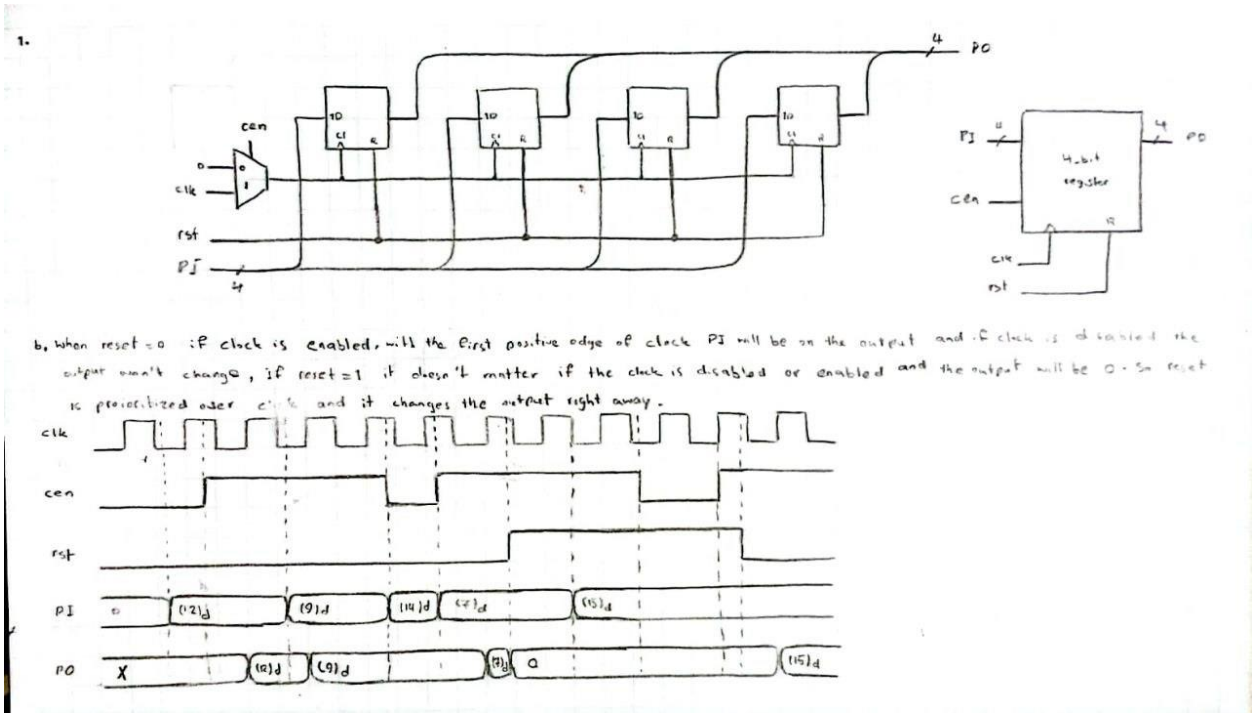
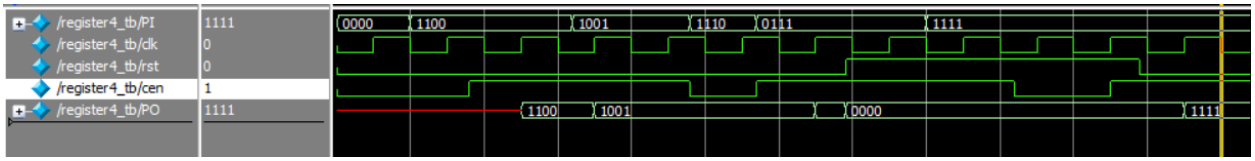
1.



