VLSI using Verilog HDL

Cetpa Infotech Pvt. Ltd

Sector 16, Noida

Ramandeep Kaur

Trainer

6 weeks training(from 11 June to 22 July, 2015)

Jimmy Rajpal(Trainee)

ECE

9912102188

Jaypee Institute of Information Technology

Sector 128,Noida

**CONTENT**

1. Acknowledgement
2. Introduction to VLSI
3. Introduction to Verilog and the Designing Tool used(Xilinx 14.1i)
4. Designing in Verilog HDL
5. Data types in Verilog HDL
6. Introduction Of Different Modelling Style
   1. Gate Level Modelling
   2. Dataflow Modelling
   3. Behavioural Modelling
   4. Switch Level Modelling
   5. Register Transfer Level Modelling
7. Experiments Performed
8. Conclusion
9. Appendices
10. **Acknowledgement**

I am grateful to Cetpa Infotech Pvt. Ltd, Noida for giving me opportunity to carry out the project work in the area of VLSI during my training. I would like to also thank my institute, Jaypee Institute of Information Technology, Noida for giving permission and necessary administrative support to take up the training work at the training institute, Noida.

First of all, I would like to thank respected, Mrs. Raman deep Kaur, (trainer) who gave me the opportunity to carry out this training under his guidance. He provided me with all the necessary, which were required for the completion of my training. I also thank my college library staffs for providing me with the books related to the training. A grateful acknowledgement goes to my project mates, who helped me with timely suggestions during the entire period of references and data collections.

**Noida**

**JUNE 2015 Jimmy Rajpal**

1. **Introduction To VLSI**

**Very-large-scale integration** (**VLSI**): is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

**Levels of integration of Integrated Circuit(IC)**

1. SSI: Small Scale Integration :: Number of Transistors: less than 10
2. MSI: Medium Scale Integration:: Number of Transistors: between 10 to 100
3. LSI: Large Scale Integration :: Number of Transistors: approximately 10,000
4. VLSI: Very Large Scale Integration: : Number of Transistors: around one million
5. ULSI: Ultra Large Scale Integration :: Number of Transistors: much larger than one million

Classification of VLSI

Front End

Back End

**Front End**: considers all the logical designing and verificationpart, In simple words we can say all the work up to the Gate level or RTL Level designing and verification  considered as VLSI Front End Designing and Verification, We have multiple ways for logical designing of IC (Integrated Circuits)’s in VLSI Front End, For Example in early days when we have very less chip complexity, designers used Manual Logical Designing and they used concept of number system, Basic Gates Concept, K-maps, Boolean Mathematics, expression handling etc.

**Back End**: consider all the designing and verification part after logical designing means Gate level or RTL level designing, That may include Floor Planning, Place & Route, and All the foundry work like fabrication, packaging etc.

1. **Introduction To Verilog**

**Hardware Description Language** (**HDL**): is a specialized computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital circuits. A enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of an HDL program into a lower level specification of physical electronic components, such as the set of masks used to create an integrated circuit.

**Design by HDL**

1. ABEL: advanced Boolean expression logic
2. AHDL: Altera hardware description language
3. VHDL: VHSIC(Very High speed integrated circuit) Hardware description language
4. Verilog HDL: Verifying Logic Hardware description language
5. System Verilog: extension of Verilog HDL and OOPS concepts
6. **Designing In Verilog HDL**

* **Design Methodology**
  + **Top-Down Methodology**: The traditional method of electronic design is bottom-up. Each design is performed at the gate-level using the standard gates (refer to the Digital Section for more details). With the increasing complexity of new designs this approach is nearly impossible to maintain. New systems consist of ASIC or microprocessors with a complexity of thousands of transistors. These traditional bottom-up designs have to give way to new structural, hierarchical design methods. Without these new practices it would be impossible to handle the new complexity.
  + **Bottom-up Methodology**: The desired design-style of all designers is the top-down one. A real top-down design allows early testing, easy change of different technologies, a structured system design and offers many other advantages. But it is very difficult to follow a pure top-down design. Due to this fact most designs are a mix of both methods, implementing some key elements of both design styles.
* **Modules**: Verilog provides the concept of *module.* A module is the basic building block in Verilog. A module can be an element or a collection of lower-level design blocks. Typically, elements are grouped into modules to provide common functionality that is used at many places in design. A module provides the necessary functionality to the higher-level block through its port interface (inputs and outputs), but hides the internal implementation. This allows the designers to modify module internals without affecting the rest of the design.

**Syntax:-**

module <module\_name> (<module\_terminal\_list>);

….

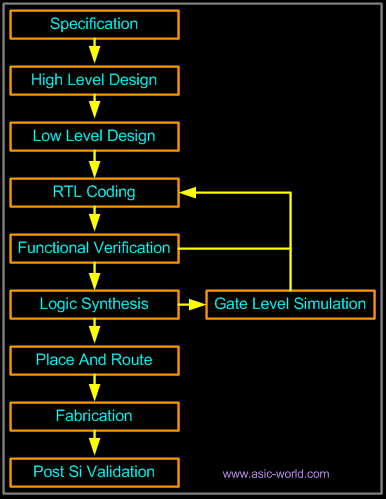
<module internals>

….

….

endmodule.

* **Verilog HDL Design Flow**



1. **Data Types In Verilog Hdl**

* **Data Values:**
  + 0::low or false
  + 1:: high or true
  + X:: unknown
  + Z:: high impedance state
* **Description of Data types**
  + **Net:** The net data types represent physical connections between structural entities, such as gates. A net does not store a value (except for the trireg net). Instead, it must be driven by a driver, such as a gate or a continuous assignment.
  + **Register**: A register is an abstraction of a data storage element. The keyword for the register data type is reg. A register stores a value from one assignment to the next. An assignment statement in a procedure acts as a trigger that changes the value in the data storage element. The Verilog language has powerful constructs that allow you to control when and if these assignment statements are executed. These control constructs are used to describe hardware trigger conditions, such as the rising edge of a clock, and decision-making logic, such as a multiplexer.
* **Data Representation**
  + **Scalar Data Representation**: A net or reg declaration without a specification is one bit wide; that is, it is scalar.
  + **Vector Data Representation**: Multiple bit net and reg data types are declared by specifying [size]identity, and are known as vectors.
* **Array Description:** Arrays are used to hold several objects of the same type constituting collection of vectors or scalars. for this, only reg identifier is used. It is represented as [size]identity[dimension].

1. **Modelling Styles Of Verilog**

Verilog is both a behavioral and a structural language. Verilog provides four levels of modeling. These levels of modeling can be defined as follows:

* 1. **Gate level Modelling**:
     + It describes elements in terms of basic primitive transistors i.e. predefined gates such as XOR, NAND, NOR, AND, OR, NOT etc , a total of 8 gates. If we have more than one gate in same program then they will execute at same time. This modeling is concurrent in nature.

Eg: xor a1(Y,A,B)

module fa\_1bit ( x, y, ci, s, co );

input x, y, ci;

output s, co;

wire im1, im2, im3;

xor ( s, x, y, ci );

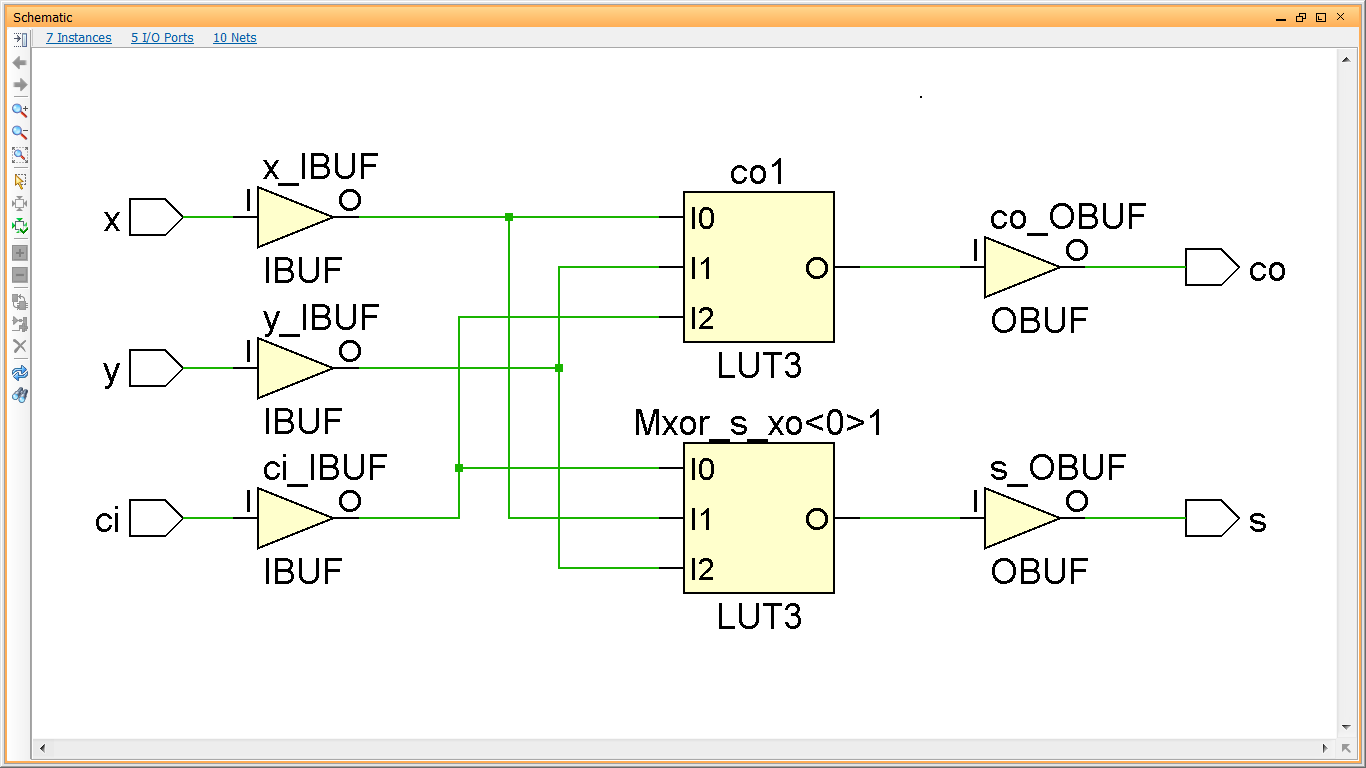
and ( im1, x, y ),

( im2, y, ci ),

( im3, ci, x );

or ( co, im1, im2, im3 );

endmodule



* 1. **Dataflow modeling:**
* It describes in terms of Boolean expression and it is concurrent in nature for Boolean expression in same program.
* **Continuous Assignment statement:** A continuous assignment in Verilog is used only in concurrent Verilog bodies. This assignment represents a net driven by a gate output or a logic function. In its simplest form, a continuous assignment begins with the ‘assign’ keyword, followed by the left-hand side net type variable, an equal sign, and a right-hand side expression.

Eg : Y=A+B;

* **Verilog operators:** Boolean operations are the most common type of operations for describing functions of hardware components at the gate or even RT level. In addition, there are operations for the behavioral or functional description of hardware. Most operations found in software languages, are also supported in Verilog.
  + **Arithmetic Operators:-**

+,-,\*,/,% (the modulus operator)

* + **Relational Operators:-**

<,>,<=(less than equal to),>=(greater than equal to)

* + **Equality Operators:-**

===,!==,==,!=

* + **Logical Operators:-**

!(logic negation),&&(logical and),||(logical or)

* + **Bitwise Operators:-**

~,&,|,^,^~,~^

* + **Reduction Operators:-**

&,~&,|,~|,^,^~or ~^

* + **Shift Operators:-**

<<,>>

* + **Concatenation Operators:-**

{ and }

* + **Conditional Operators:-**

Cond\_expr? True\_expr : false\_expr.

* 1. **Behavioral Modeling:** It describes in terms of behavior or truth table so it is best preferred when we can’t use gate level circuits such as in sequential circuits like UART(Universal Asynchronous Receiver Transmitter)
     + **Structured Procedural Statements**:
  + **Always Statements**: the modules containing ‘always’ statements are executed again and again. They are controlled by event controlled statements.
  + **Initial Statements:** the modules containing ‘initial’ statements are executed only once and are used for initialization. So calling them again in procedural statements is not allowed.

**Eg: Reference module for clock generation:**

**Initial** begin

Clk=0;

End

**Always** begin

Clk=~clk;

#17;

end

* + **Assign statement:** Verilog uses ‘assign’ statement for assigning a Boolean expression to a wire or vector of wires(nets).

module Mux2to1 (input a, b, c, output w);

wire n, m, p;

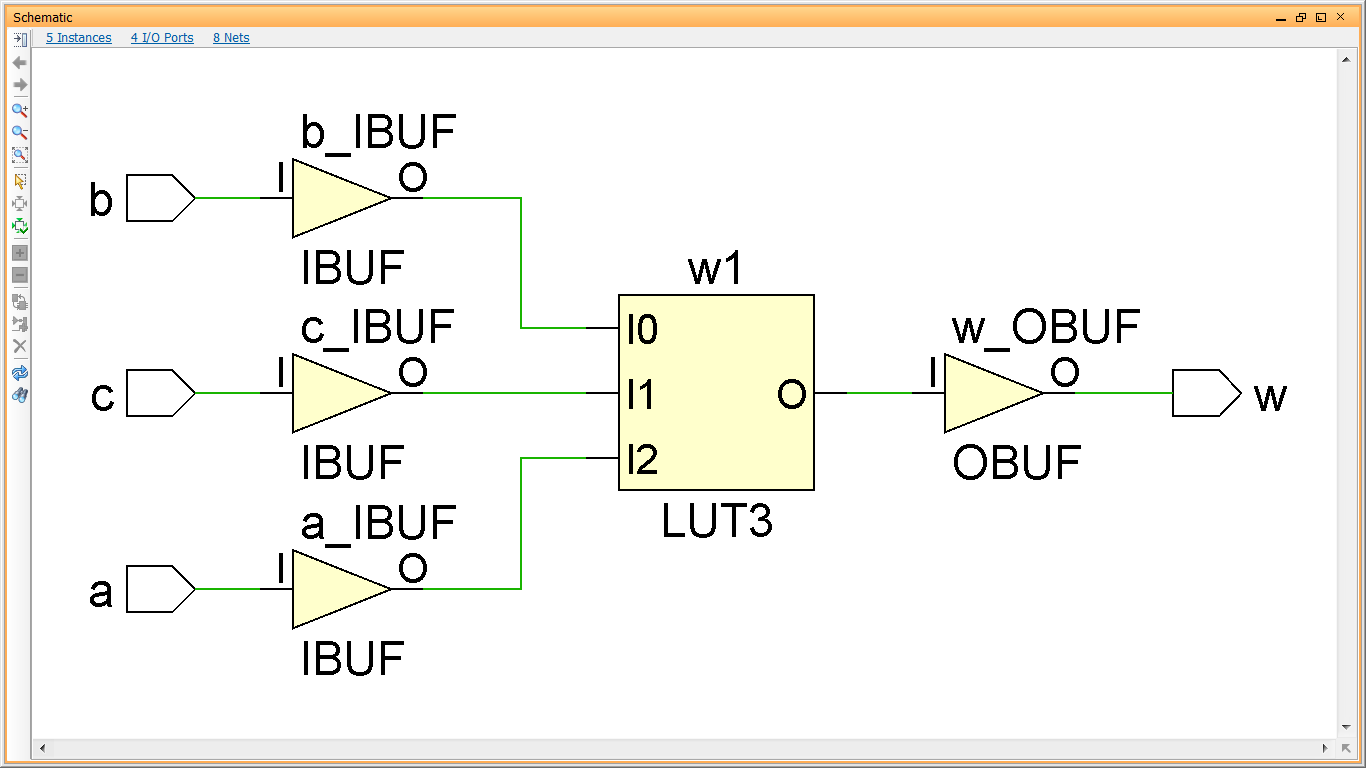
assign #3 m = a & b;

assign #3 p = n & c;

assign #6 n = ~b;

assign #2 w = m | p;

endmodule



* **Loops:** Verilog has four looping statements like any other programming language.
  + **Forever :** forever loop executes continually
    - syntax: **forever <**statement**>**
  + **Repeat**: Executes for a fixed number of time.
    - Syntax: repeat (<number>)<statement>
  + **While**: Executes as long as an expression evaluates true.
    - Syntax: while (<expression>)<statement>
  + **For**: It is same as used in other programming languages.
    - Syntax: for (<initial assignment>;<exprssion>,<step assignment>)<statement>
* **Block Statements**
  + **Parallel block**
  + **Sequential block**
  1. **Switch level Modeling**: It describes in terms of basic primitive transistors i.e. in CMOS, PMOS, NMOS etc. This is the most complex designing model because we have to write in transistor form. It has not synthesizable delay.
  2. **Register Transfer Level:** This is an additional level which is frequently used in Verilog. Designs using the register-transfer level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing possibility; operations are scheduled to occur at certain times. Modern definition of a RTL code is “Any code that is synthesizable is called RTL code”.

1. **Experiments practiced**

**Experiment-1: Half adder**

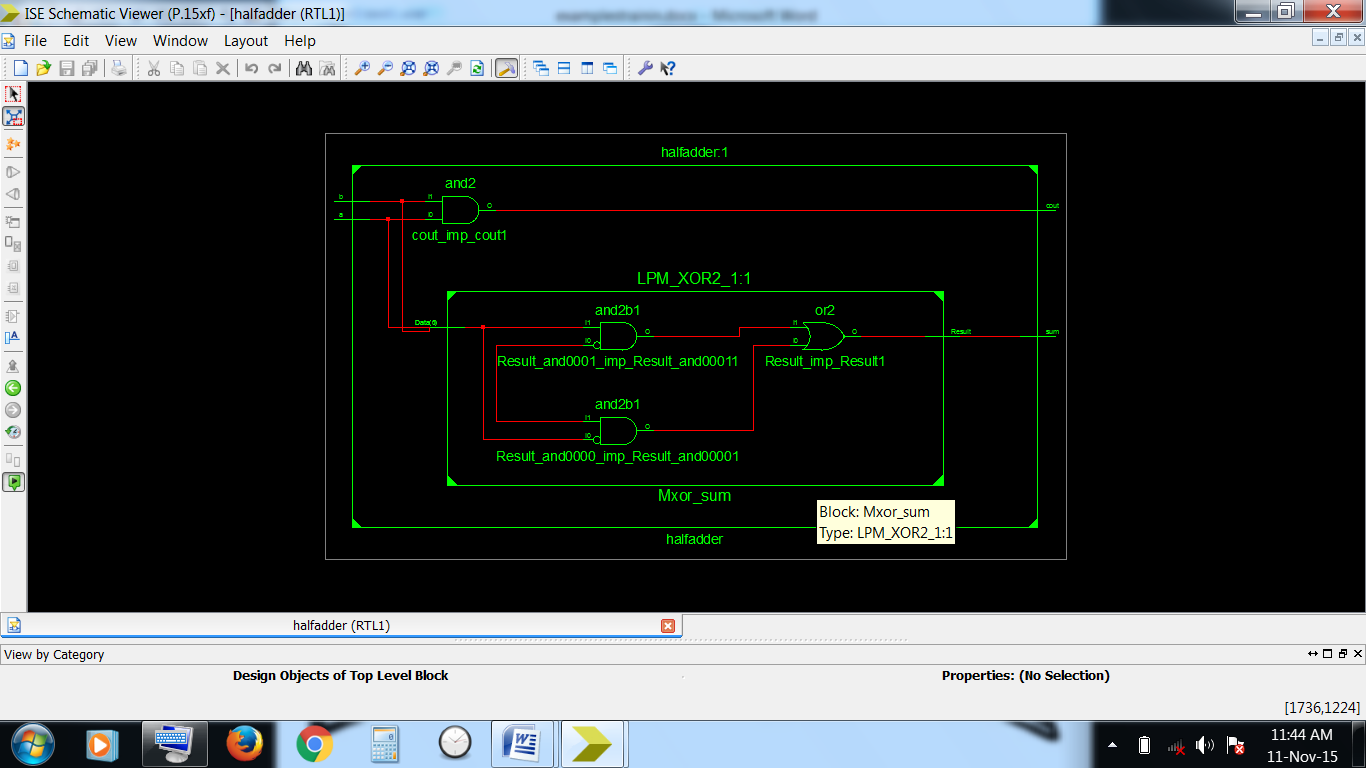
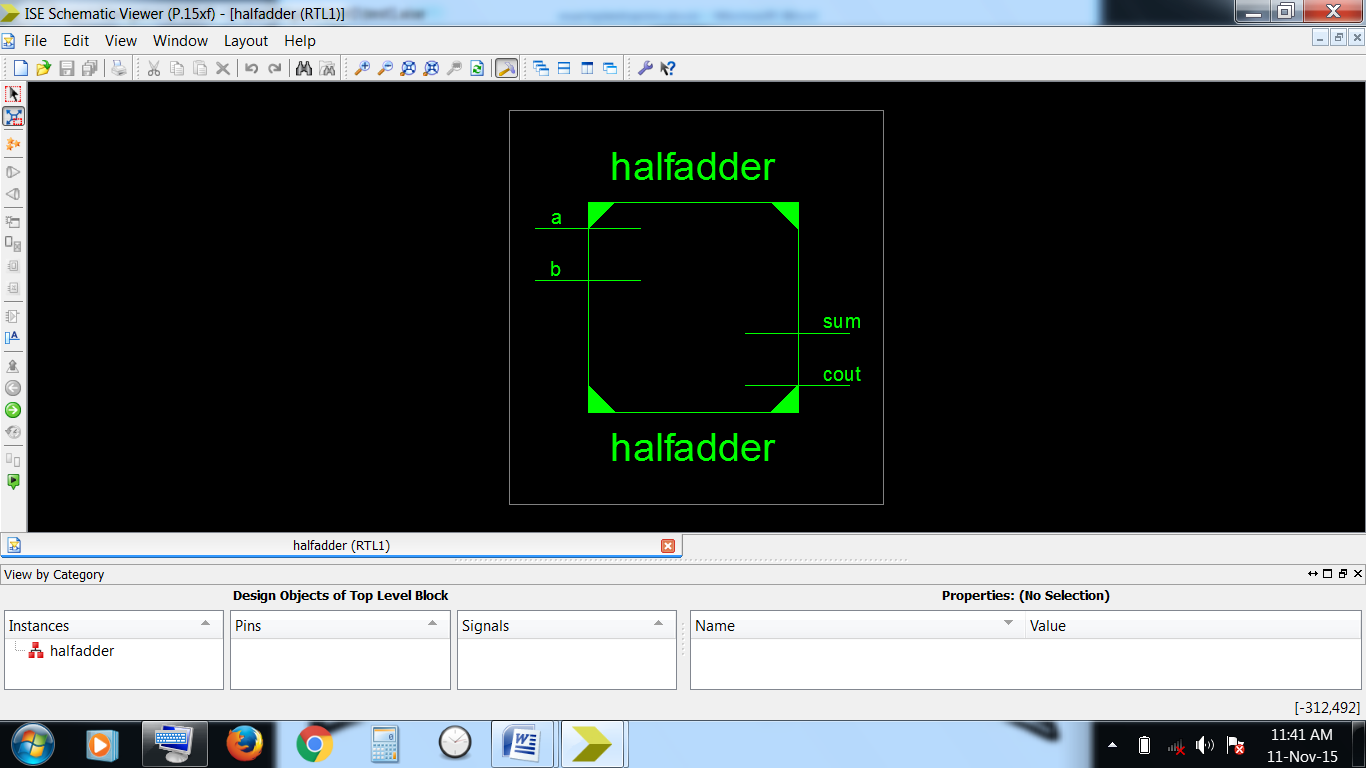
**Code:**

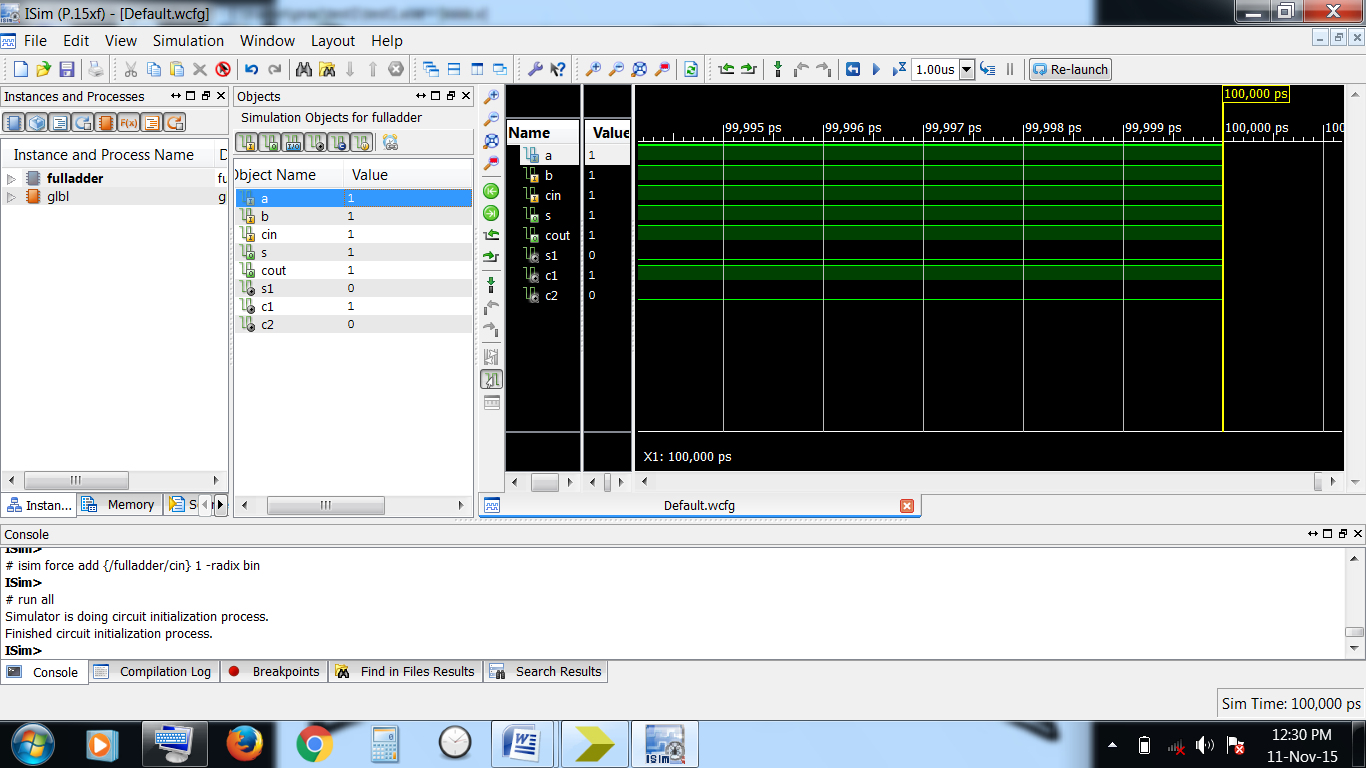
Module halfadder(input a,b, output sum,cout);

xor a1(sum,a,b);

and a2(cout,a,b);

endmodule





**Experiment-2: Full adder**

module fulladder( input a,b,cin, output s,cout);

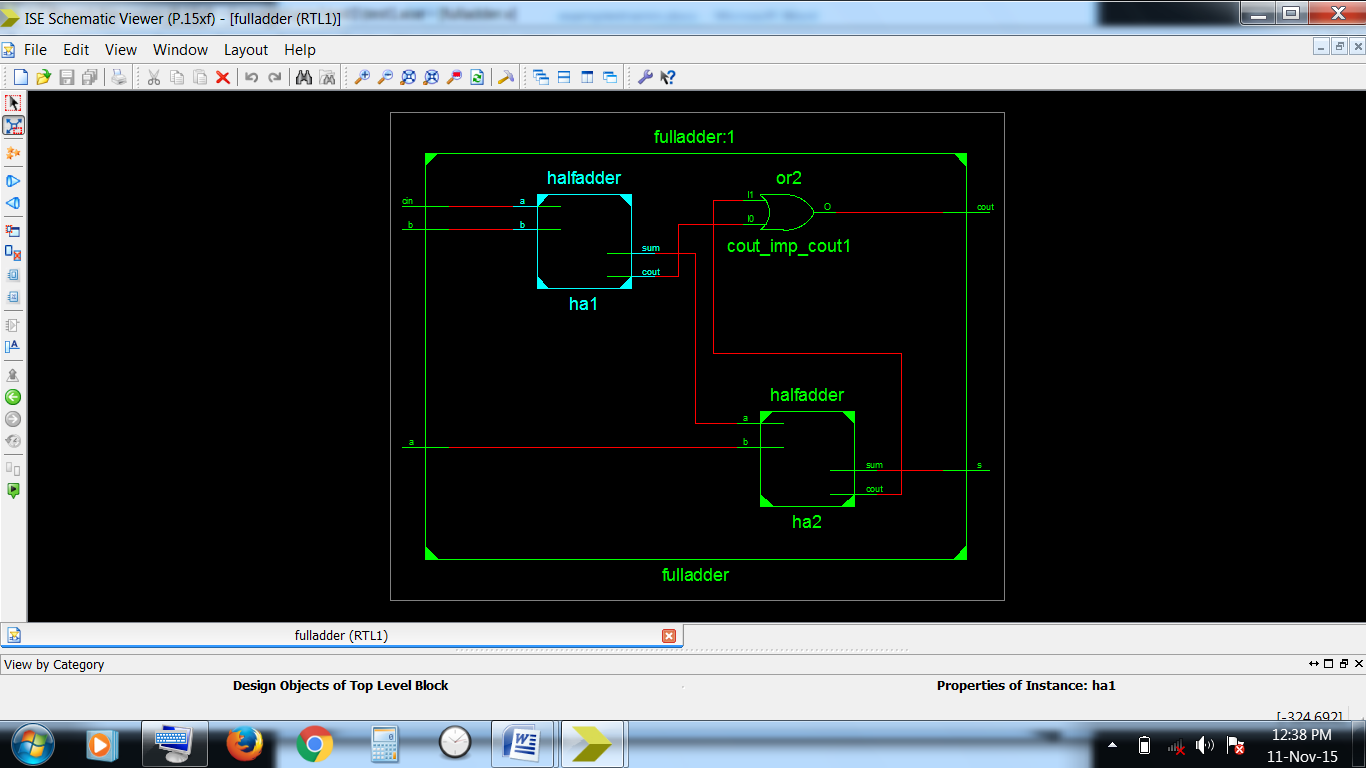
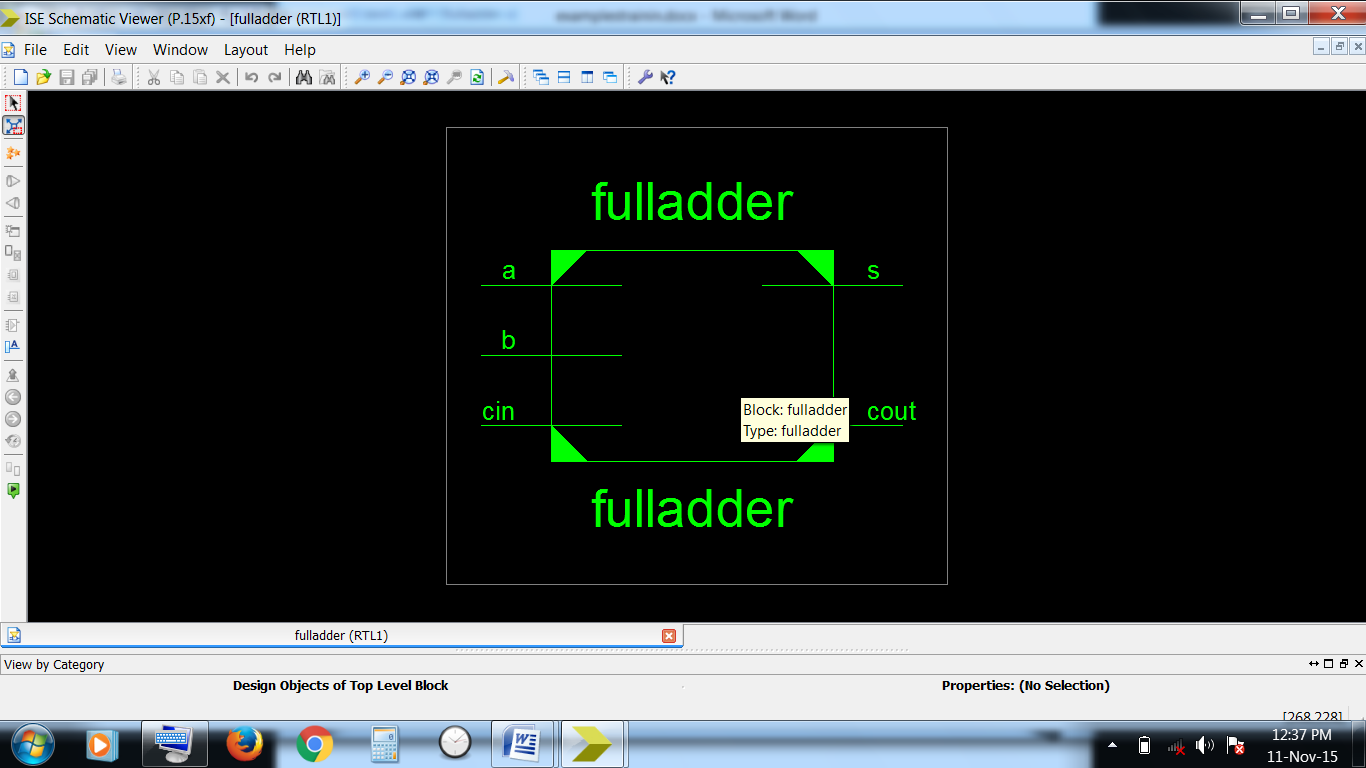
wire s1,c1,c2;

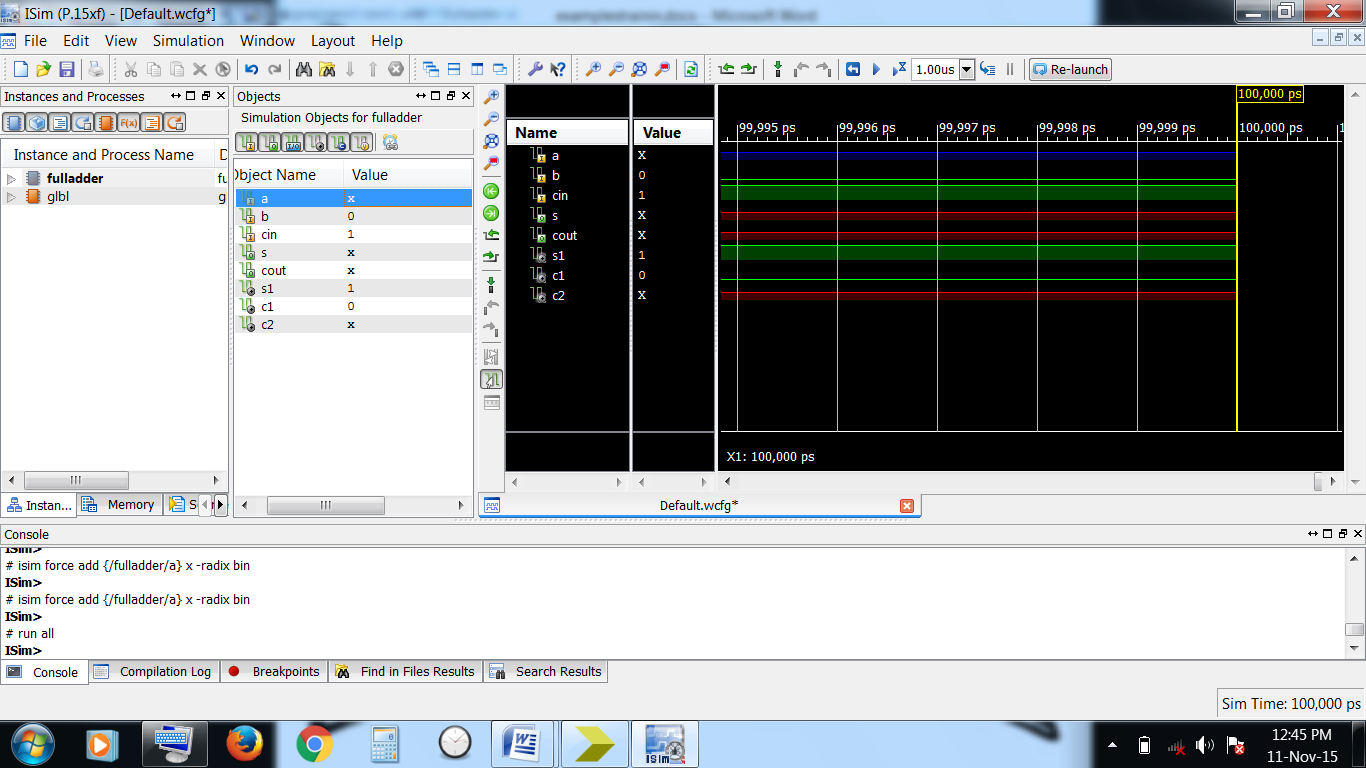
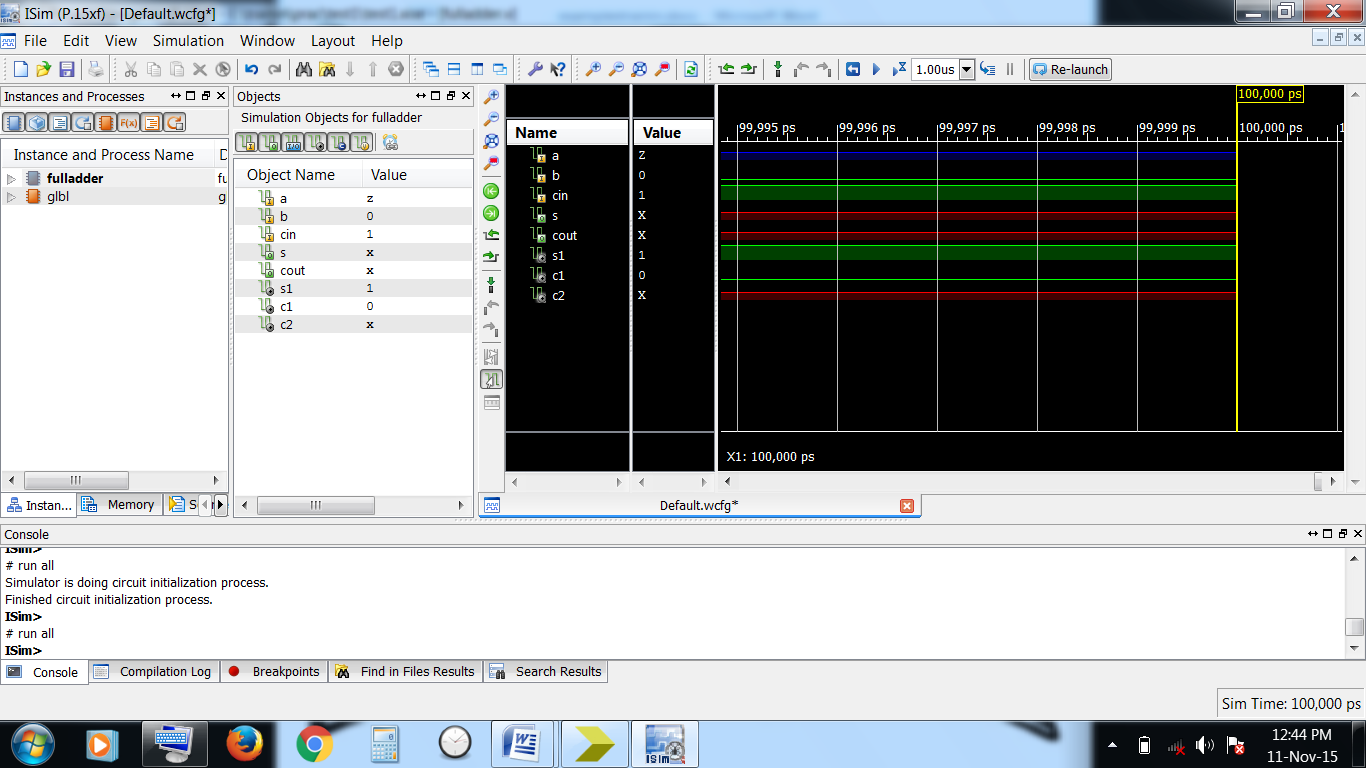
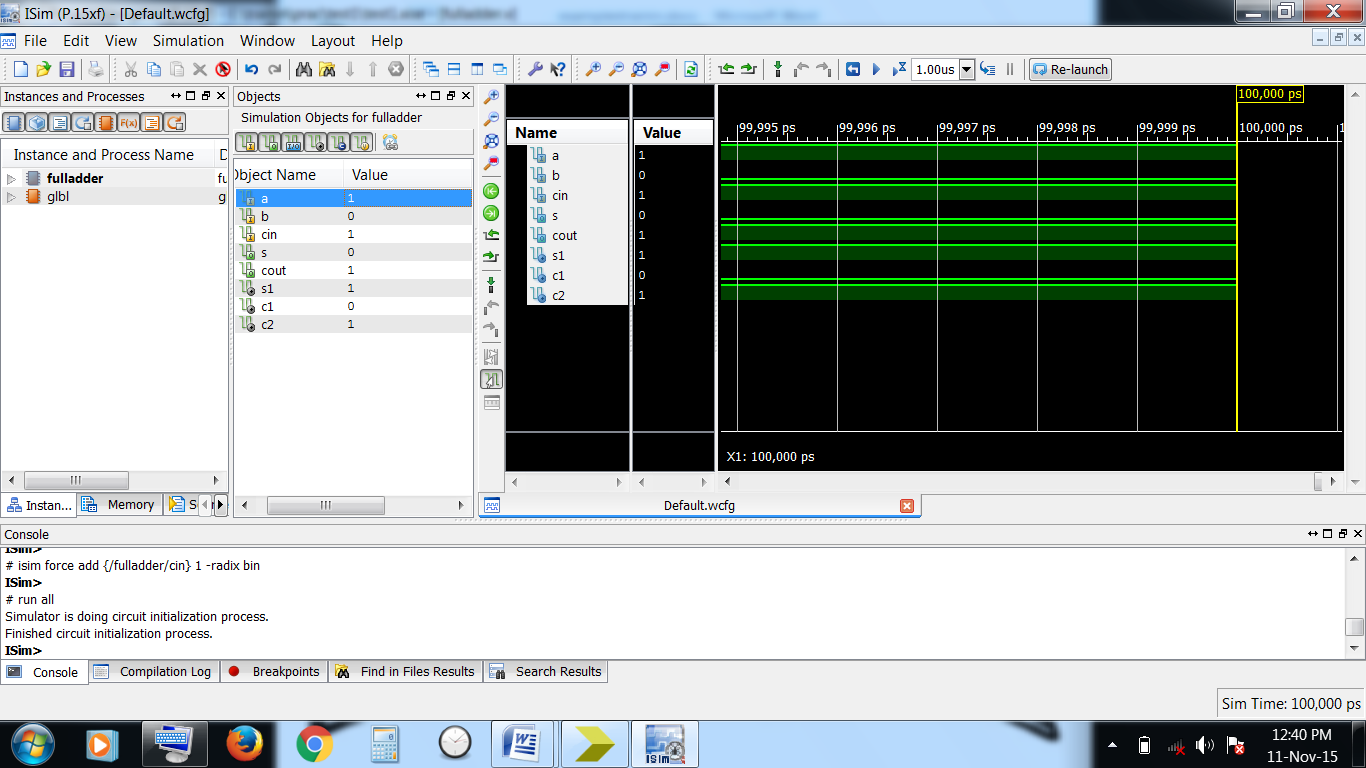
halfadder ha1(cin,b,s1,c1);

halfadder ha2(s1,a,s,c2);

or or1(cout,c1,c2);

endmodule





**Experiment-3:2 to 1 multiplexer**

**Code:**

module Mux2to1 (input a, b, c, output w);

wire n, m, p;

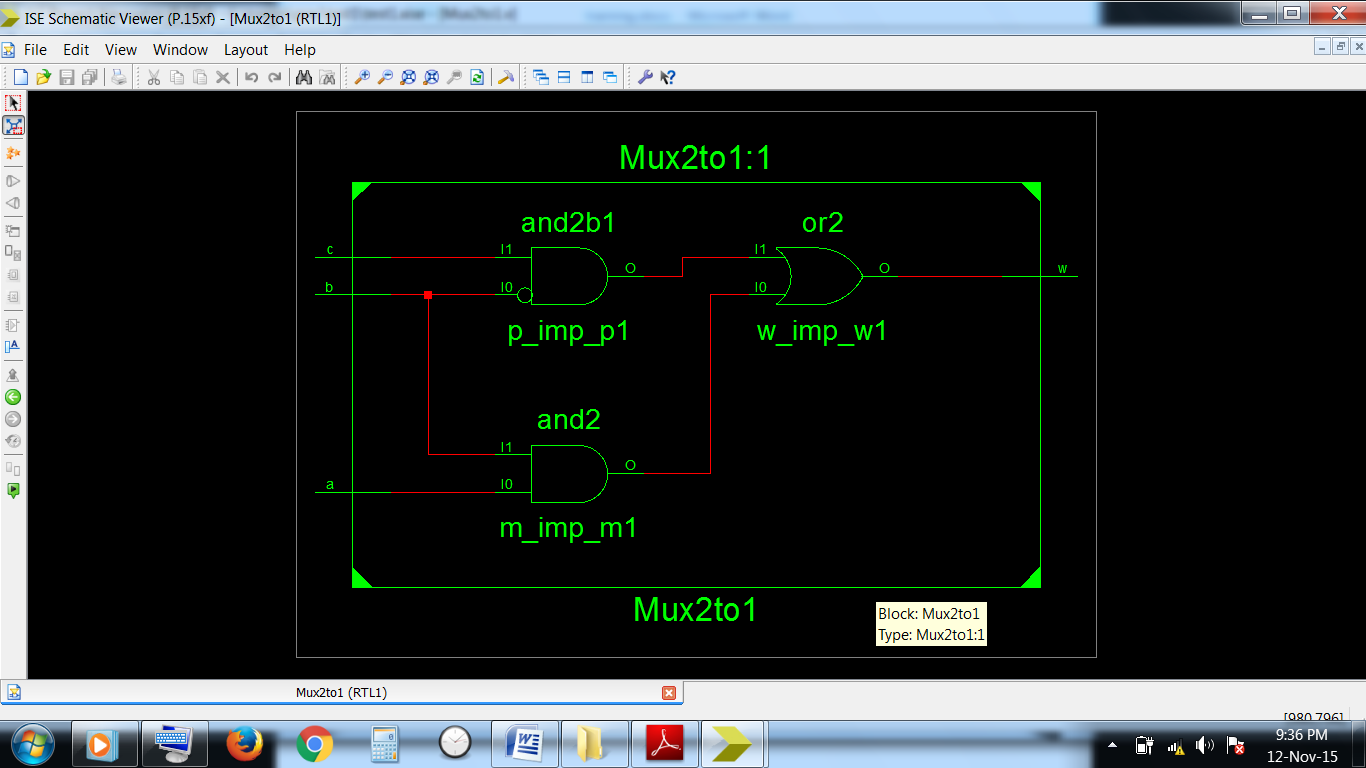
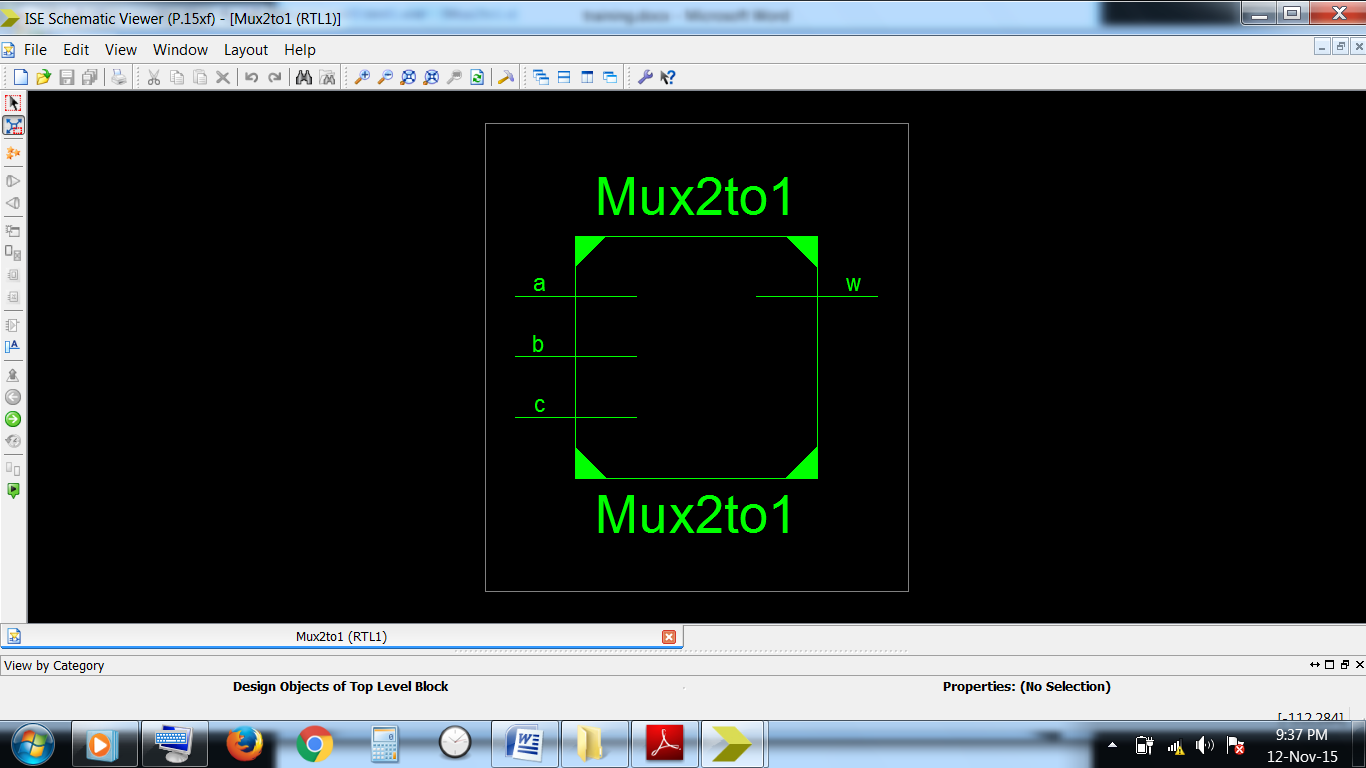
assign #3 m = a & b;

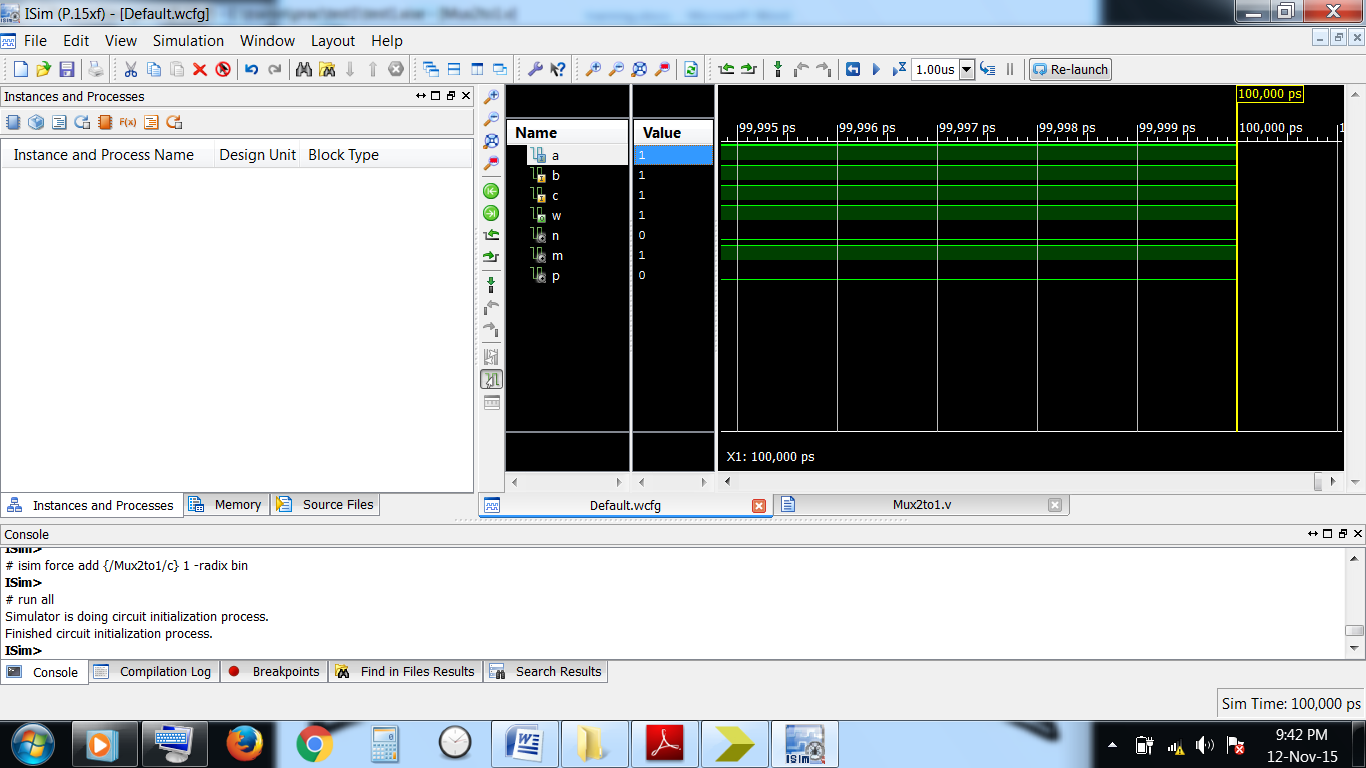
assign #3 p = n & c;

assign #6 n = ~b;

assign #2 w = m | p;

endmodule





**Experiment-4:D flip flop**

**Code:**

module dff (input clk, set, rst, d, output reg q);

always @(posedge clk or posedge set or posedge rst)

if( set )

assign q = 1'b1;

else if( rst )

assign q = 1'b0;

else

assign q = d;

endmodule

module structural\_lfsr #(parameter [3:0] seed=4'b0)

(input clk, init, sin, output sout);

wire im1, im2, im3, im4, im5;

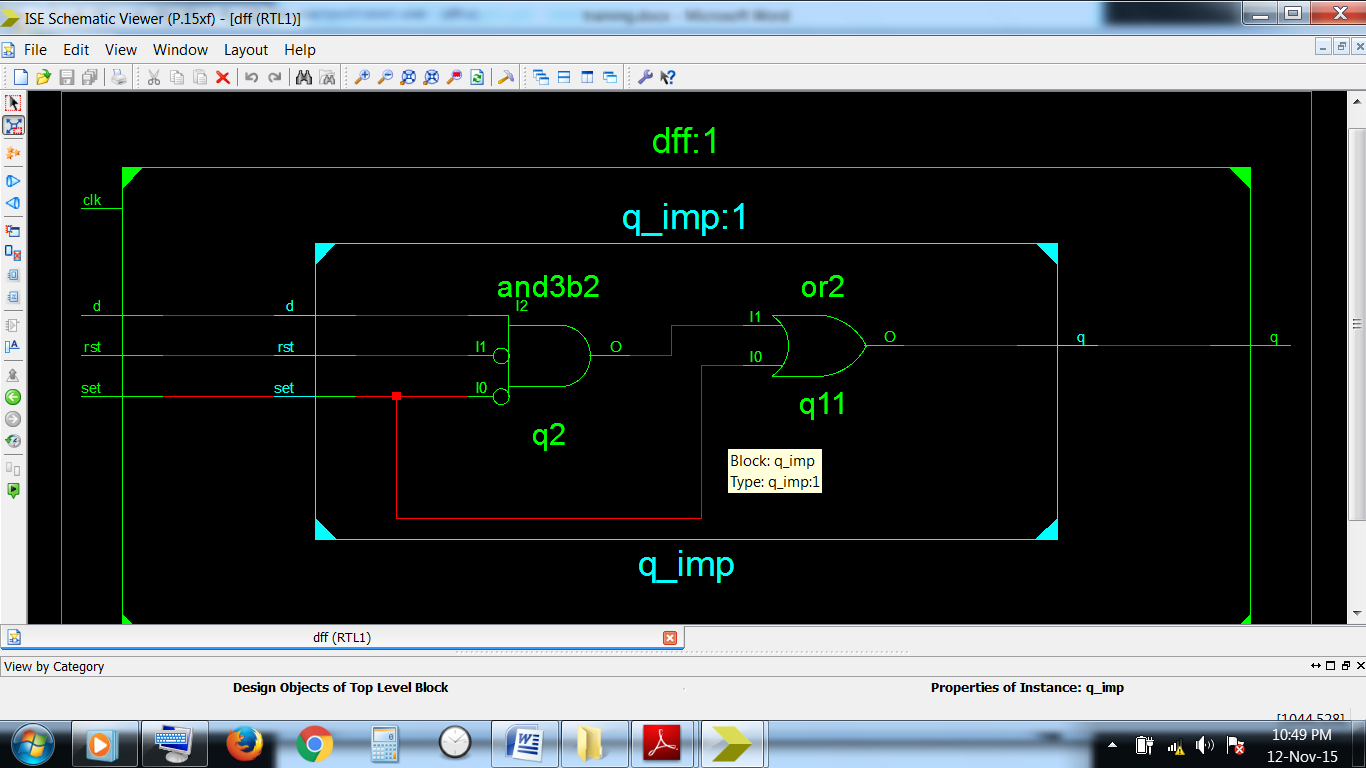
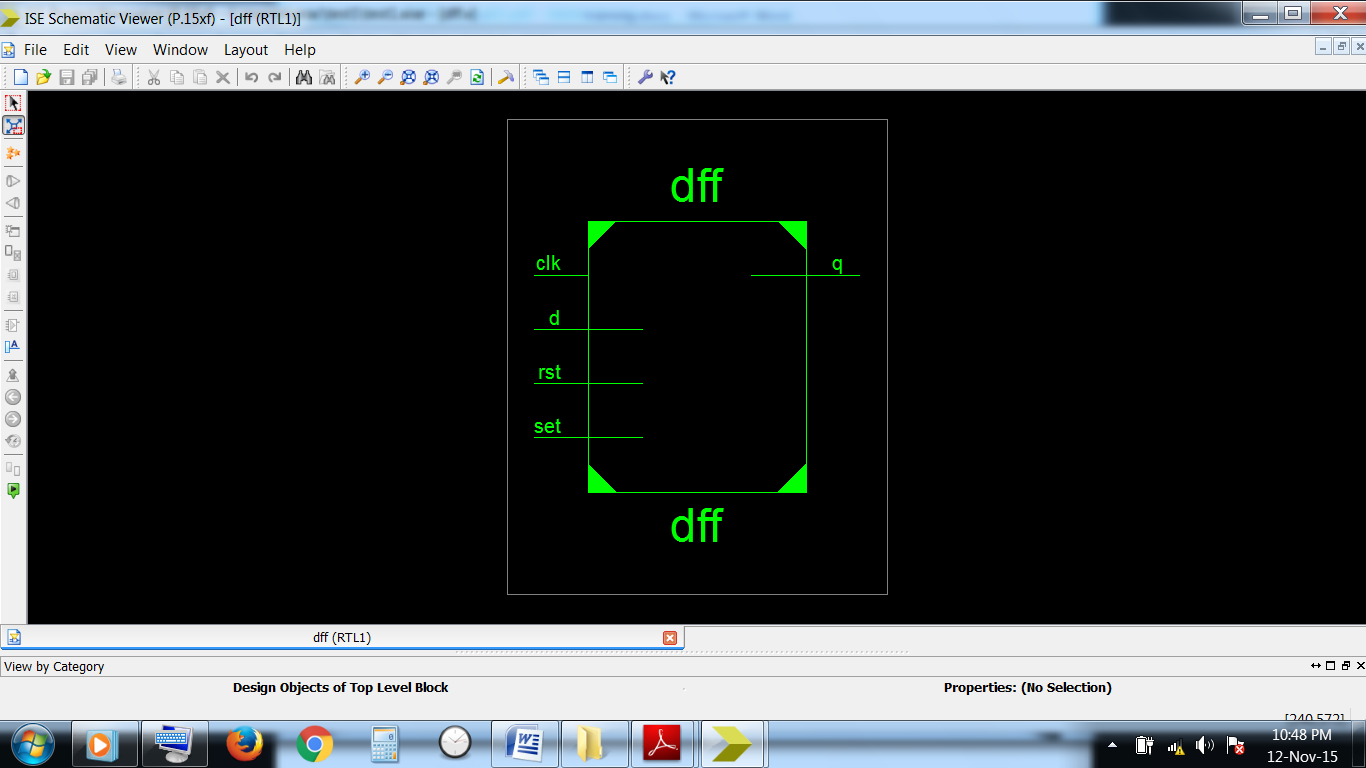
dff ff[3:0] ( clk, {4{init}}&seed, {4{init}}&~seed,

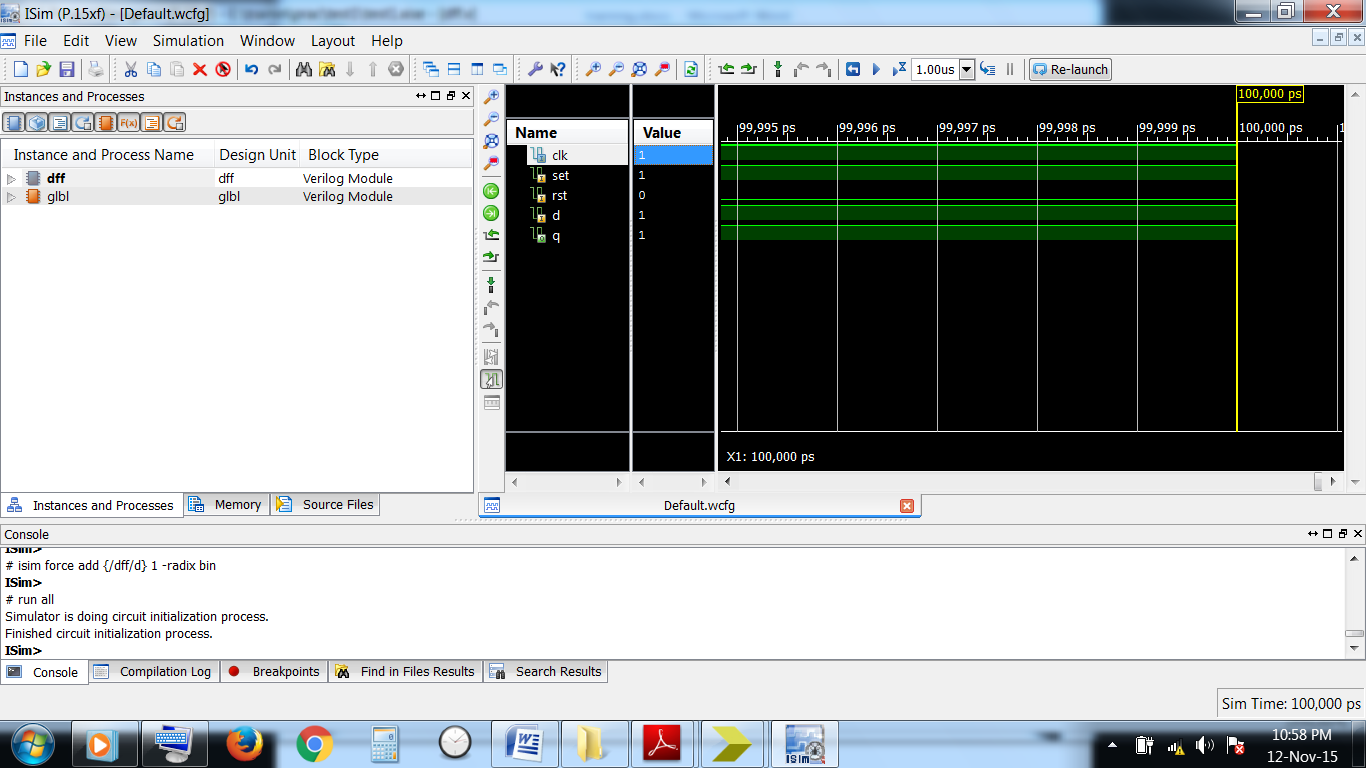
{im1,im2,im4,im5}, {im2,im3,im5,sout} );

xor ( im1, sin, sout );

xor ( im4, im3, sout );

endmodule





1. **Conclusion**

It can be concluded from above that Verilog HDL is a language which is user-friendly and economic. Verilog provides designers to put their own designs and can easily simulate and synthesize the designs and then can implement it. Verilog HDL is a hardware description language which provided the RTL description of the designs for the digital designs.

There are less chances of faults in the design which are designed using Verilog. So it helped in reducing faults also.

Overall, Verilog HDL is a very useful language used for hardware designing.

1. **Appendices**

**Xilinx ISE**

It is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while the ModelSim logic simulator is used for system-level testing. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and ChipScope Pro.