DIGITAL DESIGN LAB 5

DIVYAM PATEL

B20EE082

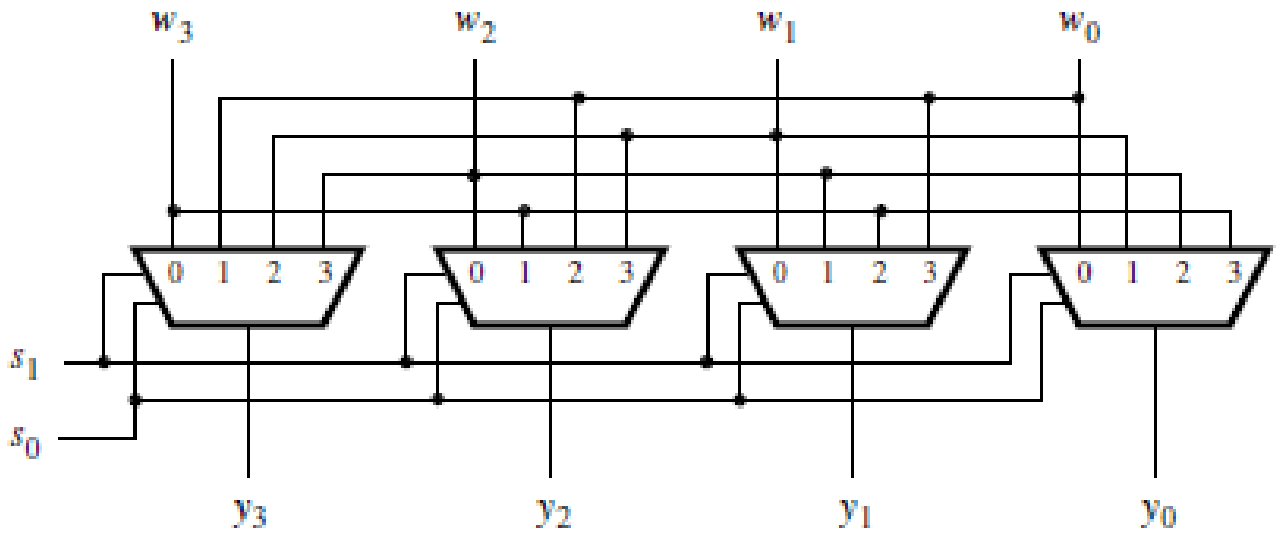
1. Write a program to implement a Barrel Shifter.

A barrel shifter is a specialized digital electronic circuit with the purpose of shifting an entire data word by a specified number of bits by only using combinational logic, with no sequential logic used.

TRUTH TABLE

s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

CIRCUIT DIAGRAM



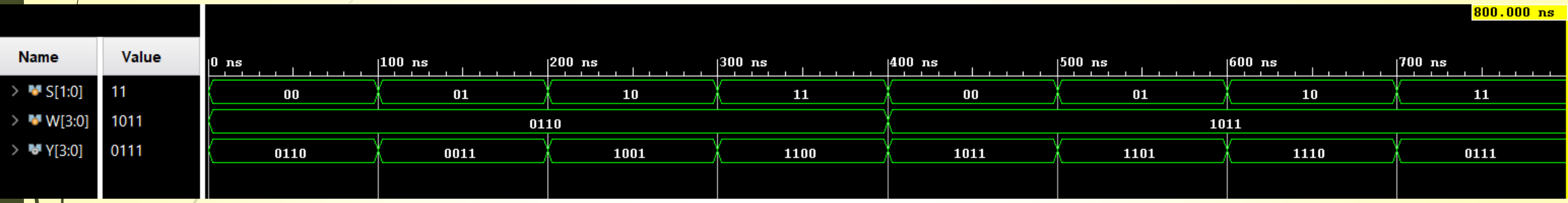
CODE

```
module barrel_shifter(S, W, Y);
    input [1:0]S;
    input [3:0]W;
    output [3:0]Y;
    wire [3:0] T;
    assign {T, Y} = {W,W} >> S;
endmodule
```

