



Centre of Excellence in VLSI

Verilog Lab Manual

www.maver-silicon.com

Maven Silicon Confidential

All the presentations, books, documents [hard copies and soft copies], labs and projects [Source Code] that you are using and developing as part of the training course are the proprietary work of Maven Silicon and it is fully protected under copyright and trade secret laws. You may not view, use, disclose, copy, or distribute the materials or any information except pursuant to a valid written license from Maven Silicon.

Table of Contents

Lab Instructions	4
Lab - 1: Verilog Syntax, Instantiation	15
Lab - 2: Operators	17
Lab - 3: Combinational Logic Design	19
Lab - 4: Sequential Logic Design	21
Lab - 5: Memory Logic Design	23
Lab - 6: FSM Design	25

Lab Instructions

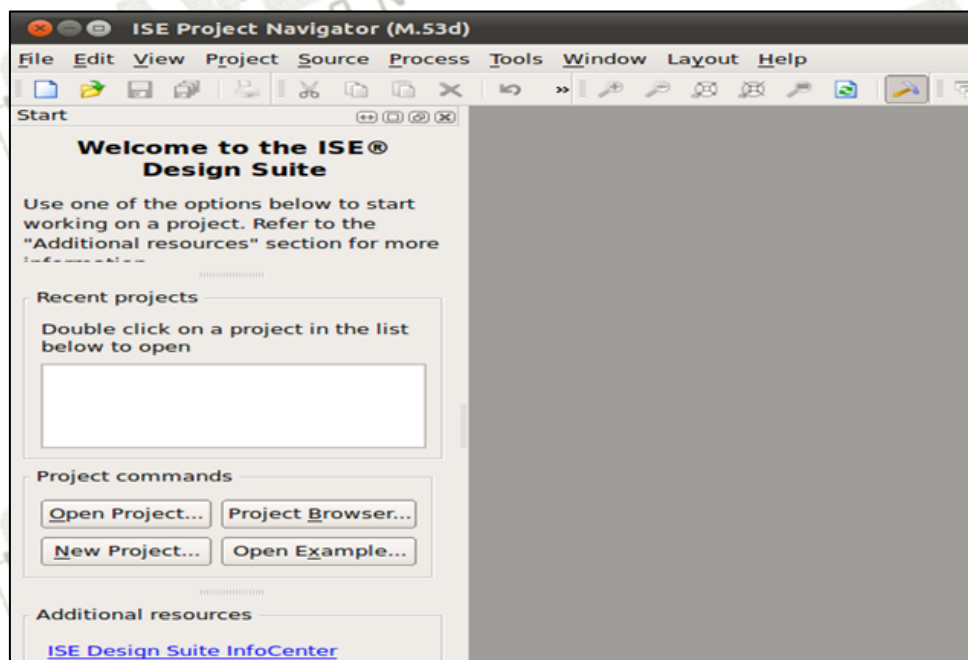
1. The recommended editor is vi or gvim editor
2. Xilinx Isim is used to run the simulation.
3. Xilinx XST is used for running synthesis.
4. The following directory structure is followed for all the lab exercises:

- sim/ - used to create project directory
- rtl/ - contains Verilog RTL code
- tb/ - contains Verilog Testbench

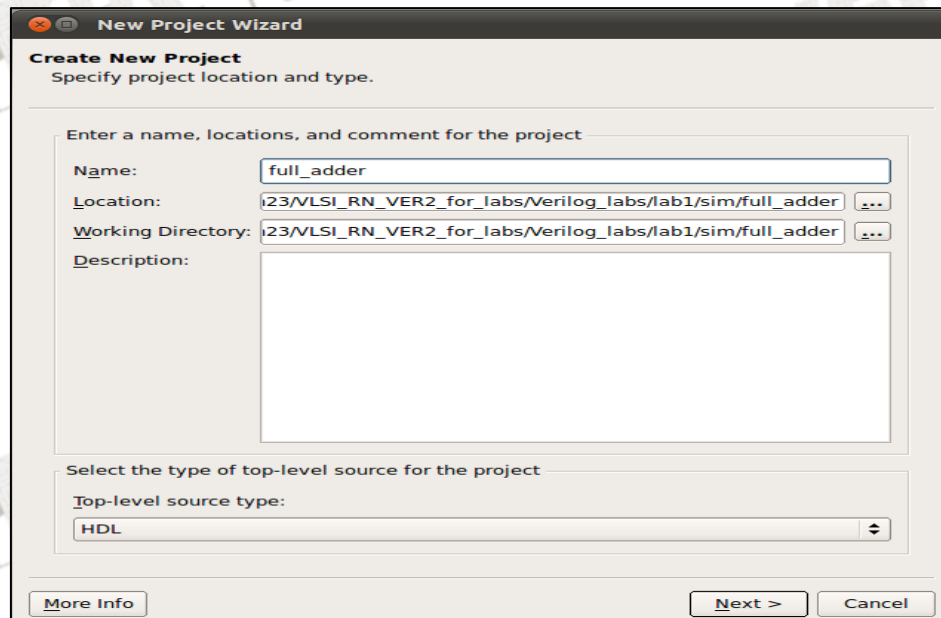
The simulation process involves different steps such as:

- a. Compilation
 - b. Elaboration
 - c. Simulation
5. Please find the steps for simulation process as mentioned below:

Step1: Invoke the GUI interface of the Xilinx application either from Linux terminal by typing the binary **\$ ise&** or from the Desktop short-cut icon on Windows. You will find the GUI interface opens up as shown below:



Step2: Then click on File-New-Project as shown below inorder to create a new Project.



Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location: ...

