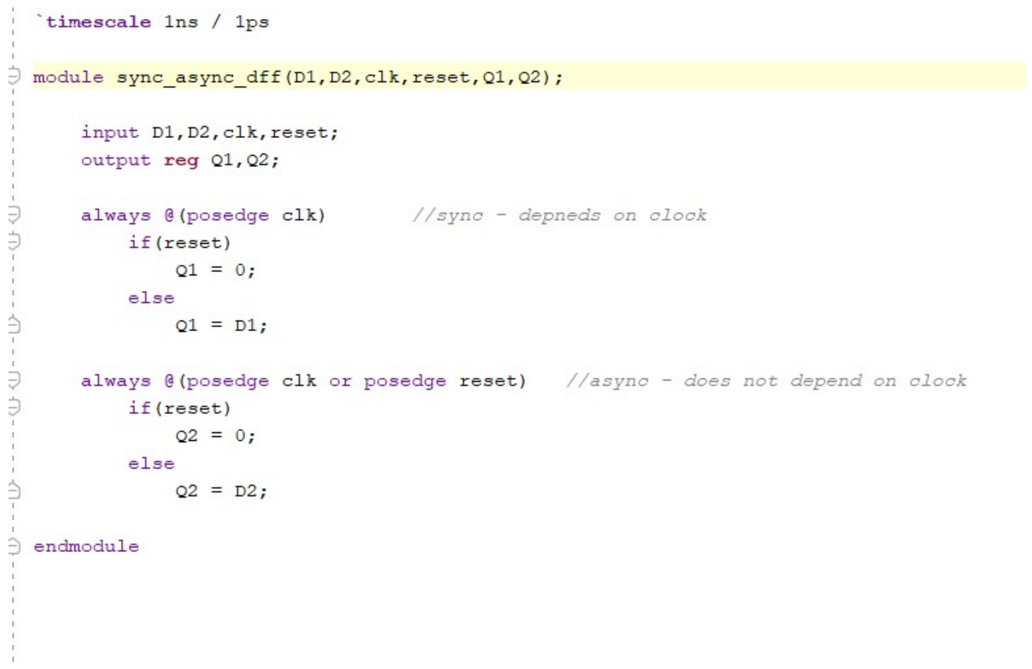
Digital Design Lab 6 Report

Palaskar Adarsh Mahesh
B20EE087

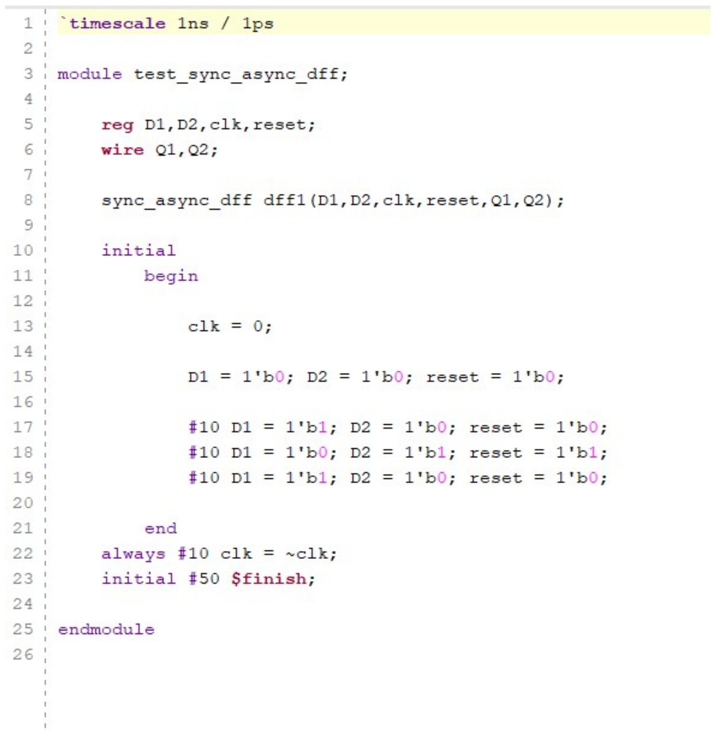
March 4, 2022

1 D flip flop with synchronous and asynchronous reset:



```verilog
`timescale 1ns / 1ps

module sync_async_dff(D1,D2,clk,reset,Q1,Q2);

    input D1,D2,clk,reset;
    output reg Q1,Q2;

    always @(posedge clk)       //sync - depneds on clock
        if(reset)
            Q1 = 0;
        else
            Q1 = D1;

    always @(posedge clk or posedge reset)   //async - does not depend on clock
        if(reset)
            Q2 = 0;
        else
            Q2 = D2;

endmodule
```

