Digital Systems Design

Engr 378

Lab 3

Latches, Flip-flops and Sequential Circuits

Date:
3/9/21

Grade:

By:
Aaron Luong and Richard Couto

Problem Analysis

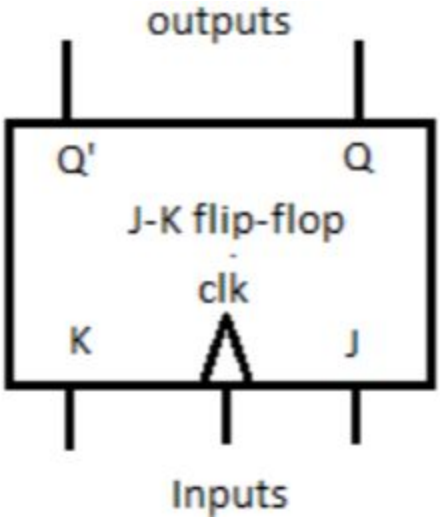


Fig 1: JK flip Flop

We need to run a two-bit counter capable of counting 0 to 3 and also has reset capabilities. Our limitation for this task was to exclusively use at most 2 JK flip flops. This would require coding for the specific Verilog modules for each component as well as a module to call both in a single system. Again these will be set in a testbench to be modeled.

Hardware Design

It took several rough designs to arrive at a functional physical setup. Our first design was logically accurate. What we had to make adjustments for was the delivery of the clock signal as a high/low source. This had both timing and waveform-shape issues. The following design corrected for this but had some unaccounted for states causing the second bit to act inappropriately during reset periods.

In our successful setup, We tied both J and K inputs to high, ensuring that the only state we could enter was toggle. This is essential because the counter needs to sequentially move with the clock, represented by the alternation in binary. The second J and K inputs will be tied to whatever the current output of the first bit would be, ensuring that on the next clock pulse our bit would transfer to the next bit location. Then we would incorporate the reset feature within the JK flip flop module ensuring that every instance of that module would be reset when desired.

