Faculty of Engineering

Ain Shams University

CSE 620 - Advanced Computer Architecture

Project:

HDL Testbench

By: Youssef Mahmoud Zaki Al Arnaouty

ID: 2002388

Table of Contents

1. Introduction

In this project, I have used Xilinx's ISE Design Suite 14.7, it includes ISim for HDL code simulation, and a schematic viewer for generating RTL and Technology schematics (target based RTL).

I have implemented 5 examples in Verilog with their testbenches, provided simulation results, waveforms, and RTL schematics.

I have tried to implement a different testing strategy for every module while making sure that all modules are synthesizable.

2. AND Gate (Sensitive)

Based on the and2_sensitivity.vhd file.

	Name	Bits
Inputs	a, b	1 Bit each
Outputs	c	1 Bit

Operation: The outputs is set as the logical AND between the 2 inputs.

Verilog code:

```
module AND_Sens(
    input a,
    input b,
    output reg c
    );
always @ (a,b)
 c<=a&b;

endmodule
```

Testbench code:

```
module ANd_Sens_tb;
   // Inputs
   reg a;
   reg b;
   // Outputs
   wire c;
   // Instantiate the Unit Under Test (UUT)
   AND_Sens uut (.a(a), .b(b), .c(c));
   initial begin
   $monitor ("a=%d  |  b=%d  |   result=%d", a, b, c);
      a = 0;
      b = 0;
      #10
      a= 0;
      b=1;
      #10
      a= 1;
      b=0;
      #10
      a= 1;
      b=1;
   end
endmodule
```

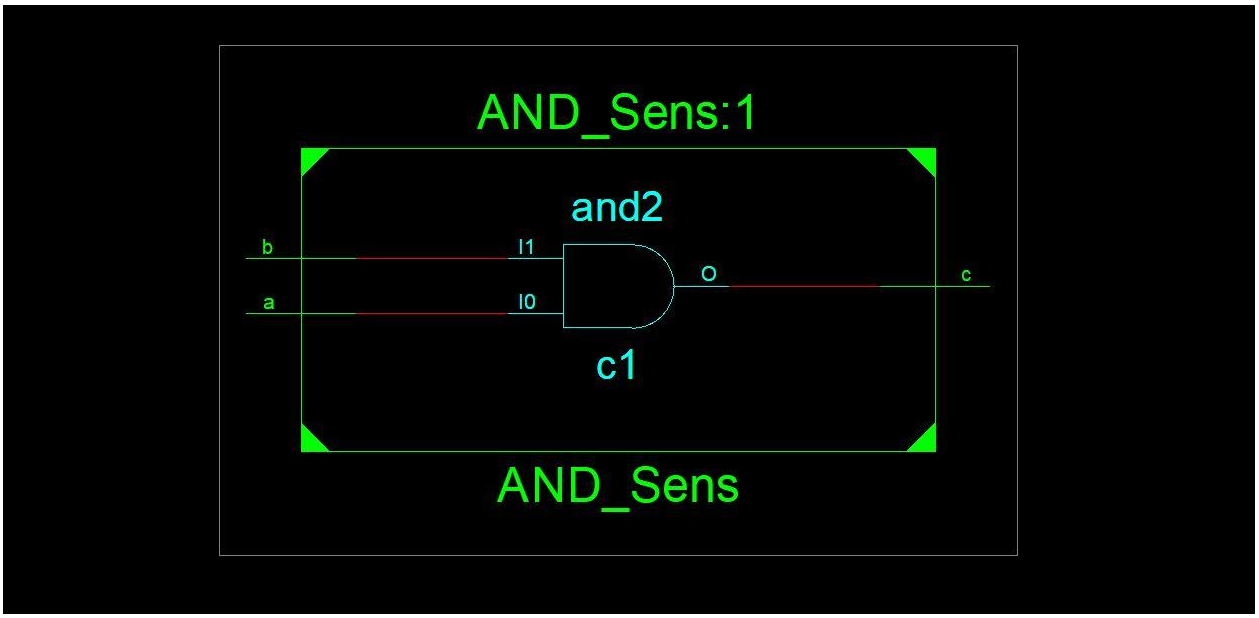
Project : HDL Testbench

Testing strategy:

As there are only 2 inputs, an exhaustive test can be done easily, generating a truth table.

```
Console
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
a=0 | b=0 | result=0
a=0 | b=1 | result=0
a=1 | b=0 | result=0
a=1 | b=1 | result=1
```

RTL Schematic:



3. Counter

Based on counter.vhd file.

