescale 1ns / 1ps

module ring\_counter(

input clock, clr,

output reg Q0, reg Q1, reg Q2, reg Q3

);

always @(posedge clock) begin

if(clr==1) begin

Q0 = 1;

Q1 = 0;

Q2 = 0;

Q3 = 0;

end

else begin

if(Q3==1) begin

Q3 = Q2;

Q2 = Q1;

Q1 = Q0;

Q0 = 1;

end

else begin

Q3 = Q2;

Q2 = Q1;

Q1 = Q0;

Q0 = 0;

end

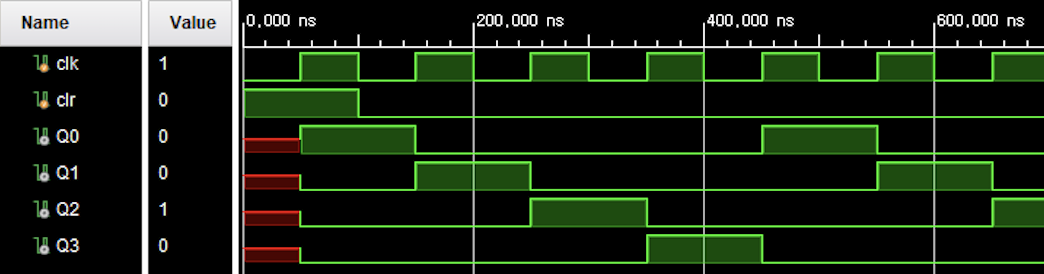
end

end

endmodule

<4-bit ring counter의 verilog code>

구현한 형태를 보면, 위에서 작성한 shift register의 형태에서 if(Q3==1)의 조건문이 추가된 형태임을 파악할 수 있다. 이는 마지막 flip-flop의 이전 output이 필요없는 4-bit shift register과 다르게, 마지막 output이 맨 처음 flip-flop의 input으로 들어가야 하기 때문에, 이와 같이 code를 작성했다. 이에 대한 simulation 결과는 아래와 같다.

<4-bit ring counter의 simulation 결과>

위 결과를 보면, 초기화로 인해 생긴 1이 계속 shift되며, 마지막에 Q3 = 1인 상태에서 shift가 일어난 이후, 이 1이 없어지는 것이 아닌, 다시 Q0로 들어가 count가 반복되는 형태를 파악할 수 있다. 또한 4-bit ring counter의 경우에는 0~3의 clock을 count할 수 있다.

**3. 4-bit Up/Down counter의 결과 및 simulation 과정**

Up/Down counter은 up(+1), down(-1)의 양방향의 count을 수행할 수 있는 회로로, Up/Down’ 라는 제어선을 통해서 up count을 수행할지, down count을 수행할지 결정하게 된다. 이때 JK flip-flop의 toggle(J = 1, K = 1)을 활용하게 되며, 4-bit Q3Q2Q1Q0에 대해 toggle이 되는 조건은 다음과 같다.

(1) Q0는 clock pulse가 들어올 때마다 toggle 된다.

(2) Q1은 up count인 경우 Q0 = 1, down count인 경우 Q0 = 0일 때 toggle 된다.

(3) Q2는 up count인 경우 Q1Q0 = 11, down count인 경우 Q1Q0 = 00일 때 toggle 된다.

(4) Q3은 up count인 경우 Q2Q1Q0 = 111, down count인 경우 Q2Q1Q0 = 000일 때 toggle 을 수행한다.

따라서 Verilog code는 아래와 같다.

`timescale 1ns / 1ps

module up\_down\_counter(

input up, clock, clr,

output reg Q0, reg Q1, reg Q2, reg Q3, reg digit, reg seg1, reg seg2, reg seg3, reg seg4, reg seg5, reg seg6, reg seg7

);

always @(posedge clock) begin

if(clr==1) begin

digit = 0;

Q0 = 0;

Q1 = 0;

Q2 = 0;