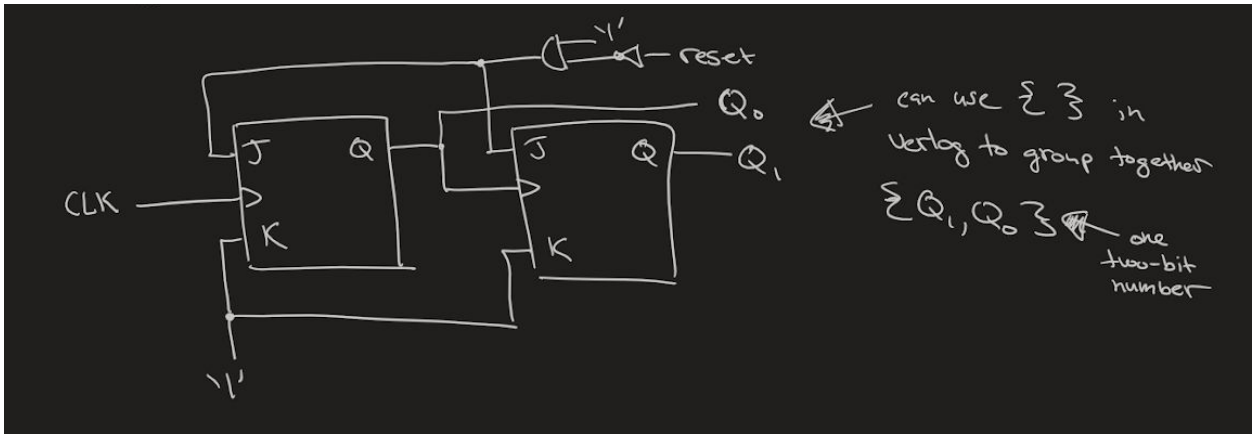


Fig 2: Counter Draft 1

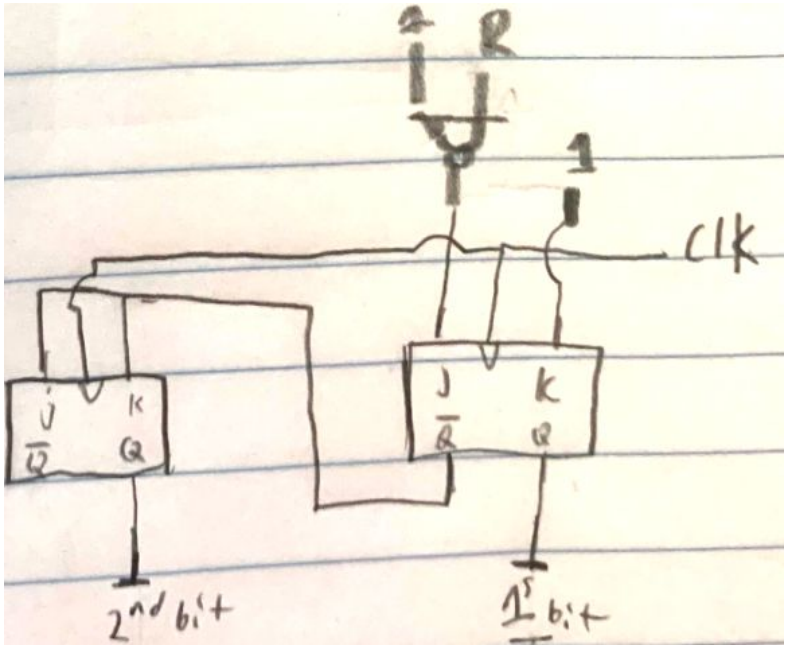


Fig 3: Counter Draft 2

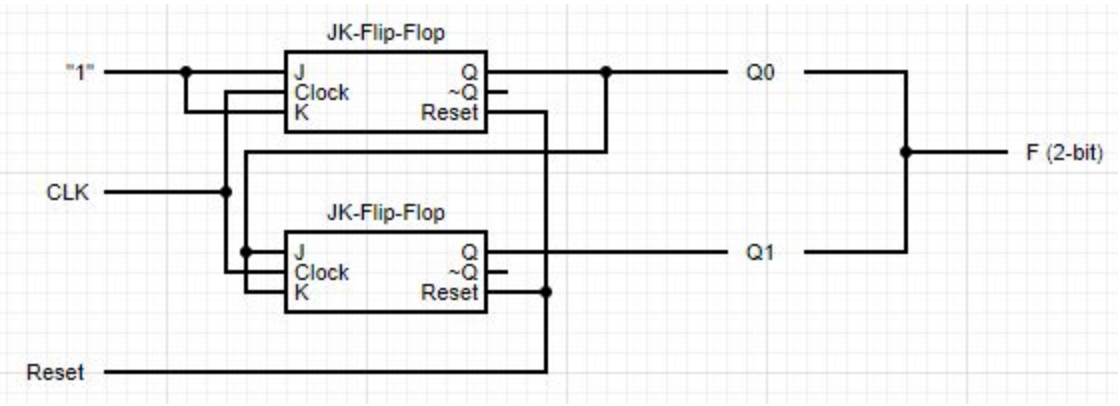


Fig. 4:Final Counter Design

Verilog Modeling

For the lab, we begin by coding the D-Latch. The D-Latch has two inputs and one output. The two inputs are the D and the clk, where clk represents the enable input of the D-Latch. When enable is active, then Q will change to whatever D is currently, and when enable is not active then Q stays the same.

D-Latch:

```
module DF (Q, D, clk);

input D;
input clk;

output reg Q;

always @ (posedge clk)
begin
        if (D == 1)
        Q <= 1;
        else
        Q <= 0;
 end

endmodule
```

The next module that needs to be written is for the JK-Flip-Flop. This module has four inputs and one output. The J and the K are the standard inputs for a JK-Flip-Flop and the clock input of the JK-Flip-Flop is connected to the CLK input. The reset is added to this module for the other part of this lab which is the counter. It is necessary here because without it one of the two JK-Flip-Flops won't have a starting value, which would be that it won't work correctly. Finally, we have Q which changes depending on what value J and K are at.

JK-Flip-Flop:

```
module JK(J, K, Q, CLK, reset);
        input J, K, CLK, reset;
        output reg Q;

        always @(posedge CLK) begin
                        if (reset == 1)begin
                                Q <= 0;
                        end
                        else if(J == 1'b0 && K == 1'b0)begin
```

```
                Q <= Q;
            end
            else if (J == 1'b0 && K == 1'b1)begin
                Q <= 1'b0;
            end
            else if (J == 1'b1 && K == 1'b0)begin
                Q <= 1'b1;
            end
            else if (J == 1'b1 && K == 1'b1)begin
                Q <= ~Q;
            end
        end
endmodule
```

Finally, the last module, not including the highest level module is the counter module. What this module does is take in a clock and a reset value. When reset is high then the output, F, will become zero and the counter won't start counting until reset is low again. Two wires are meant to connect to the output ports of the JK-Flip-Flop and then are concatenated together to form a two-bit number that is given to F where it then returns the value to the main module. The reason why the reset was on the JK-Flip-Flop instead of on this module is so that the second JK-Flip-Flop has a starting value to flip so in a way, the two flip-flops have preloaded values.

Counter (2-bit):

```
module counter(CLK, reset, F);
    input CLK, reset;
    output [1:0] F;
    wire J, K, Q0, Q1;

    assign K=1'b1;
    assign J=1'b1;

    JK JK_1(J, K, Q0, CLK, reset);
    JK JK_2(Q0, Q0, Q1, CLK, reset);
    assign F = {Q1, Q0};
endmodule
```

Last, but not least, is the parent module where all the modules are brought together to work together. This module is a place where all the different modules explained early can work together so is sort of the main class of a program.

"Parent" Class:

```
module Lab3(D, G, J, K, Q_JK, Q_D, reset, CLK, F);
```

```
    input J, K, reset, CLK, D, G;
    output Q_JK, Q_D;
    output [1:0] F;

    DF DLatchTest(Q_D, D, G);
    JK JKtest(J, K, Q_JK, CLK, reset);
    counter Countertest(CLK, reset, F);

endmodule
```

Results/ Verification

```
module Lab3Test();
    reg D, G, J, K, reset, CLK;
    wire [1:0] F;
    wire Q_JK, Q_D;
    parameter time_out = 100;

    initial $monitor (D, G, J, K, Q_JK, Q_D, reset, CLK, F);
    always
    begin
        #0 #5 G=0; D=0;CLK=0;J=0;K=0;reset=1;
        #10 G=1; D=0;
        #5 G=0;
        #5 G=1; D=1;
        #5 G=0; D=0;
        #5 G=0; D=1;

        #5 reset=0;
        #5 CLK=0; J=0; K=0;
        #5 CLK=0; J=0; K=1;
        #5 CLK=0; J=1; K=0;
        #5 CLK=0; J=1; K=1;
        #5 CLK=1; J=0; K=0;
        #5 CLK=0;
        #5 CLK=1; J=0; K=1;
        #5 CLK=0;
        #5 CLK=1; J=1; K=0;
        #5 CLK=0;
        #5 CLK=1; J=1; K=1;

        #5 CLK=0; reset=0;
        #5 CLK=1;
        #5 CLK=0;
```

```
        #5 CLK=1;
        #5 CLK=0;
        #5 CLK=1;
        #5 CLK=0;
        #5 CLK=1;
        #5 CLK=0;
        #5 CLK=1;
        #5 CLK=0;
        #5 CLK=1; reset=1;

    end
    Lab3 test(D,G,J,K,Q_JK,Q_D,reset,CLK,F);

endmodule
```

The testbench is based on the last testbench that we made, but the parameters have changed and the time delay between each line of code. The CLK oscillates from zero to one so that there can be a positive edge of the clock so that the JK-Flip-Flops can work.

