DIGITAL DESIGN LAB 6

DIVYAM PATEL

B20EE082

1. Write a program to implement a D f/f with synchronous and asynchronous reset.

D flip-flop with synchronous reset means the output can reset to zero with the reset input but only with the clock, which makes the reset input dependent on the clock pulse; without clock pulse reset will not be able to set the output Q to zero, which will give us a synchronous output always.

In asynchronous reset the Flip Flop does not wait for the clock and sets the output right at the edge of the reset. In Synchronous Reset, the Flip Flop waits for the next edge of the clock (rising or falling as designed), before applying the Reset of Data.

SYNCHRONOUS RESET

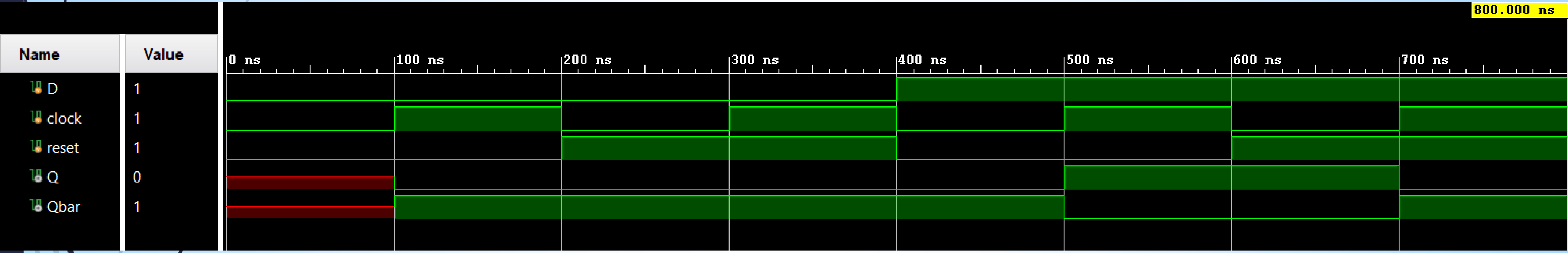
CODE

```
module Synchronous_D(Q,Qbar,D,clock,reset);
    input D,clock,reset;
    output reg Q;
    output Qbar;
    assign Qbar = ~Q;
    always @(posedge clock)
    begin
    if(reset)
        Q<=0;
    else
        Q<=D;
    end
endmodule
```

TESTBENCH

```
module test_synchronous();
reg D,clock,reset;
wire Q,Qbar;
Synchronous_D SD(Q,Qbar,D,clock,reset);
initial
begin
clock = 0;
end
always
#100 clock=~clock;
initial #800 $finish;
initial
begin
clock = 1'b0; D = 1'b0; reset = 1'b0;
#100 D = 1'b0; reset = 1'b0;
#100 D = 1'b0; reset = 1'b1;
#100 D = 1'b0; reset = 1'b1;
#100 D = 1'b1; reset = 1'b0;
#100 D = 1'b1; reset = 1'b0;
#100 D = 1'b1; reset = 1'b1;
#100 D = 1'b1; reset = 1'b1;
end
initial #800 $finish;
endmodule
```

SIMULATION



Name	Value
D	1
clock	1
reset	1
Q	0
Qbar	1

ASYNCHRONOUS RESET

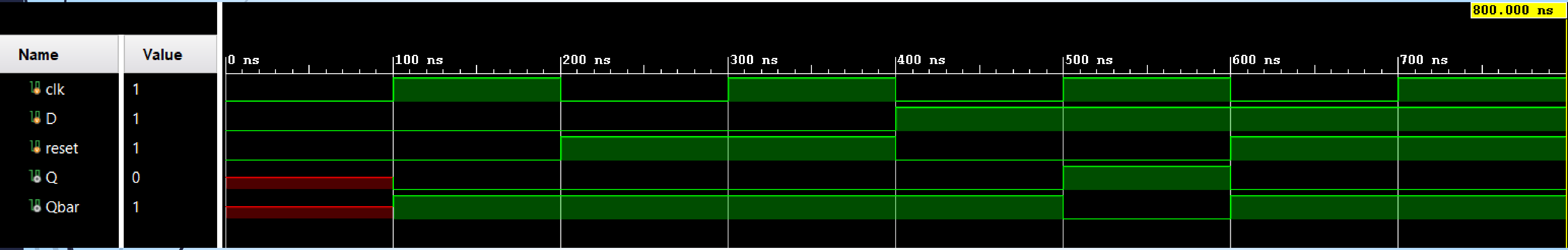
CODE

```
module Asynchronous_D(Q,Qbar,clk,D,reset);
    input D,clk,reset;
    output reg Q;
    output Qbar;
    assign Qbar = ~Q;
    always @(posedge clk,posedge reset)
    begin
    if(reset)
        Q<=0;
    else
        Q<=D;
    end
endmodule
```

