CA4-Report

Digital Systems I

Spring 1402-03
Digital Systems I – ECE894

Amirmortaza Rezaee
810101429

Problem 1

Figures 1.1 and 1.2 illustrate the circuit we are trying to build:

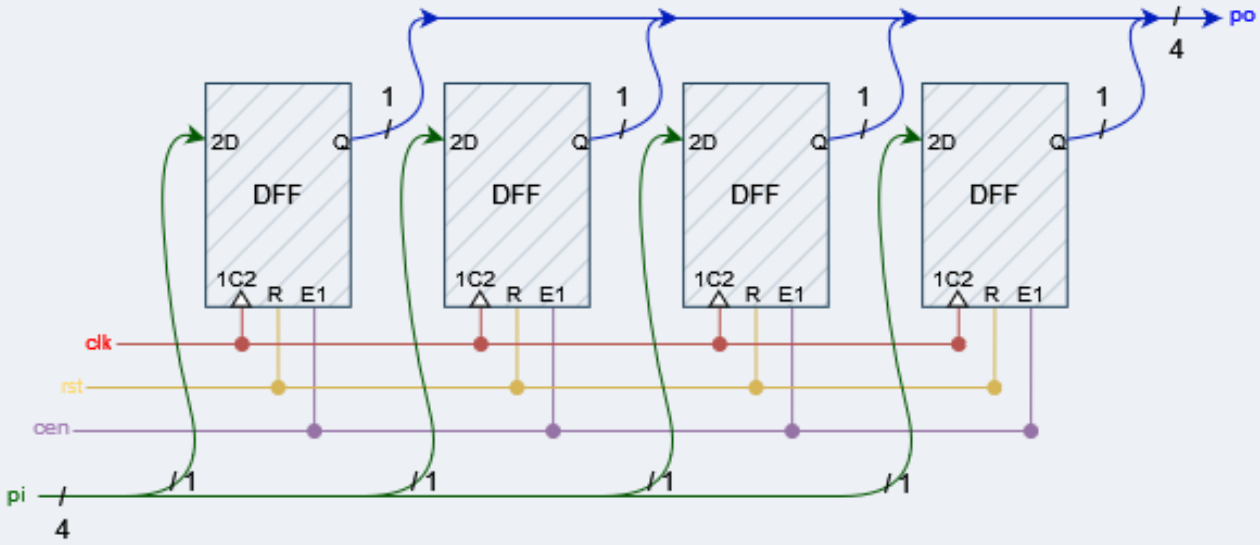

Figure 1.1

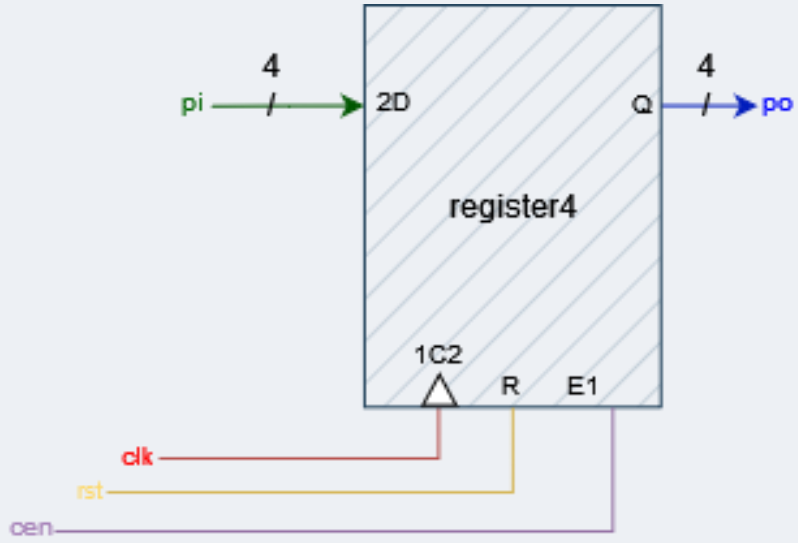

Figure 1.2

The System-Verilog description for this 4-bit register:

```
module register4(input [3:0] PI, input rst, clk, cen, output logic [3:0] PO);

    always @(posedge clk , posedge rst) begin
        if (rst) PO <= 4'd0;
        else
            PO <= (cen) ? PI : PO;
    end
endmodule
```