	Name	Bits
Inputs	Clk	1 Bit
Outputs	qd	4 Bits

Operation: the module is a free-running counter that starts from 0 and increments mod 16 on each rising edge of the clock.

Verilog Code:

```
module mycounter(
    input clk,
    output [3:0] qd
    );
// reg to hold the counting value intialized to zero
reg [3:0] cnt = 0;
assign qd = cnt;
always@(posedge clk)
   begin
         cnt <= (cnt +1);
   end
endmodule
```

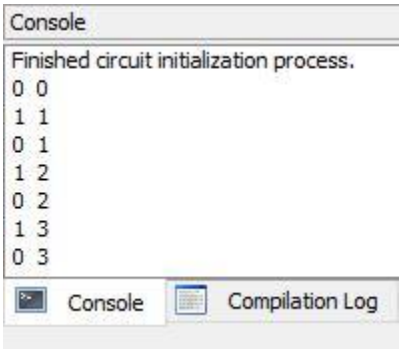
Testbench Code:

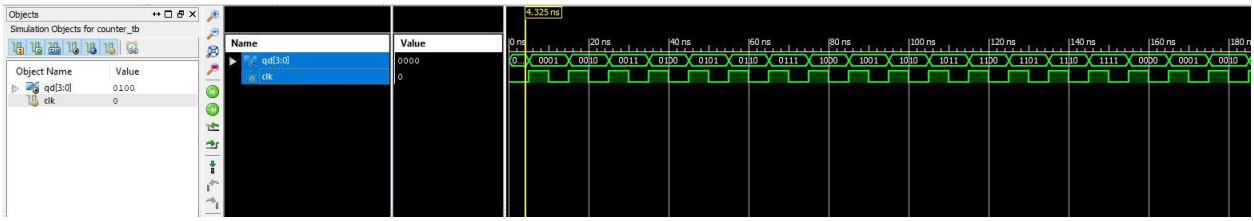
```
module counter_tb;
   // Inputs
   reg clk;
   // Outputs
   wire [3:0] qd;
   // Instantiate the Unit Under Test (UUT)
   mycounter uut ( .clk(clk), .qd(qd));
   initial
   begin
      // Generating clock cycles (square wave)
      clk=0;
      forever #5 clk=~clk;
   end
   initial begin
         $monitor(clk,qd );
   end
endmodule
```

Project : HDL Testbench

Testing Strategy:

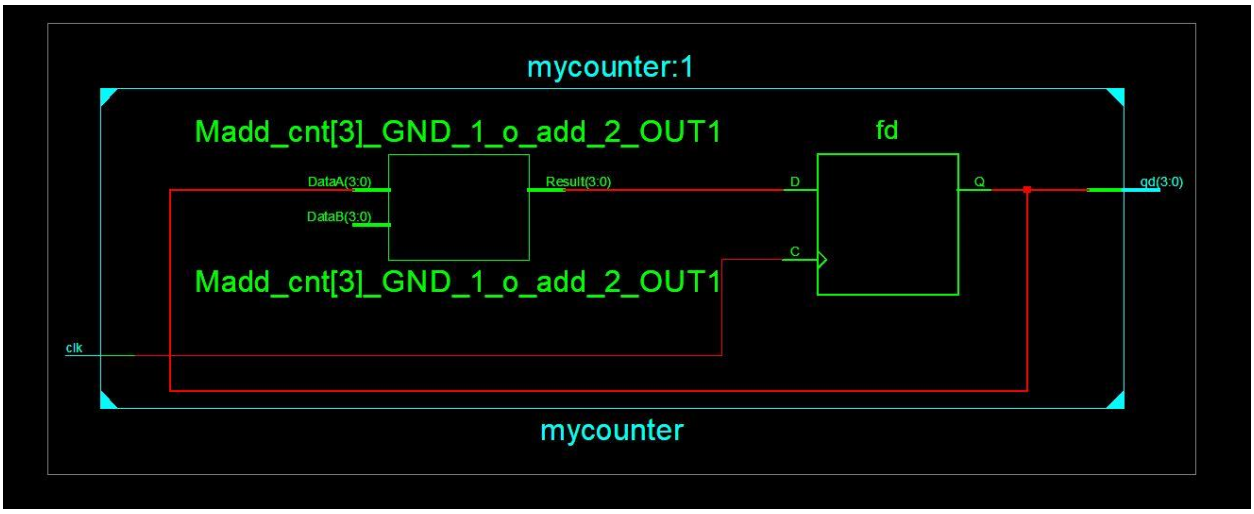
By generating a square wave, the output is incremented only on the clock's positive edge, also we check that after overflowing the counter is reset to zero.





After the counting is overflew, it resets back to zero as if the incrementation is done in mod 16.

RTL Schematic:



4. Full Adder

Based on full_adder.vhd file.

	Name	Bits
Inputs	a, b	4 Bits
	C_in	1bit
Outputs	s	4 Bits
	c_out	1 bit

Operation: perform bitwise addition to the inputs to produce the output and carryout signal.

Verilog Code:

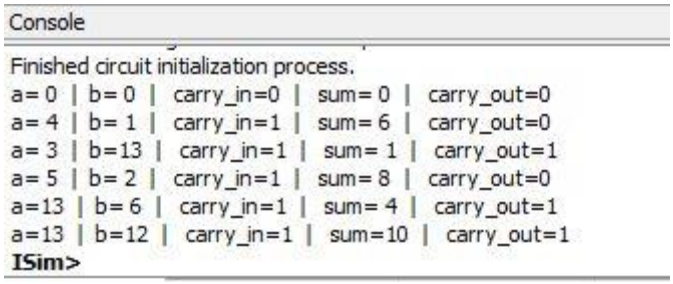
```
module myFullAdder(  input [3:0] a,
                     input [3:0] b,
                     input c_in,
                     output c_out,
                     output [3:0] sum);
   assign {c_out, sum} = a + b + c_in;
endmodule
```

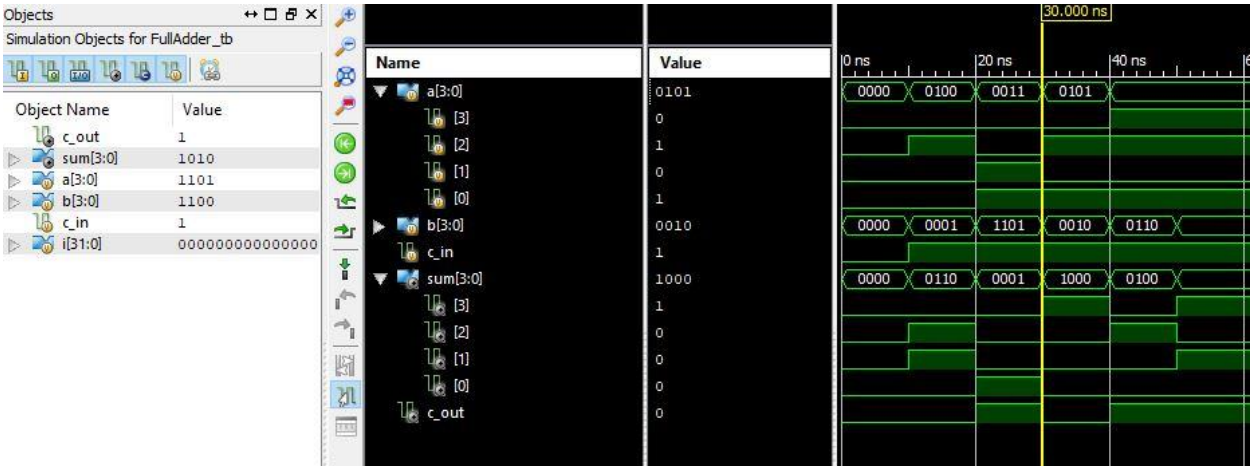
Testbench code:

```
module FullAdder_tb;
   // Inputs
   reg [3:0] a;
   reg [3:0] b;
   reg c_in;
   // Outputs
   wire c_out;
   wire [3:0] sum;
   //Variables
   integer i;
   // Instantiate the Unit Under Test (UUT)
   myFullAdder uut (.a(a),.b(b),.c_in(c_in),.c_out(c_out), .sum(sum));
   initial begin
      a <= 0;
      b <= 0;
      c_in <= 0;
      $monitor ("a=%d  |  b=%d  |   carry_in=%d  |   sum=%d  |   carry_out=%d", a, b, c_in, sum,c_out);
        // Use a for loop to apply random values to the input
      for (i = 0; i < 5; i = i+1) begin
         #10
         a <= $random;
         b <= $random;
         c_in <= $random;
      end
   end
endmodule
```

