실험 Verilog-12: 결과보고서

전공: 컴퓨터공학 학년: 2 학번: 20171645 이름 박찬우

1. 목적

Verilog를 통해 Counter에 대해 이해한다.

2. 요구 사항

1) 2-bit counter의 결과 및 simulation 과정에 대해서 설명하시오.

Counter는 Clock의 edge에 따라 결과에 1을 더하는 연산을 진행하고, 2-bit counter의 경우 두 개의 비트만 저장하므로 0~3의 값을 저장할 수 있다. 3을 초과하게 되면 다시 0으로 초기화된다. 이를 Truth table로 나타내면 아래와 같다.

현재상태	다음상태		Output Z		D1 D2	
	x=0	x=1	x=0	x=1	x=0	x=1
00	00	01	0	0	00	01
01	01	10	0	0	01	10
10	10	11	0	0	10	11
11	11	00	0	1	11	00

이러한 Truth table을 바탕으로 JK-Flip Flop 2개를 사용해 2-bit Counter를 아래와 같은 Verilog Code로 구현할 수 있다.

```
`timescale 1ns / 1ps
module jk_ff(input j,k,clk,output reg q,qc);
    initial begin
    q=0;
    qc=1;
    end
    always@(negedge clk)begin
       if (j==0&&k==0) begin
       q<=q;
```

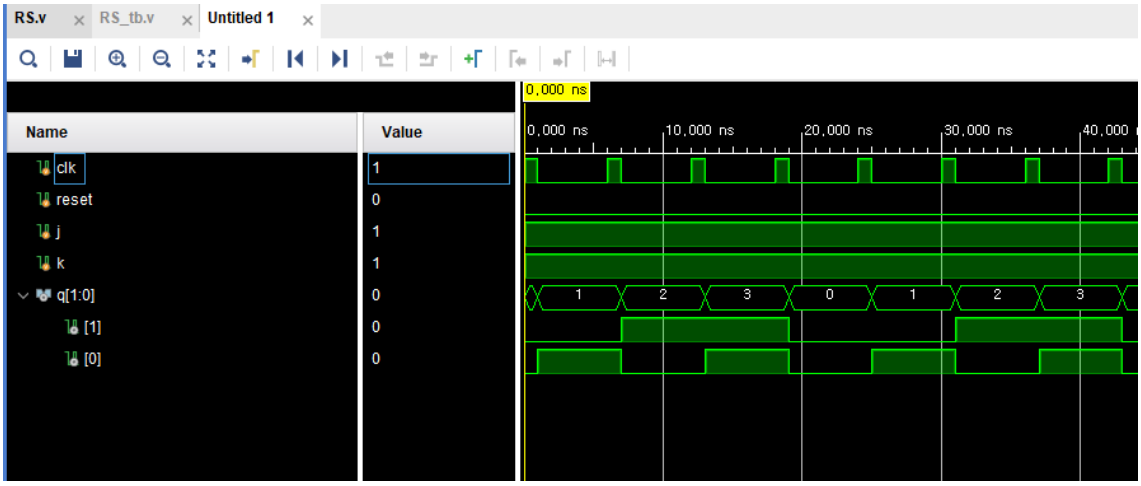
```
        qc<=qc;
        end
        if (j==0&&k==1) begin
        q=0;
        qc=1;
        end
        if (j==1&&k==0) begin
        q=1;
        qc=0;
        end
        if (j==1&&k==1) begin
        q<=qc;
        qc<=q;
        end
    end
endmodule
module inv(
    input clk, reset, j,k,
    output wire[1:0] q
    );

    wire[1:0] qc;
    wire[1:0] tmpq;
    jk_ff b1(j,k,clk,tmpq[0],qc[0]);
    jk_ff b2(tmpq[0],tmpq[0],clk,tmpq[1],qc[1]);

    and(q[0],~reset,tmpq[0]);
    and(q[1],~reset,tmpq[1]);
endmodule
```

첫번째 JK flipflop의 연산 결과를 tmpq[0]에 저장하고, 이를 두번째 JK flipflop의 J,K에 넣어 결과를 tmpq[1]에 저장하며 Reset이 1일경우 결과가 항상 0이어야 하므로 and연산을 수행하는 간단한 process를 나타내고 있다.

