Experiment No 11

Implementation of Combinational Logic using Verilog HDL

11.1 Objectives:

After completing this experiment, student will be able to:

- Creation and compilation of a Verilog project in ModelSim
- Writing of Design Module & Stimulus Module in Verilog
- Result display and timing waveform generation in ModelSim
- Writing Verilog Modules for Different Combinational Circuits

11.2 Background Theory

Hardware description languages (HDLs) are used to design, simulate and verify complex VLSI circuits. HDLs support multiple levels of abstraction and can simulate concurrency found in hardware circuits. Logic synthesis tools are used to convert the RTL design in HDL into gate level circuit schematics Designers can verify functionality of their design at a higher abstraction level before going into technology specific details. There are two main Hardware description Languages

- Verilog HDL
- VHDL

Verilog is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits. Hardware description languages such as Verilog are similar to software programming languages because they include ways of describing the propagation time and signal strengths (sensitivity).

11.3 Components of Verilog Simulator

Verilog is a case sensitive language. Verilog keywords are in lower case letters. Some important syntax to be kept in mind while doing Verilog programming.

- Identifiers (Names given to Modules)
- Upper and lower case letters from the alphabet
- Digits (0, 1, ..., 9), underscore (_)
- Max length of 1024 symbols
- $ Symbol (only for system tasks)
- Terminate lines with semicolon ‘;’
- Single line comments
- // A single-line comment goes here
- Multi-line comments – /* Multi-line comments*/

Verilog Design has two main blocks as shown in figure 11.1

- Design Block
- Stimulus Block (Test Bench)

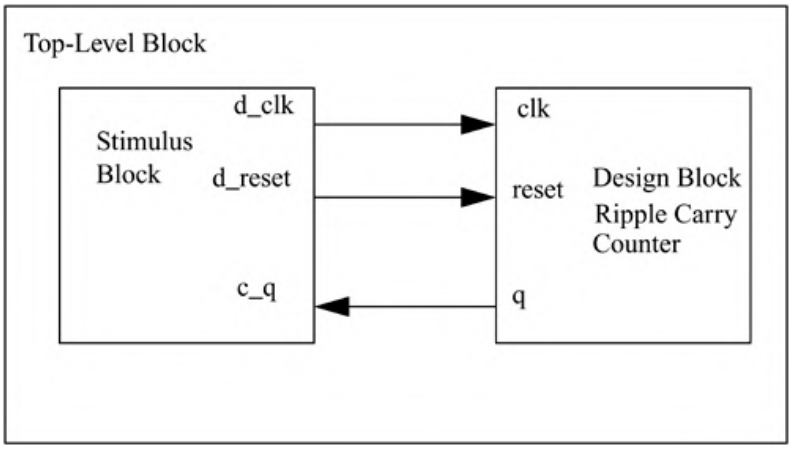

Figure 11.1: Stimulus and Design Blocks Instantiated in a Dummy Top-Level Module

11.3.1 Design Block

Design block is the actual design in the form of Verilog module with all its instantiated modules. Modules are the basic building blocks in Verilog. Modules are used in a design by instantiation. An instance of a module has a unique identity and is different from other instances of the same module. Each instance has an independent copy of the internals of the module. It is important to understand the difference between modules and instances. Ports provide the interface by which a module can communicate with its environment. For example, the input/output pins of an IC chip are its ports. The environment can interact with the module only through its ports. The internals of the module are not visible to the environment. This provides a very powerful flexibility to the designer. The internals of the module can be changed without affecting the environment as long as the interface is not modified. Ports are also referred to as terminals.

Table 5 Port Declaration

Verilog Keyword	Types of Port
Input	Input Port
Output	Output Port
Inout	Bidirectional Port

Each port in the port list is defined as input, output, or inout, based on the direction of the port signal.

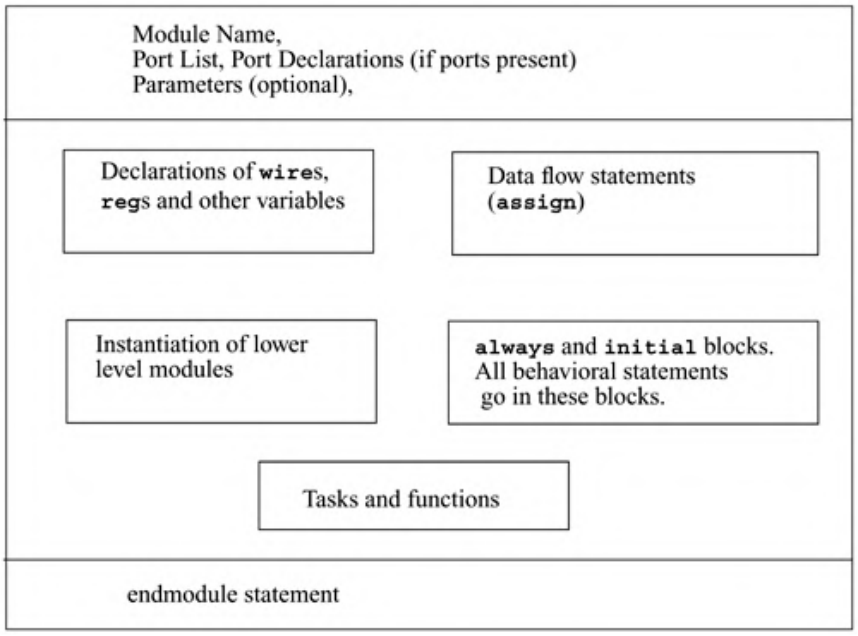

Figure 11.2: Components of a Verilog Module

A module definition always begins with the keyword module. The module name, port list, port declarations, and optional parameters must come first in a module definition. Port list and port declarations are present only if the module has any ports to interact with the external environment. The five components within a module are: variable declarations, dataflow statements, instantiation of lower modules, behavioral blocks, and tasks or functions. These components can be in any order and at any place in the module definition. The endmodule statement must always come last in a module definition. All components except module, module name, and endmodule are optional and can be mixed and matched as per design needs. Verilog allows multiple modules to be defined in a single file. The modules can be defined in any order in the file.

11.3.2 Stimulus Block (Test Bench)

Once a design block is completed, it must be tested. The functionality of the design block can be tested by applying stimulus and checking results. We call such a block the stimulus block. It is good practice to keep the stimulus and design blocks separate. The stimulus block can be written in Verilog. A separate language is not required to describe stimulus. The stimulus block is also commonly called a test bench. Different test benches can be used to thoroughly test the design block. Stimulus block is used for testing of Design block, also called Test bench. Stimulus module instantiates Design module, provides inputs and collects outputs. Stimulus module and Design module should be kept separate. Stimulus module is never part of RTL design.

11.3 Gate Level Primitive

At gate level, the circuit is described in terms of gates (e.g., and, nand). Hardware design at this level is intuitive for a user with a basic knowledge of digital logic design because it is possible to see a one-to-one correspondence between the logic circuit diagram and the Verilog

description. A logic circuit can be designed by use of logic gates. Verilog supports basic logic gates as predefined primitives. These primitives are instantiated like modules except that they are predefined in Verilog and do not need a module definition. All logic circuits can be designed by using basic gates. These gates are instantiated to build logic circuits in Verilog. Examples of gate instantiations are shown below. More than two inputs can be specified in a gate instantiation. Gates with more than two inputs are instantiated by simply adding more input ports in the gate instantiation

Table 11.1: Gate instantiations

Gate	Parameters	Instantiation
And	OUT, IN1, IN2	a1(OUT, IN1, IN2);
Nand	OUT, IN1, IN2	na1(OUT, IN1, IN2);
Or	OUT, IN1, IN2	or1(OUT, IN1, IN2);
Nor	OUT, IN1, IN2	or1(OUT, IN1, IN2);
Xor	OUT, IN1, IN2	x1(OUT, IN1, IN2);
Xnor	OUT, IN1, IN2	nx1(OUT, IN1, IN2);
Nand	OUT, IN1, IN2, IN3	na1_3inp(OUT, IN1, IN2, IN3);

11.3.1 Verilog Operators

Operators are of three types: unary, binary, and ternary. Unary operators precede the operand. Binary operators appear between two operands. Ternary operators have two separate operators that separate three operands as shown below.

```
a = ~ b; // ~ is a unary operator. b is the operand
a = b && c; // && is a binary operator. b and c are operands
a = b ? c : d; // ?: is a ternary operator. b, c and d are operands
```

Almost all operators in Verilog are allowed for logic synthesis. Table 11.1 is a list of the operators allowed. Only operators such as === and !== that are related to x and z are not allowed, because equality with x and z does not have much meaning in logic synthesis. While writing expressions, it is recommended that you use parentheses to group logic the way you want it to appear. If you rely on operator precedence, logic synthesis tools might produce an undesirable logic structure.

Table 11.2 Verilog HDL Operators for Logic Synthesis

Operator Type	Operator Symbol	Operation Performed
Arithmetic	* / + - % + -	Multiply divide add subtract modulus unary plus unary minus
Logical	! && ||	logical negation logical and logical or
Relational	> < >= <=	greater than less than greater than or equal less than or equal

Equality	== !=	Equality inequality
Bit-wise	~ & | ^ ^~ or ~^	bitwise negation bitwise and bitwise or bitwise ex-or bitwise ex-nor
Reduction	& ~& | ~| ^ ^~ or ~^	reduction and reduction nand reduction or reduction nor reduction ex-or reduction ex-nor
Shift	>> << >>>	right shift left shift arithmetic right shift

11.4 Simulation Tool-ModelSim

ModelSim is a multi-language environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Intel Quartus Prime, PSIM, Xilinx ISE or Xilinx Vivado. Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

11.4.1 Project Creation in ModelSim

Below are the steps for creation of a new project in ModelSim.

- **Invoke ModelSim**
  Invoke ModelSim from Desktop by double clicking on the ModelSim icon. (Or from start→programs→ …menu). Close '**Welcome to ModelSim 5.7g**' dialog box if it appears. Main ModelSim Window is composed of Workspace window & Output window. Other windows can be opened from 'View' Menu. Main ModelSim 5.7g window is shown below:

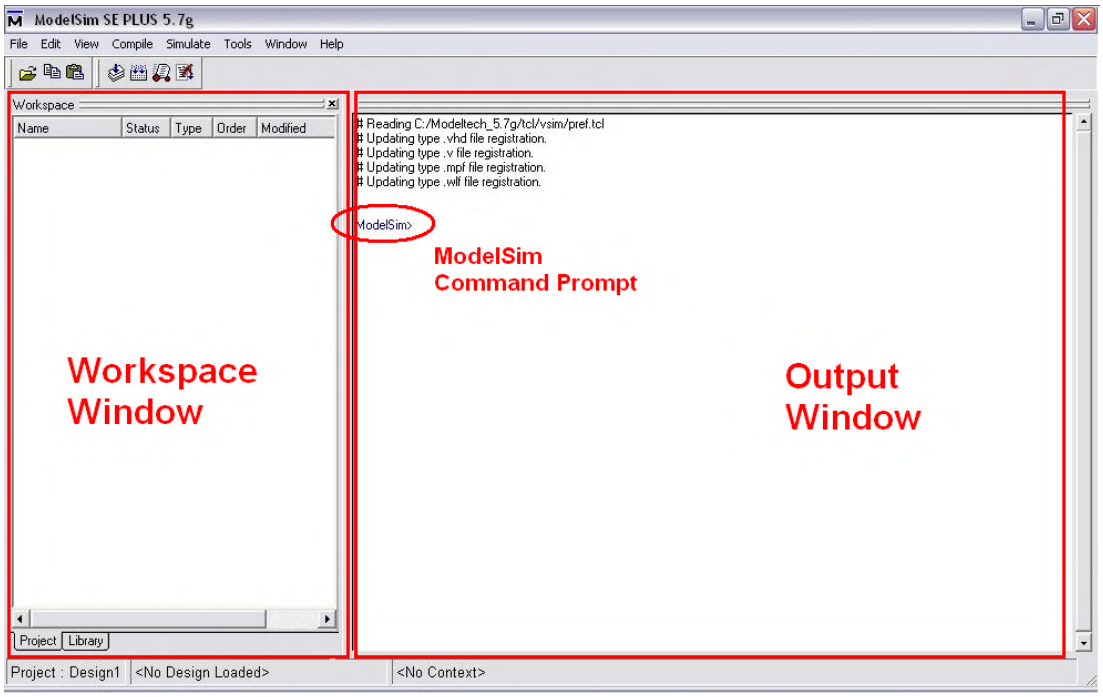

Figure 11.3: Main ModelSim Window

- **Create a ModelSim Project**

Steps for creating a new Verilog Project in ModelSim are listed below:

1. Create a new folder with name 'Design1' at your desired location
2. Click on File→ New→Project on the drop down menu.
3. You will see Dialog Box 'Create Project' (Figure 11.4). Enter project name; Write project name as 'Design1'
4. Choose project location; Select 'Design1' folder you already created through Browse button
5. The default library name is 'work'. Leave it unchanged and Click OK button

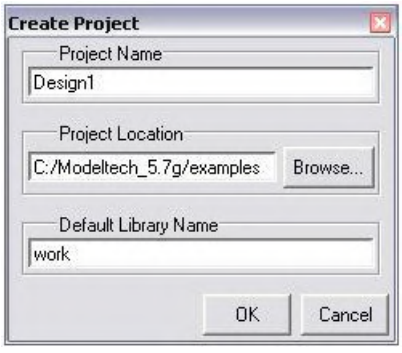

Figure 11.4: Create Project Dialog Box

- **Add Items to Project**
  Next you will see Dialog Box 'Add Items to Project' (Figure 11.5), click the Close button

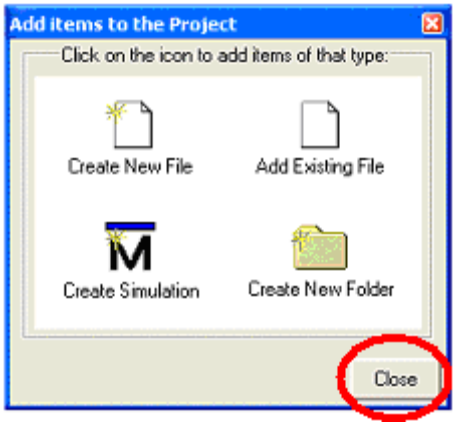

Figure 11.5: Add Items to Project Dialog Box

- **Writing Design Block & Stimulus Block**
  Verilog Simulation is composed of two blocks: Design Block and Stimulus or Test-bench block. We will write Verilog modules for these two blocks in separate files and add these files to our project 'Design1'.

- **Adding Files to the Project**
  We have an empty project 'Design1'. Now we will add a new file to the project. Click on File→Add to Project→New File. Choose Verilog as the file type. In the File name: box enter the desired file name, in this case the file is named "FA.v". Click on the OK button. The "FA.v" file has been added to the project.

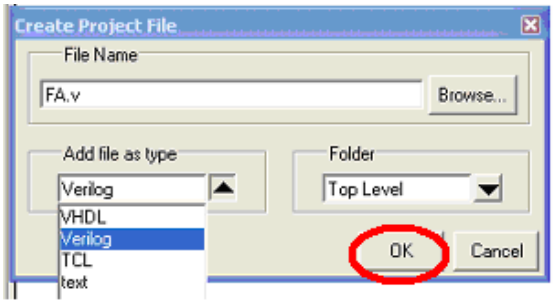

Figure 11.6: Adding File to the Project

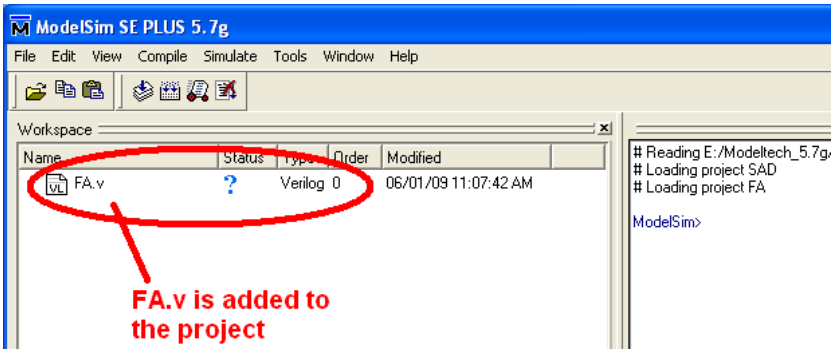

Figure 11.7: File added in Workspace Window

- Double-click on the 'FA.v' to show the file contents. An empty file 'FA.v' will appear. You are now ready to specify the FA module's functionality. Enter the FA module Verilog code shown below in 'FA.v' file. Save changes to the 'FA.v' file by clicking on *File→Save* (or simply click on icon)

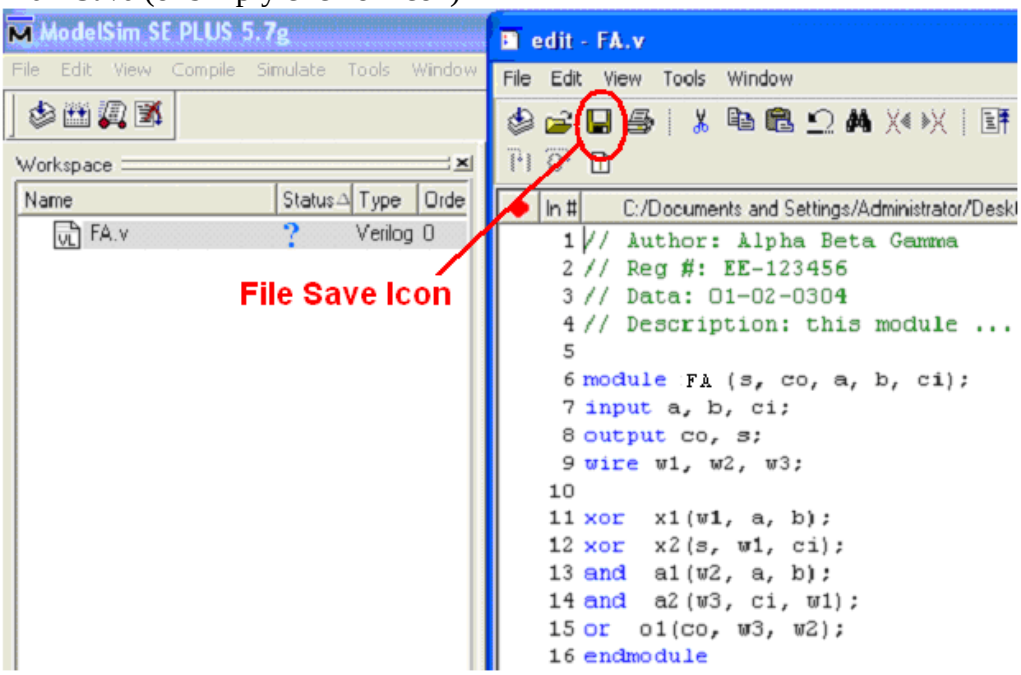

Figure 11.8: Editing the FA.v File

- Next we want to add the Stimulus module or test-bench module to the project. Add a new file to the project named 'tb_FA.v' by following steps already mentioned. Next enter the Verilog code for test-bench module shown below:

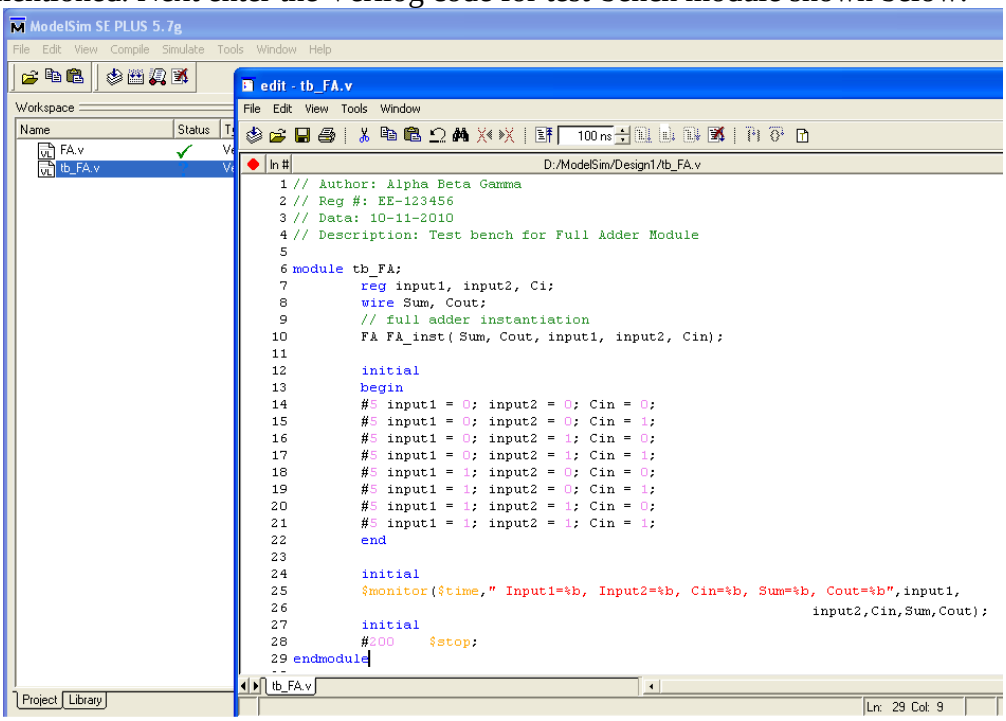

Figure 11.9: Editing tb_FA.v File

- **Compilation and Simulation of Project**

After saving both ‘FA.v’ and ‘tb_FA.v’, next we check syntax of Verilog source code in both files. Click on the Compile Menu and select Compile All. After project compilation, next we simulate the design in modelsim.

- **Timing waveform in Wave Window**
  To add the test bench module ‘tb_FA’ to wave window, right-click the module and select Add to Wave as shown below. Wave Window will appear with all the signals from module tb_FA added.

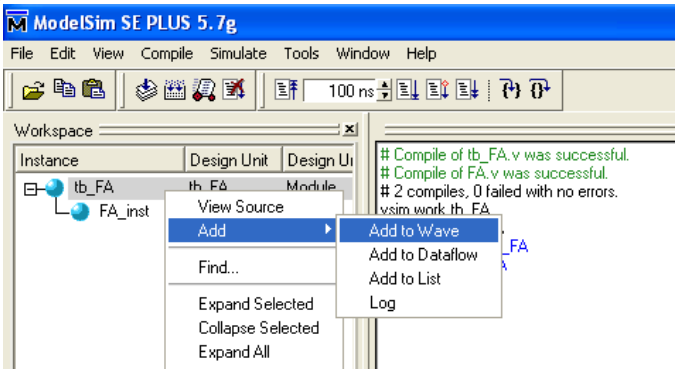

Figure 11.10: Adding the waveform

In the wave Window, click on the Run and then zoom icon/buttons to view the timing waveforms for project simulation. Functionality of Full Adder design can be verified by analyzing this timing waveform.

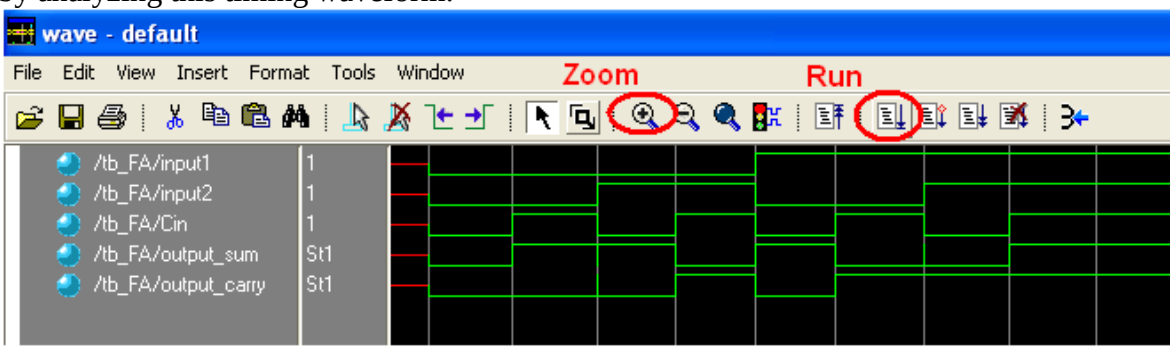

Figure 11.11: Simulation waveform

- Timing waveform can also be saved to file .For this click on File in Wave Window and select Save Image. Timing waveform will be saved as bitmap image in project folder. Timing waveform can also directly copied into word docs by using the select all and copy command from the ‘edit menu’ in wave window.

11.5 Lab Activities

11.5.1 Task-1:

Implement the resulting logic circuit of Full Adder Subtractor on ModelSim and perform truth table based verification.

11.5.2 Task-2:

Implement the resulting logic circuit as a Verilog Module on ModelSim and write a testbench to perform truth table based verification.

F (A, B, C) = Σ (1, 2, 3, 5, 6, 7)

11.5.3 Task-3:

Below is Circuit diagram of a 4 to 1 MUX

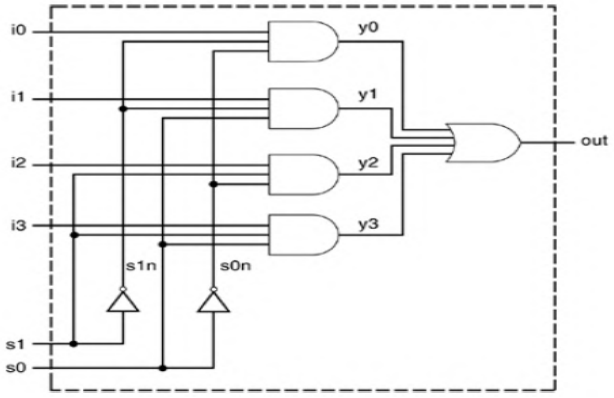

Figure 11.13: Task-2 Logic Diagram

- Write gate-level Verilog module for 4 to 1 MUX. Also write its Stimulus in Verilog. Your Stimulus should check output for different combinations of input & select lines.
- Create a ModelSim project, add 4 to 1 MUX module and its stimulus to project, run and verify the simulation. Save resultant timing waveform in a separate file

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date: _____**

LABORATORY SKILLS ASSESMENT (Psychomotor)

Total Marks: 100

Criteria (Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< 90%	Level 4 90%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate skills and/or strategies to write and compile the programs.	Selects and applies partially appropriate skills and/or strategies required by the programs.	Selects and applies the considerably appropriate strategies and/or skills specific to the programs.	Selects and applies the completely appropriate strategies and/or skills specific to the programs.	
Practical Implementation (20)	Makes major critical errors in applying procedural knowledge related to the Implementation of Combinational Logic using Verilog HDL	Makes numerous critical errors in applying procedural knowledge related to the Implementation of Combinational Logic using Verilog HDL	Makes minor non-critical errors in applying procedural knowledge related to the Implementation of Combinational Logic using Verilog HDL	Applies the procedural knowledge in optimized ways related to the Implementation of Combinational Logic using Verilog HDL	
Program Logic (20)	Program logic has many errors with majority of contradictory conditions.	Program logic has some errors with several contradictory conditions.	Program logic is mostly correct, but may contain occasional errors or redundant/ contradictory conditions.	Program logic is correct, with no known errors, and no redundant or contradictory conditions.	
Syntax Correctness and Results (20)	Program does not follow proper syntax of the Implementation of Combinational Logic using Verilog HDL and does not produce desired results for most inputs.	Program partially follows the proper syntax of the Implementation of Combinational Logic using Verilog HDL and produces appropriate results for few inputs. .	Program adequately follows the proper syntax of the Implementation of Combinational Logic using Verilog HDL and produces appropriate results for most inputs.	Program completely follows proper syntax of the Implementation of Combinational Logic using Verilog HDL and produces appropriate results for all inputs tested.	
Use of Software Tool (10)	Uses software tool, with limited competence.	Uses software tool, with some competence.	Uses software tool, with considerable competence.	Uses software tool, with a high degree of competence.	
Safety (10)	Requires Constant reminders to follow safety procedures.	Requires some reminders to follow safety procedures.	Follows safety procedures with only minimal reminders.	Routinely follows safety procedures.	
				Marks Obtained	

Instructor's Signature: _____ **Date:** _____

LABORATORY SKILLS ASSESMENT (Affective)

Total Marks: 40

Criteria (Max. Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < 90%	Level 4 90% ≤ S ≤ 100%	Score (S)
Introduction (5)	Very little background information provided or information is incorrect	Introduction is brief with some minor mistakes	Introduction is nearly complete, missing some minor points	Introduction complete and well-written; provides all necessary background principles for the experiment	
Procedure (5)	Many stages of the procedure are not entered on the lab report.	Many stages of the procedure are entered on the lab report.	The procedure could be more efficiently designed but most stages of the procedure are entered on the lab report.	The procedure is well designed and all stages of the procedure are entered on the lab report.	
Data Record (10)	Data is brief and missing significant pieces of information.	Data provides some significant information and has few critical mistakes.	Data is almost complete but has some minor mistakes.	Data is complete and relevant. Tables with units are provided. Graphs are labeled. All questions are answered correctly.	
Data Analysis (10)	Data is presented in very unclear manner. Error analysis is not included.	Data is presented in ways (charts, tables, graphs) that are not clear enough. Error analysis is included.	Data is presented in ways (charts, tables, graphs) that can be understood and interpreted. Error analysis is included.	Data are presented in ways (charts, tables, graphs) that best facilitate understanding and interpretation. Error analysis is included.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Cognitive)

Total Marks: 10

(If any) **Marks Obtained**	

Instructor's Signature: _____ **Date: _____**