ENGR 378 Digital Systems Design

Exp. 3: Latches, Flip-Flops and Sequential Circuits

Submitted by:

Björn Franzén
Kristian Lauszus

Sept. 25, 2014

Demonstration……..…..____/5

Report: Presentation ……...____/2

Problem analysis…….…….____/2

Design work…….____/5

Results……..……..____/5

Conclusion/Discussion…….…….____/1

Total……..…….____/ 20

Problem Analysis

For this lab we had to describe and synthesize a D latch, JK flip-flop and a two bit counter.

Hardware Design

Our approach to designing the counter was to first write down the truth table for the JK latch and realizing that we could implement the counter by cascading two of them. To get the intended function of the counter reversing its counting at either 0 or 3 we connected the inputs of a 2-bit MUX to Q and ‘Q and the output to clk. See the picture of our pre-lab report.

Our design synthesized into the following hardware realization.

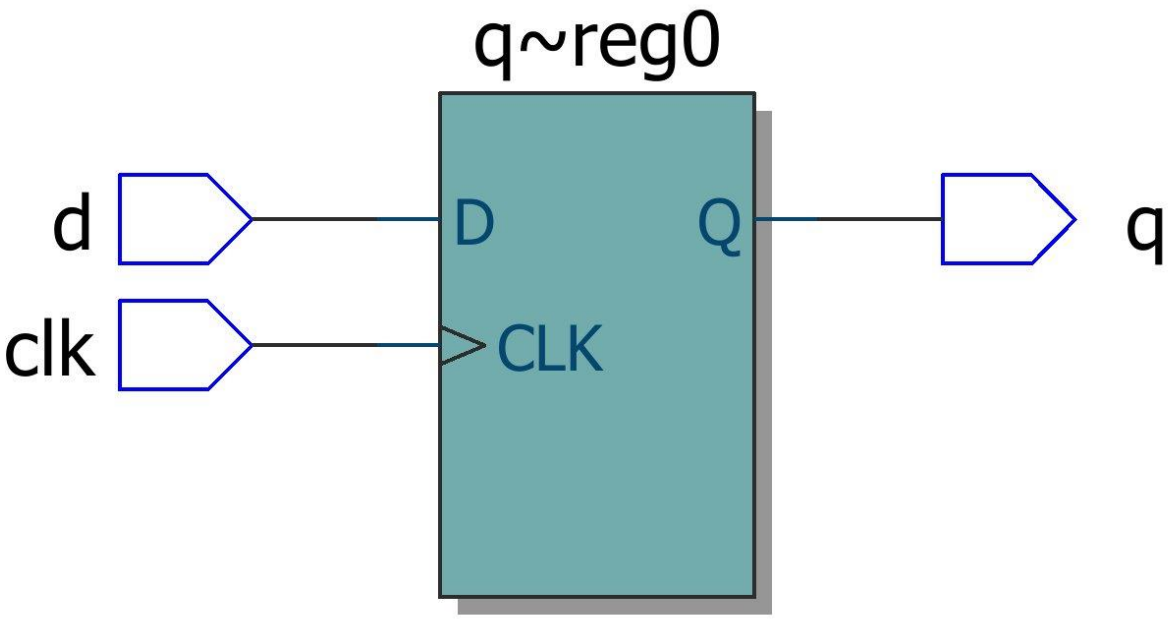

Figure 1 - Hardware synthesis of the dlatch.

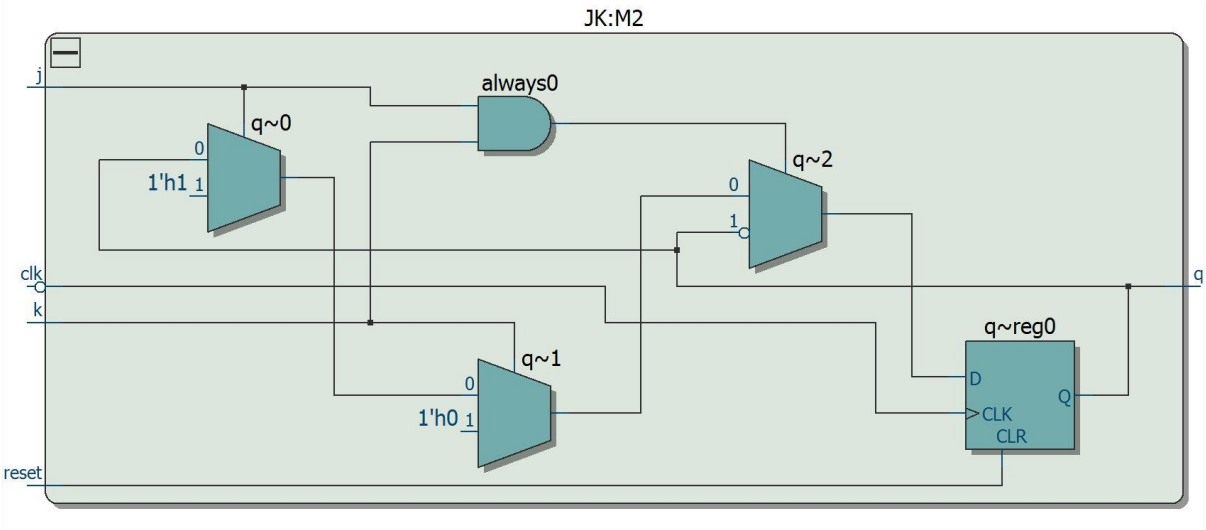

Figure 2 - Hardware synthesis of the JK Flip-Flop.

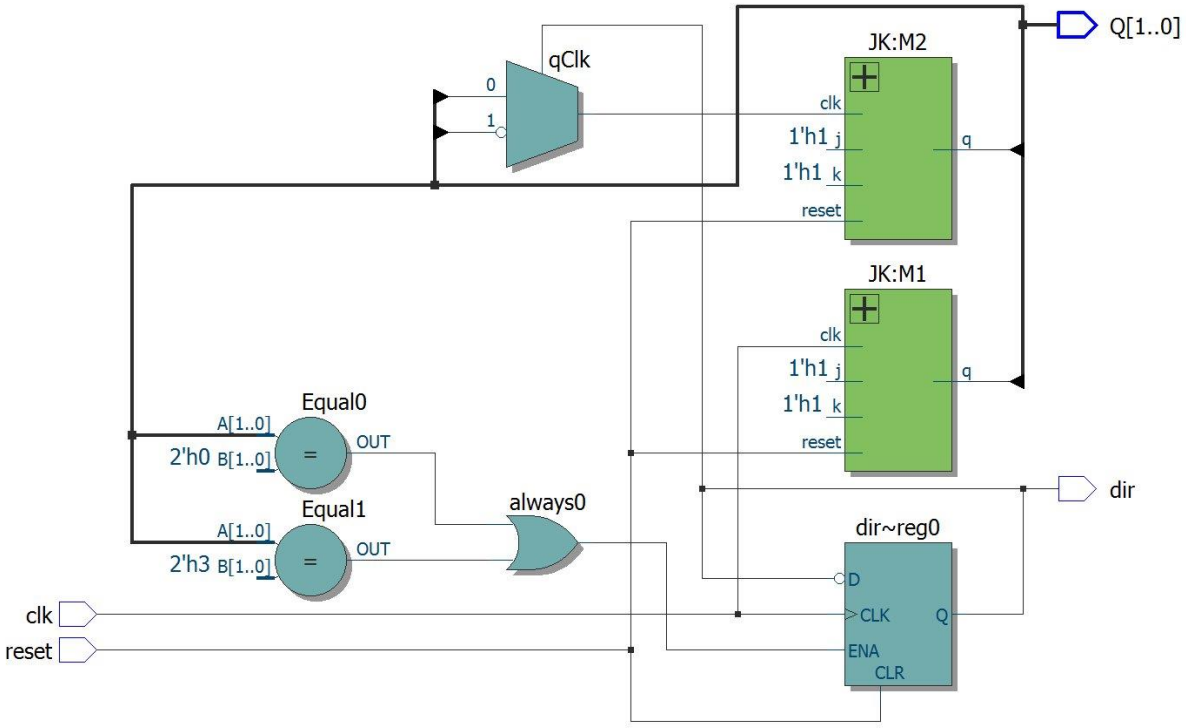

Figure 3 - Hardware synthesis of the 2bit counter.

Verilog Modelling

We broke down our design into the same elements as in our pre-lab report. That is, we constructed a positive-edge triggered JK flip-flip and connected those appropriately in the top design. To implement the MUX we used a behavioural always block.