Q3 = 0;

end

else begin

digit = 1;

if(up==1) begin

seg1 = 0;

seg2 = 1;

seg3 = 1;

seg4 = 1;

seg5 = 1;

seg6 = 1;

seg7 = 0;

if(Q0&Q1&Q2) begin

Q3 = ~Q3;

end

if(Q0&Q1) begin

Q2 = ~Q2;

end

if(Q0) begin

Q1 = ~Q1;

end

Q0 = ~Q0;

end

else begin

seg1 = 0;

seg2 = 1;

seg3 = 1;

seg4 = 1;

seg5 = 1;

seg6 = 0;

seg7 = 1;

if(~Q0&~Q1&~Q2) begin

Q3 = ~Q3;

end

if(~Q0&~Q1) begin

Q2 = ~Q2;

end

if(~Q0) begin

Q1 = ~Q1;

end

Q0 = ~Q0;

end

end

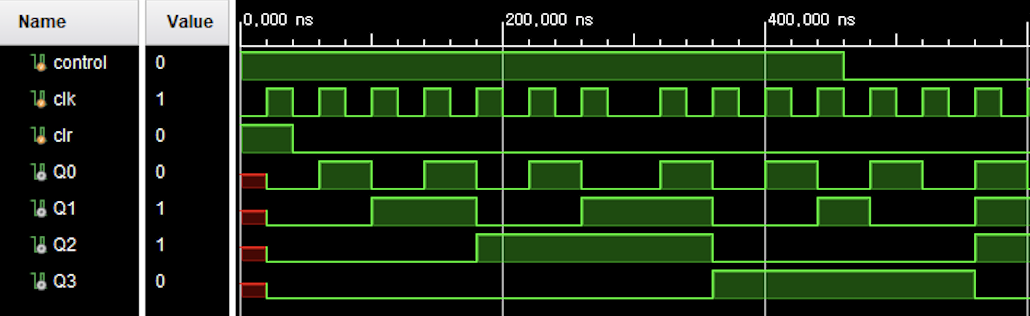
end

endmodule

<4-bit Up/Down counter의 Verilog code>

위 code에서 digit, seg1~seg7 output은 FPGA 상에서 up count인 경우 7-segment display에 U, down count인 경우 d를 표시하기 위해 사용한 output이다. 또한 위에서 설명한 toggle 조건이 성립하는 경우에, Q3 = ~Q3와 같이 toggle을 수행하도록 code을 작성했다.

이에 대한 simulation 결과는 아래와 같다.



<4-bit Up/Down counter의 simulation 결과>

simulation에서 control input은 Up/Down’ 제어선이며, clr을 통해서 맨 처음 output = 0000으로 만들고 count를 시작했다. 이후에는 up, down count가 각각 제대로 동작함을 파악할 수 있다.

**4. 결과 검토 및 논의 사항**

이번 실습에서는 shift register, ring counter, Up/Down counter을 구현하며, shift register의 작동 구조와 함께, 활용되는 방법을 파악했다. 또한 이전까지 구현한 up 방향으로만 count를 수행하는 회로들과 다르게, Up/Down counter 이라는 새로운 개념과, 이를 직접 구현해보기도 했다. 결과적으로는 각 경우가 적절한 형태로 구현됨을 파악할 수 있었다. 약간의 문제점이 존재한다면, 실제로 위 회로들은 synchronous(동기식) 형태로 만들어져 있는데, 개인적으로 구현한 방법은 asynchronous(비동기식) 형태로 구현했기 때문에, shift register의 경우에는 shift의 순서를 약간 조절해주어야 했으며, ring counter의 경우에는 Q3의 값에 따라 두 가지 case를 나누어서 code를 작성해야 했다. 이의 해결방법, 혹은 synchronous 형태로 구현할 수 있는 방법을 알아보고 싶다.

**5. 추가 이론 조사 및 작성**

위에서 설명한 것처럼, 이번 실습을 synchronous 형태로 작성하는 방법에 대해서 알아보자. C, C++와 같은 언어에는 이미 익숙한 array(배열) 형태를 다양한 방식으로 활용하는데, Verilog 또한 이와 같은 자료형이 존재하며, 이를 통해서 synchronous 형태로 이번 실습을 구현해낼 수 있다. 4-bit shift register를 구현하기 위한 형태는 아래와 같다.

(1) module의 output을 output reg [3:0] Q 로 만든다.

(2) module 내부에서 reg [3:0] save 를 를 정의한다.

(3) always @(posedge) begin ~~ block의 내부에서 save의 각 원소에 원하는 형식을 대입한다.(save[0] = input, save[1] = Q[0], save[2] = Q[1], save[3] = Q[2] 형태)

(4) 이후 Q <= save 형태로 대입을 수행하면 한꺼번에 synchronous 형태로 구현할 수 있게 된다.

위 과정을 통해 동기식 형태로 shift register을 구현할 수 있다. 사실 save에 각 원소를 할당하는 것이 동기식은 아니지만, output Q에 할당하는 것이 동기식 형태이기 때문에, 위에서 구현한 4-bit shift register보다 위 과정을 통해 구현한 shift register이 더 실제 구현에 가깝다고 생각할 수 있을 것 같다.