Verilog Assignment #1

Nehal Khaled Ibrahim 7118

1- Logic Gates (NOT, AND, OR, XOR, NOR, NAND) for two-4-bits inputs:

Code

testbench.sv

```
module tb_logic_gates_4bit;
  reg [3:0] a;
  reg [3:0] b;
  wire [3:0] not_a,not_b,and_out,or_out,xor_out,nor_out,nand_out;
  logic_gates_4bit g (a,b,not_a,not_b,and_out,or_out,xor_out,nor_out,nand_out);
    initial begin
    $dumpvars(1,g);
    a = 4'b0000;
    b = 4'b1111;
    $display("a=%b",a);
    $display("b=%b",b);
    #10
    $display("not_a=%b",not_a);
    $display("not_b=%b",not_b);
    $display("and_out=%b",and_out);
    $display("or_out=%b",or_out);
    $display("xor_out=%b",xor_out);
    $display("nor_out=%b",nor_out);
    $display("nand_out=%b",nand_out);
    #10;
    a = 4'b1100;
    b = 4'b0011;
    $display("a=%b",a);
    $display("b=%b",b);
    #10
    $display("not_a=%b",not_a);
    $display("not_b=%b",not_b);
    $display("and_out=%b",and_out);
    $display("or_out=%b",or_out);
    $display("xor_out=%b",xor_out);
    $display("nor_out=%b",nor_out);
    $display("nand_out=%b",nand_out);
    #10;
    a = 4'b1010;
    b = 4'b0101;
    $display("a=%b",a);
    $display("b=%b",b);
    #10
    $display("not_a=%b",not_a);
    $display("not_b=%b",not_b);
    $display("and_out=%b",and_out);
    $display("or_out=%b",or_out);
    $display("xor_out=%b",xor_out);
    $display("nor_out=%b",nor_out);
    $display("nand_out=%b",nand_out);
    #10;
    a = 4'b1111;
    b = 4'b1111;
    $display("a=%b",a);
    $display("b=%b",b);
    #10
    $display("not_a=%b",not_a);
    $display("not_b=%b",not_b);
    $display("and_out=%b",and_out);
    $display("or_out=%b",or_out);
    $display("xor_out=%b",xor_out);
    $display("nor_out=%b",nor_out);
    $display("nand_out=%b",nand_out);
    #10;
    $finish;
  end
endmodule
```

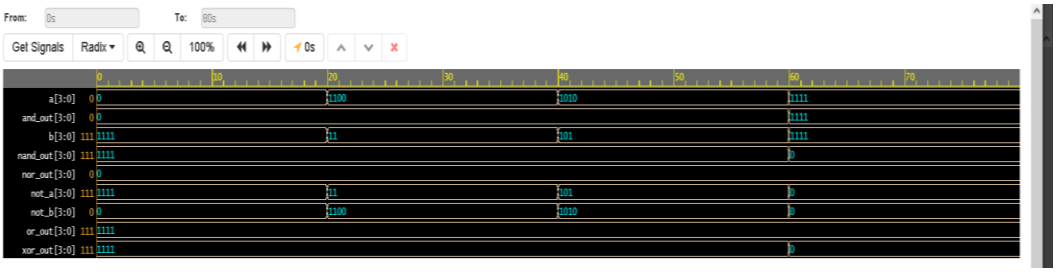
design.sv

```
module logic_gates_4bit(
  input [3:0] a,
  input [3:0] b,
  output [3:0] not_a,
  output [3:0] not_b,
  output [3:0] and_out,
  output [3:0] or_out,
  output [3:0] xor_out,
  output [3:0] nor_out,
  output [3:0] nand_out
);
  assign not_a = ~a;
  assign not_b = ~b;
  assign and_out = a & b;
  assign or_out = a | b;
  assign xor_out = a ^ b;
  assign nor_out = ~(a | b);
  assign nand_out = ~(a & b);
endmodule
```

Output:

```
a=0000
b=1111
not_a=1111
not_b=0000
and_out=0000
or_out=1111
xor_out=1111
nor_out=0000
nand_out=1111
a=1100
b=0011
not_a=0011
not_b=1100
and_out=0000
or_out=1111
xor_out=1111
nor_out=0000
nand_out=1111
a=1010
b=0101
not_a=0101
not_b=1010
and_out=0000
or_out=1111
xor_out=1111
nor_out=0000
nand_out=1111
a=1111
b=1111
not_a=0000
not_b=0000
and_out=1111
or_out=1111
xor_out=0000
nor_out=0000
nand_out=0000
Finding VCD file...
```

Signal:



2- AND gate for one-8-bits input:

Code :

testbench.sv

```
module tb_logic_gates_4bit;
  reg [7:0] a;
  reg  y;
  and_gate_8bit ay(a,y);
    initial begin
    a = 8'b11001111;
    $display("a=%b",a);
    #10
    $display("y=%b",y);
    #10;
    $finish;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0,a,y);
  end
endmodule
```

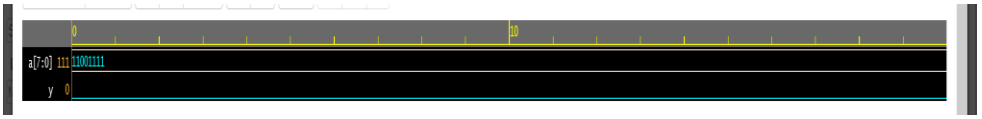
design.sv

```
module and_gate_8bit(input [7:0] a,output y);
  assign y=a[7]&a[6]&a[5]&a[4]&a[3]&a[2]&a[1]&a[0];
endmodule
```