이러한 Verilog code를 아래와 같은 Simulation code를 통해 Simulation 한 결과는 아래와 같다.

```
`timescale 1ns / 1ps
module inv_tb;
reg clk,reset,j,k;
wire [1:0] q;
inv func(
.clk(clk),
.j(j),
.k(k),
.reset(reset),
```

```
.q(q));
initial begin
clk = 1'b1;
reset = 1'b0;
j = 1'b1;
k = 1'b1;
end
always begin
clk = #1 ~clk;
clk = #5 ~clk;
end
initial begin
#48
reset = ~reset;
#10
$finish;
end
endmodule
```

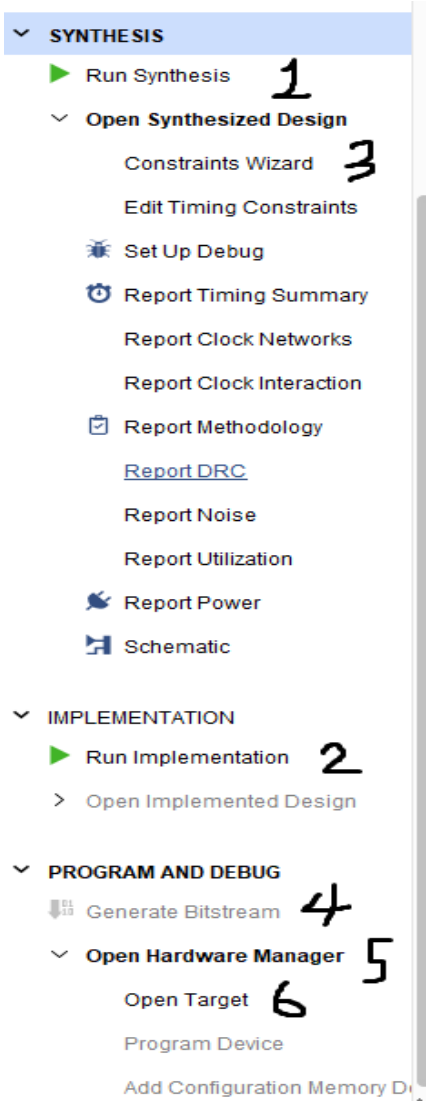


결과를 보면, Output인 Q의 값이 0에서 시작하여 Clk의 falling edge마다 1씩 증가해 1,2,3을 거쳐 다시 0으로 초기화되는 과정을 볼 수 있다. 즉, Clk의 입력마다 결과가 1씩 증가하는 2-bit Counter가 정상적으로 동작하는 것을 볼 수 있다.

