pscpr

VERILOG PROJECT

N-bit Multiplier with Binary to BCD Converter

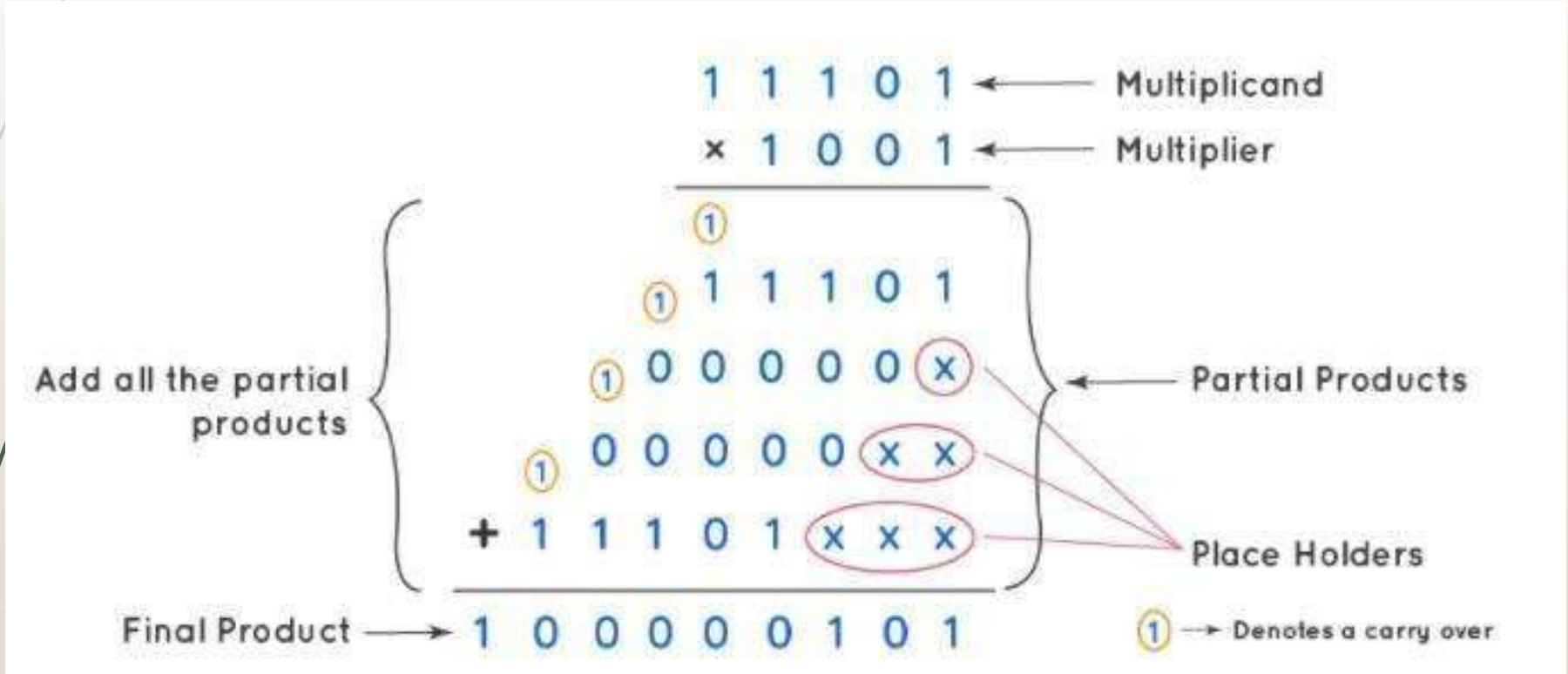
PER

INTRODUCTION

- ✓ This project is designed as an n-bit multiplier of two numbers. The multiplication is performed using the **shift and add method** of multiplying two numbers.
- ✓ Once the multiplication is completed, the output result is also converted to its **binary-coded decimal (BCD) representation**.
- ✓ This is done using the **double dabble method** of converting binary numbers to binary-coded decimal(BCD) .
- ✓ In the output, both the binary and the binary-coded decimal (BCD) representation is shown.