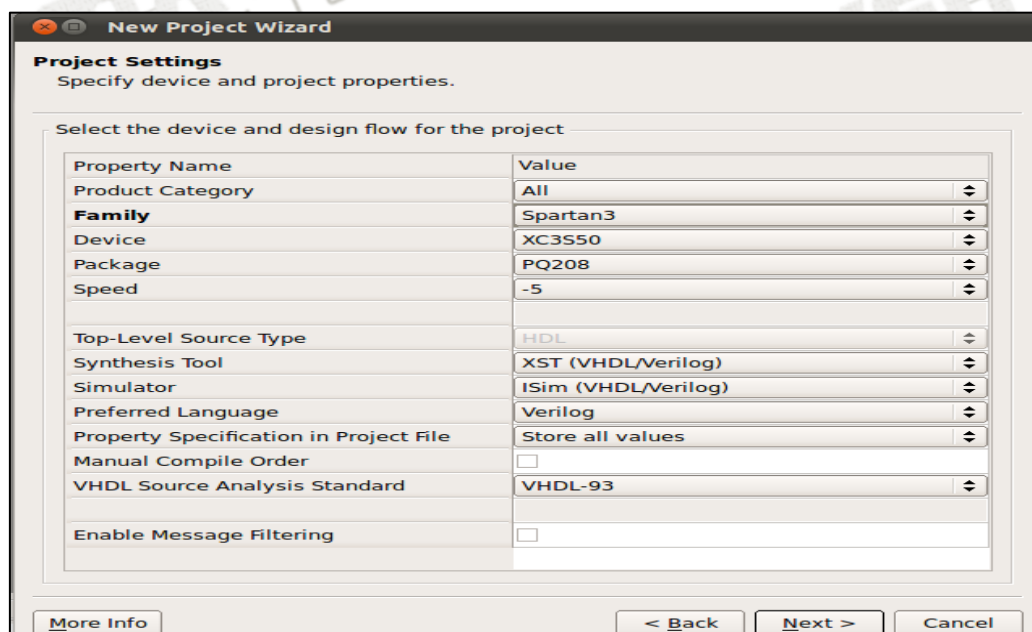
Working Directory: ...

Description:

Select the type of top-level source for the project

Top-level source type:

Step3: Then enter the **Name** of the project (recommended is name of the module), select the **Location** where you want to create the project (recommended is the sim directory) and then click **Next**, which opens up the following wizard where you need to select “Isim (VHDL/Verilog)” under the tab “Simulator” and configure the fields “Family, Device, Package, Speed”. Then click on **Next** then **Finish**.

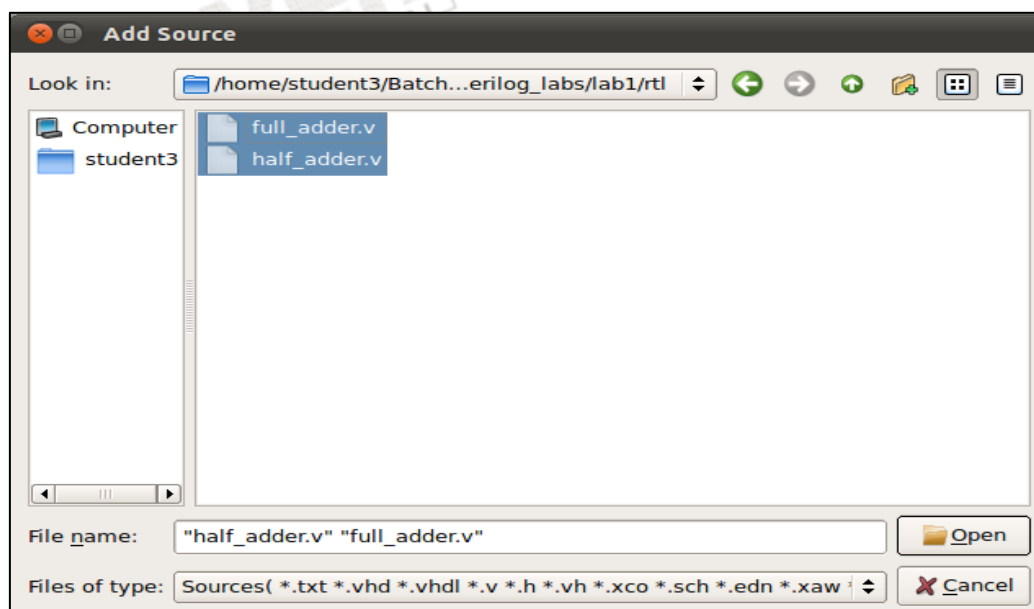
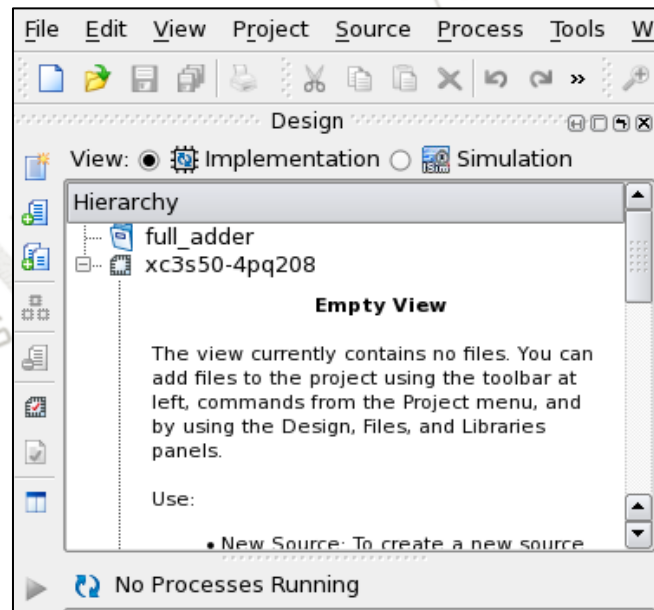