TESTBENCH

```
module test_barrel_shifter();
reg [1:0]S;
reg [3:0]W;
wire [3:0]Y;

barrel_shifter BS(S, W, Y);
initial
    begin;
        S = 2'b00; W = 4'b0110;
        #100 S = 2'b01; W = 4'b0110;
        #100 S = 2'b10; W = 4'b0110;
        #100 S = 2'b11; W = 4'b0110;
        #100 S = 2'b00; W = 4'b1011;
        #100 S = 2'b01; W = 4'b1011;
        #100 S = 2'b10; W = 4'b1011;
        #100 S = 2'b11; W = 4'b1011;
    end
initial #800 $finish;
endmodule
```

SIMULATION



s1	s0	y3	y2	y1	y0
0	**0**	0	1	1	0
0	**1**	0	0	1	1
1	**0**	1	0	0	1
1	**1**	1	1	0	0

s1	s0	y3	y2	y1	y0
0	**0**	1	0	1	1
0	**1**	1	1	0	1
1	**0**	1	1	1	0
1	**1**	0	1	1	1

2. Write a Program to implement a 32 bit ALU.

- ALU i.e. Arithmetic Logic Unit is the circuit which can perform both arithmetic as well as logical operations.
- It is basically the calculator of the computer.
- Based on the input we give, it performs the corresponding operation.
- The control unit will supply the data needed by the ALU from memory or from any input devices.

Operation	Input	32 bit Output
Clear	000	0
Addition	001	A + B
Subtraction	010	A - B
A * 2	011	Left shift
A / 2	100	Right Shift
A AND B	101	A & B
A OR B	110	A | B
A XOR B	111	A^B

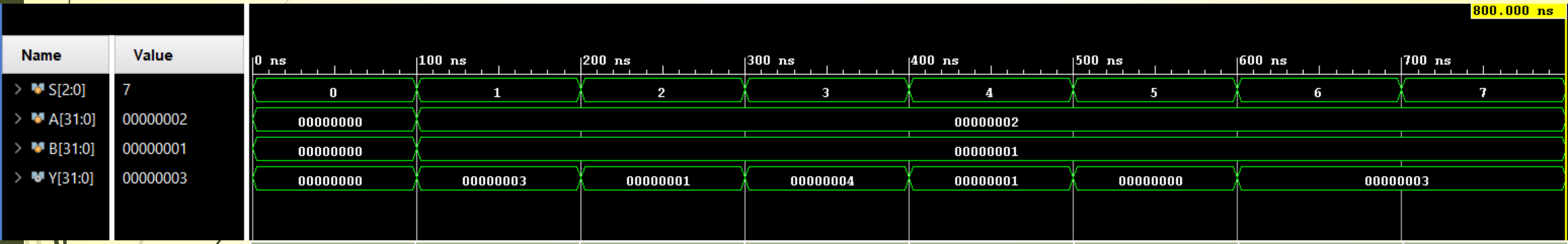
CODE

```
module alu_32_bit(S, A, B, Y);
input[2:0] S;
input[31:0] A;
input[31:0] B;
output reg[31:0] Y;
always@(S)
    case(S)
        3'b000: Y = 0; //Clear
        3'b001: Y = A+B; //Addition
        3'b010: Y = A-B; //Subtraction
        3'b011: Y = A<<1; //Left Shift
        3'b100: Y = A>>1; //Right Shift
        3'b101: Y = A&B; //AND
        3'b110: Y = A|B; //OR
        3'b111: Y = A^B; //XOR
    endcase
endmodule
```