SHIFT AND ADD ALGORITHM

Consider two numbers **29 and 9**
Their binary representation is **11101 and 1001**



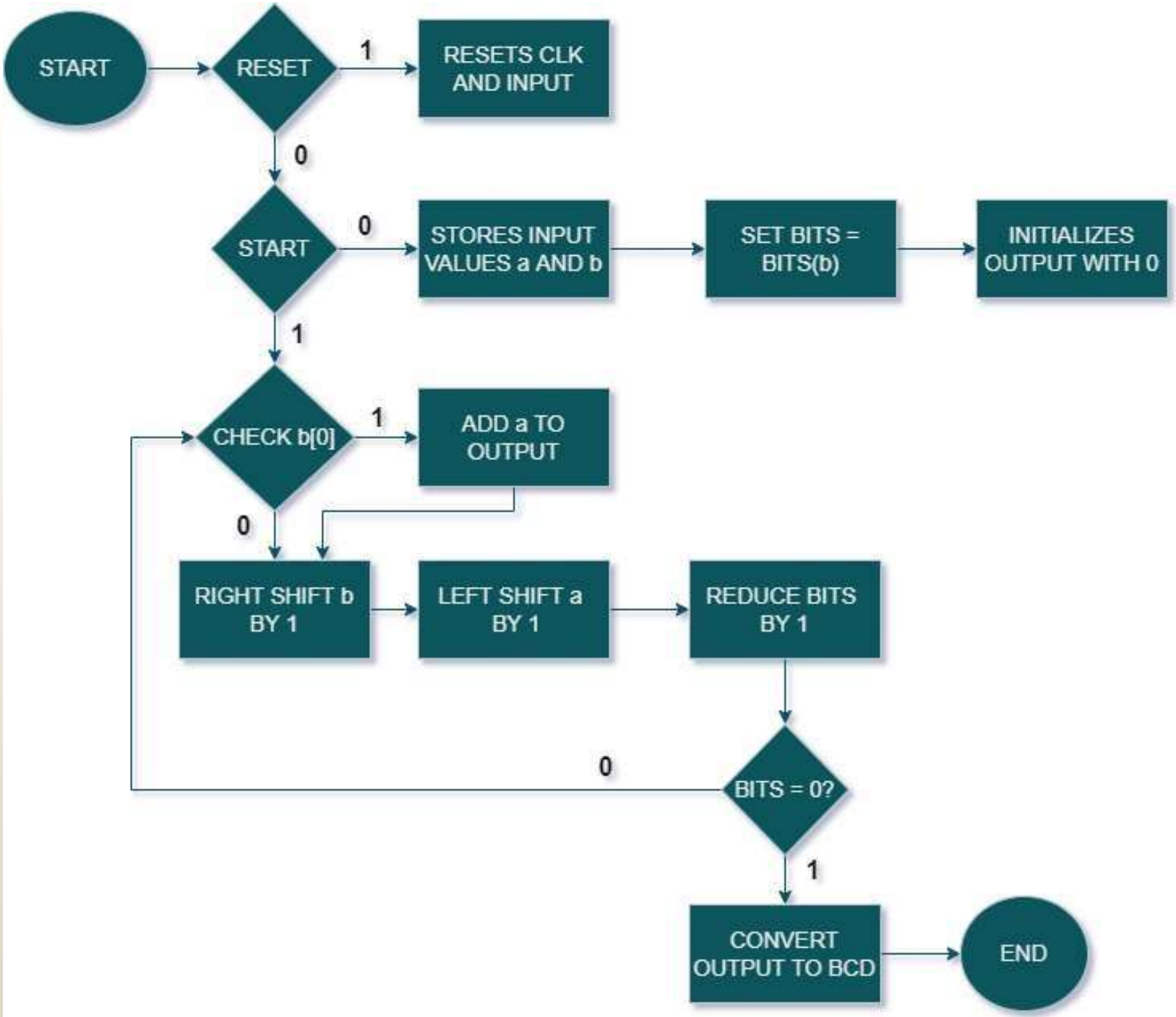
The result is **261**
Their binary representation is **100000101**

