College of Engineering, Pune
Dept. of Electronics and Telecommunication Engineering

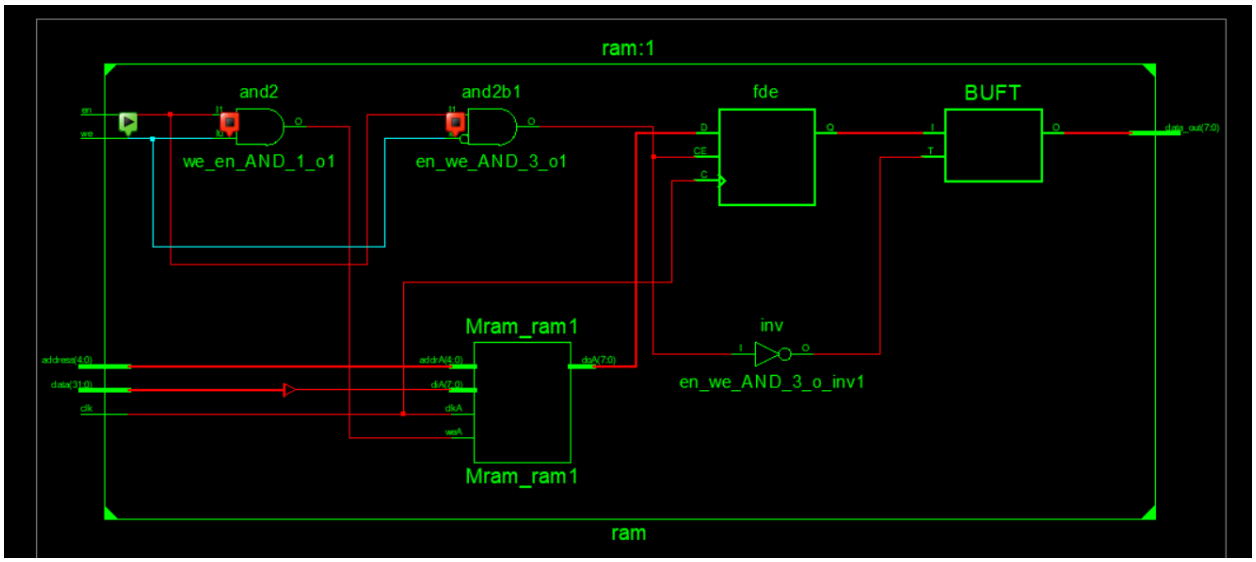
Course : RTL Simulation and Synthesis Lab (FY VLSI & ES)
Year : 2020-21

RTL Lab 8 Report :

