**Digital System design Processing Lab**

Lab Journal: 02



Name: M.OMAR

Class: BCE-6

Enrollment No: 01-132182-024

**Lab # 02**

**Introduction to Verilog HDL and Gate Level Modelling**

**Objective:**

* **The objective of this lab is to give a comprehensive tutorial of Verilog Hardware Descriptive Language (HDL).**

**Procedure:**

* Open eda playground Software.
* Make a new script and name it on the name of your lab.
* Make your tasks in the same folder and keep them at the same directory.
* Make functions and name them same as your script name.
* Make sure that the main function and the other functions are in the same directory.
* Run the given code and see the output.

**INTRODUCTION:**

**Background information**

There are two types of circuit Sequential circuit and Combinational circuit. Sometimes we have to make the circuit designs that are very big and complex, So it is better to first test that design through coding in Verilog language. By this technique we can save our time and resources. We know the how Half adder circuit and full adder circuit works. But in this lab, we are using Verilog language to make Adder circuit.

**Procedure:**

Full adder is a circuit that is made of 2 half adder and is use to sum three-bit number. In this Lab we are using three methods to write full adder circuit code in Verilog.

1. **Data flow level**

This level of abstraction is higher than the gate level. Expressions, operands and operators characterize this level. Most of the operators used in dataflow modeling are common to software programmers, but there are a few others that are specific to HW design. Operators that are used in expressions for dataflow modeling are given in Table 2.3. At this level every expression starts with the keyword assign. Here is a simple example where two variables a and bare added to produce c: assign c a + b; The value on wire c is continuously driven by the result of the arithmetic operation. This assignment statement is also called ‘continuous assignment’. In this statement the right-hand side must be a variable of type wire, whereas the operands on the left-hand side may be of type wire or reg.

1. **Gate level**

The code at gate level is built from Verilog primitives. These primitives are built in gate level models of basic functions, including nand, nor, and, or, xor, buf and not. Modeling at this level requires describing the circuit using logic gates. This description looks much like an implementation of a circuit in a basic logic design course. Delays can also be modeled at this level.

1. **Behavioral**

The behavioral level is the highest level of abstraction in Verilog. This level provides high level language constructs like for, while, repeat, if else and case. Designers with a software programming background already know these constructs

**Task 01**

Write stimulus module for  
 **- OR gate  
 - AND gate  
 - NOT gate  
 - XOR gate  
 - XNOR gate  
 - NAND gate  
 - NOR gate**

Code: Stimulus for and,nand,.or,nor,xor,xnor are all same and is as follows

// Code your testbench here

// or browse Examples

module myand\_tb;

reg a1,b1;

wire y1;

myand myand\_tb(.a(a1),.b(b1),.y(y1));

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

// $monitor(a1,b1,y1);

a1=1'b0;

b1=1'b0;

#1 $display("a1:%b, b1:%b, y1:%b",a1,b1,y1);

#1

a1=1'b0;

b1=1'b1;

#1 $display("a1:%b, b1:%b, y1:%b",a1,b1,y1);

#1

a1=1'b1;

b1=1'b0;

#1 $display("a1:%b, b1:%b, y1:%b",a1,b1,y1);

#1

a1=1'b1;

b1=1'b1;

#1 $display("a1:%b, b1:%b, y1:%b",a1,b1,y1);

end

endmodule

Stimulus for not gate:

// Code your testbench here

// or browse Examples

module my\_tb;

reg a1;

wire y1;

my my\_tb(.a(a1),.y(y1));

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

$monitor(a1,y1);

a1=1'b0;

