Experiment#5

Title: -

Introduction to Verilog and implementation of Basic Logic gates using Verilog

Objective: -

- To get familiar with Verilog
- To get familiar with SynaptiCAD verilogger software
- To implement basic logic Gates

Tools required: -

- SynaptiCAD verilogger Software

Theory: -

What is Verilog and why we use it?

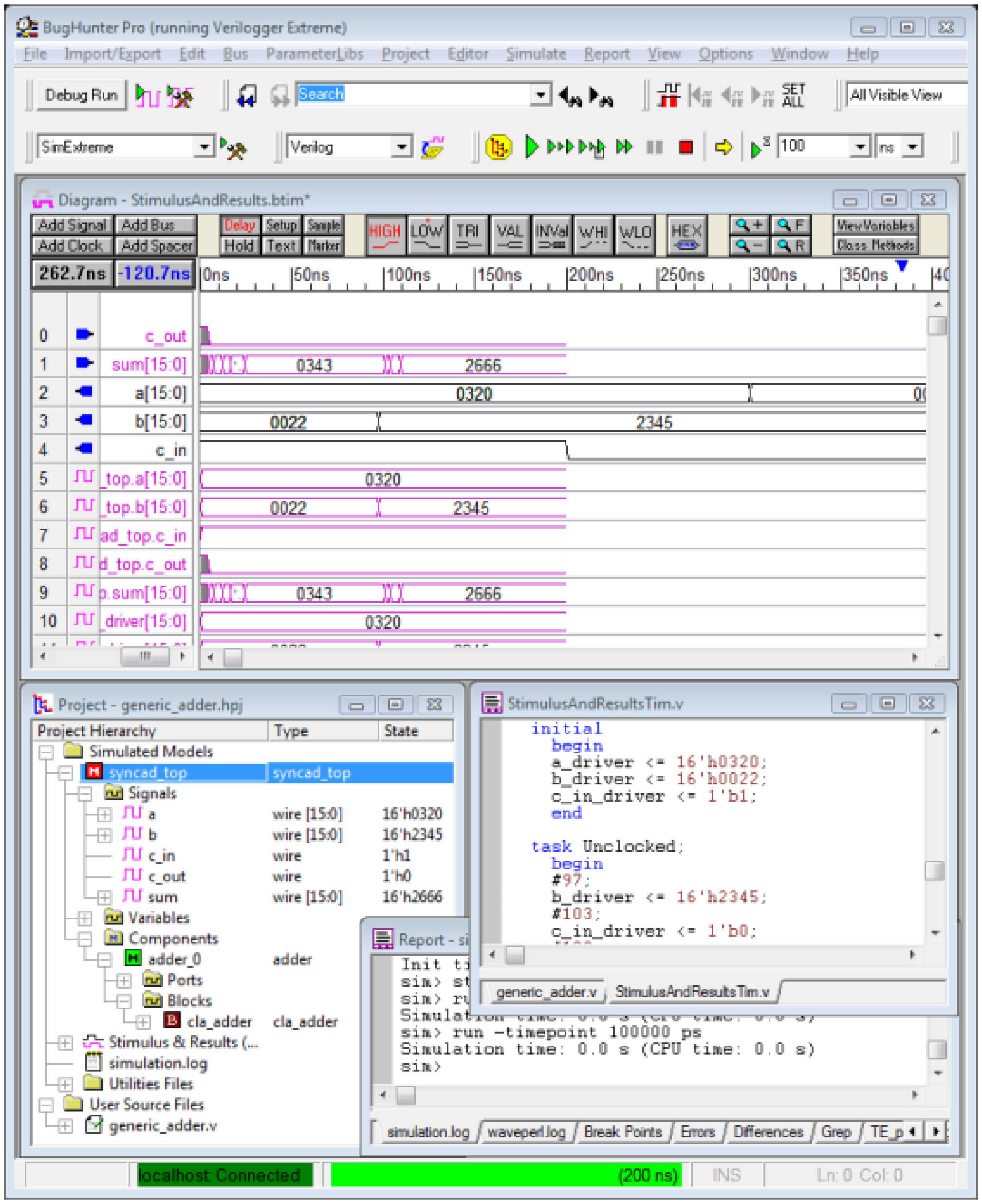
Verilog is a HDL (Hardware Description Language). It is used to model and simulate digital electronic circuits. Once a design is simulated, tested and ready for 'tape-out' to the fab, it can be synthesized to produce gate level designs that are then translated to physical design. In some ways, you can think of Verilog HDL as the code that represents the highest level abstraction of electronic circuits. Verilog is a hardware description language. It is different from general purpose programming languages in that it is specifically used to model hardware. So in Verilog you have the ability to specify registers, wires, gates, clock, etc. It is very useful when you have hardware specs on paper, and you want to simulate and test it first before synthesizing the circuit, thus saving time and money. The language is also pretty simple, only requires knowledge of digital logic, and it has syntax similar to C. The "Veri" part of the name is short for verification, and Verilog was designed for simulating digital hardware to verify that it operated correctly prior to fabrication. Some of the language constructs pertain to board-level logic (e.g. tri-state buses), but it is mostly known as a digital IC modeling language. It was the "sign off" simulation language for chip design for many years, and still is in some quarters. System Verilog is a pure extension of Verilog, but has not appreciably improved the sign-off level capabilities of the language, it mostly added test-bench level constructs (that could have been done in C/C++ or other OO languages), and sequenced assertions which help with verification.