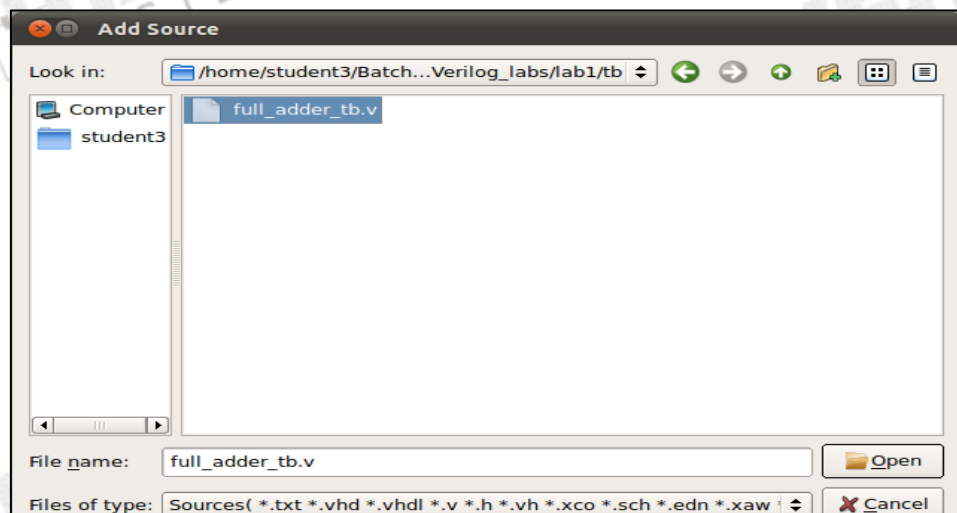
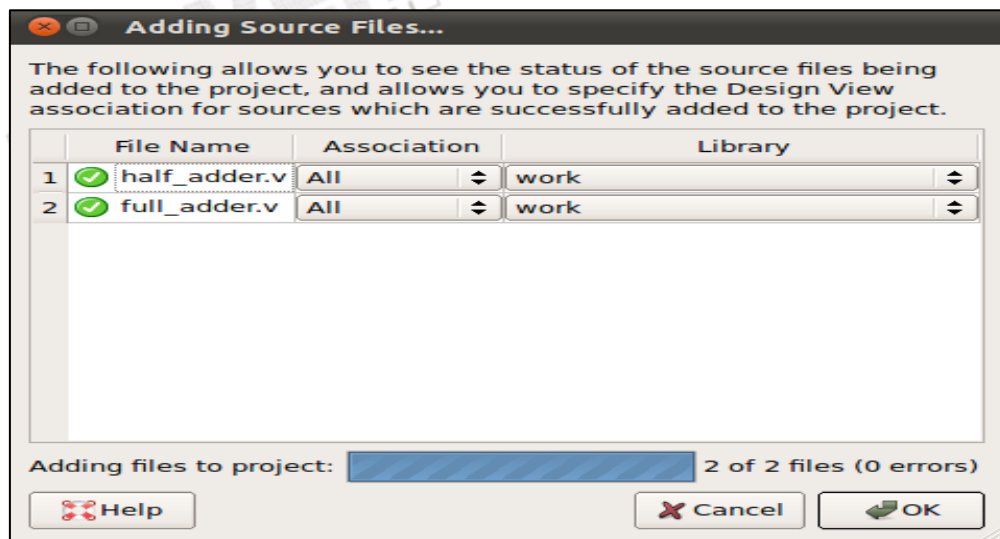
Project Settings
Specify device and project properties.

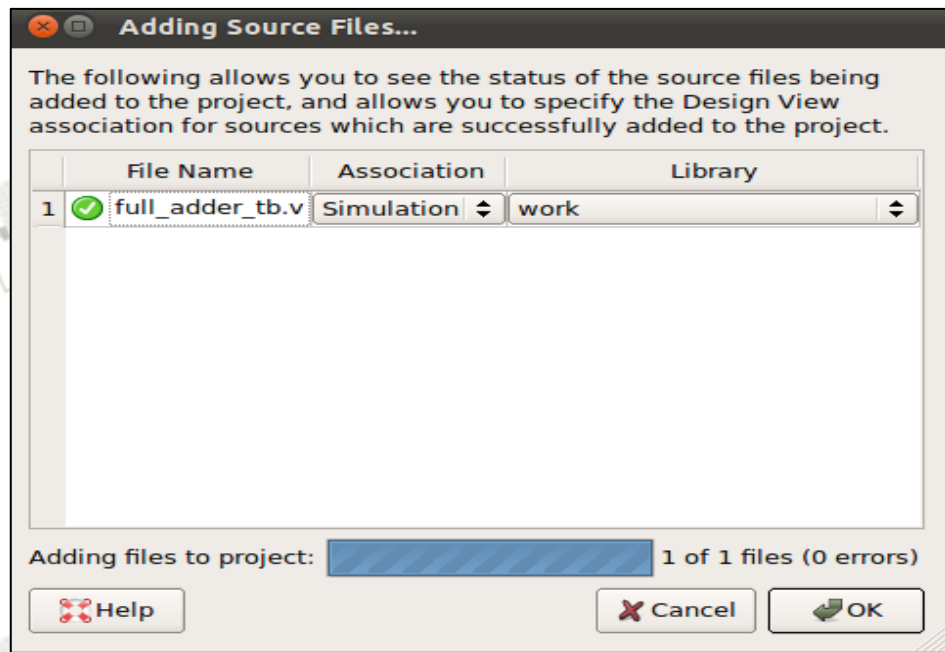
Select the device and design flow for the project

Property Name	Value
Product Category	All
Family	Spartan3
Device	XC3S50
Package	PQ208
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

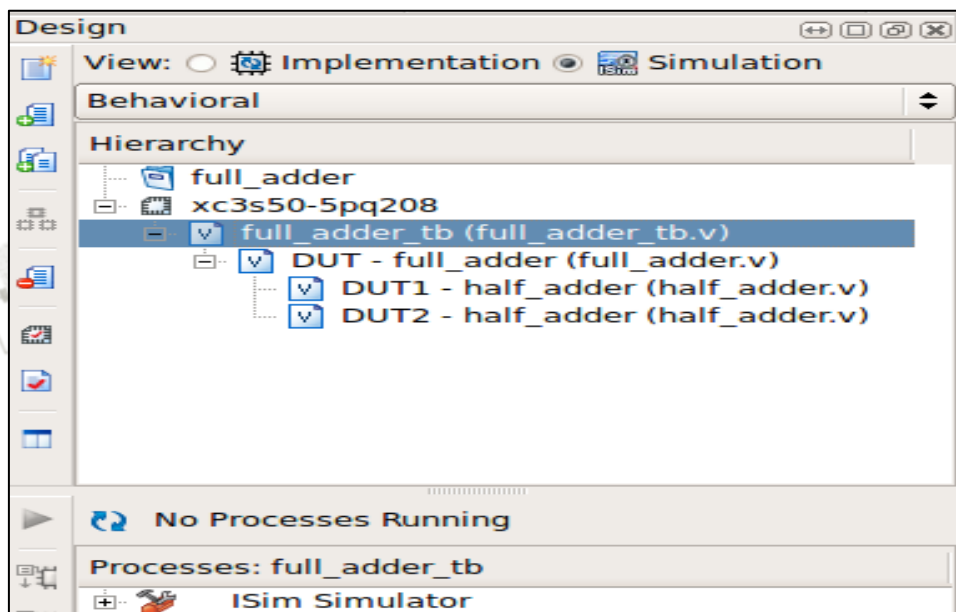
Step4: Then select **Simulation** under **View** and right click on the highlighted device name **xc3s50-5pq208** under **Hierarchy**, then select **Add Source** -select the RTL source files along with the Testbench files, click **Open** then **OK**. Check the same in the below snapshots.



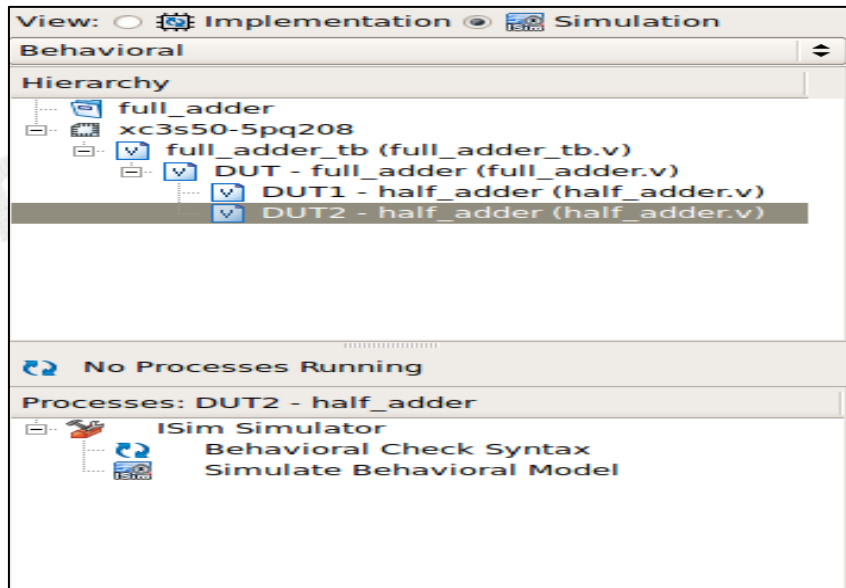




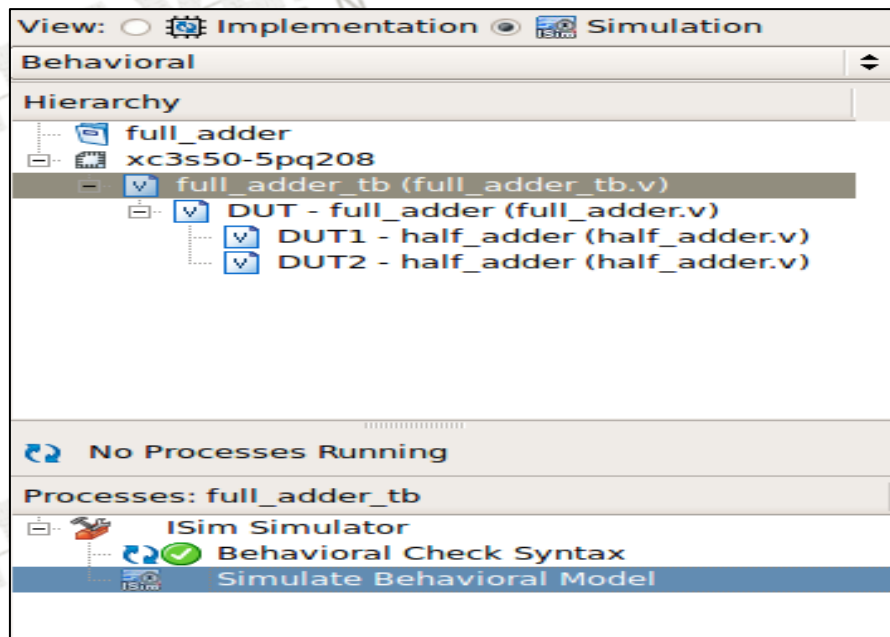
Step5: Under the completion of the above steps, the source files will be updated on the ISE Project Navigator window as shown below with the test bench file being set as the top module .



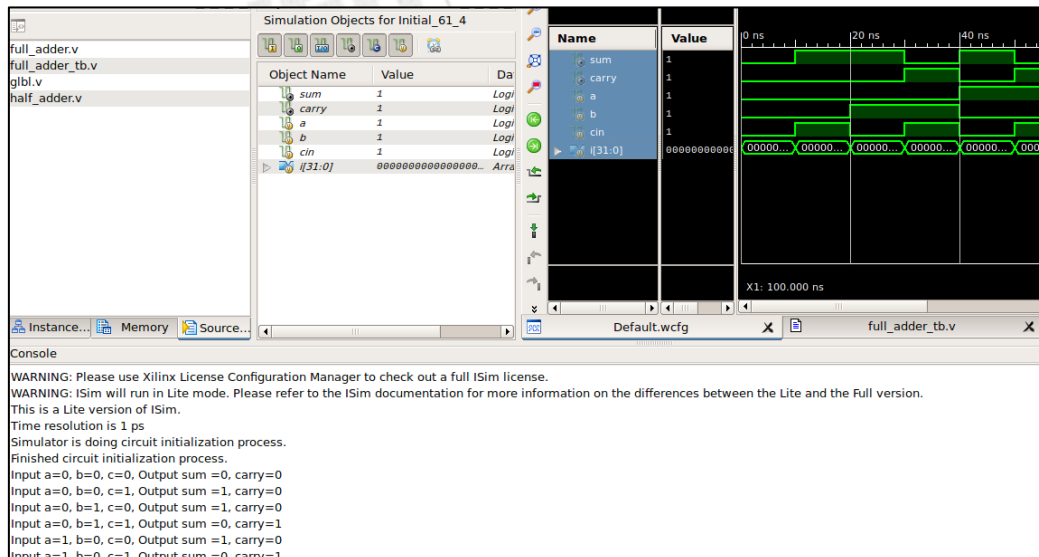
Step6: Then highlight an individual module, select the **Isim Simulator**-click on the drop down '+' & then double click on **Behavioural Check syntax**. This will show if any syntax error is there or not as shown below:



Step7: Then select the top testbench file and simulate as shown below:



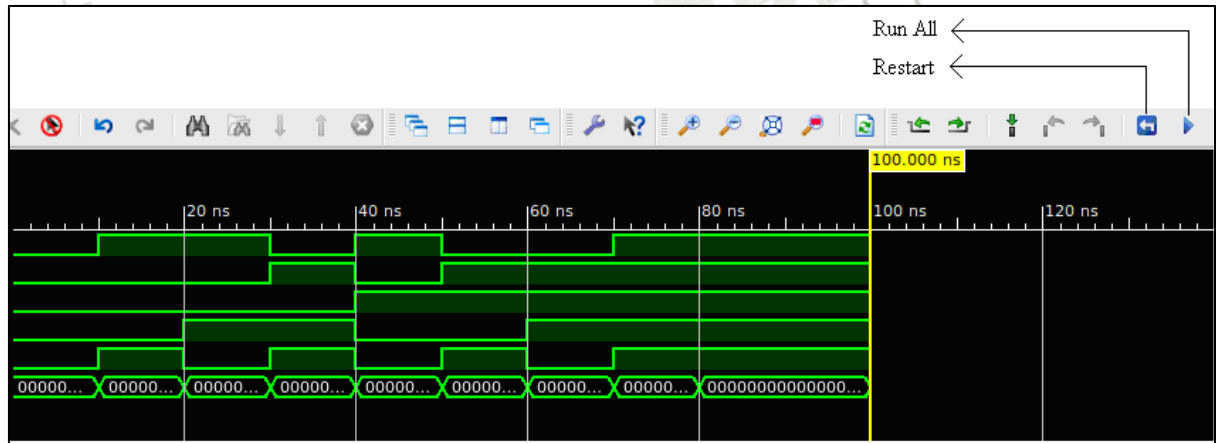
Step8: The above step will pop up another window called as “**Isim**” as shown below that displays both the Batch mode and Waveform mode of output. In order to undock the waveform window right click on the tab “Default.wcfg”- float . The waveform window will be expanded as shown below with the console output:



Step9: Things to be noted here:

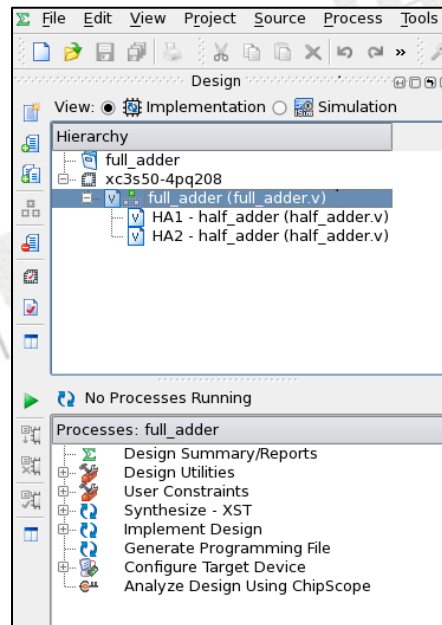
1) By default the top level testbench signals will get added to the waveform window and will get simulated. In order to restart and re-run the simulation, click on restart button

Run –all as shown below. This will end the simulation process.

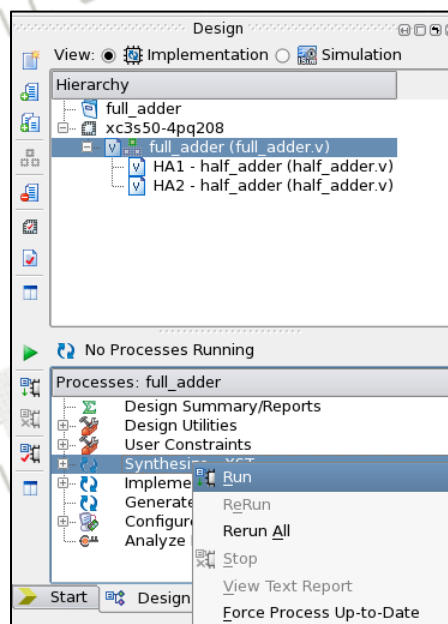


6. Please find the steps for synthesis process as mentioned below:

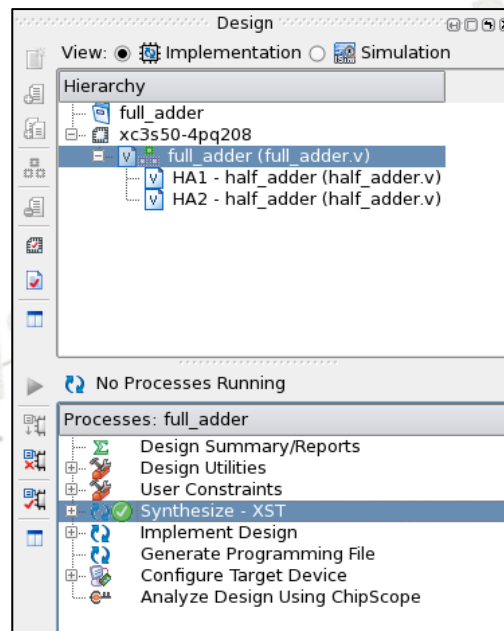
Step1: Select the **View** to Implementation mode as shown below. This will only select the RTL files & not the testbench file.



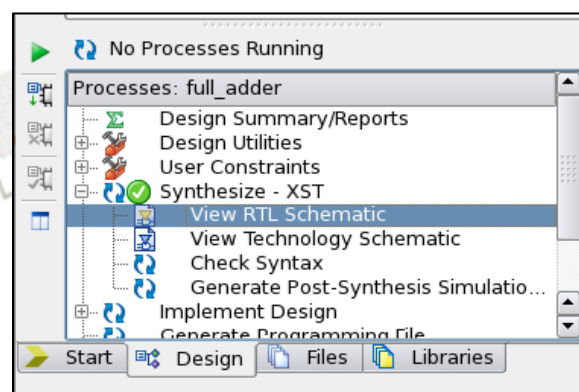
Step2: Then with the top module as highlighted & set as top, as shown below, run the synthesis process.



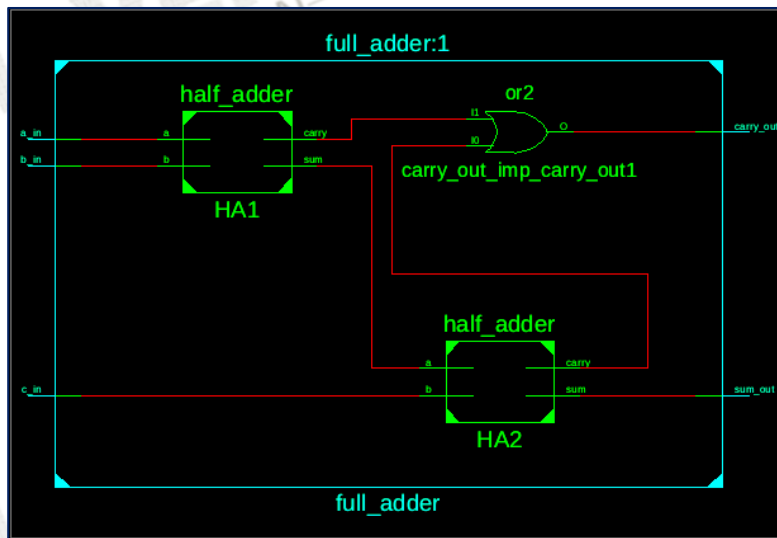
Step3: This will check if any synthesis issues are there & if synthesis run successfully, gives a green check as shown below.



Step4: Then click on the drop down & click on **View RTL Schematic** as shown below:



Step5: The next window that opens up will show the logical netlist as shown below:

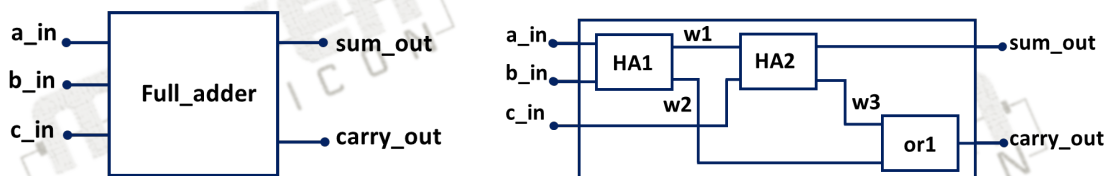


Lab - 1: Verilog Syntax, Instantiation

Objective : *To understand the different modelling styles in Verilog and learn different port mapping methods.*

Exercise :

Write RTL description and testbench for the Full Adder circuit using half adders and OR gate. The block diagram of full adder, along with complete connections of full adder using half adder and OR gate is shown below.



Working Directory : Verilog_labs/lab1/rtl

Source Code : half_adder.v, full_adder.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Write down the directions for the ports.
- ✓ Declare the internal wires.
- ✓ Instantiate the Half-Adders.
- ✓ Instantiate the OR gate.

Working Directory : Verilog_labs/lab1/tb

Source Code : full_adder_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the full_adder top with order based port mapping.
- ✓ Understand the global variables required for testbench.
- ✓ Understand how stimulus is generated using a for loop & delays.

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

How to instantiate the sub-blocks in the top RTL module and do port mapping.

Assignments :

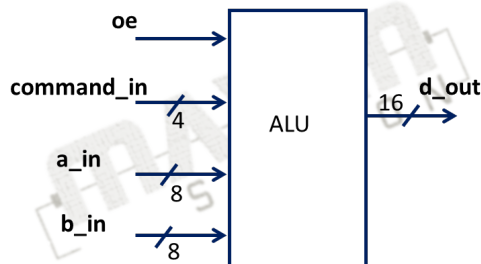
1. Write RTL for 1bit Full adder using Dataflow abstraction and verify the same using a Testbench.
2. Write RTL for 2x4 decoder using Dataflow abstraction and verify the same using a Testbench.
3. Write RTL for 8x3 priority encoder using structural model and verify the same using a Testbench.
4. Write RTL for the 4 bits Ripple carry Adder using 1-bit Full adder and verify the same using a Testbench.
5. Write RTL for 4:1 Mux using 2:1 Muxes and verify the same using a Testbench.
6. Write RTL for 4x1 Mux using Decoder & tri-state buffers and verify the same using a Testbench.
7. Write RTL to design a bidirectional buffer and verify the same using a Testbench.

Lab - 2: Operators

Objective : *To understand RTL description for an Arithmetic Logic Unit using arithmetic and logical operators.*

Exercise :

Write RTL description and testbench for an Arithmetic Logic Unit using arithmetic and logical operators. The block diagram and instructions set for ALU are shown below.



Working Directory : Verilog_labs/lab2/rtl

Source Code : alu.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Write down the functionality of ALU based on commands given.
- ✓ Understand the tri-state output logic.
- ✓ Within always@(command) begin
 - The ALU has to perform 16 different operations using command input on 8bits inputs **a_in** and **b_in**.
 - If the MSB of the **command_in** input is low then the arithmetic operations are performed.
 - If the MSB of **command_in** input is high then the logical operations are performed.
 - Use case construct to perform the operations.
 - The output is of 16bits. Input **oe** enables the ALU, i.e. when **oe** is low, output is at high impedance.

Working Directory : Verilog_labs/lab2/tb

Source Code : alu_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the design ALU.
- ✓ Understand about the global variables required for testbench.
- ✓ Write a task named "initialize" to initialize the inputs of DUT.
- ✓ Within initial begin
 - Understand how tasks are called and values are passed by "Call by Value" method.
 - Also understand how \$monitor is monitoring the changes in the outputs

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

Explore the different operators concept in Verilog.

Assignments :

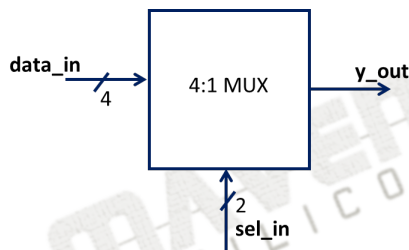
1. Practise all operators.

Lab - 3: Combinational Logic Design

Objective : *To understand, how to write RTL(behavioral) abstraction for a Multiplexor.*

Exercise :

Write RTL (Behavioral) description and test bench for a 4:1 Multiplexer circuit. The block diagram of 4:1 multiplexer is shown below.



Working Directory : Verilog_labs/lab3/rtl

Source Code : mux4_1.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Define the port directions with proper datatypes.
- ✓ Write the MUX behaviour as a parallel logic.
- ✓ Within always@(.) begin
 - Use case construct to represent the behavior of 4:1Mux using its TruthTable as per the select lines.
 - Use Blocking assignments.

Working Directory : Verilog_labs/lab3/tb

Source Code : mux4_1_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the Design.
- ✓ Write down the variables required for testbench.
- ✓ Declare a task to initialize inputs of DUT to 0.
- ✓ Declare tasks with arguments for driving stimulus to DUT.
- ✓ Call the tasks from procedural block.
- ✓ Use \$monitor task to display inputs and outputs.
- ✓ Use \$finish task to terminate the simulation at 100ns.
- ✓ Within initial begin
 - Call the tasks & pass values by "Call by Value" method.

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

Understand the behavioral abstraction level for designing a combinational circuit.

Assignments :

1. Write RTL description and test bench for 3:8 Decoder.
2. Write RTL description and testbench for 8:3 Priority encoder.

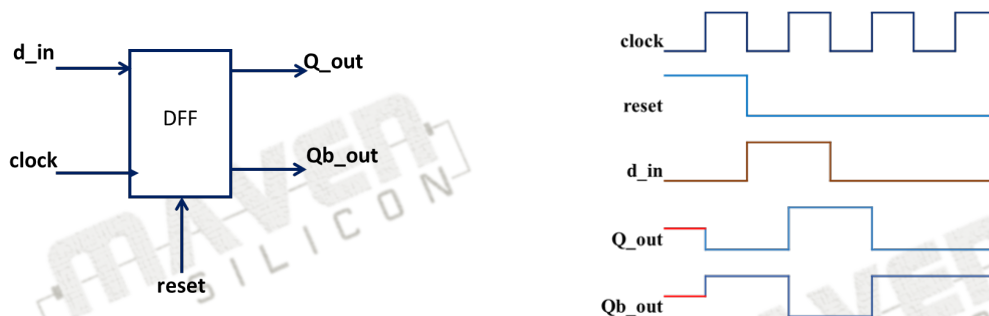
Lab - 4: Sequential Logic Design

Objective : To understand how to write RTL(behavioral) abstraction for a DFF.

Exercise :

Write RTL description and testbench for the D flip-flop.

The block diagram, input and output waveforms of D flip-flop is shown below.



Working Directory : Verilog_labs/lab4/rtl

Source Code : dff.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Define the port directions with proper datatypes.
- ✓ Understand the behavioral logic for D flip-flop functionality.
- ✓ Write the logic for **Qb_out**.
- ✓ Within always@(posedge clk) begin
 - Understand the usage of if construct to reset the DFF and in the else part the normal operation of the Flip-flop.
 - Understand the usage of Non-blocking assignments.

Working Directory : Verilog_labs/lab4/tb

Source Code : dff_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the Design.
- ✓ Understand about the global variables required for testbench.
- ✓ Define a parameter constant with name "CYCLE" which is equal to 10.
- ✓ Understand the clock generation logic.
- ✓ Understand the event based timing control statements within the tasks for generating the stimulus.
- ✓ Use \$monitor to display the various inputs and outputs.
- ✓ Within initial begin
 - Understand how the tasks are called & values are passed by "Call by Value" method.

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

Understand the behavioral abstraction level for designing a sequential circuit.
Also analyze the timing diagram for the circuit using waveform.

Assignments :

1. Write RTL and Testbench for SR latch using Gate level modelling.
2. Write RTL and Testbench for JK Flip Flop, using parameter declaration for the respective scenarios (HOLD, TOGGLE, SET, RESET).
3. Write RTL and Testbench for a T Flip Flop using D Flip Flop.
4. Write RTL and Testbench for a 4-bit synchronous and loadable binary up counter.
5. Write RTL and Testbench to design a 4-bit MOD-12 loadable binary synchronous up counter.
6. Write RTL and Testbench to design a 4-bit loadable binary synchronous up-down counter.