Figure 1: Verilog code for D flip flop with synchronous and asynchronous reset:



```
`timescale 1ns / 1ps

module test_sync_async_dff;

    reg D1,D2,clk,reset;
    wire Q1,Q2;

    sync_async_dff dff1(D1,D2,clk,reset,Q1,Q2);

    initial
        begin

            clk = 0;

            D1 = 1'b0; D2 = 1'b0; reset = 1'b0;

            #10 D1 = 1'b1; D2 = 1'b0; reset = 1'b0;
            #10 D1 = 1'b0; D2 = 1'b1; reset = 1'b1;
            #10 D1 = 1'b1; D2 = 1'b0; reset = 1'b0;

        end
    always #10 clk = ~clk;
    initial #50 $finish;

endmodule
```

Figure 2: Test bench for D flip flop with synchronous and asynchronous reset

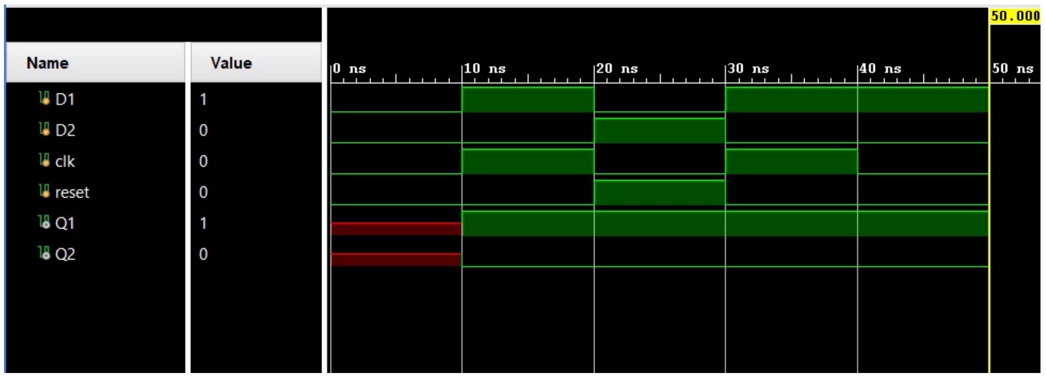
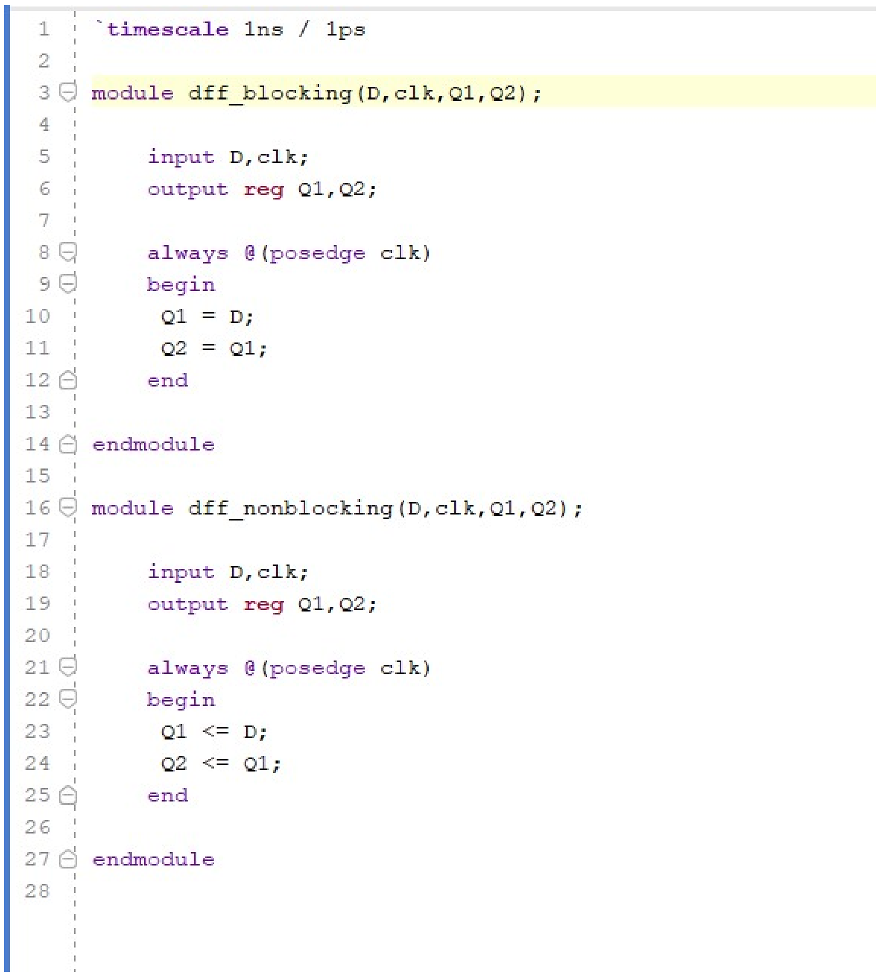