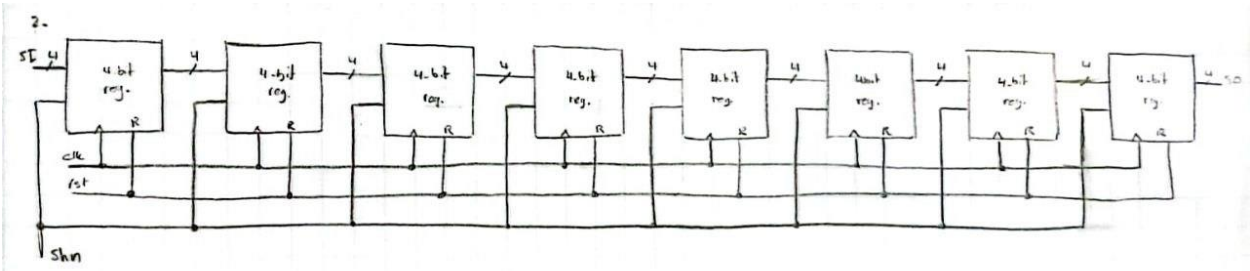
b, when reset = 0 if clock is enabled, will the first positive edge of clock PI will be on the output and if clock is disabled the output won't change, if reset = 1 it doesn't matter if the clock is disabled or enabled and the output will be 0. So reset is prioritized over clk % and it changes the output right away.


```
1   `timescale 1ns/1ns
2
3   module register4(input [3:0] PI , input clk , cen , rst , output reg [3:0] PO);
4       always @ (posedge clk , posedge rst) begin
5           if (rst) PO <= 4'd0;
6           else if (cen) PO <= PI;
7       end
8   endmodule
9
```

```
1   `timescale 1ns/1ns
2
3   module register4_tb();
4       reg [3:0] PI = 4'd0;
5       reg clk = 1'b0;
6       reg rst = 1'b0;
7       reg cen = 1'b0;
8       wire [3:0] PO;
9
10      register4 UUT (PI , clk , cen , rst , PO);
11
12      always #5 clk <= ~clk;
13
14      initial begin
15          #10 PI = 4'd12;
16          #8 cen = 1'b1;
17          #14 PI = 4'd9;
18          #16 cen = 1'b0;
19          PI = 4'd14;
20          #9 cen = 1'b1;
21          PI = 4'd7;
22          #12 rst = 1'b1;
23          #11 PI = 4'd15;
24          #12 cen = 1'b0;
25          #13 cen = 1'b1;
26          #4 rst = 1'b0;
27          #15 $stop;
28      end
29  endmodule
30
```



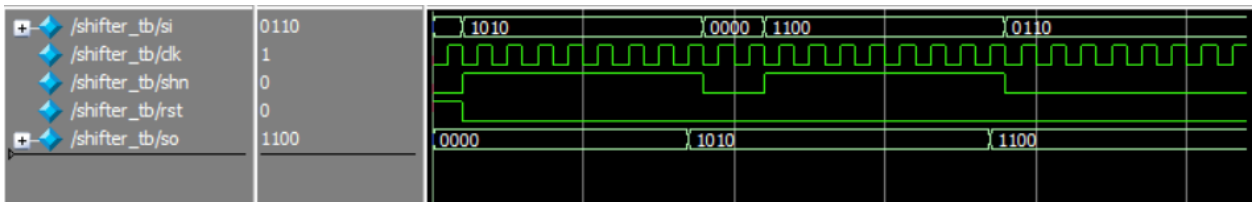
2.



b. The simulation shows that it takes exactly 8 clock cycles to shift the data to last register and we have 8 registers so shifting the data to next register takes 1 clock cycle.

```
1   `timescale 1ns/1ns
2
3   module shifter (input [3:0] si , input clk , shn , rst , output reg [3:0] so);
4       wire [3:0] stage [0:7];
5
6       genvar i;
7       generate
8           for (i = 0; i < 8; i = i + 1) begin : shift_stage
9               if (i == 0) begin
10                  register4 stage_reg (si , clk , shn , rst , stage[i]);
11              end else begin
12                  register4 stage_reg (stage[i-1] , clk , shn , rst , stage[i]);
13              end
14          end
15      endgenerate
16
17      assign so = stage[7];
18
19  endmodule
20
```

```
1   `timescale 1ns/1ns
2
3   module shifter_tb();
4       reg [3:0] si;
5       reg clk = 1'b0;
6       reg shn;
7       reg rst;
8
9       wire [3:0] so;
10
11      shifter UUT(si , clk , shn , rst , so);
12
13      always #5 clk <= ~clk;
14
15      initial begin
16          si = 4'b0000;
17          shn = 1'b0;
18          rst = 1'b1;
19          #10 rst = 1'b0;
20          si = 4'b1010;
21          shn = 1'b1;
22
23          #80 shn = 1'b0;
24          si = 4'b0000;
25
26          #20 si = 4'b1100;
27          shn = 1'b1;
28
29          #80 si = 4'b0110;
30          shn = 1'b0;
31
32          #80 $stop;
33      end
34  endmodule
35
```

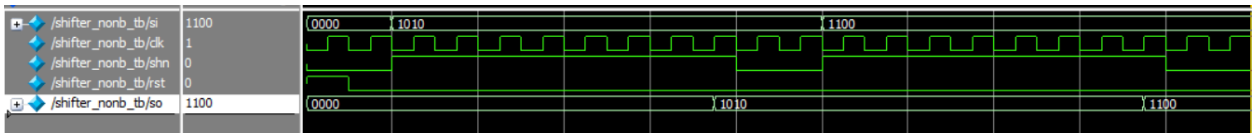


3.

a.

```
1   `timescale 1ns/1ns
2
3   module shifter_nonb(input [3:0] si , input clk , shn , rst , output reg [3:0] so);
4       reg [3:0] s_array [0:7];
5       integer i;
6
7       always @(posedge clk or posedge rst) begin
8           if (rst) begin
9               for (i = 0; i < 8; i = i + 1) begin
10                  s_array[i] <= 4'd0;
11              end
12          end
13          else if (shn) begin
14              s_array[0] <= si;
15              for (i = 7; i > 0; i = i - 1) begin
16                  s_array[i] <= s_array[i-1];
17              end
18          end
19      end
20
21      assign so = s_array[7];
22
23  endmodule
24
25
```

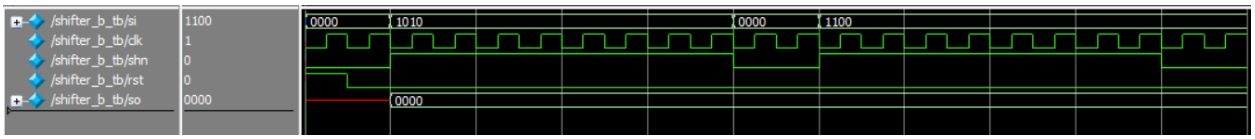
```
1   `timescale 1ns/1ns
2
3   module shifter_nonb_tb();
4       reg [3:0] si;
5       reg clk = 1'b0;
6       reg shn;
7       reg rst;
8       wire [3:0] so;
9
10      shifter_nonb UUT(si, clk, shn, rst, so);
11
12      always #5 clk = ~clk;
13
14      initial begin
15          si = 4'b0000;
16          shn = 0;
17          rst = 1;
18          #10 rst = 0;
19          #10 shn = 1;
20          si = 4'b1010;
21          #80 shn = 0;
22          #20 si = 4'b1100;
23          shn = 1;
24          #80 shn = 0;
25          #20 $stop;
26      end
27  endmodule
28
29
```



b.

```
1   `timescale 1ns/1ns
2
3   module shifter_b(input [3:0] si , input clk , shn , rst , output reg [3:0] so);
4       reg [3:0] s_array [0:7];
5       integer i;
6
7       always @(posedge clk or posedge rst) begin
8           if (rst) begin
9               for (i = 0; i < 8; i = i + 1) begin
10                  s_array[i] = 4'd0;
11              end
12          end
13          else if (shn) begin
14              s_array[0] = si;
15              for (i = 7; i > 0; i = i - 1) begin
16                  s_array[i] = s_array[i-1];
17              end
18          end
19      end
20
21      assign so = s_array[7];
22
23  endmodule
24
```

```
1   `timescale 1ns/1ns
2
3   module shifter_b_tb();
4       reg [3:0] si;
5       reg clk = 1'b0;
6       reg shn;
7       reg rst;
8       wire [3:0] so;
9
10      shifter_b UUT(si , clk , rst , shn , so);
11
12      always #5 clk = ~clk;
13
14      initial begin
15          si = 4'b0000;
16          shn = 1'b0;
17          rst = 1'b1;
18          #10 rst = 1'b0;
19          #10 si = 4'b1010;
20          shn = 1'b1;
21          #80 shn = 1'b0;
22          si = 4'b0000;
23          #20 si = 4'b1100;
24          shn = 1'b1;
25          #80 shn = 1'b0;
26          #20 $stop;
27      end
28  endmodule
29
```



C.

8. c. Like we use blocking method (=) the target variable will be updated right away during the simulation but in non-blocking assignment (<=) variables will be updated at the end of each cycle in simulation and it's better for describing sequential logic.