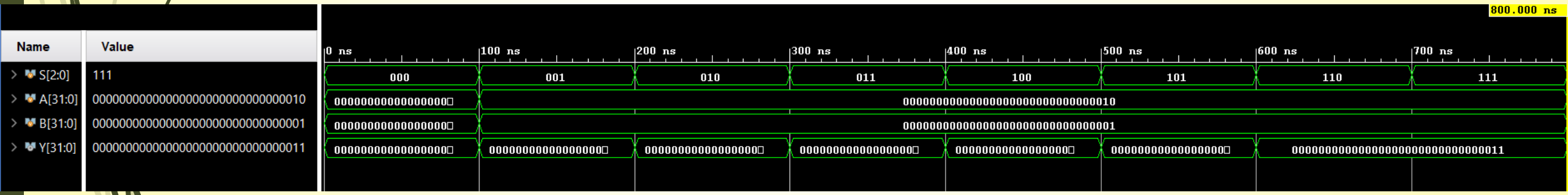
TESTBENCH

```
module test_alu_32_bit;
reg [2:0] S;
reg [31:0] A;
reg [31:0] B;
wire [31:0] Y;

alu_32_bit ALU(S, A, B, Y);
initial
begin
    S = 3'b000; A = 2'b00; B = 2'b00;
    #100 S = 3'b001; A = 2'b10; B = 2'b01;
    #100 S = 3'b010; A = 2'b10; B = 2'b01;
    #100 S = 3'b011; A = 2'b10; B = 2'b01;
    #100 S = 3'b100; A = 2'b10; B = 2'b01;
    #100 S = 3'b101; A = 2'b10; B = 2'b01;
    #100 S = 3'b110; A = 2'b10; B = 2'b01;
    #100 S = 3'b111; A = 2'b10; B = 2'b01;
end
initial #800 $finish;
endmodule
```

SIMULATION



Clear	Addition	Subtraction	Left Shift	Right Shift	AND	OR	XOR



3. Write a program to implement a 4 line to 2 line priority encoder using
a. Casex statements b. For loop

4 to 2 Priority Encoder
This is also referred to as 4- bit priority, which consists of 4 inputs and 2 output lines. Since an encoder contains 2^n input lines and n output lines.

- D3, D2, D1, D0 are the inputs
- A and B are the outputs
- V is the valid bit indicator
- D3 input is the highest priority input
- D0 is the lowest priority input.

The **truth table of a 4 to 2 priority encoder** is shown below.

D3	D2	D	D0	A	B	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

4 to 2 Priority Encoder Using Casex Statements

CODE

```
module priority_encoder(A, D, V);
input [3:0]D;
output reg [1:0]A;
output reg V;
always@(D)
    begin
    V = 1;
        casex(D)
            4'b0001:A = 2'b00;
            4'b001x:A = 2'b01;
            4'b01xx:A = 2'b10;
            4'b1xxx:A = 2'b11;
            default:begin
                        V = 0;
                        A = 2'bxx;
                    end
        endcase
    end
endmodule
```

TESTBENCH

```
module test_priority_encoder_casex();
reg [3:0] D;
wire [1:0] A;
wire V;

priority_encoder PEC(A, D, V);
initial
begin
    D = 0;
    #100 D = 4'b0000;
    #100 D = 4'b0001;
    #100 D = 4'b0010;
    #100 D = 4'b0100;
    #100 D = 4'b1000;
end
initial #600 $finish;
endmodule
```

4 to 2 Priority Encoder Using For Loop

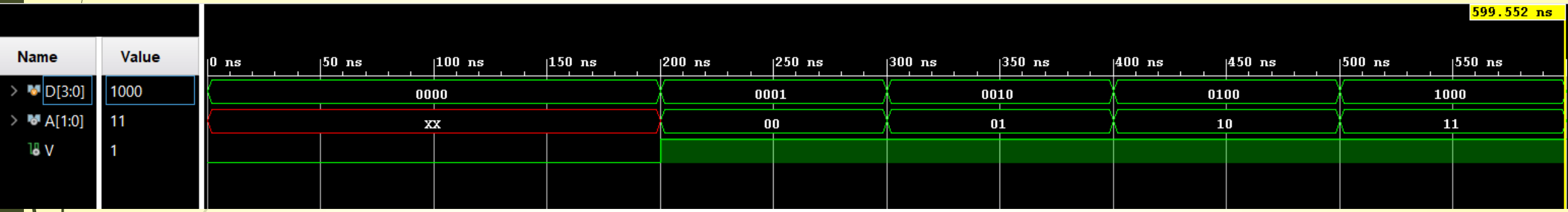
CODE

```
module priority_encoder_forloop(A, D, V);
input [3:0]D;
output reg [1:0] A;
output reg V;
integer k;
always @(D)
begin
    A = 2'bxx;
    V = 0;
    for (k = 0; k < 4; k = k+1)
        if (D[k])
        begin
            A = k;
            V = 1;
        end
end
endmodule
```