Mis : 122035014
Name : Swapnil Hanmant Phalke

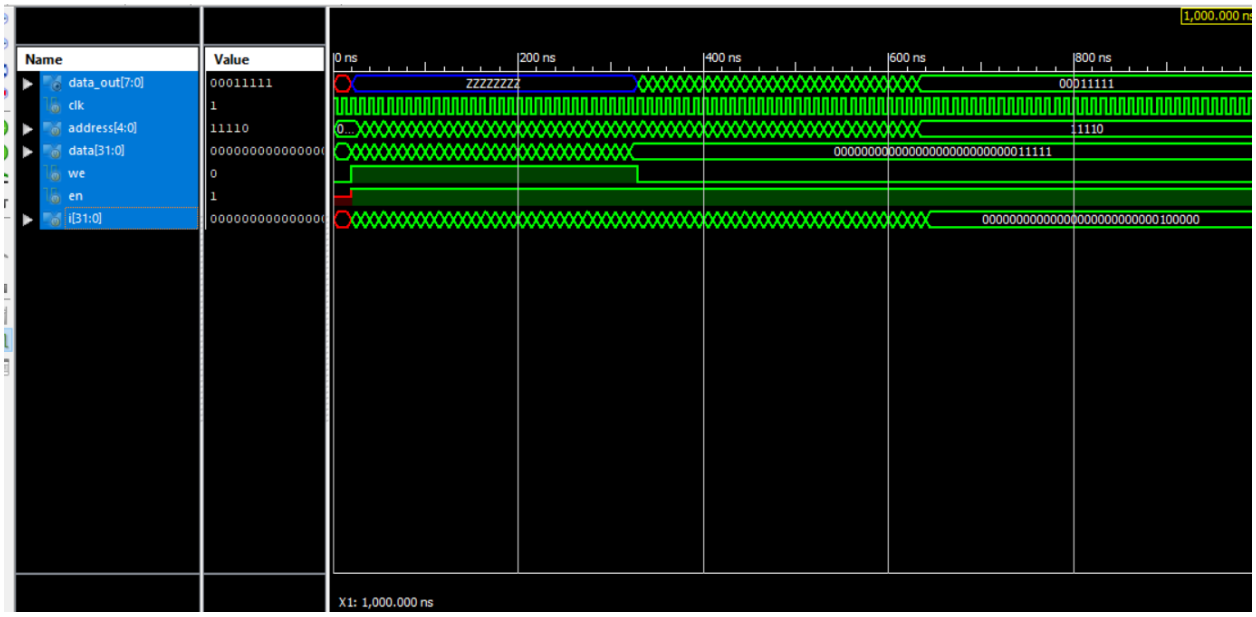
RAM

```
module ram(clk,address, data,we,en,data_out
    );
input clk,we,en;
input [4:0]address;
input [31:0]data;
reg [7:0]temp_data;
output  [7:0]data_out;

reg [7:0]ram[31:0];

always@(posedge clk)
begin
if(we & en)
ram[address]=data;
end

always@(posedge clk)
begin
if(!we & en)
temp_data=ram[address];
end

assign data_out= en &!we ?temp_data : 'hz;

endmodule
```




```
module ramtestbench;

        // Inputs
        reg clk;
        reg [4:0] address;
        reg [31:0] data;
        reg we;
        reg en;

        // Outputs
        wire [7:0] data_out;

        // Instantiate the Unit Under Test (UUT)
        ram uut (
                .clk(clk),