이러한 Verilog Code를 그림과 같은 과정을 통해 아래와 같은 .xdc 파일과 함께 FPGA에 연결해 확인해보니 동일한 결과가 나타남을 실제 회로를 통해 확인할 수 있었다.

```
set_property IOSTANDARD LVCMOS18 [get_ports j]
set_property IOSTANDARD LVCMOS18 [get_ports k]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports q[1]]
set_property IOSTANDARD LVCMOS18 [get_ports q[0]]
```

```
set_property PACKAGE_PIN J4 [get_ports j]
set_property PACKAGE_PIN L3 [get_ports k]
set_property PACKAGE_PIN K3 [get_ports clk]
set_property PACKAGE_PIN M2 [get_ports reset]
set_property PACKAGE_PIN F15 [get_ports q[1]]
set_property PACKAGE_PIN F13 [get_ports q[0]]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]_OBUF}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```



2) 4-bit decade counter의 결과 및 simulation 과정에 대해서 설명하시

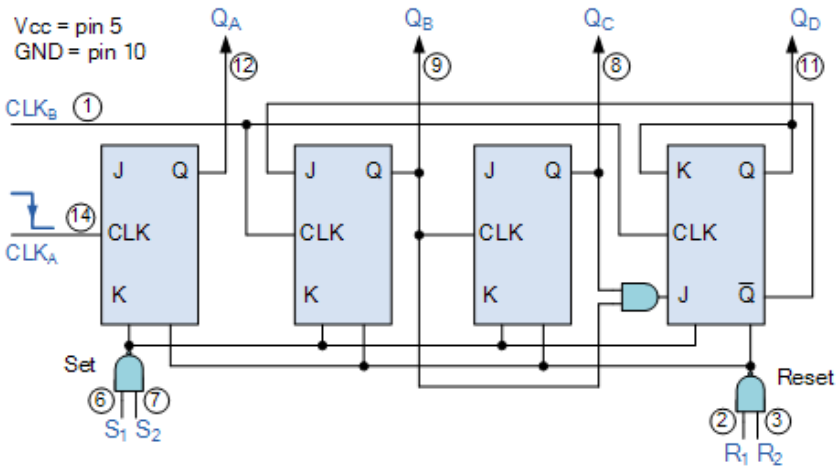
현재상태	다음상태		Output		D1 D2 D 3D4	
	x=0	x=1	x=0	x=1	x=0	x=1
0000	0000	0001	0	0	0000	0001
0001	0001	0010	0	0	0001	0010
0010	0010	0011	0	0	0010	0011
0011	0011	0100	0	0	0011	0100
0100	0100	0101	0	0	0100	0101
0101	0101	0110	0	0	0101	0110
0110	0110	0111	0	0	0110	0111
0111	0111	1000	0	0	0111	1000
1000	1000	1001	0	0	1000	1001
1001	1001	0000	0	1	1001	0000

오.

4-bit decade counter는 2bit와 유사하게 작동하지만, decade counter이기 때문에 0~9까지의 값만 저장한 후 10 대신 다시 0으로 초기화되는 특징을 가지고 있다. 4-bit decade counter의 Truth table은 아래와 같다.

이러한 Truth table을 바탕으로 2-bit counter와 유사하게 JK-flipflop을 4개 활용하여 다음과 같은 회로를 구성한다.

이러한 회로를 verilog code로 나타내면 아래와 같다.

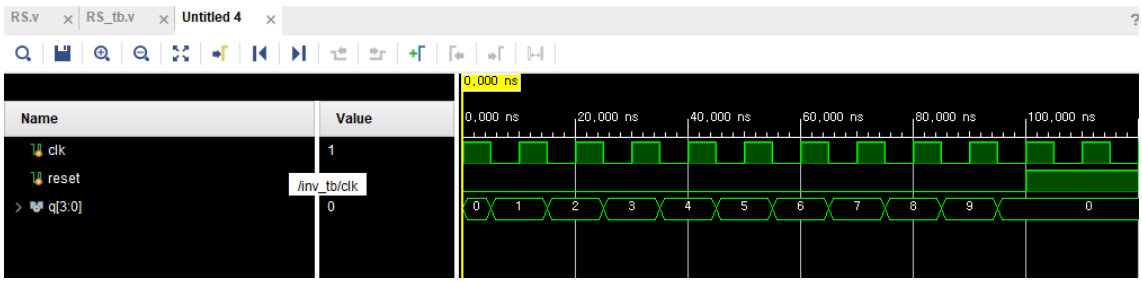