Figure 1.3

```
`timescale 1ns/1ns
module Reg4TB ();
    logic [3:0] T1PI = 4'd0;
    logic T1clk = 1'b0;
    logic T1rst = 0;
    logic T1cen = 0;
    wire [3:0] T1PO;

    register4 CUT1 (T1PI, T1rst, T1clk, T1cen, T1PO);

    always #10 T1clk <= ~T1clk;

    initial begin
        #7 T1PI = $random;
        #5 T1cen = 1'b1;
        #5 T1PI = $random;
        #15 T1cen = 1'b0;
        #7 T1cen = 1'b1;
        #9 T1PI = $random;
        #5 T1rst = 1'b1;
        #12 T1PI = $random;
        #5 T1cen = 1'b0;
        #9 T1cen = 1'b1;
        #14 T1rst = 1'b0;
        #100 $stop;
    end

endmodule
```

Figure 1.4

In this Testbench, clock's positive edge occurs every 20 NS. So:

t=7 -> clk=0, cen=0, rst=0, pi changes -> po =0000

t=10 -> clk=1, cen=0, rst=0 -> po =0000

t=12 -> clk=1, cen=1, rst=0 -> po =0000

t=17 -> clk=1, cen=1, rst=0 pi changes -> po =0000

t=20 -> clk=0, cen=1, rst=0 -> po =0000

t=30 -> clk=1, cen=1, rst=0 -> ***po =pi***

t=32 -> clk=1, cen=0, rst=0 -> po =pi

t=39 -> clk=1, cen=1, rst=0 -> po =pi

t=40 -> clk=0, cen=1, rst=0 -> po =pi

t=48 -> clk=0, cen=1, rst=0 pi changes -> po =previous value of pi

t=50 -> clk=1, cen=1, rst=0 -> ***po =pi***

t=53 -> clk=1, cen=1, rst=1 -> ***po =0000***

t=60 -> clk=0, cen=1, rst=1 -> po =0000

t=65 -> clk=0, cen=1, rst=1 pi changes -> po =0000

t=70 -> clk=1, cen=0, rst=1 -> po =0000

t=79 -> clk=1, cen=1, rst=1 -> po =0000

t=80 -> clk=0, cen=1, rst=1 -> po =0000

t=90 -> clk=1, cen=1, rst=1 -> po =0000

t=93 -> clk=1, cen=1, rst=0 -> po =0000

t=100 -> clk=0, cen=1, rst=0 -> po =0000

t=110 -> clk=1, cen=1, rst=0 -> ***po =pi***

The figure below, shows the result of this simulation:

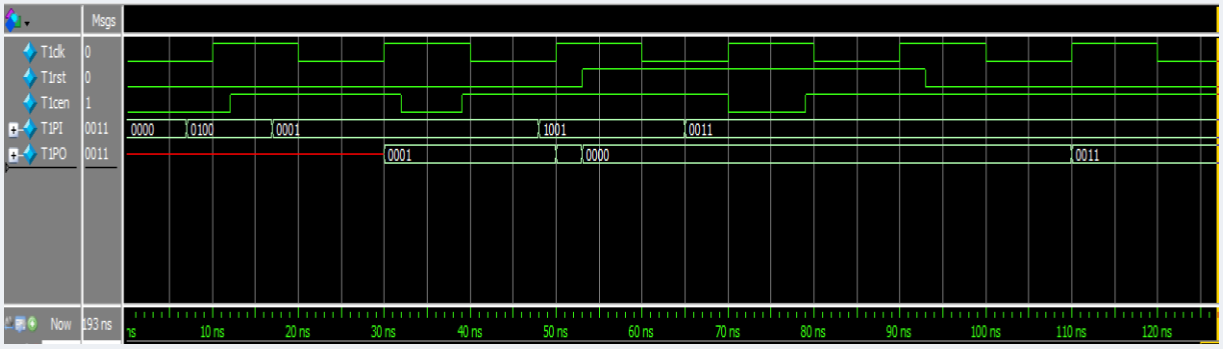

Figure 1.5

As it's obvious, the structure works properly as expected. Which means the output becomes 0000 as rst becomes 1 unaware to the clock; and the output only changes at the positive edges of the clock when cen is 1!