DOUBLE DABBLE ALGORITHM

Consider the number **151**.
Its binary representation is **10010111**

BCD2	BCD1	BCD0	Binary Input	
0000	0000	0000	10010111	
0000	0000	0001	0010111.	← 1
0000	0000	0010	010111..	← 2
0000	0000	0100	10111...	← 3
0000	0000	1001	0111....	← 4
0000	0000	1100	0111....	ADD
0000	0001	1000	111.....	← 5
0000	0001	1011	111.....	ADD
0000	0011	0111	11.....	← 6
0000	0011	1010	11.....	ADD
0000	0111	0101	1.....	← 7
0000	1010	1000	1.....	ADD ADD
0001	0101	0001	.....	← 8

FLOW CHART


START
RESET
1
RESETS CLK AND INPUT
0
START
0
STORES INPUT VALUES a AND b
SET BITS = BITS(b)
INITIALIZES OUTPUT WITH 0
1
CHECK b[0]
1
ADD a TO OUTPUT
0
RIGHT SHIFT b BY 1
LEFT SHIFT a BY 1
REDUCE BITS BY 1
0
BITS = 0?
1
CONVERT OUTPUT TO BCD
END

VERILOG CODE

```
module multiplier(
      out, a_in, b_in, clk, start, reset, finish, bcd
   );

parameter N = 8;
```

```
// Outputs
output[(((N*2)/3)+1)*4-1:0] bcd;
output[(N*2)-1:0] out;
output finish;
```

Declaring Outputs

```
// Inputs
input start;
input clk;
input reset;
input [N-1:0] a_in;
input [N-1:0] b_in;
```

Declaring Inputs

```
// Reference registers
reg[(((N*2)/3)+1)*4-1:0] bcd_reg = 0;
reg[(N*2)-1:0] out_reg;
reg finish_reg = 0;
reg [(N*2)-1:0] a_in_reg;
reg [(N*2)-1:0] b_in_reg;
reg [8:0] bits;
```

Creating Reference Registers

VERILOG CODE ...continued

```
// Continuous assignment
assign bcd = bcd_reg;
assign out = out_reg;
assign finish = finish_reg;

integer i;
```

Continuous Assignments

```
// Reset clk and inputs
always @(negedge reset)
begin
    out_reg = 0;
    a_in_reg = 0;
    b_in_reg = 0;
end
```

Reset clk and inputs

```
always @(posedge clk)
begin
    if(!reset)
    begin
        case(start)
            1'b0: begin
                a_in_reg = a_in;
                b_in_reg = b_in;
                bits = N;
                finish_reg = 0;
                out_reg = 0;
                bcd_reg = 0;
                $display("Values loaded into the input register!");
            end
```

Load Input Values

VERILOG CODE ...continued

```
            1'b1: begin
               if(b_in_reg[0]==1)
               begin
                  out_reg = out_reg + a_in_reg;
               end
               bits = bits - 1;
               a_in_reg = a_in_reg<<1;
               b_in_reg = b_in_reg>>1;
            end
         endcase
         if(bits==0)
         begin
            $display("Multiplication completed!");
            finish_reg = 1'b1;
         end
            //----------------------------------------------------------------
            // conversion of binary to bcd
            for(i=0;i<(N*2);i=i+1)
            begin
               if (3 <= (((N*2)/3)+1)*4-1 && bcd_reg[3:0] >= 5) bcd_reg[3:0] = bcd_reg[3:0] + 3;
               if (7 <= (((N*2)/3)+1)*4-1 && bcd_reg[7:4] >= 5) bcd_reg[7:4] = bcd_reg[7:4] + 3;
               if (11 <= (((N*2)/3)+1)*4-1 && bcd_reg[11:8] >= 5) bcd_reg[11:8] = bcd_reg[11:8] + 3;
               if (15 <= (((N*2)/3)+1)*4-1 && bcd_reg[15:12] >= 5) bcd_reg[15:12] = bcd_reg[15:12] + 3;
               bcd_reg = {bcd_reg[(((N*2)/3)+1)*4-2:0],out_reg[(N*2)-1-i]};
            end
            //----------------------------------------------------------------
            $display("Conversion of binary to BCD completed!");
         end
      end
   end
end
endmodule
```