```
`timescale 1ns / 1ps

module four_bit_decade_counter(
    input clk, reset,
    output wire [3:0] q
    );

    wire[3:0] qc;
    wire[3:0] tmpq;
    wire[6:0] tmpwire;
    jk_ff b1(1'b1,1'b1,clk,tmpq[0], qc[0]);
    and(tmpwire[0], tmpq[0], qc[3]);
    jk_ff b2(tmpwire[0], tmpwire[0], clk, tmpq[1], qc[1]);
    and(tmpwire[1], tmpq[0], tmpq[1]);
    jk_ff b3(tmpwire[1], tmpwire[1], clk, tmpq[2], qc[2]);
    and(tmpwire[2], tmpwire[1], tmpq[2]);
    and(tmpwire[3], tmpq[0], tmpq[3]);
    or(tmpwire[4], tmpwire[2], tmpwire[3]);
    jk_ff b4(tmpwire[4], tmpwire[4], clk, tmpq[3], qc[3]);

    and(q[0], ~reset, tmpq[0]);
    and(q[1], ~reset, tmpq[1]);
    and(q[2], ~reset, tmpq[2]);
    and(q[3], ~reset, tmpq[3]);
   endmodule

module jk_ff(
    input j, k, clk,
    output reg q, qc
);
    initial begin
        q=0;
        qc=1;
    end
    always@(negedge clk)begin
        if(j==0&&k==0) begin
            q<=q;
            qc<=qc;
        end
        if(j==0&&k==1) begin
            q=0;
```

```
            qc=1;
        end
        if(j==1&&k==0) begin
            q=1;
            qc=0;
        end
        if(j==1&&k==1)begin
            q<=qc;
            qc<=q;
        end
    end
endmodule
```

이러한 Verilog Code를 아래와 같은 simulation code로 simulation한 결과는 아래와 같다.

```
`timescale 1ns / 1ps
module inv_tb;
reg clk,reset;
wire [3:0] q;
inv func(
.clk(clk),
.reset(reset),
.q(q));
initial begin
clk = 1'b1;
reset = 1'b0;
end
always begin
clk = #5 ~clk;
end
initial begin
#100
reset = ~reset;
#100
$finish;
end
endmodule
```



그림을 확인하면 알 수 있듯, clk의 Falling edge마다 output의 값이 0에서 1,2,3,...으로 진행되다가 9를 넘기면 0으로 초기화된다. 그 뒤, reset이 1로 활성화되자 clk에 상관없이 output이 항상 0으로 고정되는걸 확인할 수 있다. 즉, 4-bit decade counter가 정상적으로 동작하는걸 확인할 수 있다.
마지막으로, 아래와 같은 .xdc 파일을 통해 FPGA와 연결해 동일하게 작동하는 것을 확인할 수 있었다.

```
set_property IOSTANDARD LVCMOS18 [get_ports j]
set_property IOSTANDARD LVCMOS18 [get_ports k]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports q[0]]
set_property IOSTANDARD LVCMOS18 [get_ports q[1]]
set_property IOSTANDARD LVCMOS18 [get_ports q[2]]
set_property IOSTANDARD LVCMOS18 [get_ports q[3]]
set_property PACKAGE_PIN J4 [get_ports j]
set_property PACKAGE_PIN L3 [get_ports k]
set_property PACKAGE_PIN K3 [get_ports clk]
set_property PACKAGE_PIN M2 [get_ports reset]
set_property PACKAGE_PIN F15 [get_ports q[3]]
set_property PACKAGE_PIN F13 [get_ports q[2]]
set_property PACKAGE_PIN F14 [get_ports q[1]]
set_property PACKAGE_PIN F16 [get_ports q[0]]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[3]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[3]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[2]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[2]_OBUF}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```

3) 4-bit 2421 decade counter의 결과 및 simulation 과정에 대해서 설명하시오.

2421 decade counter의 경우 이전의 4-bit decade counter와 거의 동일하지만, counter 동작시에 0100 (4) 뒤에 0101 (5)가 아닌 1011 (B)가 되고, 그대로 진행되다가 1111 (F) 뒤에 다시 0으로 초기화되는 방식의 counter이다. 이와같은 특징을 아래 2421 decade counter의 Truth table을 통해 확인할 수 있다.