                .address(address),
                .data(data),
                .we(we),
                .en(en),
                .data_out(data_out)
        );
```

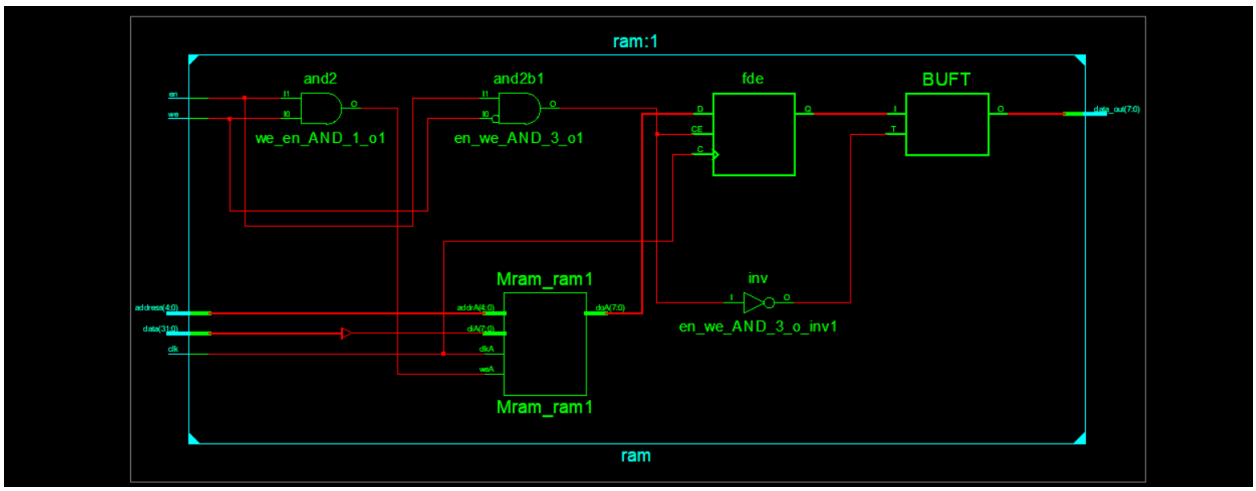
```
 always
       #5 clk = ~clk;
  integer i;
    initial begin
       // Initialize Inputs

       clk = 1;
       we = 0;
       data = 0;
       address =0;
       #20;
       //Write all the locations of RAM
       en= 1;
       we= 1;
      for(i=1; i <= 31; i = i + 1) begin
          data= i;
          address= i-1;
          #10;
       end
       en= 1;
       we= 0;
       //Read from port 1, all the locations of RAM.

       for( i=1; i <= 31; i = i + 1) begin
          address = i-1;
          #10;
       end

    end

endmodule
```



DualPort RAM

```
module dualport(clk1,din1,din2,cs,addr1,addr2,wen1,wen2,oen1,oen2,dout1,dout2);
        input cs,clk1,wen1,oen1,wen2,oen2;
        input [4:0] addr1;
        input [4:0] addr2;
        input [7:0] din1,din2;
        output [7:0] dout1,dout2;

reg [7:0] temp_data1,temp_data2;
reg [7:0] addr_ram1 [31:0];
reg [7:0] addr_ram2 [31:0];

always @ (posedge clk1)
```

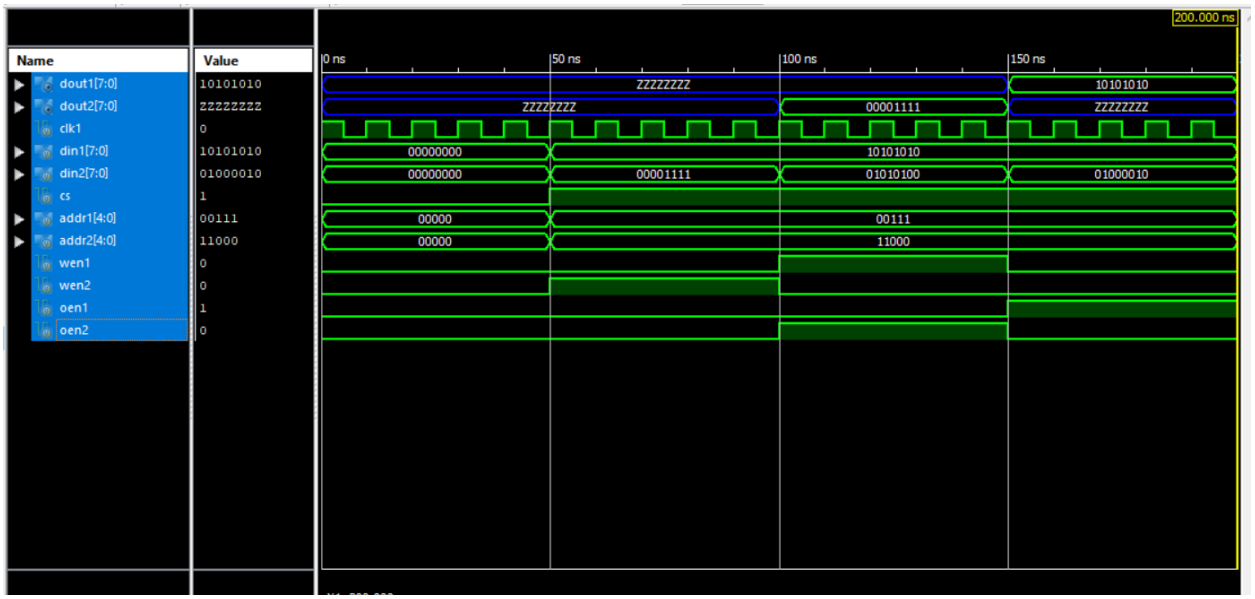
```
begin
        if (clk1 & cs & wen1)
                addr_ram1[addr1] <= din1;
end

always @ (posedge clk1)
begin
        if (clk1 & cs & wen2)
                addr_ram2[addr2] <= din2;
end

always @ (posedge clk1)
begin
        if (clk1 & cs & !wen1)
                temp_data1 <= addr_ram1[addr1];
end

always @ (posedge clk1)
begin
        if (clk1 & cs & !wen2)
                temp_data2 <= addr_ram2[addr2];
end

assign dout1 = (cs & !wen1 & oen1) ? temp_data1 : 8'bzzzzzzzz;
assign dout2 = (cs & !wen2 & oen2) ? temp_data2 : 8'bzzzzzzzz;

Endmodule
```



```
module dualport_ram_tb;

	// Inputs
	reg clk1;
	reg [7:0] din1;
	reg [7:0] din2;
	reg cs;
	reg [4:0] addr1;
	reg [4:0] addr2;
	reg wen1;
	reg wen2;
	reg oen1;
	reg oen2;

	// Outputs
	wire [7:0] dout1;
	wire [7:0] dout2;

	// Instantiate the Unit Under Test (UUT)
	dualport_ram_ uut (
		.clk1(clk1),
		.din1(din1),
		.din2(din2),
		.cs(cs),
		.addr1(addr1),
		.addr2(addr2),
		.wen1(wen1),
		.wen2(wen2),
		.oen1(oen1),
		.oen2(oen2),
		.dout1(dout1),
		.dout2(dout2)
	);

	initial begin
	clk1=1;
	forever clk1=#5 ~clk1;
	end
```

```
initial begin
        // Initialize Inputs
        din1 = 0;
        din2 = 0;
        cs = 0;
        addr1 = 0;
        addr2 = 0;
        wen1 = 0;
        wen2 = 0;
        oen1 = 0;
        oen2 = 0;
        #50;

        din1 = 8'b10101010;
        din2 = 8'b00001111;
        cs=1;
        addr1 = 5'b00111;
        addr2 = 5'b11000;
        wen1 = 0;
        wen2 = 1;
        oen1 = 0;
        oen2 = 0;
        #50;

        din1 = 8'b10101010;
        din2 = 8'b01010100;
        cs=1;
        addr1 = 5'b00111;
        addr2 = 5'b11000;
        wen1 = 1;
        wen2 = 0;
        oen1 = 0;
        oen2 = 1;
        #50;

        din1 = 8'b10101010;
        din2 = 8'b01000010;
        cs=1;
        addr1 = 5'b00111;
```