TESTBENCH

```
module test_asynchronous();
reg clk,D,reset;
wire Q,Qbar;
Asynchronous_D AD(Q,Qbar,clk,D,reset);
initial
begin
clk = 1'b0; D = 1'b0; reset = 1'b0;
#100 D = 1'b0; reset = 1'b0;
#100 D = 1'b0; reset = 1'b1;
#100 D = 1'b0; reset = 1'b1;
#100 D = 1'b1; reset = 1'b0;
#100 D = 1'b1; reset = 1'b0;
#100 D = 1'b1; reset = 1'b1;
#100 D = 1'b1; reset = 1'b1;
end
always
#100 clk = ~clk;
initial #800 $finish;
endmodule
```

SIMULATION



800.000 ns
Name
Value
clk
1
D
1
reset
1
Q
0
Qbar
1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns

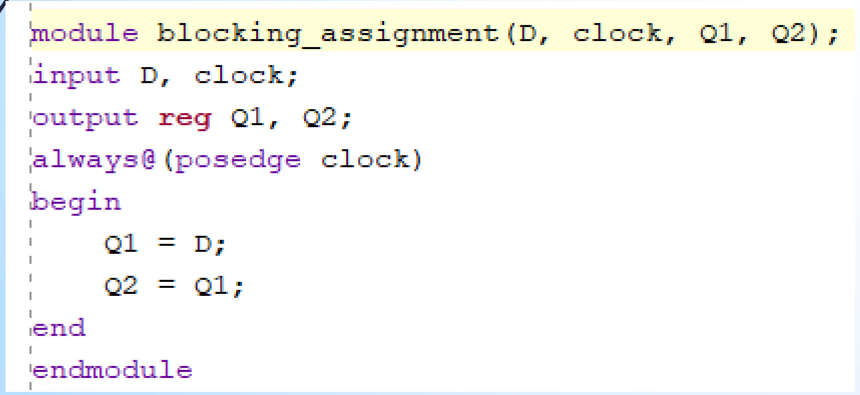
2. Test the operation of blocking and non-blocking assignments using two D flip flops.

BLOCKING ASSIGNMENTS :
Blocking assignment statements are assigned using **=** and are executed one after the other in a procedural block.
However, this will not prevent execution of statments that run in a parallel block.

NON-BLOCKING ASSIGNMENTS :
Non-blocking assignment allows assignments to be scheduled without blocking the execution of following statements and is specified by a **<=** symbol.