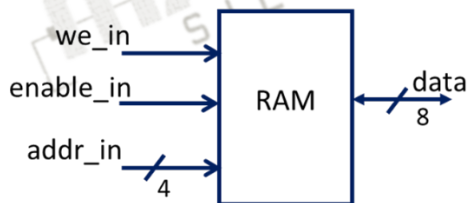
Lab - 5: Memory Logic Design

Objective : *To understand how to write RTL(behavioral) abstraction for an asynchronous Single Port RAM*

Exercise :

Write RTL description and testbench for the Single Port RAM, which is 8 bit wide and has 16 memory locations. The data can be written on a memory location by providing its address and making “we” high and “oe” low. The data can be read from a memory location by providing the address and making “oe” high and “we” low. This RAM has single port for data writing and reading.

The block diagram of the single port RAM is shown below.



Working Directory : Verilog_labs/lab5/rtl

Source Code : ram.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Declare a 8 bit wide memory having 16 locations.
- ✓ Understand the logic for writing data into a memory location
- ✓ Understand the logic of reading data from a memory location
- ✓ Within always@(...) begin
 - Understand how the memory write operation is done using a fixed size array such that when write is asserted, read is low.
- ✓ Also understand how read operation is done using continuous concurrent process as data which is an inout port is always of wire type.

Working Directory : Verilog_labs/lab5/tb

Source Code : ram_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the RAM module and connect the ports.
- ✓ Write a task named "stimulus" to assign data into "addr" and "tempd" inputs through i and j variables.
- ✓ Understand the various tasks used in this testbench.
- ✓ Understand how a temp reg variable is used to generate stimulus and drive the same to the inout port of the DUT via a wire type.
- ✓ Within initial begin
 - Call the tasks & pass values by "Call by Value" method.

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

Understand how fixed size array is used to model a Memory design.

Also analyze how inout ports are driven from the Testbench.

Assignments :

1. Write RTL to design a 16x8 synchronous dual-port RAM & verify the same using Testbench.
2. Write RTL to design a 8x16 asynchronous dual-port RAM & verify the same using Testbench.
3. Write RTL to design a 4-bit right shift Serial-In-Serial-out shift register & verify the same using Testbench.
4. Write RTL to design a clock buffer & verify the same using Testbench.
5. Write RTL to implement a FIFO for the below mentioned specification:

A synchronous FIFO with a size 16x8 (8 bits wide & 16 bytes depth) supports simultaneous read and write operations. Empty, Full are status signals based on the read or write operations.

Inputs: clock, reset(asynchronous active low), data_in, read_n and write_n

Outputs: data_out, full, empty

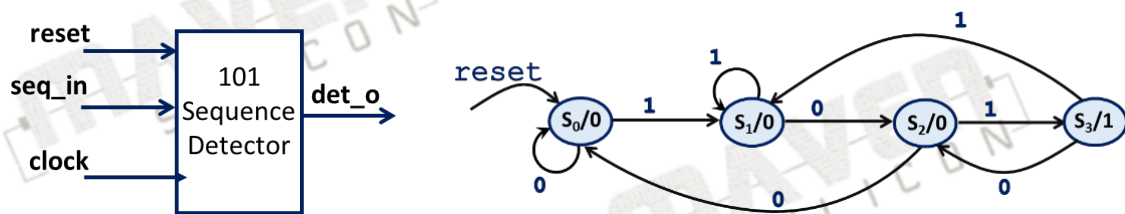
Lab - 6: FSM Design

Objective : To understand how to write RTL(behavioral) abstraction for an overlapping sequence detector 101 using Moore FSM.

Exercise :

Write RTL description and testbench for the Sequence Detector that detects “101” from input data stream.

The block diagram and state diagram for the “101” sequence detector are shown below.



Working Directory : Verilog_labs/lab6/rtl

Source Code : seq_det.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Declare the states as parameter "S0", "S1", "S2", "S3".
- ✓ Write down the port direction.
- ✓ Write down the sequential logic for present state.
- ✓ Understand the combinational logic for next state.
- ✓ Write down the logic for output dout.
- ✓ Within always@(posedge clk) begin
 - Write the present state logic using Non-blocking assignments.
- ✓ Within always@(...) begin
 - Understand the next state logic using Blocking assignments.
- ✓ Also understand how output is calculated using continuous concurrent assignments.

Working Directory : Verilog_labs/lab6/tb

Source Code : seq_det_tb.v

Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Generate clock, using parameter "CYCLE".
- ✓ Write a task named "initialize" to initialize the input din of sequence detector.
- ✓ Write a task named "RESET" to reset the design.
- ✓ Write a task named "stimulus" which provides input to design on negedge of clock.
- ✓ Within initial begin
 - Call the tasks & pass values by “Call by Value” method in such a way that the sequence 101 is passed in an overlapping manner.

Simulation Process :

- ✓ Create a project
- ✓ Add the source files RTL & TB
- ✓ Simulate the TB
- ✓ Observe the waveform & console output

Synthesis Process :

- ✓ Create a project
- ✓ Add the RTL files
- ✓ Synthesize the code
- ✓ Observe the RTL schematic

Learning outcomes :

Understand how to use internal registers to write both present state & next state logic.
Also analyze how output logic is written for a Moore based sequence detector.

Assignments :

Design a synchronous logic control unit for Vending machine.

The machine can take only two types of coins of denomination 1 and 2 in any order. It delivers only one product that is priced Rs. 3. On receiving Rs. 3, the product is delivered by asserting an output $X=1$ which otherwise remains 0. If it gets Rs. 4, then the product is delivered by asserting X and also a coin return mechanism is activated by output $Y = 1$ to return a Re. 1 coin. There are two sensors to sense the denomination of the coins that give binary output as shown in the following table. The clock speed is much higher than human response time, i.e. no two coins can be deposited in same clock cycle.

I	J	Coin
0	X	No coin dropped
1	0	One Rupee
1	1	Two Rupees