Problem 2

Figure 2.1 illustrates the circuit we are trying to build using 4-bit registers of previous problem:

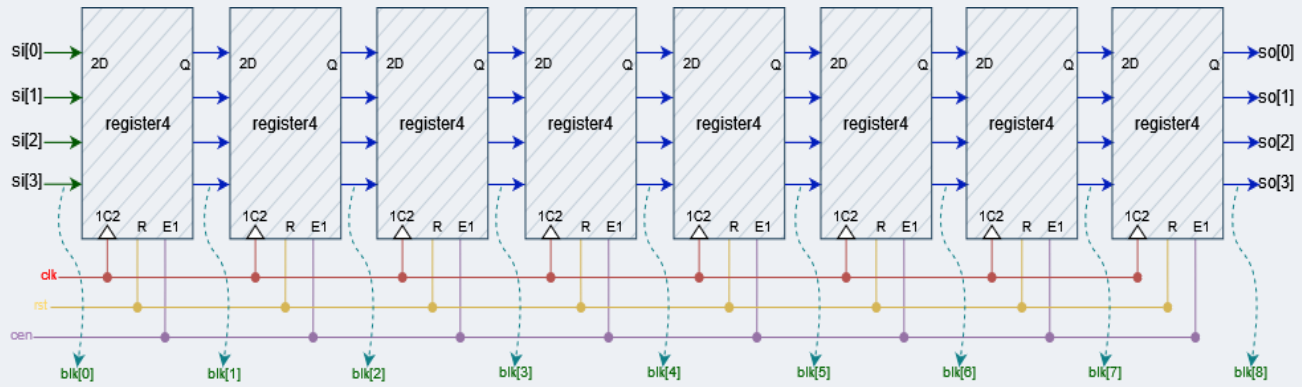

Figure 2.1

*As it's shown in the diagram above, we need to declare a 2-D set of wires (here **blk**), that blk [0] [0:3] equals to si [0:3] and blk [8][0:3] equals to so[0:3]. And the other elements of blk are the inputs and the outputs of registers of previous problem that are cascaded together.*

Here is the System Verilog description of this structure:

```
module shifterQ2 (input  [3:0] si , input rst , clk , shn, output logic [3:0] so);
    wire [0:8][3:0] blk;
    assign so = blk[8];
    assign blk[0] = si;
    genvar i;
    generate
        for (i=0;i<8;i=i+1) begin: procedure
            register4 rgi( blk[i] , rst, clk , shn , blk[i+1]);
        end
    endgenerate
endmodule
```

Figure 2.2

Figure 2.3 is the Testbench written for this structure:

```
`timescale 1ns/1ns
module shifterQ2TB ();
    logic [3:0] T2SI = 4'd0;
    logic T2clk = 1'b0;
    logic T2rst = 0;
    logic T2shn = 0;
    wire [3:0] T2SO;

    shifterQ2 CUT2 (T2SI, T2rst, T2clk, T2shn, T2SO);

    always #5 T2clk <= ~T2clk;

    initial begin
        #1 T2rst = 1'd1;
        #2 T2rst = 1'd0;
        #4 T2shn = 1'b1;
        #2 T2SI = $random+$random;
        #8 T2SI = 4'd0;
        #160 $stop;
    end

endmodule
```

Figure 2.3

In this Testbench, the positive edge of clock occurs every 10 NS. So, as asynchronous reset turns on and off, we expect all registers to get the value of 0 at t=1 NS. Then shn turns on at t=7 NS and si [0:3] gets some random value; so, at t=10 NS at positive edge of clock, the output of first layer (blk [1]) will get the value of si, from now then every 10 NS this value will propagate to the next block. Also, si gets the value of 0000 at t=17 NS, so on the next positive edge of the clock, 0000 starts to propagate through the blocks!

Simulation results are shown in the figure below:

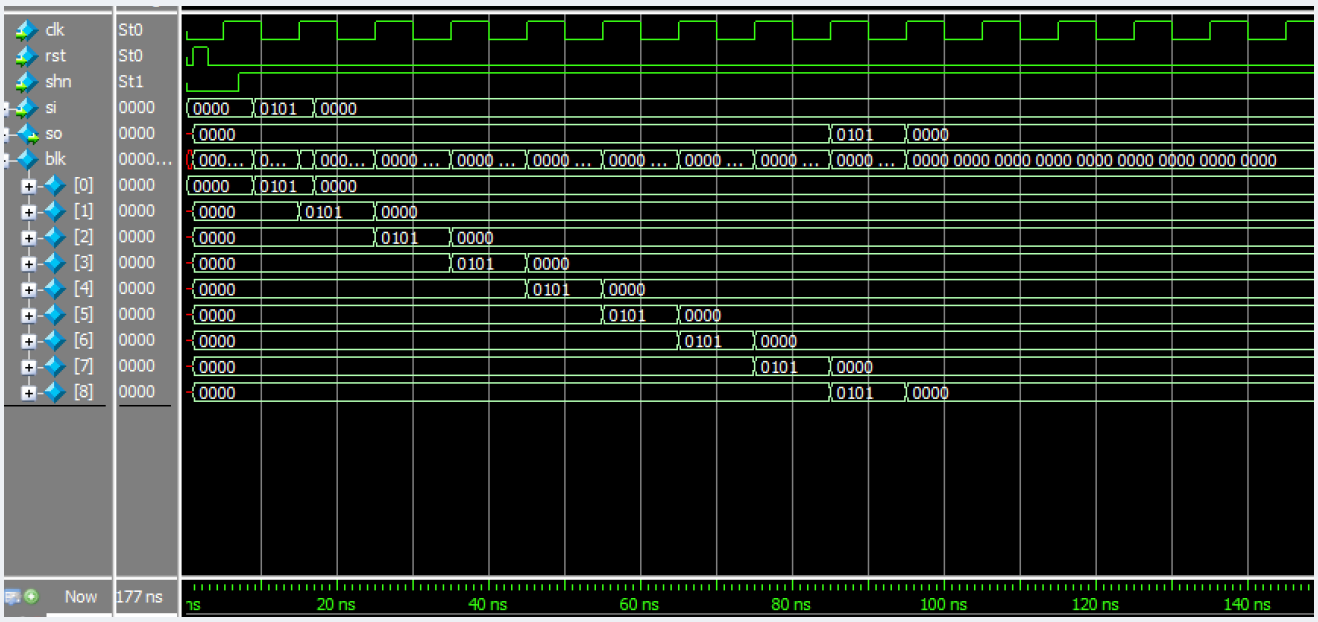

Figure 2.4

As shown in the figure, the act of shifting is done properly as expected.

Problem 3

a.

The System Verilog description of a two-dimensional array for shifting 4-bit data within the depth of the array using a ***non-blocking*** *assignment in a for-loop:*

```
module shifterQ3a (input  [3:0] si , input rst , clk , shn, output logic [3:0] so);
    logic [0:8][3:0] blk;
    assign so = blk[8];

    always @(posedge clk , posedge rst) begin
        integer i;
        blk[0] = si;
        for(i=0;i<8;i=i+1) begin
            if (rst) blk[i+1] <= 4'd0;
            else
                blk[i+1] <= (shn) ? blk[i] : blk[i+1];

        end
    end
endmodule
```

Figure 3.a.1

The simulation results of running the same Testbench for this structure:

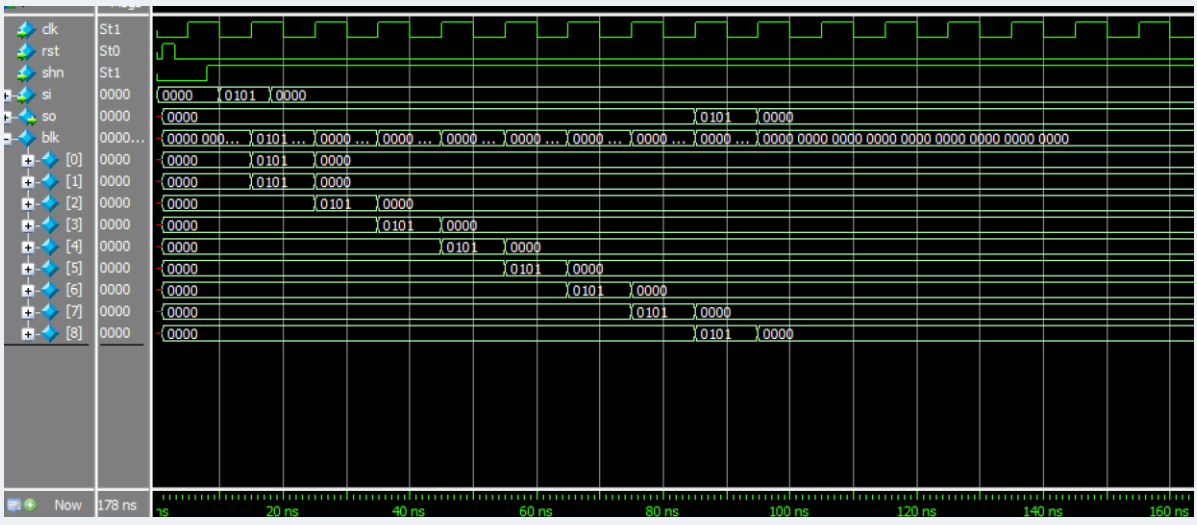

Figure 3.a.2

b.

The System Verilog description of a two-dimensional array for shifting 4-bit data within the depth of the array using a ***blocking*** *assignment in a for-loop:*

```
module shifterQ3b (input  [3:0] si , input rst , clk , shn, output logic [3:0] so);
    logic [0:8][3:0] blk;
    assign so = blk[8];

    always @(posedge clk , posedge rst) begin
        integer i;
        blk[0] = si;
        for(i=0;i<8;i=i+1) begin
            if (rst) blk[i+1] = 4'd0;
            else
                blk[i+1] = (shn) ? blk[i] : blk[i+1];

        end
    end
endmodule
```

Figure 3.b.1

The simulation results of running the same Testbench for this structure:

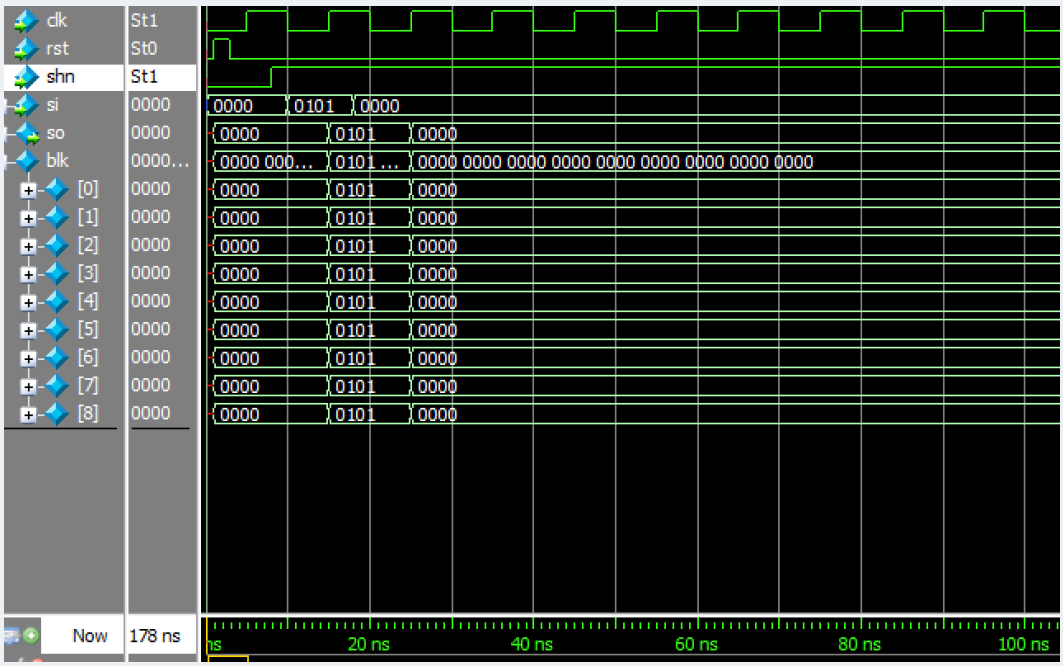

Figure 3.b.2

Using a blocking assignment causes the propagation of input through the output in just one clock.

To fix this problem, we can use a descending order in the for-loop instead of an ascending order, as shown below:

```
module shifterQ3bfixed (input  [3:0] si , input rst , clk , shn, output logic [3:0] so);
    logic [0:8][3:0] blk;
    assign so = blk[8];

    always @(posedge clk , posedge rst) begin
        integer i;
        blk[0] = si;
        for(i=7;i>=0;i=i-1) begin
            if (rst) blk[i+1] = 4'd0;
            else
                blk[i+1] = (shn) ? blk[i] : blk[i+1];

        end
    end
endmodule
```

Figure 3.b.3

The simulation results of running the same Testbench for this structure:

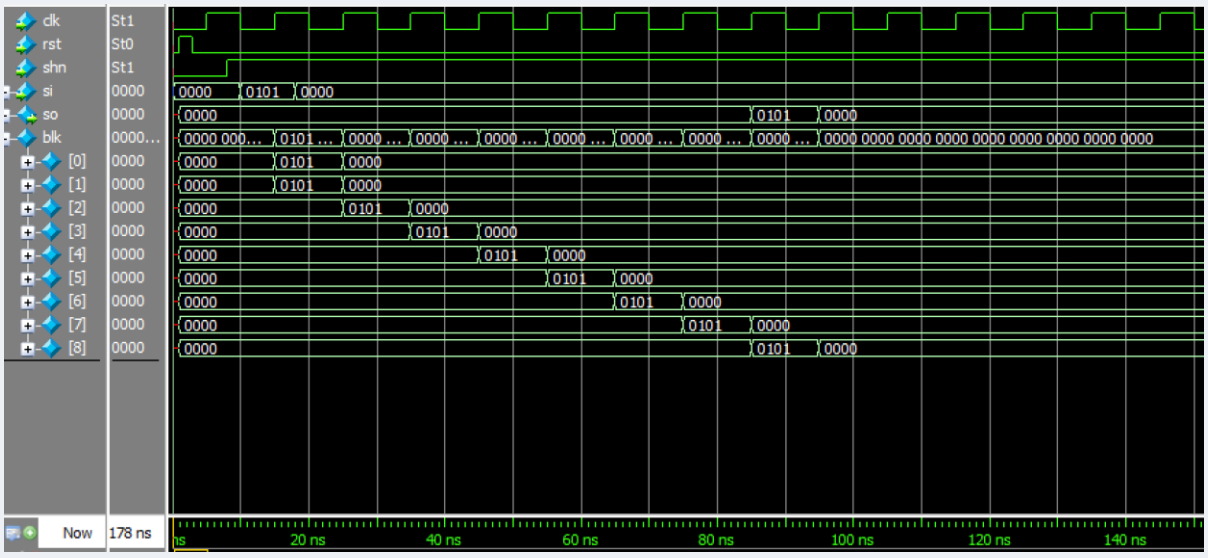

Figure 3.b.4

c.

In blocking assignments, the statements are executed sequentially in the order they appear in the code. The right-hand side of a blocking assignment is evaluated immediately, and the assignment is completed before moving on to the next statement. Actually, the value of the variable is updated immediately in the same time step.
On the other hand, in non-blocking assignments, all the assignments on the right-hand side are evaluated simultaneously, and the actual updates to the variables happen at the end of the time step. In other words, the value of the variable is updated at the end of the time step, after all other statements have been evaluated.