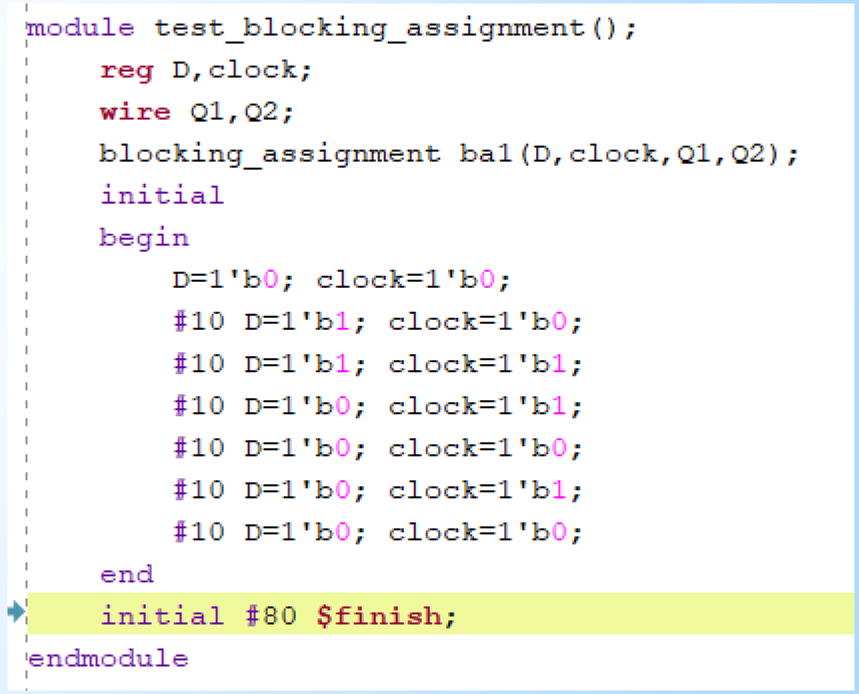
BLOCKING ASSIGNMENT

CODE



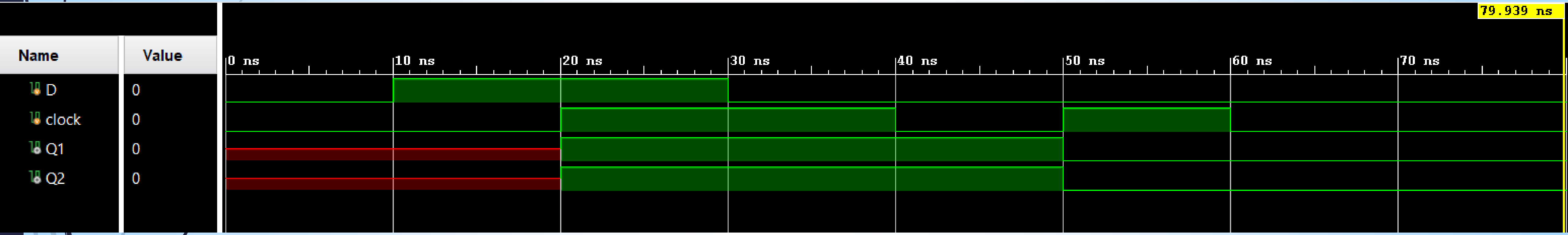
```
module blocking_assignment(D, clock, Q1, Q2);
input D, clock;
output reg Q1, Q2;
always@(posedge clock)
begin
    Q1 = D;
    Q2 = Q1;
end
endmodule
```

TESTBENCH



```
module test_blocking_assignment();
    reg D,clock;
    wire Q1,Q2;
    blocking_assignment ba1(D,clock,Q1,Q2);
    initial
    begin
        D=1'b0; clock=1'b0;
        #10 D=1'b1; clock=1'b0;
        #10 D=1'b1; clock=1'b1;
        #10 D=1'b0; clock=1'b1;
        #10 D=1'b0; clock=1'b0;
        #10 D=1'b0; clock=1'b1;
        #10 D=1'b0; clock=1'b0;
    end
    initial #80 $finish;
endmodule
```

SIMULATION


79.939 ns
Name
Value
D
0
clock
0
Q1
0
Q2
0
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns

NON-BLOCKING ASSIGNMENT

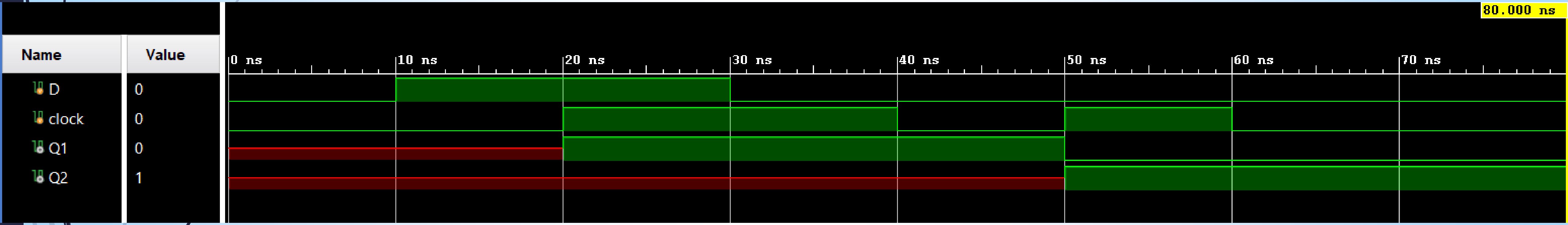
CODE

```
module unblocking_assignment(D, clock, Q1, Q2);
input D, clock;
output reg Q1, Q2;
always@(posedge clock)
begin
    Q1 <= D;
    Q2 <= Q1;
end
endmodule
```

TESTBENCH

```
module test_nonblocking_assignment();
    reg D,clock;
    wire Q1,Q2;
    unblocking_assignment ua1(D,clock,Q1,Q2);
    initial
    begin
        D=1'b0; clock=1'b0;
        #10 D=1'b1; clock=1'b0;
        #10 D=1'b1; clock=1'b1;
        #10 D=1'b0; clock=1'b1;
        #10 D=1'b0; clock=1'b0;
        #10 D=1'b0; clock=1'b1;
        #10 D=1'b0; clock=1'b0;
    end
    initial #80 $finish;
endmodule
```

SIMULATION



Name	Value
D	0
clock	0
Q1	0
Q2	1

3. Implement the 8 bit serial addition (A+B) operation using shift register and a Full Adder. Load the two registers from input using parallel loading. The result should be stored in register A. Find the number of clock cycles your design would take to implement the complete operation.