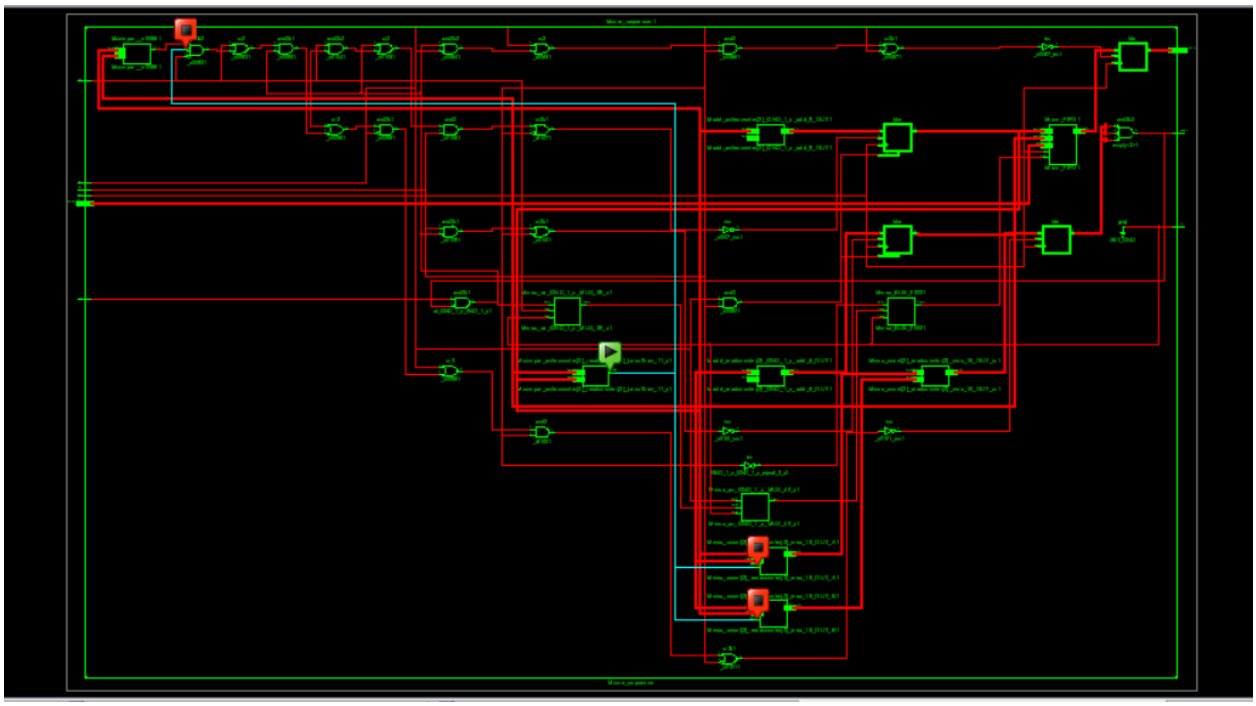
```
        addr2 = 5'b11000;
        wen1 = 0;
        wen2 = 0;
        oen1 = 1;
        oen2 = 0;
        #50;
        $finish;
    end

endmodule
```