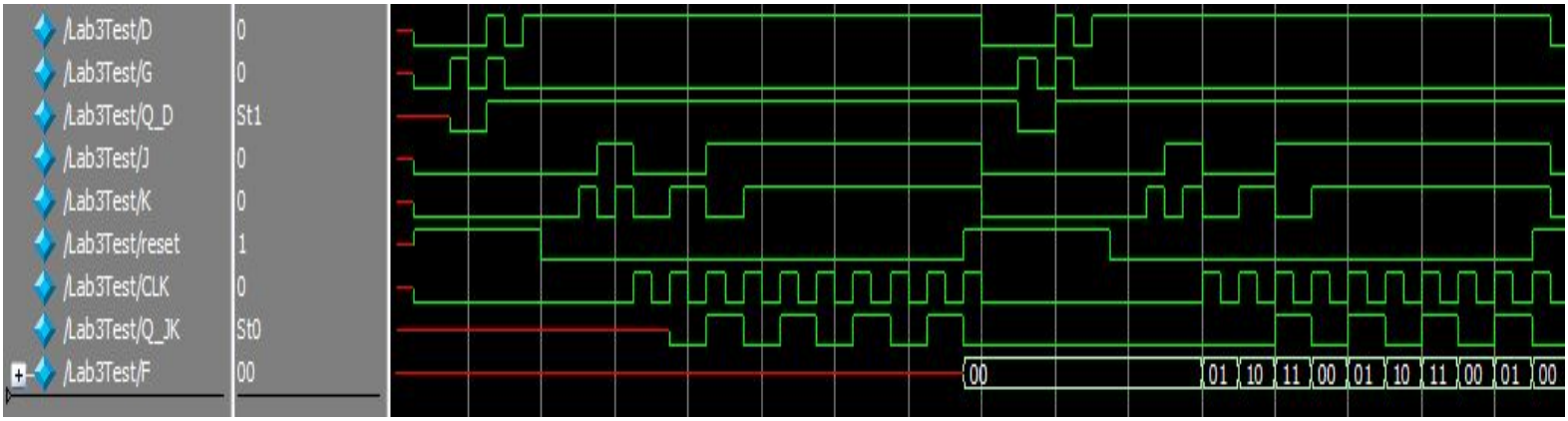


Fig 5: Testbench Results

From the results, we can see that the D-Latch works as intended and the JK-Flip-Flop works as intended as well. As for the Counter, it doesn't start until the first cycle ends because the module is still starting up and by the time it is ready to start the cycle starts again. However, during the second cycle, the module starts to count when the reset is low and the clock starts to work. This counter counts to three and then resets to zero because of the overload that happens. When the reset state goes high again, the counter resets to zero and doesn't count until reset is low and the clock starts.

Conclusion and Discussion

This counter counts to three and then resets to zero because of the overload that happens. When the reset state goes high again, the counter resets to zero and doesn't count until reset is low and the clock starts. Working through multiple designs, we found several unexpected hurdles with this task. How specific waveforms are delivered to drive a clock vs one used to represent high and low. Furthermore how some logic gates may not be able to account for all possible states. Our strongest solution was to program additional features into our modules to enforce specific desired outcomes. One of the problems that we encountered was that the counter initially didn't work with the design that we picked from one of the two prelabs. After a while of testing out different design, we figured out that the reason why one of the JK flip-flops wasn't working was that we have it flipping the Q that the JK-Flip-Flop has in its memory, but since it was always in an unknown state, it couldn't flip to a known state. To counter this we tried to program the reset into the module by updating J, but since the input for the second JK is the result of the first JK, this still didn't solve the issue. To get around this, we decided to have a reset port in the JK module to have a preset value already inside the JK-Flip-Flop so it can flip a known state.

Work Breakdown

For the work breakdown, we decided to split the work in half. Ricky worked on the D-Latch and Aaron worked on the JK-Flip-Flop. The Counter module was a joint effort between the two and so was the "parent" module. As for the testbench, Aaron wrote that because he already had the testbench code from the last lab so the only things that were changed were the parameters that needed to be changed and the results that needed to be shown in the wave format.

Prelabs

(NEXT 2 PAGES)

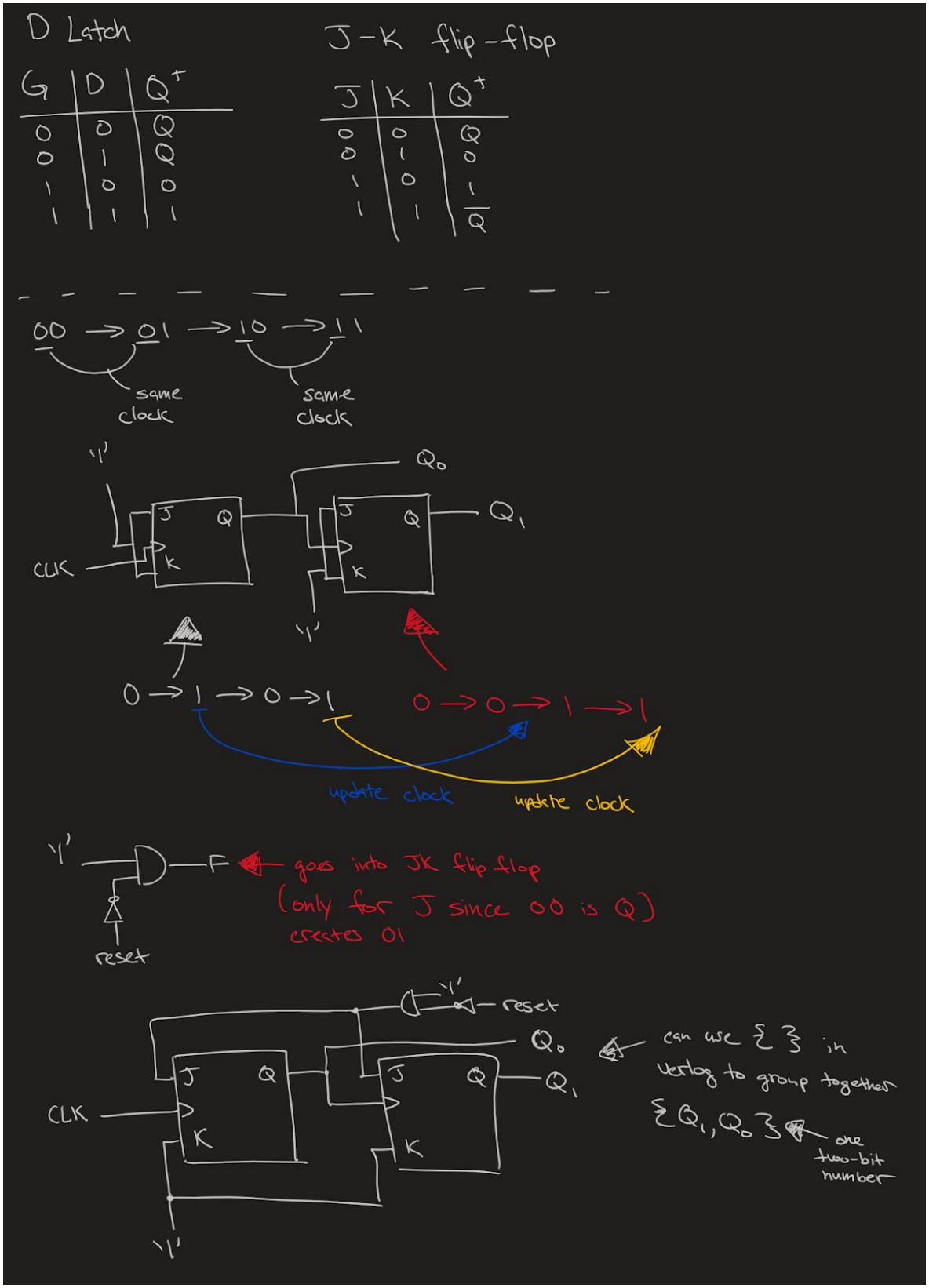


Fig 6: Aaron's Prelab

Prelab 3

1) Truth Table for D & JK

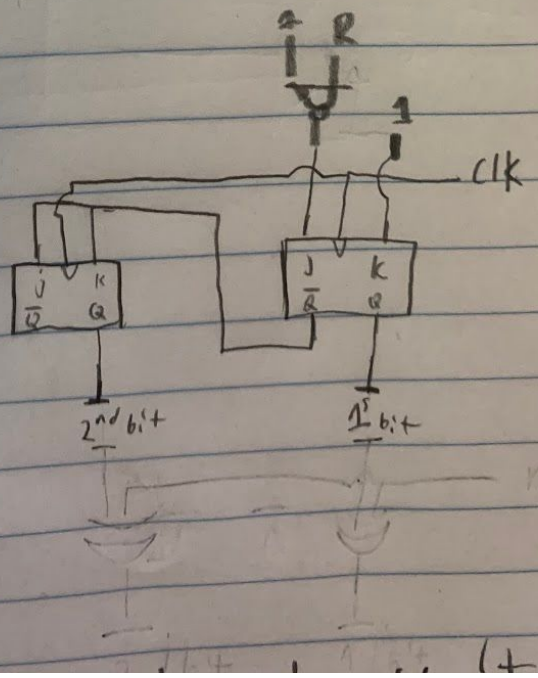
D

Q_t	D	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

JK

J	K	Q_{t+1}
0	0	Q
0	1	0
1	0	1
1	1	$\bar{Q}$

2)



Lab

Fig 7: Ricky's Prelab

HDL Source Code

```
module Lab3Test();
	reg D, G, J, K, reset, CLK;
	wire [1:0] F;
	wire Q_JK, Q_D;
	parameter time_out = 100;

	initial $monitor (D, G, J, K, Q_JK, Q_D, reset, CLK, F);
	always
	begin
		#0 #5 G=0; D=0;CLK=0;J=0;K=0;reset=1;
		#10 G=1; D=0;
		#5 G=0;
		#5 G=1; D=1;
		#5 G=0; D=0;
		#5 G=0; D=1;

		#5 reset=0;
		#5 CLK=0; J=0; K=0;
		#5 CLK=0; J=0; K=1;
		#5 CLK=0; J=1; K=0;
		#5 CLK=0; J=1; K=1;
		#5 CLK=1; J=0; K=0;
		#5 CLK=0;
		#5 CLK=1; J=0; K=1;
		#5 CLK=0;
		#5 CLK=1; J=1; K=0;
		#5 CLK=0;
		#5 CLK=1; J=1; K=1;

		#5 CLK=0; reset=0;
		#5 CLK=1;
		#5 CLK=0;
		#5 CLK=1;
		#5 CLK=0;
		#5 CLK=1;
		#5 CLK=0;
		#5 CLK=1;
		#5 CLK=0;
		#5 CLK=1;
		#5 CLK=0;
		#5 CLK=1; reset=1;
```

```
        end
        Lab3 test(D,G,J,K,Q_JK,Q_D,reset,CLK,F);

endmodule

module Lab3(D, G, J, K, Q_JK, Q_D, reset, CLK, F);
        input J, K, reset, CLK, D, G;
        output Q_JK, Q_D;
        output [1:0] F;

        DF DLatchTest(Q_D, D, G);
        JK JKtest(J, K, Q_JK, CLK, reset);
        counter Countertest(CLK, reset, F);

endmodule

module counter(CLK, reset, F);
        input CLK, reset;
        output [1:0] F;
        wire J, K, Q0, Q1;

        assign K=1'b1;
        assign J=1'b1;

        JK JK_1(J, K, Q0, CLK, reset);
        JK JK_2(Q0, Q0, Q1, CLK, reset);
        assign F = {Q1, Q0};
endmodule

module JK(J, K, Q, CLK, reset);
        input J, K, CLK, reset;
        output reg Q;

        always @(posedge CLK) begin
                        if (reset == 1)begin
                                Q <= 0;
                        end
                        else if(J == 1'b0 && K == 1'b0)begin
                                Q <= Q;
                        end
                        else if (J == 1'b0 && K == 1'b1)begin
                                Q <= 1'b0;
                        end
                        else if (J == 1'b1 && K == 1'b0)begin
```

```
                        Q <= 1'b1;
                    end
                    else if (J == 1'b1 && K == 1'b1)begin
                        Q <= ~Q;
                    end
                end
endmodule

module DF (Q, D, clk);

input D;
input clk;

output reg Q;

always @ (posedge clk)
begin
        if (D == 1)
        Q <= 1;
        else
        Q <= 0;
 end

endmodule
```