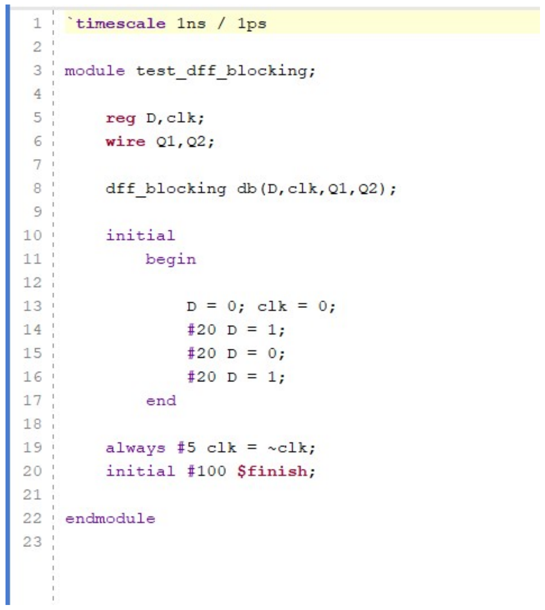
Figure 3: Simulation wave forms:

2 Testing the operation of blocking and non-blocking assignments using two D flip flops:



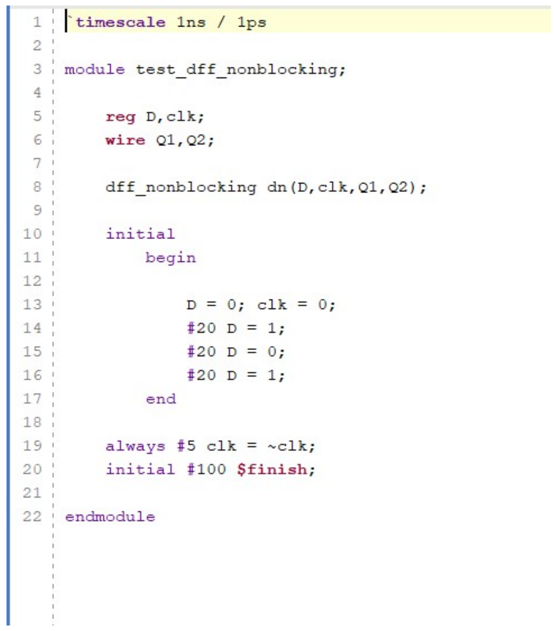
```verilog
`timescale 1ns / 1ps

module dff_blocking(D,clk,Q1,Q2);

    input D,clk;
    output reg Q1,Q2;

    always @(posedge clk)
    begin
     Q1 = D;
     Q2 = Q1;
    end

endmodule

module dff_nonblocking(D,clk,Q1,Q2);

    input D,clk;
    output reg Q1,Q2;

    always @(posedge clk)
    begin
     Q1 <= D;
     Q2 <= Q1;
    end

endmodule
```

Figure 4: Verilog code



```
`timescale 1ns / 1ps

module test_dff_blocking;

    reg D,clk;
    wire Q1,Q2;

    dff_blocking db(D,clk,Q1,Q2);

    initial
        begin

            D = 0; clk = 0;
            #20 D = 1;
            #20 D = 0;
            #20 D = 1;
        end

    always #5 clk = ~clk;
    initial #100 $finish;

endmodule
```

