실험 Verilog-13: 결과보고서

전공: 컴퓨터공학 학년: 2 학번: 20171645 이름 박찬우

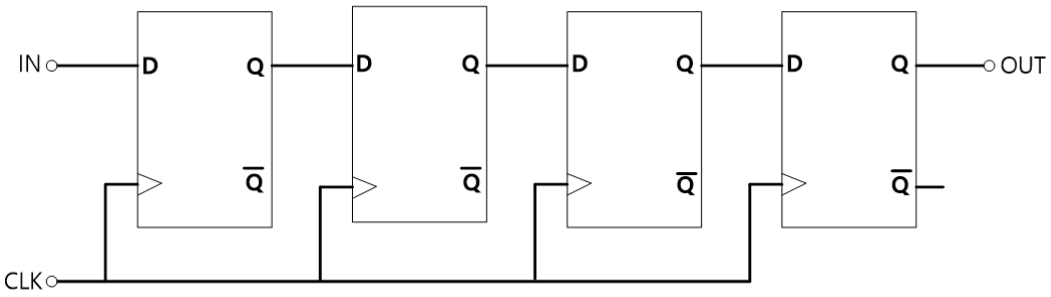
1. 목적

Verilog를 통해 4-bit shift register, ring counter, up-down counter에 대해 이해한다.

2. 요구 사항

1) 4-bit shift register의 결과 및 simulation 과정에 대해 설명하시오.

Shift Register는 일련의 플립플롭이 직렬로 연결되어 하나의 플립플롭의 출력이 다음 플립플롭의 입력으로 가게 되는 회로이다. 따라서, CLK가 들어올때마다 각 플립플롭의 메모리 상태는 다음 플립플롭으로 한칸씩 밀려나게 된다. 보통 Shift Register는 아래와 같은 구조를 따른다. 이때, D 플립플롭 대신 JK 플립플롭이나 T 플립플롭을 이용하기도 한다.



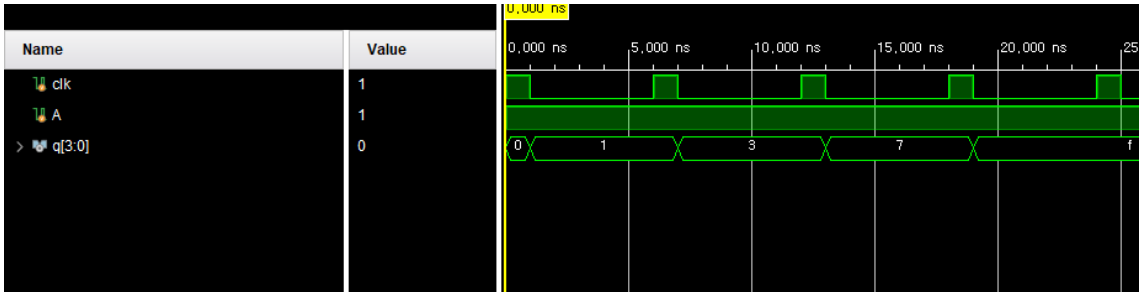
다음 구조를 Verilog Code로 코딩한 결과는 아래와 같다.

```
`timescale 1ns / 1ps
module dff(d,clk,q,qb);
    input d,clk;
    output q,qb;
    reg q,qb;
    initial
    begin
        q=1'b0;
        qb=1'b1;
    end
    always @(negedge clk)
    begin
        if (d==0) begin
```

```
        q=0;
        qb=1;
        end
        else begin
        q=1;
        qb=0;
        end
    end
endmodule
module shift_reg(A,clk,q);
    input A;
    input clk;
    output wire [3:0] q;
    wire [3:0] q;
    wire qb0,qb1,qb2,qb3;

       dff df0(A,clk,q[0],qb3);
       dff df1(q[0],clk,q[1],qb2);
       dff df2(q[1],clk,q[2],qb1);
       dff df3(q[2],clk,q[3],qb0);
 endmodule
```

각 D-플립플롭의 출력 결과가 다음 플립플롭의 입력으로 들어가는 구조를 module shift_reg에서 확인할 수 있다.
이제 아래와 같은 testbench code로 simulation을 실행해본다.

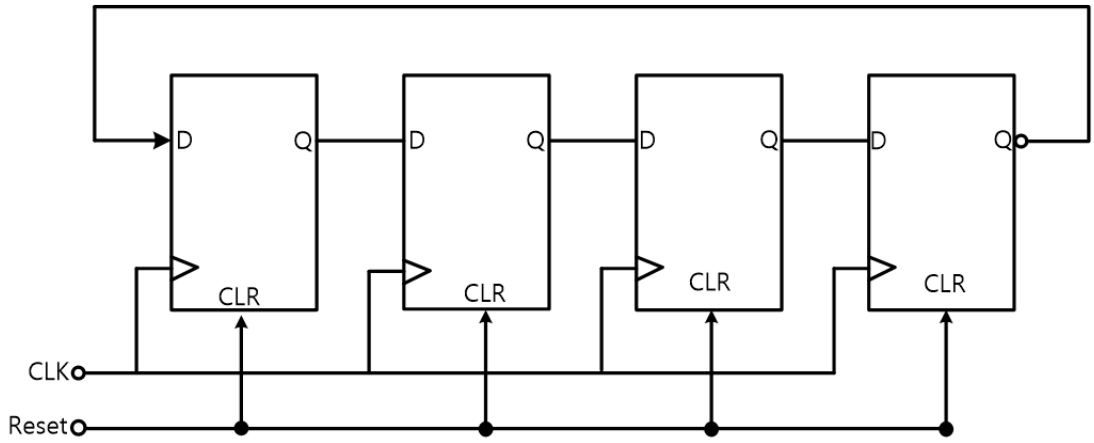
```
`timescale 1ns / 1ps
module tb_shiftreg4;
reg clk,A;
wire [3:0] q;
shift_reg t(.clk(clk), .A(A), .q(q));
initial begin
clk = 1'b1;
A=1'b1;
end
always begin
clk = #1 ~clk;
clk = #5 ~clk;
end
initial begin
#100
$finish;
end
endmodule
```



보는 바와 같이 입력 A가 항상 1일때 Output Q의 값이 1 (0001) -> 3 (0011) -> 7 (0111) -> 15(1111)로 변화하는걸 확인할 수 있다. 이는 input인 A가 첫번째 자리에 들어가고, 그에따라 각 자리의 값이 하나씩 밀려나는 걸 보여준다.

2) Ring Counter의 결과 및 simulation 과정에 대해 설명하시오.

Ring Counter는 앞선 Shift Register와 거의 동일한 구조를 갖지만, 이번에는 Input 값을 입력 받는 대신 맨 마지막 플립플롭의 Output 값이 첫번째 플립플롭의 Input 값으로 들어간다. CLK의 펄스마다 데이터는 다음 플립플롭으로 한칸씩 밀려나게 된다. 즉, 아래와 같은 구조를 따른다.



위와 같은 회로를 Verilog Code로 코딩한 결과는 아래와 같다.