Output:

```
a=11001111
VCD info: dumpfile dump.vcd opened for output.
y=0
```

Signal:



3- 2:1 MUX for two-4-bits inputs:

Code:

testbench.sv

```
module tb_mux_2to1_4bit;
  reg [3:0] a;
  reg [3:0] b;
  reg s;
  wire [3:0] out;
  mux_2to1_4bit dut (.a(a),.b(b), .s(s),.out(out));
  initial begin
    a = 4'b0000;
    b = 4'b1111;
    s = 0;
    $display("a=%b, b=%b, s=%b",a,b,s);
    #1 $display("output=%b",out);
    #10;
    a = 4'b1010;
    b = 4'b0101;
    s = 1;
    $display("a=%b, b=%b, s=%b",a,b,s);
    #1 $display("output=%b",out);
    #10;
    a = 4'b1100;
    b = 4'b0011;
    s = 0;
    $display("a=%b, b=%b, s=%b",a,b,s);
    #1 $display("output=%b",out);
    #10;
    a = 4'b1111;
    b = 4'b1111;
    s = 1;
    $display("a=%b, b=%b, s=%b",a,b,s);
    #1 $display("output=%b",out);
    #10;
    $finish;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1,dut);
  end
endmodule
```

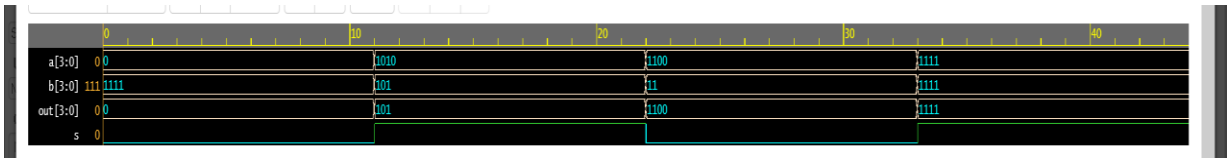
design.sv

```
module mux_2to1_4bit(input [3:0] a,input [3:0] b,input s,output [3:0
  assign out = s ? b : a;
endmodule
```

Output:

```
a=0000, b=1111, s=0
VCD info: dumpfile dump.vcd opened for output.
output=0000
a=1010, b=0101, s=1
output=0101
a=1100, b=0011, s=0
output=1100
a=1111, b=1111, s=1
output=1111
```

Signal:



4- 4:1 MUX for four-4-bits inputs:

Code:

testbench.sv

```
    reg [3:0] c;
    reg [3:0] d;
    wire [3:0] out;
    reg [1:0] s;
    integer i;
    mux_4to1  mux (.a (a),.b (b),.c (c),.d (d),.s (s),.out (out));
    initial begin
      $monitor ("[%0t] s=%b a=%b b=%b c=%b d=%b out=%b", $time, s, a, b, c, d,
out);
        s <= 0;
        a <= $random;
        b <= $random;
        c <= $random;
        d <= $random;
        for (i = 1; i < 4; i=i+1) begin
          #5 s <= i;
        end
        #5 $finish;
    end
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(1,mux);
    end
endmodule
```

design.sv

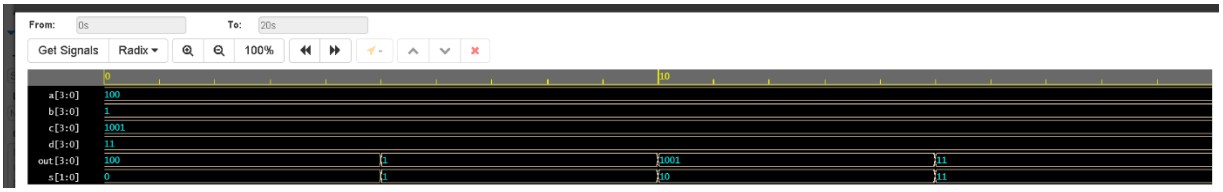
```
module mux_4to1 ( input [3:0] a,input [3:0] b,input [3:0] c,input [3:0] d,input
[1:0] s,output reg [3:0] out);
    assign out = s[1] ? (s[0] ? d : c) : (s[0] ? b : a);

endmodule
```

Output:

```
[0] s=00 a=0100 b=0001 c=1001 d=0011 out=0100
[5] s=01 a=0100 b=0001 c=1001 d=0011 out=0001
[10] s=10 a=0100 b=0001 c=1001 d=0011 out=1001
[15] s=11 a=0100 b=0001 c=1001 d=0011 out=0011
```

Signal:



4- Full Adder:

Code:

testbench.sv

```
module tb_fulladd;
    reg a;
    reg b;
    reg cin;
    wire sum,cout;
    wire p,g;
    fulladd fa (.a (a),.b (b),.cin (cin),.cout (cout),.sum (sum));
    initial begin
        a = 1'b1;
        b = 1'b0;
        cin=1'b1;
      $display ("a=%b b=%b cin=%b", a, b, cin);
      #1
      $display ("sum=%b", sum);
      $display ("cout=%b", cout);
    end
    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(1,fa);
    end
endmodule
```

design.sv

```
module fulladd (input a,input b,input cin,output cout, output sum);
    wire p,g;
    assign p=a^b;
    assign g=a&b;
    assign sum=p^cin;
    assign cout = g|(p&cin);
endmodule
```

Output:

```
a=1 b=0 cin=1
VCD info: dumpfile dump.vcd opened for output.
sum=0
cout=1
```

Signal: