CSCI 355 - Lab 6 Report

Student: Luka Karanovic
Student #: 665778833
Instructor: Ajay Shrestha

1 - Lab Procedures

6.1 Write the structural (i.e., using gate primitives) Verilog code for an SR Latch using two NOR gates with the module interface as shown below. Simulate using Vivado ML to verify the operation. Write Verilog testbench to simulate your design. Show screenshot of the Waveform window indicating correct operation.

Verilog:

<u>Filename: SRLatchNor.v</u>

```
module SRLatchNor(input wire s,
    input wire r,
    output wire q,
    output wire q_n);

    nor(q_n, s, q);
    nor(q, r, q_n);

endmodule
```

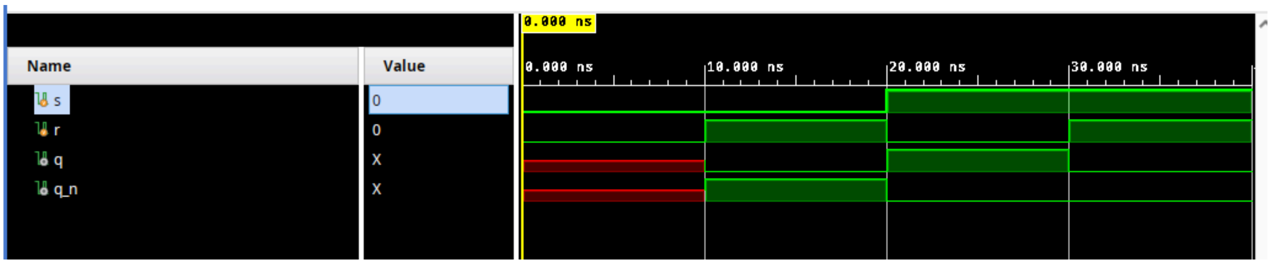
Testbench:

<u>Filename: SRLatchNor_tb.v</u>

```
module SRLatchNor_tb();
    reg s, r;
    wire q, q_n;

    SRLatchNor ut(s, r, q, q_n);

    initial begin
        s = 0;
        r = 0;
        #10;

        s = 0;
        r = 1;
        #10;

        s = 1;
        r = 0;
        #10;

        s = 1;
        r = 1;
        #10;

        $finish;
    end
```

```
endmodule
```

Timing Diagram:



- Since q and q_n didn't have any values before, they are in an undetermined state when s and r are both 0. They follow the characteristic table of an SR Latch

6.2 Write the behavioral Verilog code for a gated D Latch (with Enable input) with the module interface as shown below. Simulate using Vivado ML to verify the operation. Write Verilog testbench to simulate your design. Show screenshot of the Waveform window indicating correct operation.

Verilog:

Filename: GatedDLatch.v

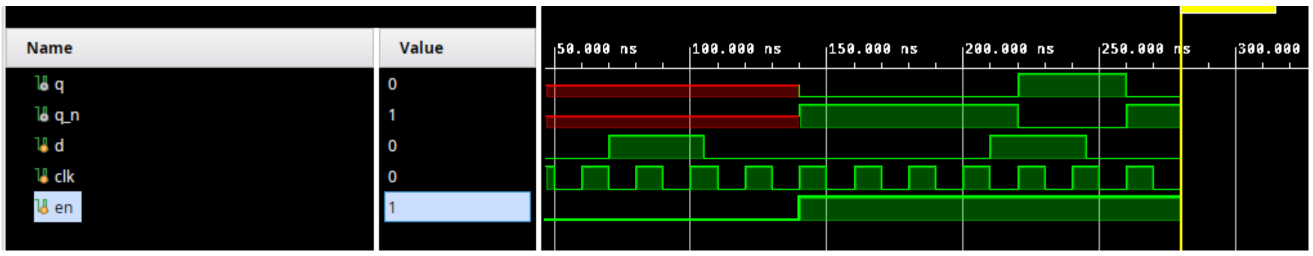
```
module GatedDLatch(
    input wire clk,
    input wire d,
    input wire en,
    output reg q,
    output reg q_n
    );

    always @(clk) begin
        if (clk & en) begin
            q = d;
            q_n = ~q;
        end
    end

endmodule
```

Testbench:

Filename: GatedDLatch_tb.v

```
module GatedDLatch_tb();
    wire q, q_n;
    reg d, en;
    reg clk = 0;
    GatedDLatch ut(clk, d, en, q, q_n);

    always begin
        clk = ~clk;
        #10;
    end


    initial begin
        en = 0; #35;
        d = 0; #35;
        d = 1; #35;
        d = 0; #35;

        en = 1; #35;
        d = 0; #35;
        d = 1; #35;
        d = 0; #35;
        $finish;
     end

endmodule
```

Timing Diagram:



- Q matches D on the positive edge of clk, and only when enabled.

6.3 Write the structural (i.e., using gate primitives) Verilog code for a positive edge triggered D FlipFlop with asynchronous reset (active low) using two modified Gated D Latches (include the "reset_n") as shown in figure below. Simulate using Vivado ML to verify the operation. Write Verilog testbench to simulate your design. Show screenshot of the Waveform window indicating correct operation.

Verilog:
Filename: DFlipFlop.v

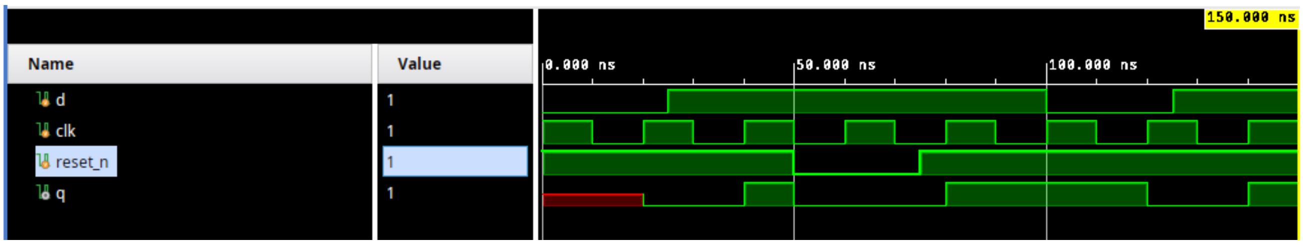
```
module DFlipFlop(input wire clk,
    input wire d,
    input wire reset_n,
    output wire q);
    wire w1, w2, w3, w4, w5, w6;

    wire nclk, nd, q_n;
    not(nclk, clk);
    not(nd, d);

    nand(w1, d, reset_n, nclk);
    nand(w2, nd, nclk);

    nand(w3, w1, w4);
    nand(w4, w2, reset_n, w3);

    nand(w5, w3, reset_n, clk);
    nand(w6, w4, clk);

    nand(q, w5, q_n);
    nand(q_n, w6, reset_n, q);

endmodule
```

Testbench:

Filename: DFlipFlop_tb.v

```
module DFlipFlop_tb();
    reg d = 0;
    reg clk = 0;
    reg reset_n = 1;
    wire q;

    DFlipFlop ut(clk, d, reset_n, q);

    always begin
        clk = ~clk;
        #10;
    end


    initial begin
        d = 0; #25;
        d = 1; #25;

        reset_n = 0; #25;
        reset_n = 1; #25;

        d = 0; #25;
        d = 1; #25;
        $finish;
    end


endmodule
```

Timing Diagram:



- Q always matches D on the positive edge of clk. Reset being 0 sets q to 0.

6.4 Write the behavioral Verilog code for an asynchronous up counter with the module interface as shown below. It should use the parameter "N" to set the size of the counter. Simulate using Vivado ML to verify the operation. Write Verilog testbench to simulate your design by showing all 16 counts for N=4. Show screenshot of the Waveform window indicating correct operation.