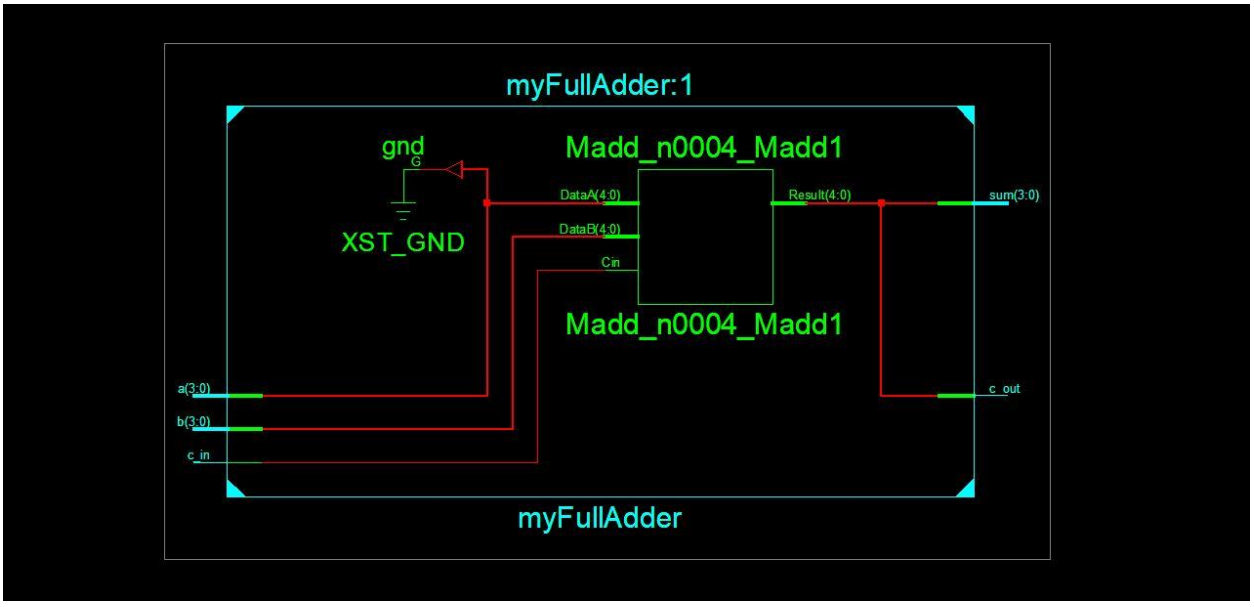
Testing strategy:

Generating random values for a, b, and the carry in signal. We can change these values by altering the seed parameter.