FIFO

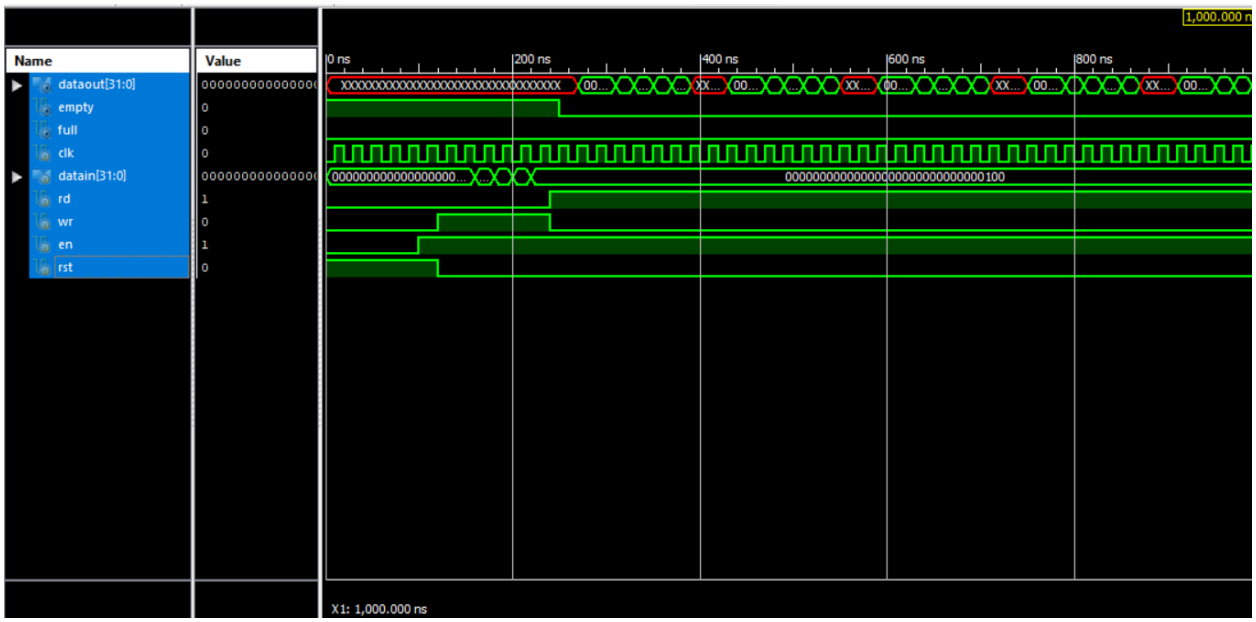
```
module FIFO_buffer(clk,datain,rd,wr,en,dataout,rst,empty,full
    );
input clk,rd,wr,en,rst;
output empty,full;
input [31:0]datain;
output reg[31:0]dataout;
reg [2:0]count=0;
reg[31:0]FIFO[0:7];
reg [2:0]readcounter=0,
                        writecounter=0;

assign empty=(count==0)?1'b1:1'b0;
assign full=(count==8)?1'b1:1'b0;
always@(posedge clk)
begin
        if(en==0);
        else
        if(rst)begin
                readcounter=0;
                writecounter=0;
        end

        else if(rd==1'b1 && count!=0)begin
                dataout=FIFO[readcounter];
```

```
            readcounter=readcounter+1;
        end

        else if(wr==1'b1 && count<8) begin
            FIFO[writecounter]=datain;
            writecounter=writecounter+1;
        end
        else if(writecounter==8)begin
            writecounter=0;
              end
            else if(readcounter==8)begin
                    readcounter=0;
                    end
            else if(readcounter>writecounter)begin
                    count=readcounter-writecounter;
                    end
            else if(writecounter>readcounter)begin
                    count=writecounter-readcounter;
          end

end
Endmodule
```



```
module FIFO_test;

	// Inputs
	reg clk;
	reg [31:0] datain;
	reg rd;
	reg wr;
	reg en;
	reg rst;

	// Outputs
	wire [31:0] dataout;
	wire empty;
	wire full;

	// Instantiate the Unit Under Test (UUT)
	FIFO_buffer uut (
		.clk(clk),
		.datain(datain),
		.rd(rd),
		.wr(wr),
		.en(en),
		.dataout(dataout),
		.rst(rst),
		.empty(empty),
		.full(full)
	);

	initial begin
		// Initialize Inputs
		clk = 0'b0;
		datain = 32'h0;
		rd = 1'b0;
		wr = 1'b0;
		en = 1'b0;
		rst = 1'b1;

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

```
        en = 1'b1;
        rst = 1'b1;

        // Wait 100 ns for global reset to finish
        #20;

        rst=1'b0;
        wr=1'b1;
        datain=32'h0;
        #20;

        datain=32'h0;
        #20;
        datain=32'h1;
        #20;
        datain=32'h2;
        #20;
        datain=32'h3;
        #20;
        datain=32'h4;
        #20;
        wr=1'b0;
        rd=1'b1;


    end
  always#10clk=~clk;
Endmodule
```

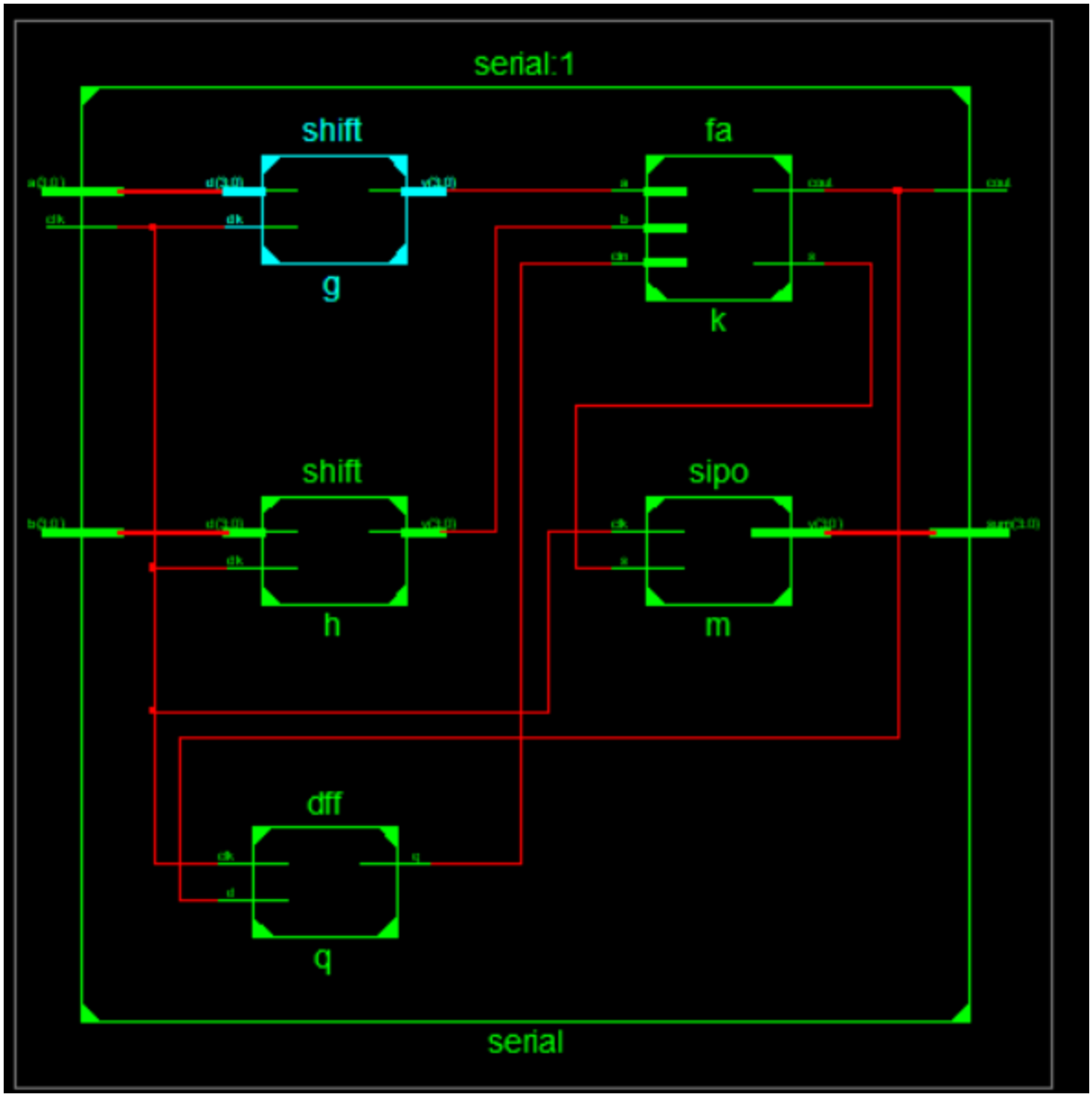


Serial Adder

```
module shift(y,d,clk);
input [3:0] d;
input clk;
output [3:0] y;
reg [3:0] y;

always @(posedge clk)
begin
y[0]=d[1];
y[1]=d[2];
y[2]=d[3];
y[3]=1'b0;
end
endmodule

//serial in parallel out register to store the 4 bit sum
module sipo(y,s,clk);
input  s;
input clk;
```

```
output [3:0] y;
reg [3:0] y;
always @(posedge clk)
begin
y={s,y[3:1]};
end
endmodule

//1 bit full adder
module fa(s,cout,a,b,cin);
input a,b,cin;
output s,cout;
assign s=a^b^cin;
assign cout=a&b|b%cin|cin&a;
endmodule

//d flipflop to store the cout of each stage
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

//main module serial adder//
module serial(sum,cout,a,b,clk);
input [3:0] a,b;
input clk;
wire [3:0] x,z;
output [3:0] sum;
output cout;
wire s,cin;
//input cin;
//initial begin
```

```
//cin=cinp;
//end
fa k(s,cout,x[0],z[0],cin);            //1 bit full adder
dff q(cin,cout,clk);                  //d flipflop to store the cout value after each 1 bit full
adder operation



sipo m(sum,s,clk);                     //serial sum(s) converted to parallel output(4 bit sum)///



shift g(x,a,clk);                  //shifts the input a
shift h(z,b,clk);                  //shifts the input b
endmodule
```


serial:1
shift
g
fa
k
shift
h
sipo
m
dff
q
serial