현재상태	다음상태		Output		D1 D2 D3 D4	
	x=0	x=1	x=0	x=1	x=0	x=1
0000	0000	0001	0	0	0000	0001
0001	0001	0010	0	0	0001	0010
0010	0010	0011	0	0	0010	0011
0011	0011	0100	0	0	0011	0100
0100	0100	1011	0	0	0100	1011
0101	1011	1100	0	0	1011	1100
0110	1100	1101	0	0	1100	1101
0111	1101	1110	0	0	1101	1110
1000	1110	1111	0	0	1110	1111
1001	1111	0000	0	1	1111	0000

이러한 Truth table을 바탕으로 작성한 Verliog Code는 아래와 같다.

```
//20191575
`timescale 1ns / 1ps
module tfto_decade_counter(
    input clk, reset,
    output wire [3:0] q
    );
    wire [3:0] tmpq;
    wire check;
    wire tmpwire;
    reg [3:0] B;
    four_bit_decade_counter fourbit(clk, reset, tmpq);
    assign                                    check                                    =
(tmpq[3]&~tmpq[2])|(tmpq[2]&tmpq[0])|(tmpq[2]&tmpq[1]&~tmpq[0]);
    always @(check)
        if(check == 1'b1) begin
          B = 4'b0110;
        end
        else begin
          B = 4'b0000;
        end
    adder4bit add0(tmpq, B, 0, q, tmpwire);
```

```
endmodule
module four_bit_decade_counter(
    input clk, reset,
    output wire [3:0] q
    );

    wire[3:0] qc;
    wire[3:0] tmpq;
    wire[6:0] tmpwire;
    jk_ff b1(1'b1,1'b1,clk,tmpq[0], qc[0]);
    and(tmpwire[0], tmpq[0], qc[3]);
    jk_ff b2(tmpwire[0], tmpwire[0], clk, tmpq[1], qc[1]);
    and(tmpwire[1], tmpq[0], tmpq[1]);
    jk_ff b3(tmpwire[1], tmpwire[1], clk, tmpq[2], qc[2]);
    and(tmpwire[2], tmpwire[1], tmpq[2]);
    and(tmpwire[3], tmpq[0], tmpq[3]);
    or(tmpwire[4], tmpwire[2], tmpwire[3]);
    jk_ff b4(tmpwire[4], tmpwire[4], clk, tmpq[3], qc[3]);

    and(q[0], ~reset, tmpq[0]);
    and(q[1], ~reset, tmpq[1]);
    and(q[2], ~reset, tmpq[2]);
    and(q[3], ~reset, tmpq[3]);
   endmodule

module jk_ff(
    input j, k, clk,
    output reg q, qc
);
    initial begin
        q=0;
        qc=1;
    end
    always@(negedge clk)begin
        if(j==0&&k==0) begin
            q<=q;
            qc<=qc;
        end
        if(j==0&&k==1) begin
            q=0;
            qc=1;
        end
        if(j==1&&k==0) begin
            q=1;
            qc=0;
        end
```