RTL Schematic:



5. Multiplexer With Selection Bits

Based on mux_sel.vhd file.

	Name	Bits
Inputs	In1, in2, in3, in4	4 Bits
	sel	2 Bits
Outputs	out	4 Bits

Operation: The output is chosen from the input according to the selection bits.

Verilog Code:

```
module mux_sel(
    input [3:0] in1,
    input [3:0] in2,
    input [3:0] in3,
    input [3:0] in4,
    input [1:0] sel,
    output reg [3:0] out
    );
always @ (in1,in2,in3,in4,sel)
case (sel)
          0: out <= in1;
          1: out <= in2;
          2: out <= in3;
          3: out <= in4;
          default: out <= 4'bz;
endcase
endmodule
```

Testbench code:

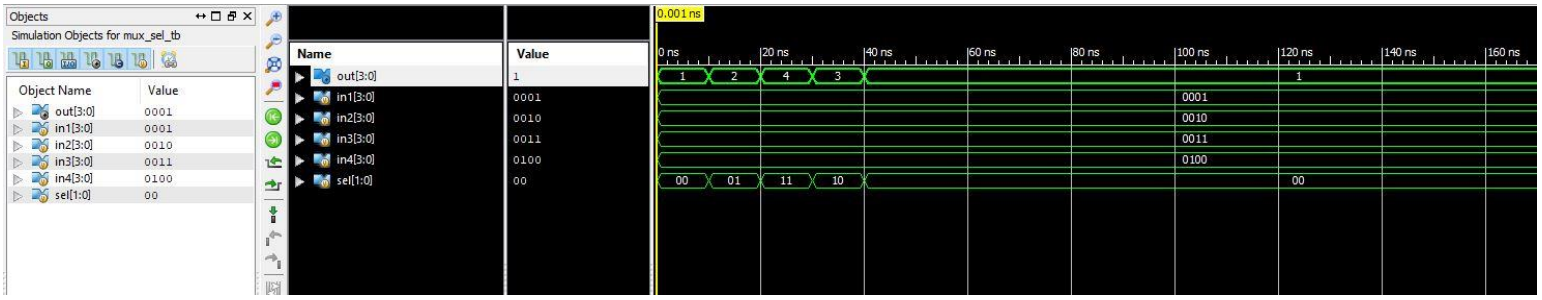
```
module mux_sel_tb;
   // Inputs
   reg [3:0] in1;
   reg [3:0] in2;
   reg [3:0] in3;
   reg [3:0] in4;
   reg [1:0] sel;
   // Outputs
   wire [3:0] out;
   // Instantiate the Unit Under Test (UUT)
   mux_sel uut (.in1(in1), .in2(in2), .in3(in3), .in4(in4), .sel(sel),.out(out));
   initial begin
   $monitor(" sel = %d, out = %d",sel,out);
      // Initialize Inputs
      in1 = 1;
      in2 = 2;
      in3 = 3;
      in4 = 4;
      sel = 0;
      #10;
       sel = 1;
       #10
       sel = 3;
       #10
       sel = 2;
       #10
       sel = 4;
       #10
       sel = 0;
   end
endmodule
```