Figure 5: Test bench for blocking assignments



```
`timescale 1ns / 1ps

module test_dff_nonblocking;

    reg D,clk;
    wire Q1,Q2;

    dff_nonblocking dn(D,clk,Q1,Q2);

    initial
        begin

            D = 0; clk = 0;
            #20 D = 1;
            #20 D = 0;
            #20 D = 1;
        end

    always #5 clk = ~clk;
    initial #100 $finish;

endmodule
```

Figure 6: Test bench for non blocking assignments

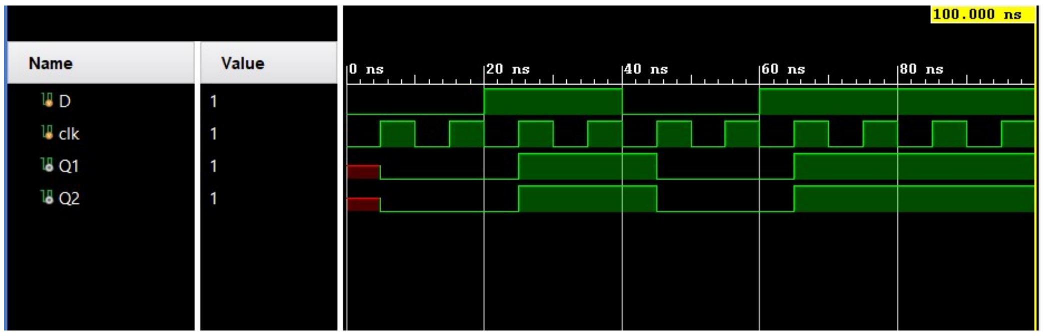


Figure 7: Simulation wave form for blocking assignments

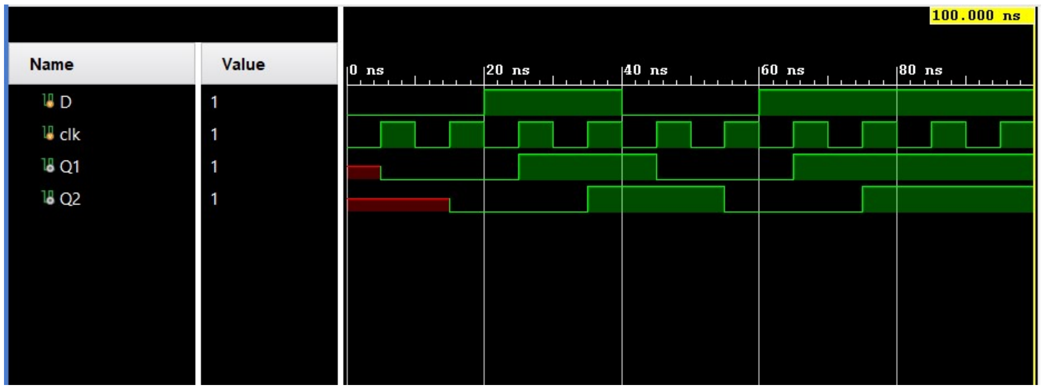
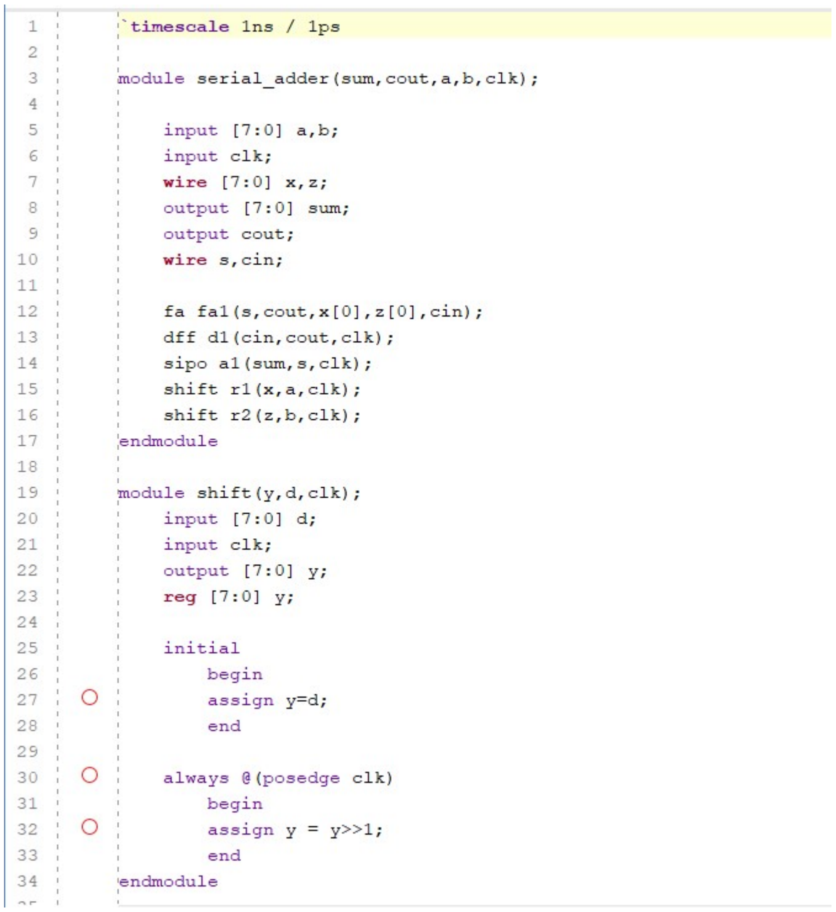


Figure 8: Simulation wave form for non blocking assignments

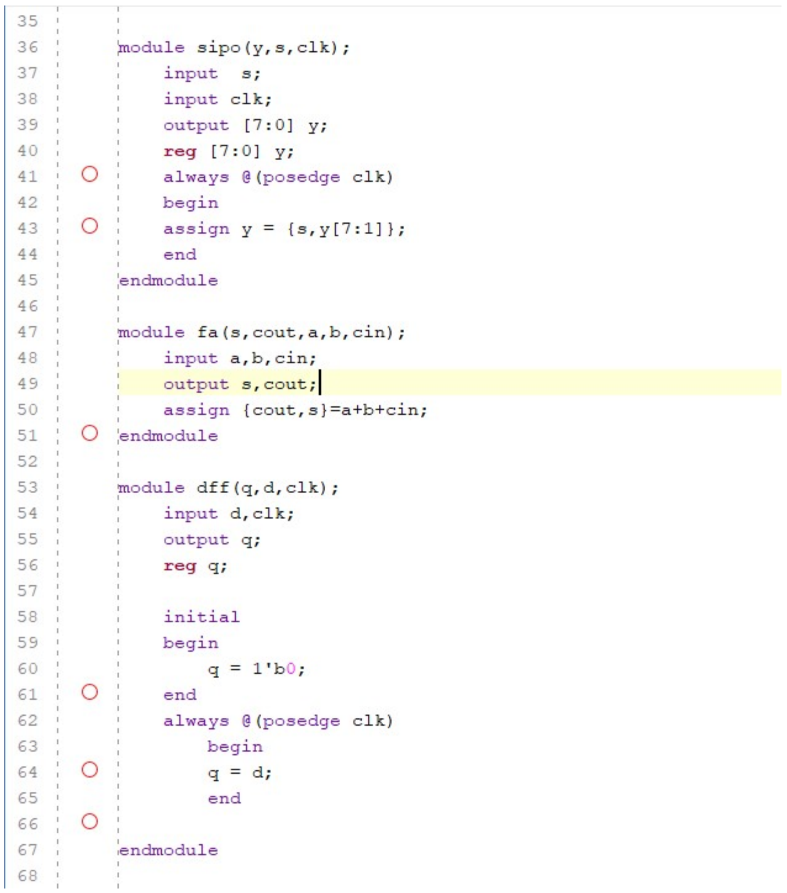
3 8 bit serial addition:

The verilog code is as follows:



```
`timescale 1ns / 1ps

module serial_adder(sum,cout,a,b,clk);

    input [7:0] a,b;
    input clk;
    wire [7:0] x,z;
    output [7:0] sum;
    output cout;
    wire s,cin;

    fa fa1(s,cout,x[0],z[0],cin);
    dff d1(cin,cout,clk);
    sipo a1(sum,s,clk);
    shift r1(x,a,clk);
    shift r2(z,b,clk);
endmodule

module shift(y,d,clk);
    input [7:0] d;
    input clk;
    output [7:0] y;
    reg [7:0] y;

    initial
        begin
        assign y=d;
        end

    always @(posedge clk)
        begin
        assign y = y>>1;
        end
endmodule
```

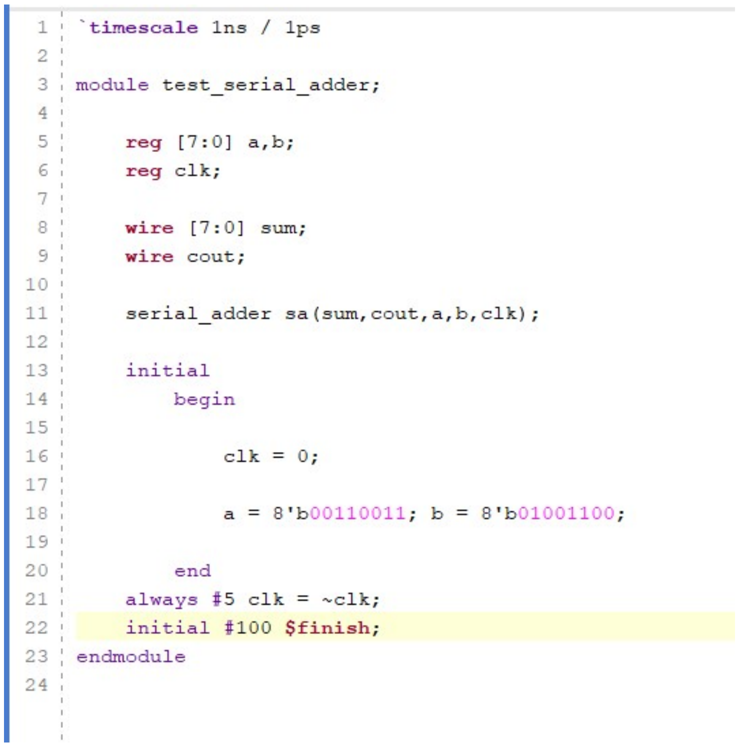
Figure 9: Verilog code for serial adder(1)



```
module sipo(y,s,clk);
    input  s;
    input clk;
    output [7:0] y;
    reg [7:0] y;
    always @(posedge clk)
    begin
    assign y = {s,y[7:1]};
    end
endmodule

module fa(s,cout,a,b,cin);
    input a,b,cin;
    output s,cout;
    assign {cout,s}=a+b+cin;
endmodule

module dff(q,d,clk);
    input d,clk;
    output q;
    reg q;

    initial
    begin
        q = 1'b0;
    end
    always @(posedge clk)
        begin
        q = d;
        end

endmodule
```

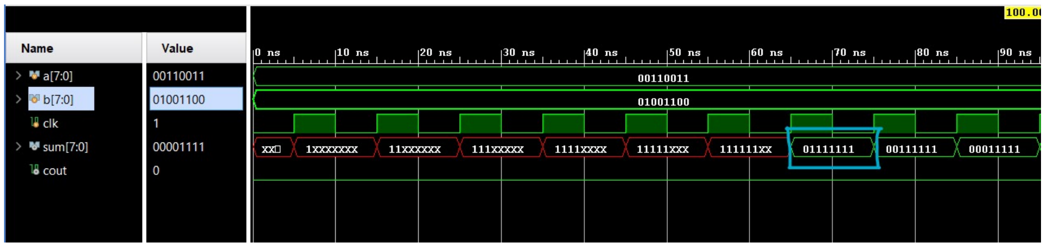
Figure 10: Verilog code for serial adder(2)

Since register a has parallel input, we can directly assign the value of sum to a to store in register a.



```
`timescale 1ns / 1ps

module test_serial_adder;

    reg [7:0] a,b;
    reg clk;

    wire [7:0] sum;
    wire cout;

    serial_adder sa(sum,cout,a,b,clk);

    initial
        begin

            clk = 0;

            a = 8'b00110011; b = 8'b01001100;

        end
    always #5 clk = ~clk;
    initial #100 $finish;
endmodule

```

Figure 11: Test bench for serial adder



Name	Value
a[7:0]	00110011
b[7:0]	01001100
clk	1
sum[7:0]	00001111
cout	0

Figure 12: Simulation wave forms

We observe that the answer is obtained after 8 clock cycles are completed.(The first input is loaded before the first clock in the simulation)