Number of clock cycles required : 9
[1 to load data in SR and 8 for addition]

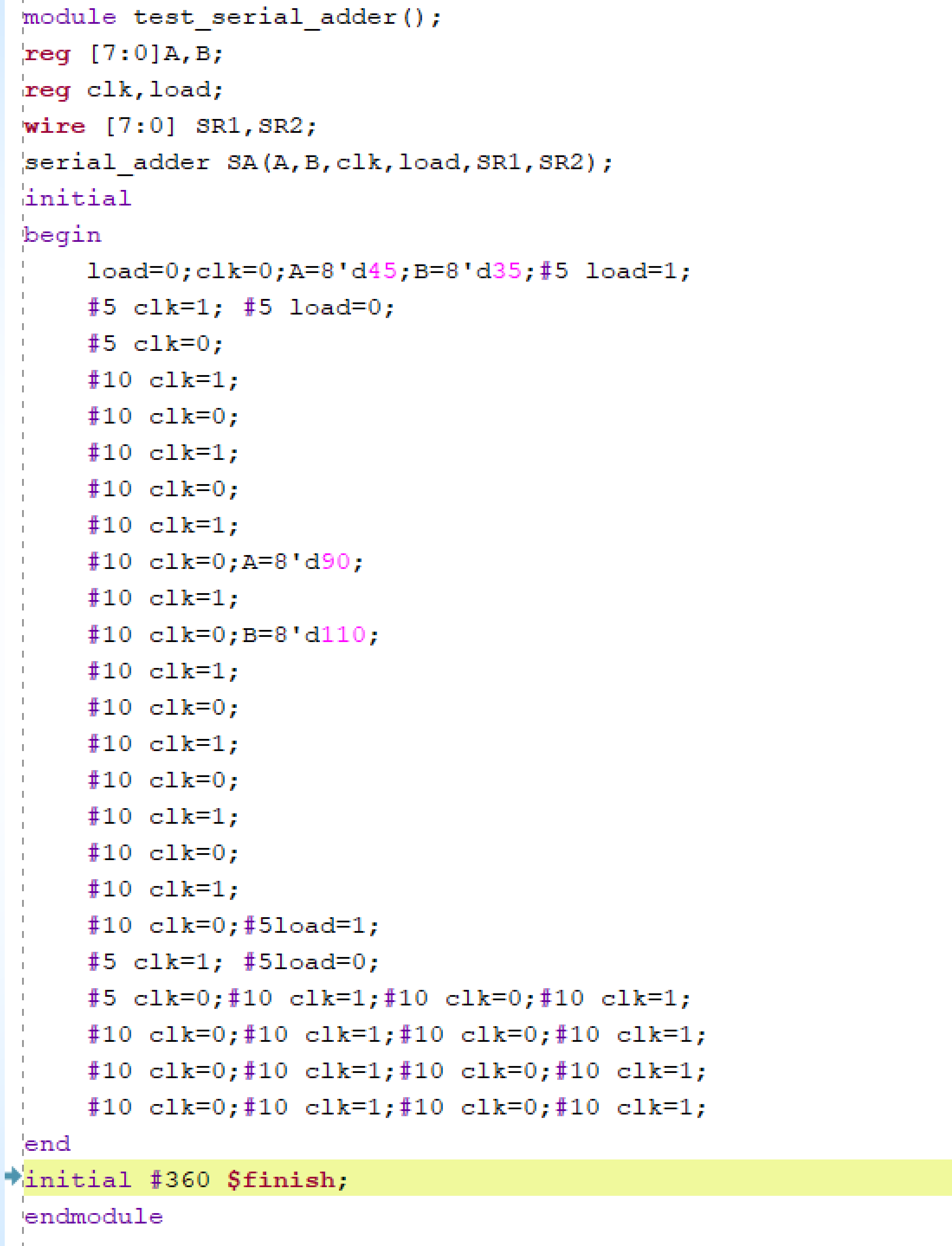
CODE

```
module serial_adder(A,B,clk,load,SR1,SR2);
input [7:0]A,B;
input clk,load;
output reg [7:0]SR1,SR2;
wire S,Carry;
reg C=0;
wire [7:0]D,E;

assign D[7]=load?A[7]:SR1[7];
assign D[6]=load?A[6]:SR1[6];
assign D[5]=load?A[5]:SR1[5];
assign D[4]=load?A[4]:SR1[4];
assign D[3]=load?A[3]:SR1[3];
assign D[2]=load?A[2]:SR1[2];
assign D[1]=load?A[1]:SR1[1];
assign D[0]=load?A[0]:SR1[0];

assign E[7]=load?B[7]:SR2[7];
assign E[6]=load?B[6]:SR2[6];
assign E[5]=load?B[5]:SR2[5];
assign E[4]=load?B[4]:SR2[4];
assign E[3]=load?B[3]:SR2[3];
assign E[2]=load?B[2]:SR2[2];
assign E[1]=load?B[1]:SR2[1];
assign E[0]=load?B[0]:SR2[0];
```

```
always @(posedge clk)
begin
if(load)
    SR1[0]=D[0];
    SR1[1]=D[1];
    SR1[2]=D[2];
    SR1[3]=D[3];
    SR1[4]=D[4];
    SR1[5]=D[5];
    SR1[6]=D[6];
    SR1[7]=D[7];

    SR2[0]=E[0];
    SR2[1]=E[1];
    SR2[2]=E[2];
    SR2[3]=E[3];
    SR2[4]=E[4];
    SR2[5]=E[5];
    SR2[6]=E[6];
    SR2[7]=E[7];
end
assign S=SR1[0]^SR2[0]^C;
assign
Carry=(SR1[0]&SR2[0])|(SR2[0]&C)|(C&SR1[0]);
```

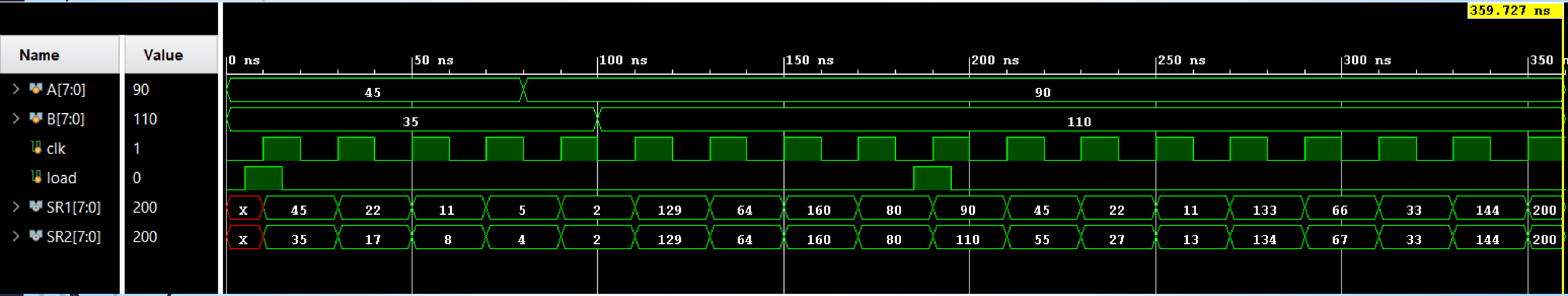
```
always @(posedge clk)
begin
   if(load==1'b0)
   begin
      if(Carry==1'bx)
         C=0;
      else
         C=Carry;
      SR1[0]=SR1[1];
      SR1[1]=SR1[2];
      SR1[2]=SR1[3];
      SR1[3]=SR1[4];
      SR1[4]=SR1[5];
      SR1[5]=SR1[6];
      SR1[6]=SR1[7];
      SR1[7]=S;
      SR2[0]=SR2[1];
      SR2[1]=SR2[2];
      SR2[2]=SR2[3];
      SR2[3]=SR2[4];
      SR2[4]=SR2[5];
      SR2[5]=SR2[6];
      SR2[6]=SR2[7];
      SR2[7]=S;
   end
   else
      C=0;
end
endmodule
```

TESTBENCH



```
module test_serial_adder();
reg [7:0]A,B;
reg clk,load;
wire [7:0] SR1,SR2;
serial_adder SA(A,B,clk,load,SR1,SR2);
initial
begin
    load=0;clk=0;A=8'd45;B=8'd35;#5 load=1;
    #5 clk=1; #5 load=0;
    #5 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;A=8'd90;
    #10 clk=1;
    #10 clk=0;B=8'd110;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;#5load=1;
    #5 clk=1; #5load=0;
    #5 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
    #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
    #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
    #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
end
initial #360 $finish;
endmodule
```

SIMULATION


359.727 ns
Name
Value
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350
A[7:0] 90
B[7:0] 110
clk 1
load 0
SR1[7:0] 200
SR2[7:0] 200
45
90
35
110
x 45 22 11 5 2 129 64 160 80 90 45 22 11 133 66 33 144 200
x 35 17 8 4 2 129 64 160 80 110 55 27 13 134 67 33 144 200

THANK YOU