Project : HDL Testbench

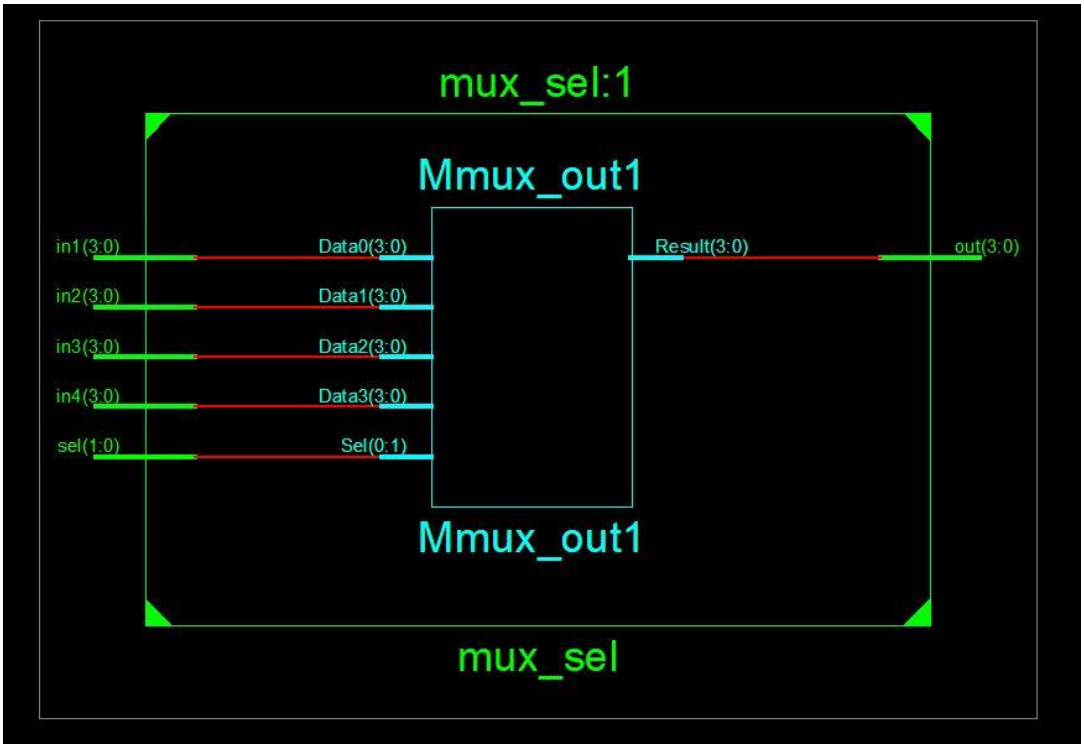
Testing Strategy:

To test this module, we test all the possible bit combinations of the “sel” bits but this time, we change the bits only one bit at a time.

```
Console
Simulator is doing circuit initialization process.
Finished circuit initialization process.
 sel = 0, out =  1
 sel = 1, out =  2
 sel = 3, out =  4
 sel = 2, out =  3
 sel = 0, out =  1
ISim>
```



RTL Schematic:



6. Priority

Based on priority.vhd file.

	Name	Bits
Inputs	sel	4 Bit
Outputs	z	3 Bits

Operation: the output is chosen according to order of 1's in the selection bits. Position 3 has the highest priority (MSB) and position 0 is the lowest (LSB).

Verilog code:

```
module Priority(
          input [3:0] sel,
          output reg [2:0] z
    );
always@(sel)
   if(sel[3])
      z <= 3'b000;
   else if (sel[2])
      z <= 3'b001;
   else if (sel[1])
      z <= 3'b010;
   else if (sel[0])
      z <= 3'b011;
    else
      z <= 3'b111;

endmodule
```

Testbench code:

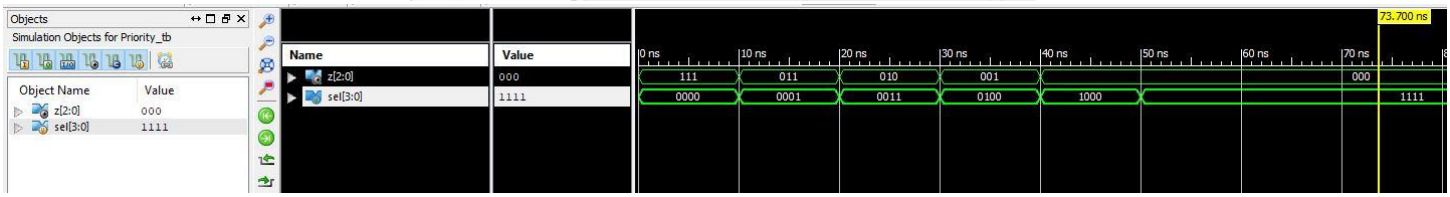
```
module Priority_tb;
   // Inputs
   reg [3:0] sel;
   // Outputs
   wire [2:0] z;
   // Instantiate the Unit Under Test (UUT)
   Priority uut (.sel(sel), .z(z));
   initial begin
   $monitor ("sel[3]=%b  |  sel[2]=%b  |  sel[1]=%b  |  sel[0]=%b  |  z=%b", sel[3], sel[2], sel[1], sel[0], z);
      // Initialize Inputs
      sel = 0;
      #10
      sel = 1;
      #10
      sel = 3;
      #10
      sel = 4;
      #10
      sel = 8;
      #10
      sel = 15;
   end
endmodule
```