Shift and Add Multiplication

Multiplication Complete

Double Dabble Algorithm

TEST BENCH

```
module multiplier_tb;
   parameter n_bits = 5;

   // Inputs
   reg [n_bits-1:0] a_in;
   reg [n_bits-1:0] b_in;
   reg clk;
   reg start;
   reg reset;

   // Outputs
   wire [(n_bits*2)-1:0] out;
   wire finish;
   wire [(((n_bits*2)/3)+1)*4-1:0] bcd;
```

Declaring Inputs and Outputs

```
   // Instantiate the Unit Under Test (UUT)
   multiplier uut (
      .out(out),
      .a_in(a_in),
      .b_in(b_in),
      .clk(clk),
      .start(start),
      .reset(reset),
      .finish(finish),
      .bcd(bcd)
   );
   defparam uut.N = n_bits;
```

Initializing the multiplier

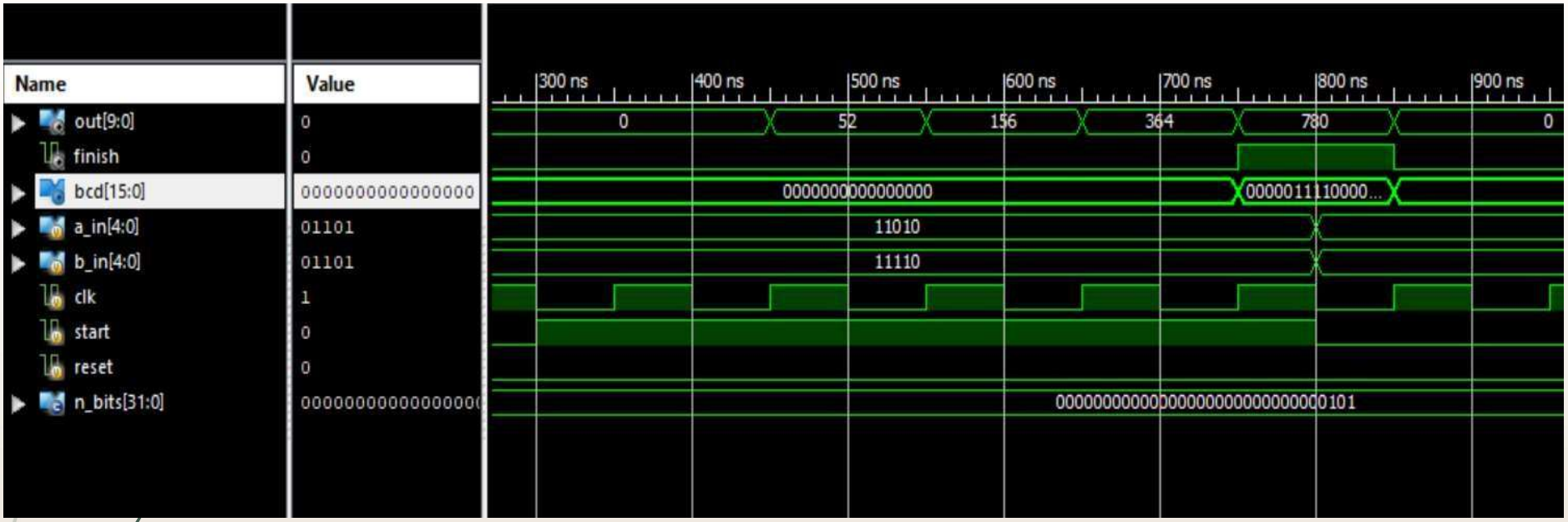
```
   initial
   begin
   forever
      #50 clk= ~clk;
   end
```