Verilog:

Filename: upcount.v

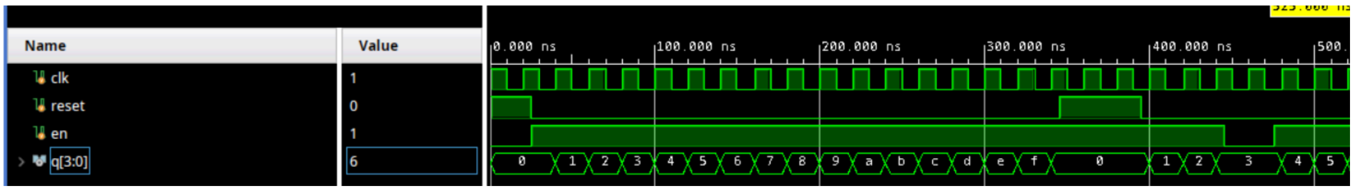
```
module upcount #(parameter N = 4) (input wire clk,
    input wire reset,
    input wire en,
    output reg[N-1: 0] q);

    always @(posedge clk) begin
        if (reset) begin
            q <= 0;
        end
        else if (en) begin
            q <= q + 1;
        end
    end
endmodule
```

Testbench:

Filename: upcount_tb.v

```
module upcount_tb();
    parameter N = 4;
    reg clk = 0;
    reg reset = 0;
    reg en = 1;
    wire [N-1:0] q;
    upcount #(N) ut(clk, reset, en, q);

    always begin
        clk = ~clk;
        #10;
    end


    initial begin
        // initialize
        reset = 1;
        en = 0;
        #25;

        // count up
        reset = 0;
        en = 1;
        #320;

        // reset
        reset = 1; #50;
        reset = 0;

        // count up again
        #50;

        // stop count for a bit (disable)
        en = 0; #30;

        // continue counting
        en = 1; #50;

        $finish;
    end

endmodule
```

Timing Diagram:



- Counts up to 15, then resets to 0 (before reset is triggered). Will start counting again.
- Disabling the upcounter pauses it.

6.5 Write the structural Verilog code for a Johnson counter (you can use generate to create instances of the DFlipFlop you created in the previous section 6.3). The module interface is shown below. It should use the parameter "N" to set the size of the counter. Simulate using Vivado ML to verify the operation. Write Verilog testbench to simulate your design for N=8. Show screenshots of the Waveform window indicating correct operation.

Verilog:
Filename: JohnsonCounter.v

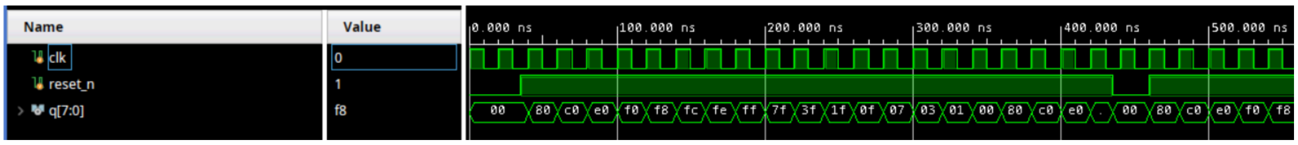
```
module JohnsonCounter #(parameter N = 8) (
   input wire reset_n,
   input wire clk,
   output wire [N-1:0] q);

   wire n_last_q;

   not(n_last_q, q[0]);


   DFlipFlop stage0(clk, n_last_q, reset_n, q[N-1]); // Assign first input as ~q of last input

   genvar i;
   generate
      for (i = 1; i < N; i = i+1) begin
         begin: bitShift
            DFlipFlop stage(clk, q[i], reset_n, q[i-1]);
         end
      end
   endgenerate
   // make flip flops and do bit shift
   always @(posedge clk) begin

   end
endmodule
```

Testbench:
Filename: JohnsonCounter_tb.v

```
module JohnsonCounter_tb();
   parameter N = 8;
   reg clk = 0;
   reg reset_n = 0;
   wire [N-1:0] q;
```

```
    JohnsonCounter #(8) ut (reset_n, clk, q);

    always begin
        clk = ~clk;
        #10;
    end

    initial begin
        #10;
        reset_n = 0;
        #25;
        reset_n = 1;

        #400;

        reset_n = 0;
        #25;
        reset_n = 1;

        #100;

        $finish;
    end

endmodule
```

Timing Diagram:



- Follows the 8-bit Johnson Counter sequence: 00 - 80 - c0 - e0 …