Project : HDL Testbench

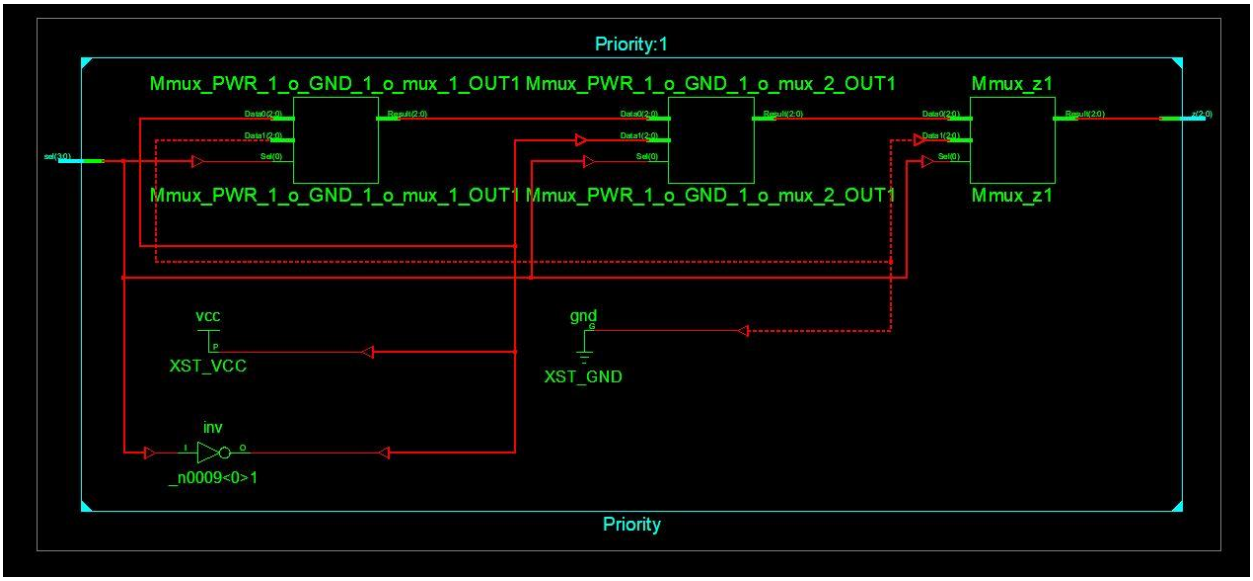
Test Strategy:

Setting all the bits successively then setting to bits at the same time.

```
Console
Finished circuit initialization process.
sel[3]=0 | sel[2]=0 | sel[1]=0 | sel[0]=0 | z=111
sel[3]=0 | sel[2]=0 | sel[1]=0 | sel[0]=1 | z=011
sel[3]=0 | sel[2]=0 | sel[1]=1 | sel[0]=1 | z=010
sel[3]=0 | sel[2]=1 | sel[1]=0 | sel[0]=0 | z=001
sel[3]=1 | sel[2]=0 | sel[1]=0 | sel[0]=0 | z=000
sel[3]=1 | sel[2]=1 | sel[1]=1 | sel[0]=1 | z=000
ISim>
```



RTL Schematic:



7. References

1- Xilinx ISE Documentation and tutorial.
2- ChipVerify's Verilog tutorial.