#1

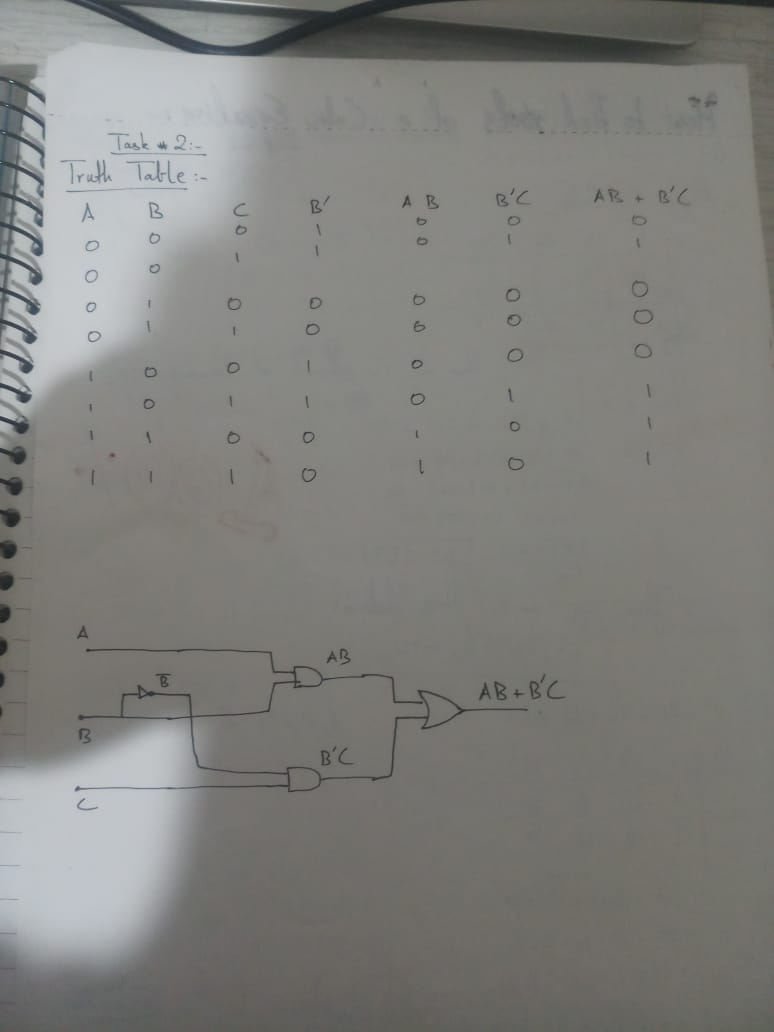
a1=1'b1;

end

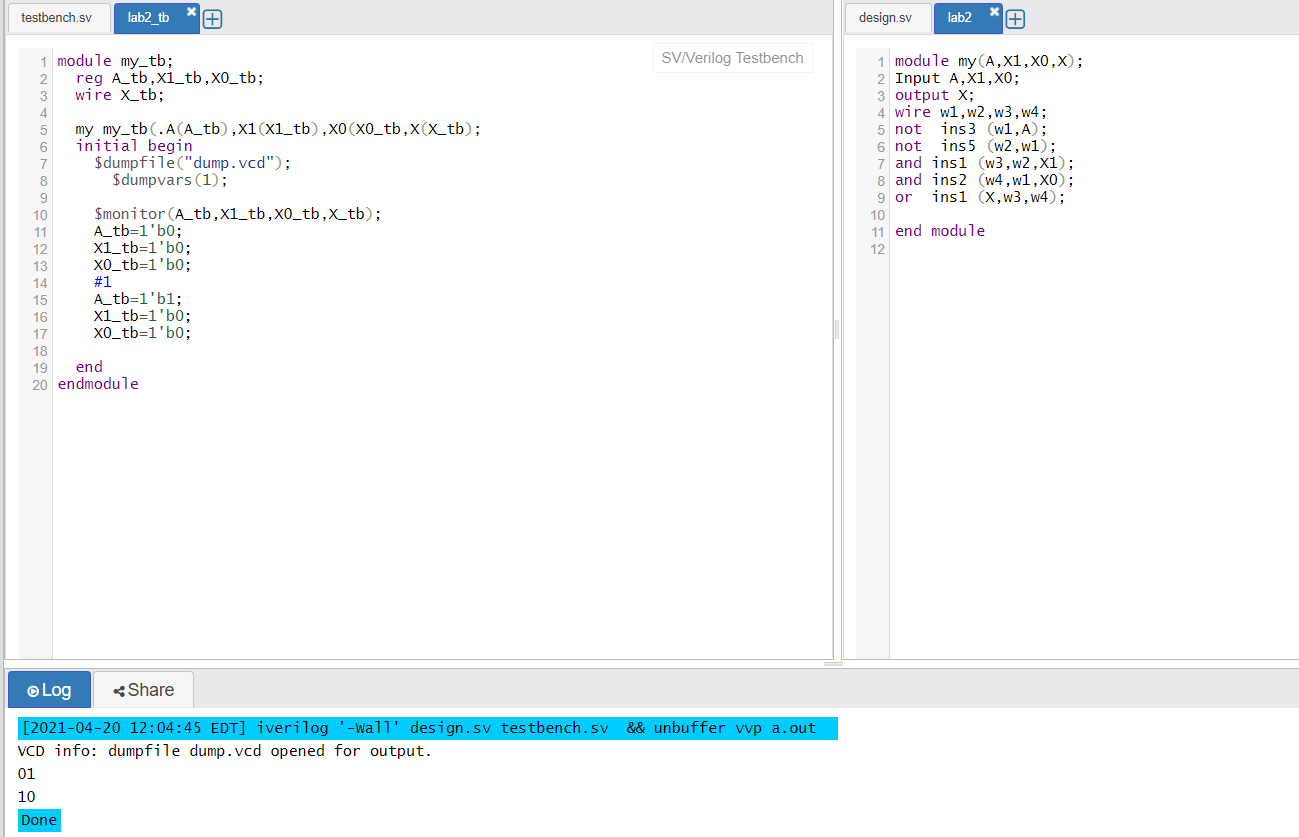
endmodule

TASK#2

* **Design a gate level diagram and truth table for the given equation and implement its design in Verilog.**
* **F=AB+B’C**
* **Code**
* module my(A,X1,X0,X);
* Input A,X1,X0;
* output X;
* wire w1,w2,w3,w4;
* not ins3 (w1,A);
* not ins5 (w2,w1);
* and ins1 (w3,w2,X1);
* and ins2 (w4,w1,X0);
* or ins1 (X,w3,w4);
* end module
* module my\_tb;
* reg A\_tb,X1\_tb,X0\_tb;
* wire X\_tb;
* my my\_tb(.A(A\_tb),X1(X1\_tb),X0(X0\_tb,X(X\_tb);
* initial begin
* $dumpfile("dump.vcd");
* $dumpvars(1);
* $monitor(A\_tb,X1\_tb,X0\_tb,X\_tb);
* A\_tb=1'b0;
* X1\_tb=1'b0;
* X0\_tb=1'b0;
* #1
* A\_tb=1'b1;
* X1\_tb=1'b0;
* X0\_tb=1'b0;
* end
* endmodule



OUTPUT:



TASK#3:

* Write a Verilog code for 2:1 MUX along with its stimulus module.

|  |  |
| --- | --- |
| * **Task 3** | **Making 2x1 Multiplexer on data flow level** |

**Code :**

Mux 2x1

module AXT(

in1,in2,s,out

);

input in1,in2,s;

output out;

assign out =s?in1:in2;

endmodule

**Setting Timing**

// Inputs

reg in1;

reg in2;

reg s;

// Outputs

wire out;

// Instantiate the Unit Under Test (UUT)

tt uut (

.in1(in1),

.in2(in2),

.s(s),

.out(out)

);

initial begin

// Initialize Inputs

in1 = 0;

in2 = 0;

s = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

in1 = 1;

in2 = 0;

s = 0;

// Wait 100 ns for global reset to finish

#100;

in1 = 0;

in2 = 1;

s = 0;

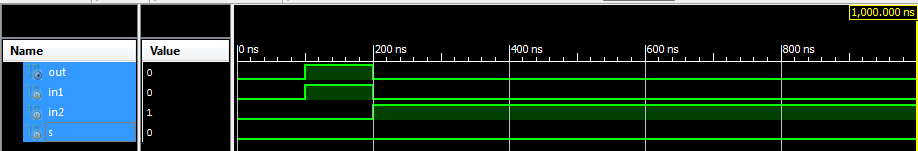
// #100

// Wait 100 ns for global reset to finish

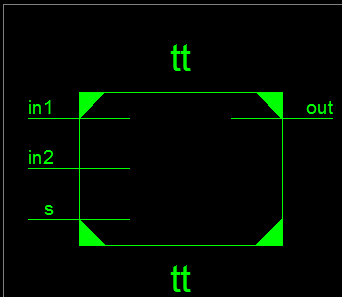
end

endmodule

**Simulation:**



**Diagram:**



TASK#4:

* **Write a Verilog code for 4x1 MUX along its stimulus module.**

|  |  |
| --- | --- |
| * **Task4** | **Making 4x1 Multiplexer using data flow level** |

**Code:**

module Multiplexer(

input I1,

input I2,

input I3,

input I4,

input [1:0]Selectlines,

output out

);

assign out = Selectlines[1]?Selectlines[0]?I4:I3:Selectlines[0]?I2:I1;

endmodule

**Setting Timing**

module MuxTime;

// Inputs

reg I1;

reg I2;

reg I3;

reg I4;

reg [1:0] Selectlines;

// Outputs

wire out;

// Instantiate the Unit Under Test (UUT)

Multiplexer uut (

.I1(I1),

.I2(I2),

.I3(I3),

.I4(I4),

.Selectlines(Selectlines),

.out(out)

);

initial begin

// Initialize Inputs

I1 = 0;

I2 = 1;

I3 = 0;

I4 = 1;

Selectlines = 0;

// Wait 100 ns for global reset to finish

#100;

I1 = 0;

I2 = 1;

I3 = 0;

I4 = 1;

Selectlines = 1;

// Add stimulus here

#50;

I1 = 0;

I2 = 1;

I3 = 0;

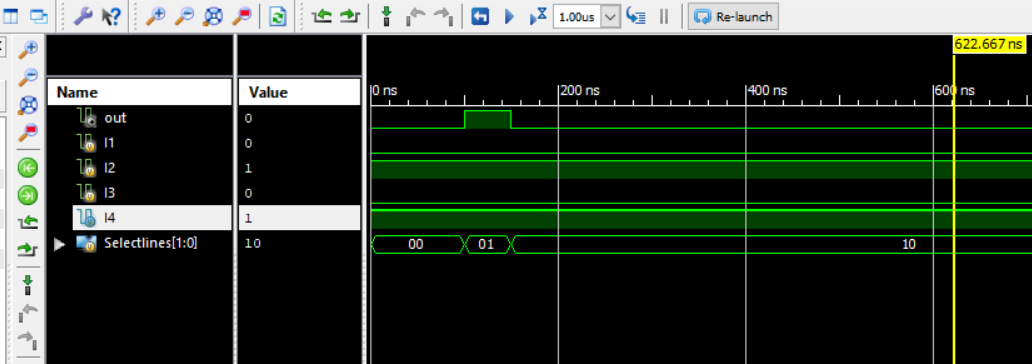
I4 = 1;

Selectlines = 2;

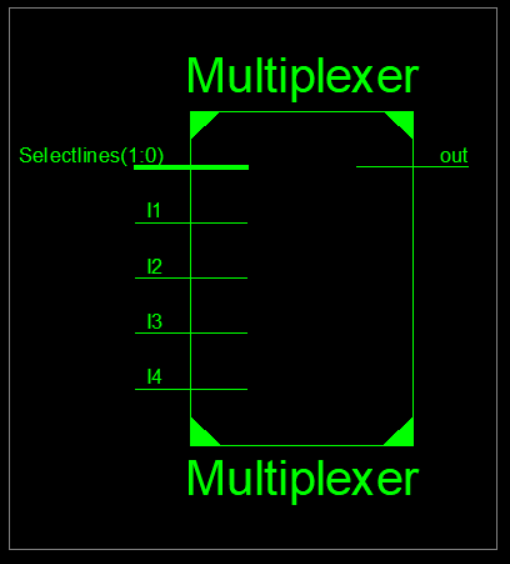
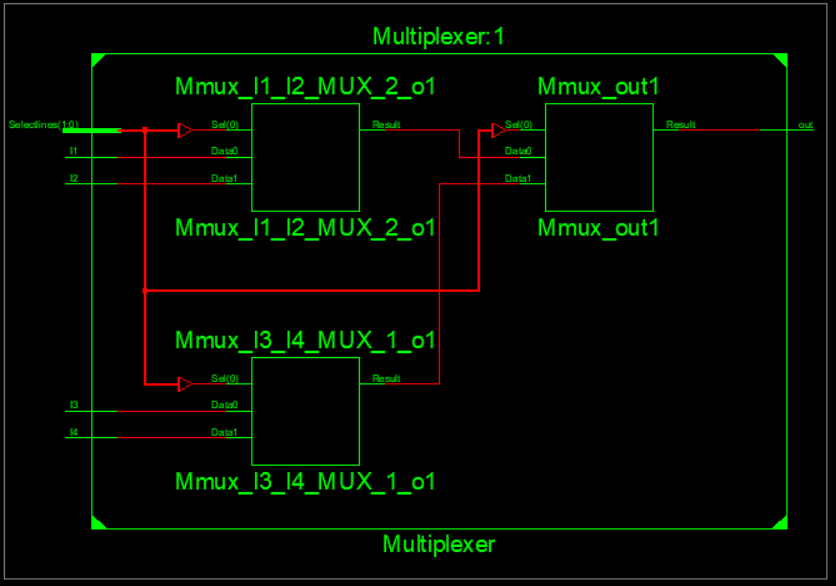
end

endmodule

**Simulation:**



**Diagram:**

TASK#5:

**Write a Verilog code for Half Adder along its stimulus module**

**Half adder Truth table**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **carry** | **sum** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

module HA(d , e , sum , carry);

input d,e;

output sum , carry;

xor (sum , d , e);

and (carry , d , e);

endmodule

module FA(X , Y , Z , S , C);

input X , Y , Z;

output S , C;

wire s1 , d1 , d2;

HA HA1(X , Y , s1 , d1),

HA2(Z , s1 , s , d2);

or g1 (C , d2 , d1);

endmodule

**Setting Timing of Inputs:**

module fulladdertiming;

// Inputs

reg X;

reg Y;

reg Z;

// Outputs

wire S;

wire C;

// Instantiate the Unit Under Test (UUT)

FA uut (

.X(X),

.Y(Y),

.Z(Z),

.S(S),

.C(C)

);

initial begin

// Initialize Inputs

X = 0;

Y = 0;

Z = 0;

// Wait 100 ns for global reset to finish

#100;

X = 1;

Y = 0;

Z = 0;

#100;

X = 1;

Y = 1;

Z = 0;

// Add stimulus here end

endmodule

**Simulation Output:**



TASK#6:

* **Write a Verilog code for Full Adder along its stimulus module.**
* **Full adder Truth table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **carry** | **sum** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Program logic code:**

module Fulladder(

input a,

input b,

input cin,

output sum,

output carry);

assign {carry,sum}=a+b+cin;

endmodule

**Setting Timing of Inputs:**

module fulladdertiming;

// Inputs

reg a;

reg b;

reg cin;

// Outputs

wire sum;

wire cout;

initial begin

// Initialize Inputs

a = 0;

b = 0;

cin = 0;

// Wait 100 ns for global reset to finish

#100;

a =1;

b=0;

#50;

a=1;

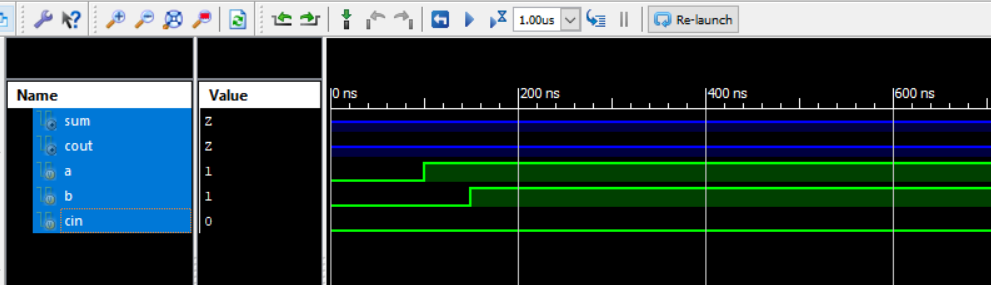
b=1;

// Add stimulus here

end

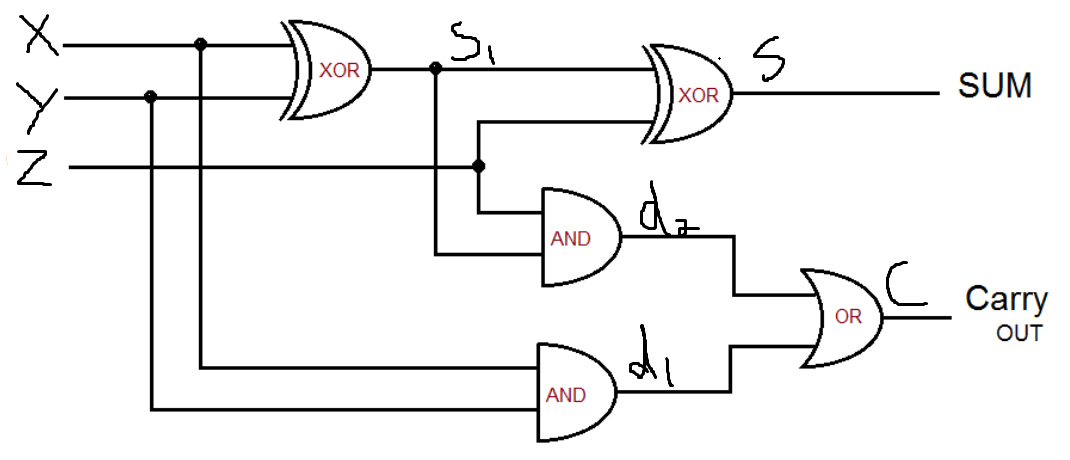
endmodule

**Simulation Output:**



**Gate flow level:**

**Program logic code:**



**Conclusion:**In this Lab (Half adder and full adder coding). We not only learn Different level of Verilog coding , and how to call previous modules In this Lab (Designing Mux Using Verilog HDL) We not only learn Different level of Verilog coding , and how to call previous modules but also We learn how to see simulation and Schematic of our coding but also We learn how to see simulation of our coding. And after performing all these tasks successfully I can say that I have learned these topics pretty much Alhamdulillah.