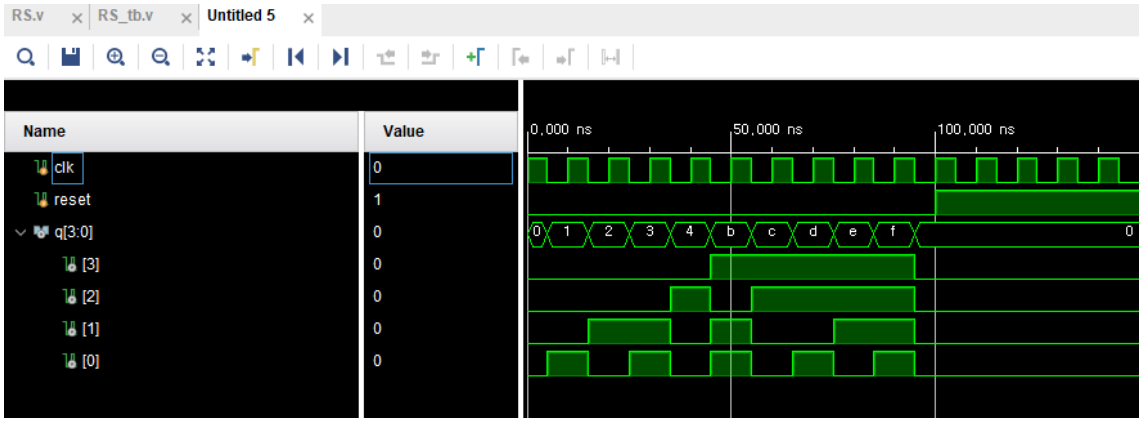
```
      if(j==1&&k==1)begin
         q<=qc;
         qc<=q;
      end
   end
endmodule
module adder1bit(A, B, Ci, S, Co);
input A, B, Ci;
output S, Co;
assign S=A^B^Ci;
assign Co=(A&B)|((A^B)&Ci);
endmodule
module adder4bit(A, B, Ci, S, Co);
input [3:0] A, B; input Ci;
output [3:0] S; output Co;
wire [3:0] A, B, S; wire Ci, Co;
wire [2:0] C;
adder1bit add1(A[0], B[0], Ci, S[0], C[0]);
adder1bit add2(A[1], B[1], C[0], S[1], C[1]);
adder1bit add3(A[2], B[2], C[1], S[2], C[2]);
adder1bit add4(A[3], B[3], C[2], S[3], Co);
endmodule
```

중간의 5->A로 가는 과정을 구현하기 위해, check를 assign해 if 조건문을 통해 5인지 확인하고, 결과에 1을 더하는 과정을 4bit binary adder의 구현을 통해 구현했다.

아래와 같은 simulation code로 simulation 한 결과는 다음과 같다.

```
`timescale 1ns / 1ps
module inv_tb;
reg clk,reset;
wire [3:0] q;
inv func(
.clk(clk),
.reset(reset),
.q(q));
initial begin
clk = 1'b1;
reset = 1'b0;
end
always begin
clk = #5 ~clk;
end
initial begin
#100
reset = ~reset;
```

```
#100
$finish;
end
endmodule
```



보이다시피 clk의 falling edge마다 결과값이 1씩 증가하다가, 5 대신 b로 결과가 들어오고, 그대로 증가하다가 1111(f)에서 clk의 falling edge를 만나면 다시 0으로 초기화되는 모습을 볼 수 있다. 또한, reset이 1일때 항상 결과가 0이 되는것 역시 확인할 수 있다.
즉, 2421 decade counter가 정상적으로 작동함을 확인할 수 있었다.

마지막으로, 아래 xdc 파일을 통해 FPGA와 연결해도 동일한 결과를 얻을 수 있었다.

```
set_property IOSTANDARD LVCMOS18 [get_ports j]
set_property IOSTANDARD LVCMOS18 [get_ports k]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports q[0]]
set_property IOSTANDARD LVCMOS18 [get_ports q[1]]
set_property IOSTANDARD LVCMOS18 [get_ports q[2]]
set_property IOSTANDARD LVCMOS18 [get_ports q[3]]
set_property PACKAGE_PIN J4 [get_ports j]
set_property PACKAGE_PIN L3 [get_ports k]
set_property PACKAGE_PIN K3 [get_ports clk]
set_property PACKAGE_PIN M2 [get_ports reset]
set_property PACKAGE_PIN F15 [get_ports q[3]]
set_property PACKAGE_PIN F13 [get_ports q[2]]
set_property PACKAGE_PIN F14 [get_ports q[1]]
set_property PACKAGE_PIN F16 [get_ports q[0]]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[1]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[0]_OBUF}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[3]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[3]_OBUF}]
```

```
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[2]}]
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets {q[2]_OBUF}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```