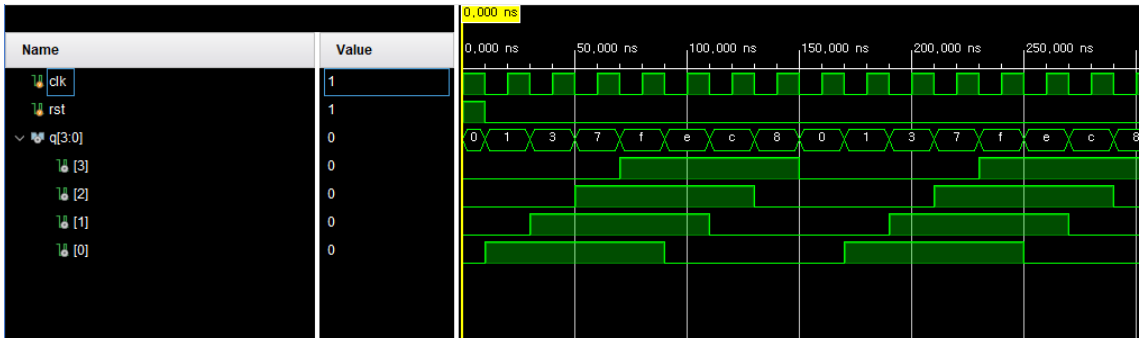
```
`timescale 1ns / 1ps
module dff(d,clk,q,qb,rst);
    input d,clk,rst;
    output q,qb;
    reg q,qb;
    initial
    begin
        q=1'b0;
        qb=1'b1;
```

```
    end
    always @(negedge clk) begin
    if (rst == 1) begin
    q=1'b0;
    qb=1'b1;
    end
    else begin
        if (d==0) begin
            q=0;
            qb=1;
            end
            else begin
            q=1;
            qb=0;
            end
        end
    end
endmodule
module shift_reg(rst,clk,q);
    input rst,clk;
    output wire [3:0] q;
    wire qb0,qb1,qb2,qb3;
    wire [3:0] tq;

    dff df0(~q[3],clk,q[0],qb0,rst);
    dff df1(q[0],clk,q[1],qb1,rst);
    dff df2(q[1],clk,q[2],qb2,rst);
    dff df3(q[2],clk,q[3],qb3,rst);

 endmodule
```

각 D-플립플롭의 출력 결과가 다음 플립플롭의 입력으로 들어가는 구조와 첫번째 입력에 마지막 플립플롭의 Output 값의 neg 값이 들어가는 구조를 module shift_reg에서 확인할 수 있다. 이때 모듈명은 앞선 shift register 작성 후 편의상 수정하지 않고 그대로 새로 작성했다.
이제 아래와 같은 testbench code로 simulation을 실행해본다.

```
`timescale 1ns / 1ps
module tb_shiftreg4;
reg clk;
reg rst;
wire [3:0] q;
shift_reg t(.clk(clk), .rst(rst), .q(q));
initial begin
clk = 1'b1;
rst = 1'b1;
end
```

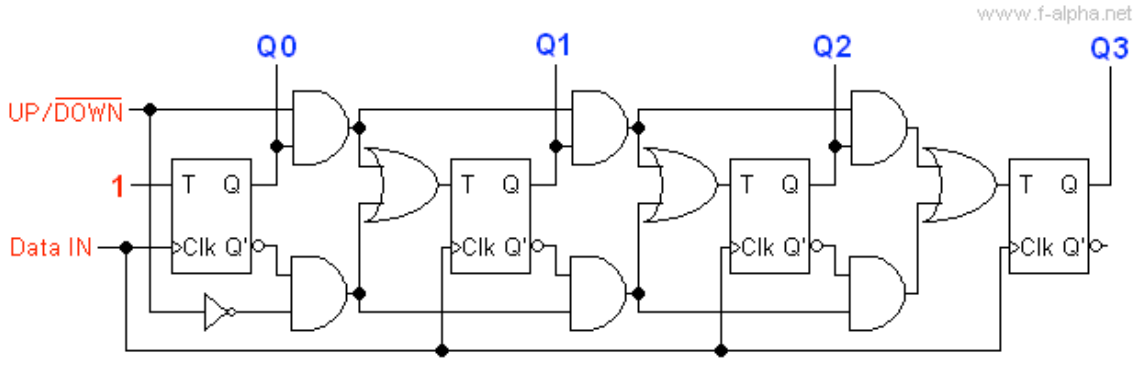
```
always clk = #10 ~clk;
initial begin
#10
rst = ~rst;
#1000
$finish;
end
endmodule
```



Simulation 결과를 보면 각 플립플롭의 값이 다음 플립플롭으로 옮겨지는 과정을 잘 확인할 수 있다.

3) Up-Down Counter의 결과 및 simulation 과정에 대해 설명하시오.

Up-down Counter는 사용자로부터 1bit 입력을 받아 각각 Up/Down인지 판단하고 그에 따라 저장된 값을 1 더하거나 뺄 수 있는 계수기를 말하는데, 사용자로부터 Up/Down을 입력받고, 4개의 T-플립플롭을 사용한 연산을 통해 Counter를 구현할 수 있다. 이는 다음과 같은 구조를 따른다.



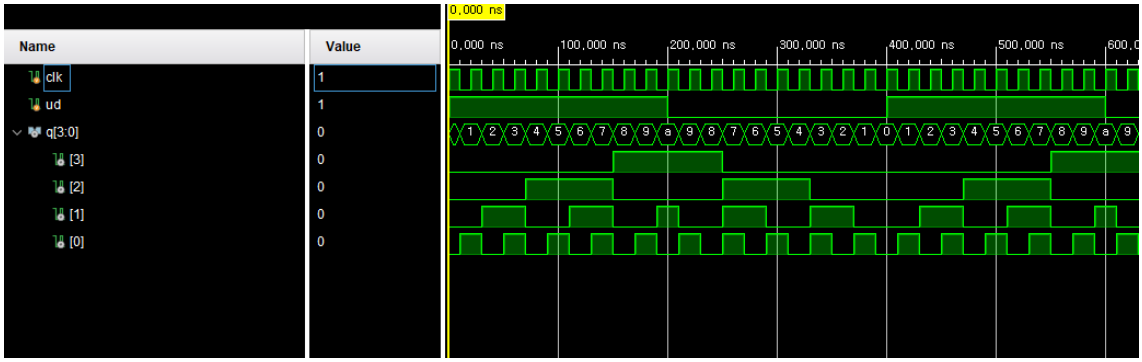
이를 바탕으로 작성한 Verilog Code는 아래와 같다.

```
`timescale 1ns / 1ps
module T(input t,clk,output reg q,qc);
    initial begin
    q=0;
```

```
    qc=1;
    end
    always@(negedge clk)begin
        if (t==1) begin
        q<=~q;
        qc<=~qc;
        end
        else begin
        q<=q;
        qc<=qc;
        end
    end
endmodule
module shift_reg(ud,clk,q);
    input ud,clk;
    output wire [3:0] q;
    wire [3:0] qb;

    T j1(1,clk,q[0],qb[0]);
    T j2((q[0] && ud) | (~ud && qb[0]), clk, q[1], qb[1]);
    T j3((q[0] && ud && q[1]) | (~ud && qb[0] && qb[1]), clk, q[2], qb[2]);
    T j4((q[0] && ud && q[1] && q[2]) | (~ud && qb[0] && qb[1] && qb[2]), clk,
q[3], qb[3]);

 endmodule
```

이러한 코드를 아래와 같은 testbench code를 통해 Simulation 해본다.

```
`timescale 1ns / 1ps
module tb_shiftreg4;
reg clk;
reg ud;
wire [3:0] q;
shift_reg t(.clk(clk), .ud(ud), .q(q));
initial begin
clk = 1'b1;
ud = 1'b1;
end
always clk = #10 ~clk;
always ud = #200 ~ud;
initial begin
#1000
$finish;
end
endmodule
```



입력 ud의 값이 1일때는 Up이다. 따라서 q의 값이 CLK의 edge마다 1씩 차례로 증가하는걸 확인할 수 있다. 그러나 중간에 ud의 값이 0으로 변하면서, q의 값이 a(10)에서 0까지 1씩 CLK의 edge마다 차례대로 떨어지고, ud의 값이 다시 1로 변하자 q값이 다시 상승하는 걸 확인할 수 있다. 즉, ud의 입력에 따라 up,down이 정상적으로 작동하는 걸 확인할 수 있다.