Results/Verification

The waveforms found in this section comes from simulation of the test benches of the respective function. These test benches can be found in the appendix. Since we weren't dealing with a particular sophisticated design we simply checked the truth table of the JK flip-flop and D-latch. For the counter we just applied a continuous clock signal and reset it once.

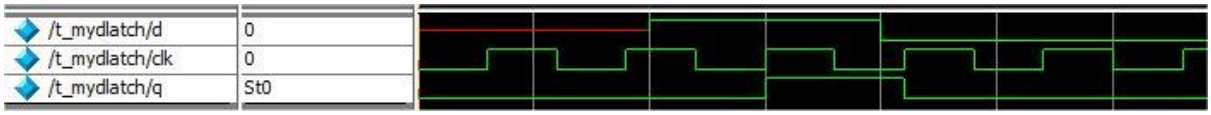

Figure 4 – Simulation waveform of the D latch.

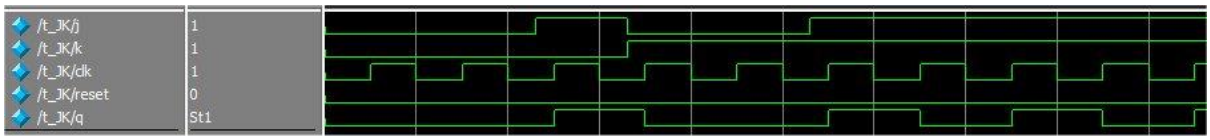

Figure 5 - Simulation waveform of the JK latch..

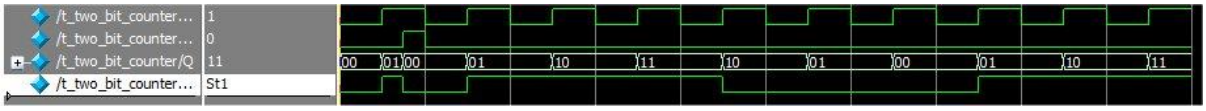

Figure 6 - Simulation waveform of the 2 bit counter.

Conclusions and Discussion

Implementing the D latch was trivial and doesn't require any explanation. When implementing the JK flip-flop, however, we had issues with inverting the output. First we used an if statement checking for if Q was undefined, i.e. "x", but it didn't work. We then used an initial q = 0 instead since it's only a simulation issue.

With the counter, however, we had issues with implementing the logic realizing the mux. First off we used an non-edge triggered always statement and an if statement inside checking for Q=0 or 3. However we then got problems with the output of our logic just producing pulses. This was because it implemented a latch and not a flip-flop for the control signal of the mux. The solution was to have the always statement be edge triggered on the clock and reset and the compiler implemented a flip-flop instead with the effect that the always statement was synchronous with the JK's.

Work Breakdown

All the assignment was created by a joint effort.

Pre-lab Report

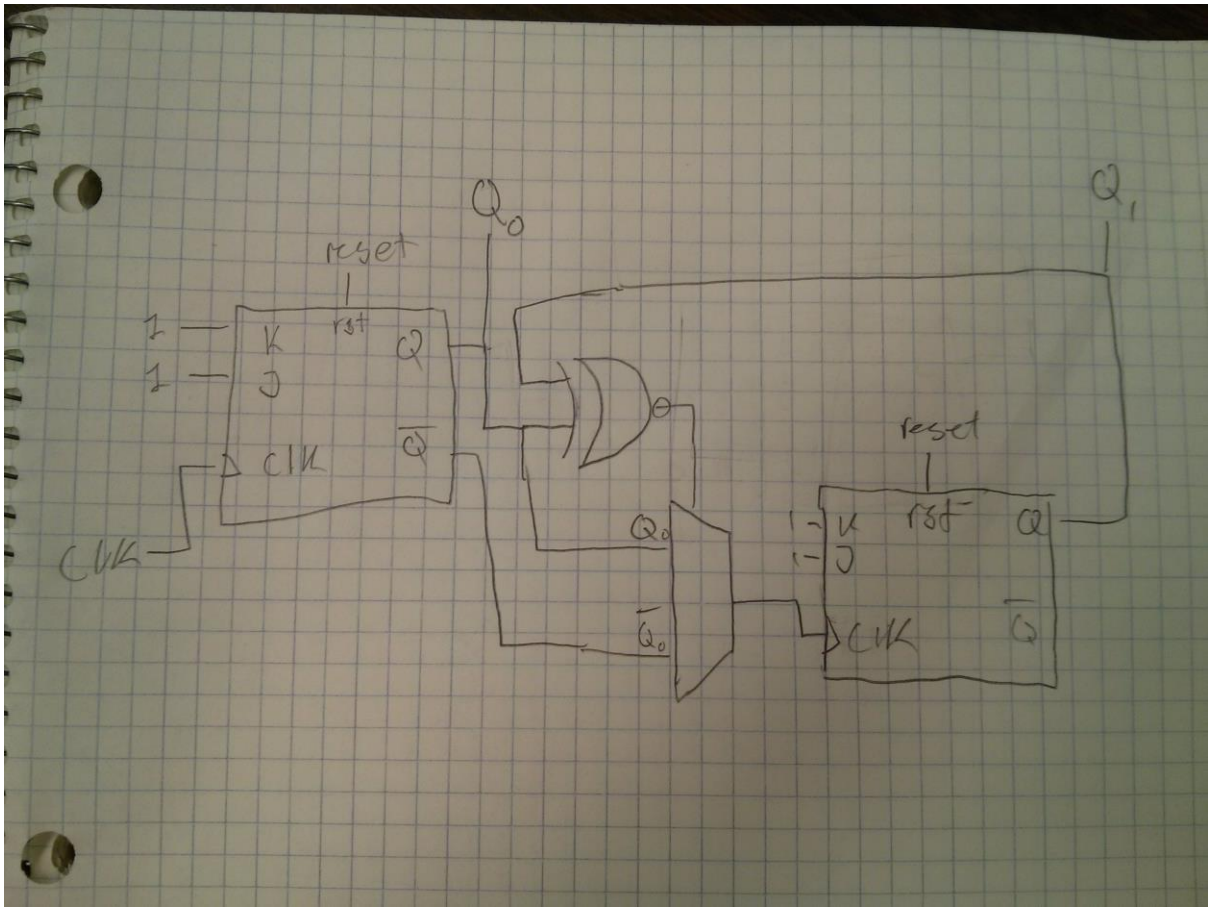

Figure 7 – The pre-lab report.

HDL Source Code

```
module two_bit_counter(
        output [1:0] Q,
        input clk, reset,
        output reg dir
);
        wire qClk;

        assign qClk = dir ? ~Q[0] : Q[0];

        JK M1(Q[0], 1'b1,1'b1, clk, reset);
        JK M2(Q[1], 1'b1,1'b1, qClk ,reset);

        initial dir = 0;

        always @ (posedge clk, posedge reset) begin
                if (reset) dir = 0;
                else if ((Q == 2'b00 | Q == 2'b11))
                        dir = ~dir;
        end
```

```
endmodule

module t_JK;
	reg j,k,clk, reset;
	wire q;

	JK M1(q,j,k,clk,reset);

	initial begin
		clk = 0; j = 0; k = 0; reset = 0;
	end
	always #5 clk = ~clk;


	initial #200 $stop;
	initial begin
		#2.5
		#10 {j,k} = 2'b00;
		#10 {j,k} = 2'b10;
		#10 {j,k} = 2'b01;
		#10
		#10 {j,k} = 2'b11;
	end


endmodule

module t_two_bit_counter;
	reg clk, reset;
	wire [1:0] Q;
	wire dir;

	two_bit_counter M1(.Q(Q),.clk(clk), .reset(reset), .dir(dir));

	initial begin
		clk = 0; reset = 0;
	end
	always #10 clk = ~clk;

	initial #200 $stop;
	initial begin
		#10;
		#5 reset = 1;
		#5 reset = 0;
	end

endmodule
module mydlatch(
	output reg q,
	input d,clk
);
	initial q = 0;

	always @ (posedge clk)
	begin
		if(d==1) q=1;
		else if(d==0) q=0;
		else q=q;
	end
endmodule

module t_mydlatch;
	reg d,clk;
	wire q;

	mydlatch M1(q,d,clk);
```

```
        initial #200 $stop;
        initial clk = 0;
        always #2.5 clk = ~clk;

        initial begin
                #10 d = 1;
                #10 d = 0;
        end
endmodule

module JK(
        output  reg q,
        input j,k,clk, reset
);

        initial q = 0;

        always @ (posedge clk, posedge reset)
        begin
                if(reset == 1) q = 0;
                else if(j&k) q=~q;
                else if(k==1) q=0;
                else if(j==1) q=1;
                else q=q;
        end
endmodule
```