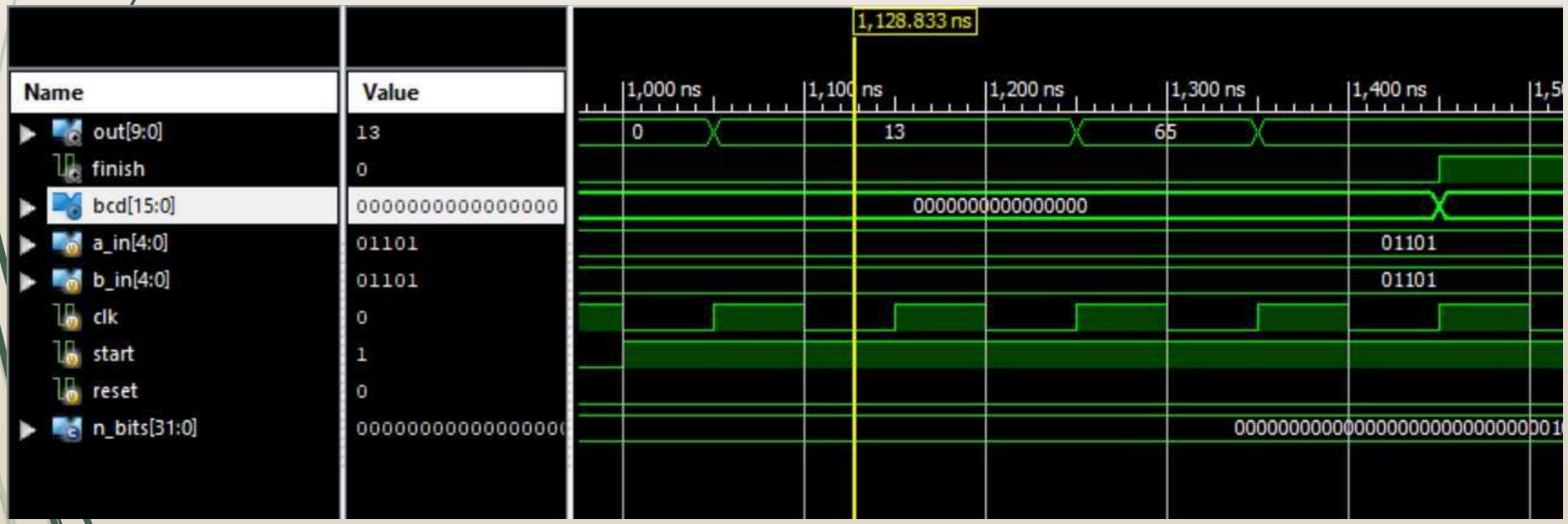
Creating clock

```
   initial begin
      // Initialize Inputs
      a_in = 0;
      b_in = 0;
      clk = 0;
      start = 0;
      reset = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      a_in = 'd26;
      b_in = 'd30;
      start = 0;
      #200
      start = 1;
      #1000
      a_in = 'd13;
      b_in = 'd13;
      start = 0;
      #200
      start = 1;
   end
```

Test Inputs

SIMULATION OUTPUT





OBSERVATIONS

- The ***out*** represent the output of the multiplication of two numbers.
- A value of 1 in ***finish*** signal indicates that the multiplication is complete while a value of 0 indicates that the multiplication is ongoing.
- The ***clk*** and ***reset*** are input signals used to set the clock and reset the circuit to initial state respectively.
- The ***bcd*** represents the binary-coded decimal representation of the output.
- The ***a_in*** represents the input multiplicand.
- The ***b_in*** represents the input multiplier.
- A value of 0 in ***start*** loads the initial value into the registers, while a value of 1 performs the multiplication.


THANK YOU