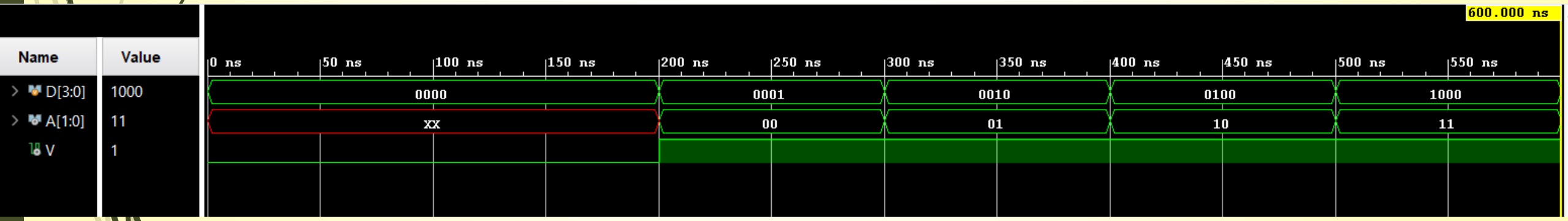
TESTBENCH

```
module test_priority_encoder_forloop();
reg [3:0] D;
wire [1:0] A;
wire V;

priority_encoder_forloop PEF(A, D, V);
initial
begin
    D = 0;
    #100 D = 4'b0000;
    #100 D = 4'b0001;
    #100 D = 4'b0010;
    #100 D = 4'b0100;
    #100 D = 4'b1000;
end
initial #600 $finish;
endmodule
```

USING CASEX STATEMENTS



USING FOR LOOP



4. Write a behavioural code for implementing

a. a BCD Adder/Subtractor Unit.

b. Multiply by 5 circuit.

BCD ADDER

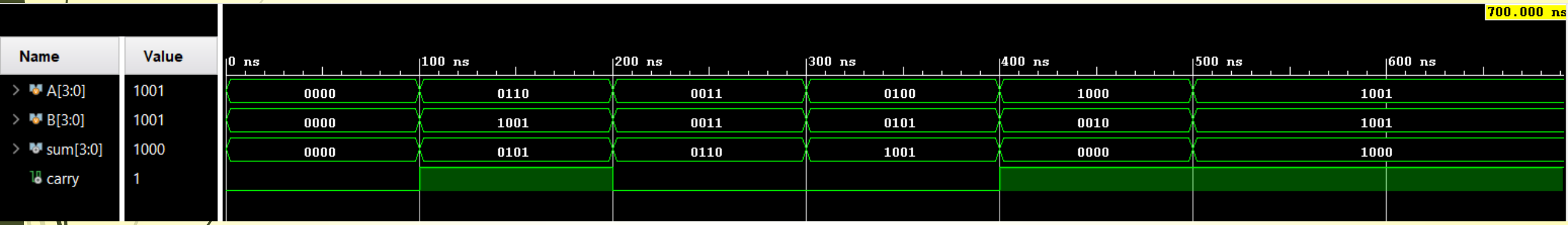
CODE

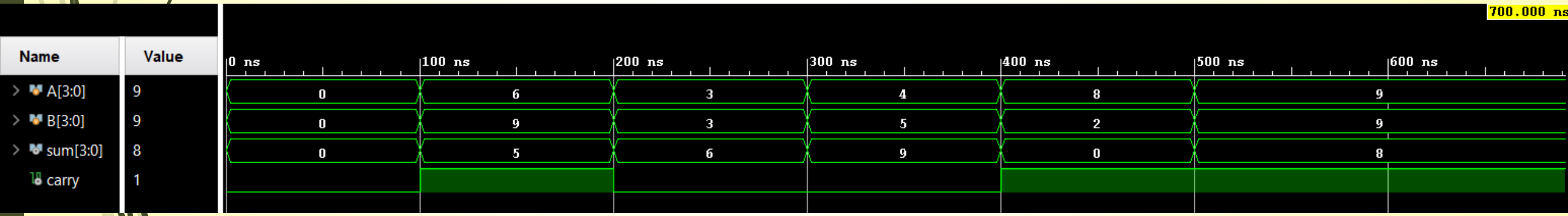
```
module bcd_adder(A,B,sum,carry);
    input [3:0] A,B;
    output [3:0] sum;
    output reg carry;
    reg [4:0] sum_temp;
    reg [3:0] sum;

    always @(A,B)
    begin
        sum_temp = A+B; //add all the inputs
        if(sum_temp > 9)
            begin
                sum_temp = sum_temp+6; //add 6, if result is more than 9.
                carry = 1;  //set the carry output
                sum = sum_temp[3:0];
            end
        else
            begin
                carry = 0;
                sum = sum_temp[3:0];
            end
    end
endmodule
```

TESTBENCH

```
module test_bcd_adder();
    reg [3:0] A;
    reg [3:0] B;
    wire [3:0] sum;
    wire carry;

    bcd_adder BA1(A,B,sum,carry);

    initial begin
        A = 0;  B = 0;  #100;
        A = 6;  B = 9;  #100;
        A = 3;  B = 3;  #100;
        A = 4;  B = 5;  #100;
        A = 8;  B = 2;  #100;
        A = 9;  B = 9;  #100;
    end
initial #700 $finish;
endmodule
```

SIMULATION


700.000 ns
Name
Value
A[3:0]
1001
B[3:0]
1001
sum[3:0]
1000
carry
1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
0000
0110
0011
0100
1000
1001
0000
1001
0011
0101
0010
1001
0000
0101
0110
1001
0000
1000


700.000 ns
Name
Value
A[3:0]
9
B[3:0]
9
sum[3:0]
8
carry
1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
0
6
3
4
8
9
0
9
3
5
2
9
0
5
6
9
0
8

BCD SUBTRACTOR

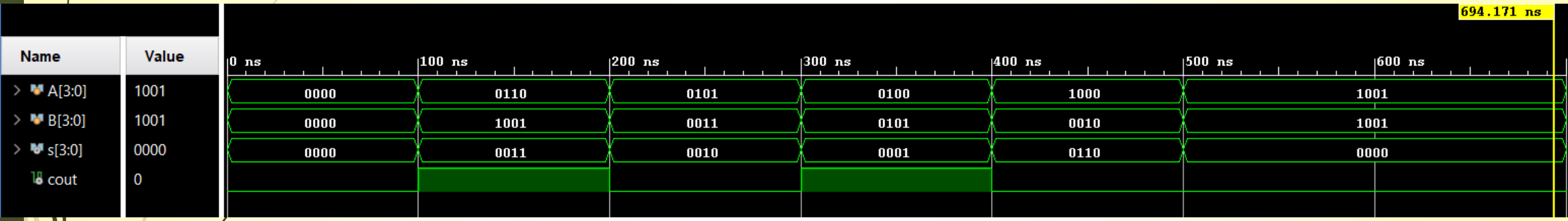
CODE

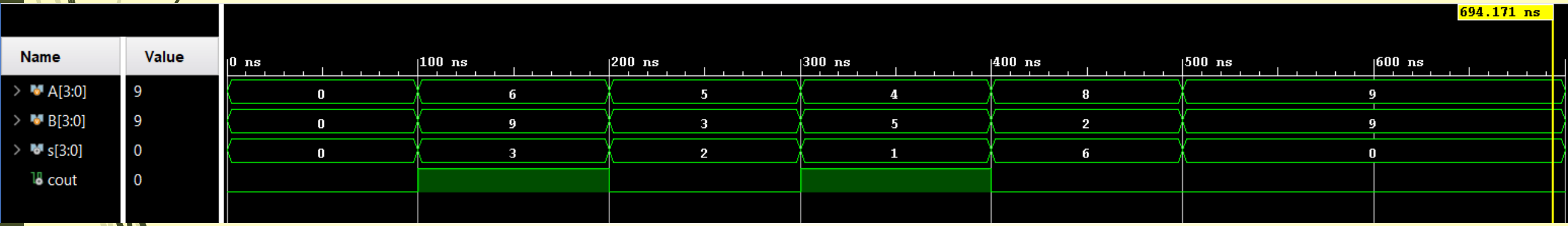
```
module bcd_subtractor(A,B,s,cout);
    input [3:0] A,B;
    output reg[3:0] s;
    output reg cout;

    always @(A,B)
    begin
        if (A>B)
            begin
                s = A - B;
                cout = 0;
            end
        else if (A<B)
            begin
                s = B - A;
                cout = 1;
            end
        else
            begin
                s = 0;
                cout = 0;
            end
    end
endmodule
```

TESTBENCH

```
module test_add_sub();
reg [3:0] A, B;
wire [3:0] s;
wire cout;


bcd_subtractor BS(A,B,s,cout);
initial
    begin
        A = 0;  B = 0;  #100;
        A = 6;  B = 9;  #100;
        A = 5;  B = 3;  #100;
        A = 4;  B = 5;  #100;
        A = 8;  B = 2;  #100;
        A = 9;  B = 9;  #100;
    end
initial #700 $finish;
endmodule
```

SIMULATION


Name
Value
A[3:0]
1001
B[3:0]
1001
s[3:0]
0000
cout
0
694.171 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
0000
0110
0101
0100
1000
1001
0000
1001
0011
0101
0010
1001
0000
0011
0010
0001
0110
0000


Name
Value
A[3:0]
9
B[3:0]
9
s[3:0]
0
cout
0
694.171 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
0
6
5
4
8
9
0
9
3
5
2
9
0
3
2
1
6
0

MULTIPLY BY 5

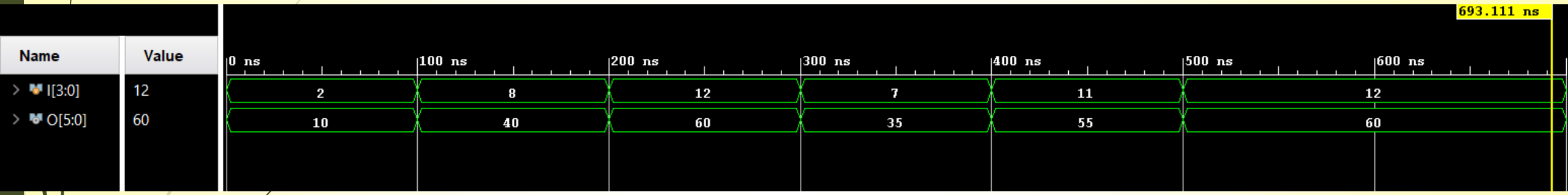
CODE

```
module multiply_5(I, O);
    input [3:0] I;
    output [5:0] O;
    assign O = (I<<2) + I;
endmodule
```

TESTBENCH

```
module test_multiply_5();
reg [3:0]I;
wire [5:0] O;
multiply_5 M1(I, O);
initial
    begin
        I = 4'b0010;  #100;
        I = 4'b1000;  #100;
        I = 4'b1100;  #100;
        I = 4'b0111;  #100;
        I = 4'b1011;  #100;
        I = 4'b1100;  #100;
    end
initial #700 $finish;
endmodule
```

SIMULATION



Output is multiplied by 5

THANK YOU