SynaptiCAD Verilogger:

VeriLogger, by SynaptiCAD is a complete design and verification environment for ASIC and FPGA designers. It contains a new type of Verilog simulation environment that combines all the features of a traditional Verilog simulator with the most powerful graphical test vector generator on the planet. Model testing is so fast in VeriLogger Pro that you can perform true bottom-up testing of every model in your design, a critical step often skipped in the race to market. Test vectors can be imported or exported from HP logic analyzers, pattern generators, and 3rd party VHDL, Verilog, and SPICE simulators for reuse. Simulation features include waveform viewing, optimized gate-level simulation, single-step debugging, point-and-click breakpoints, hierarchical browser for project management, and batch execution.

VeriLogger Screen Shot

Take a look at everything available to you in the VeriLogger Program:


BugHunter Pro (running Verilogger Extreme)
File Import/Export Edit Bus ParameterLibs Project Editor Simulate Report View Options Window Help
Debug Run
Search
SET ALL
All Visible View
SimExtreme
Verilog
100
ns
Diagram - StimulusAndResults.btim*
Add Signal Add Bus Add Clock Add Spacer
Delay Setup Sample Hold Text Marker
HIGH LOW TRI VAL INVal WHI WLO HEX
View Variables Class Methods
262.7ns -120.7ns
0ns 50ns 100ns 150ns 200ns 250ns 300ns 350ns
0 c_out
1 sum[15:0] 0343 2666
2 a[15:0] 0320
3 b[15:0] 0022 2345
4 c_in
5 _top.a[15:0] 0320
6 _top.b[15:0] 0022 2345
7 ad_top.c_in
8 d_top.c_out
9 p.sum[15:0] 0343 2666
10 _driver[15:0] 0320
Project - generic_adder.hpj
StimulusAndResultsTim.v
Report - si
simulation.log waveperl.log Break Points Errors Differences Grep TE_p
generic_adder.v StimulusAndResultsTim.v
localhost Connected
(200 ns)
INS
Ln: 0 Col: 0

Project Hierarchy	Type	State
Simulated Models		
syncad_top	syncad_top	
Signals		
a	wire [15:0]	16'h0320
b	wire [15:0]	16'h2345
c_in	wire	1'h1
c_out	wire	1'h0
sum	wire [15:0]	16'h2666
Variables		
Components		
adder_0	adder	
Ports		
Blocks		
cla_adder	cla_adder	
Stimulus & Results (...		
simulation.log		
Utilities Files		
User Source Files		
generic_adder.v		

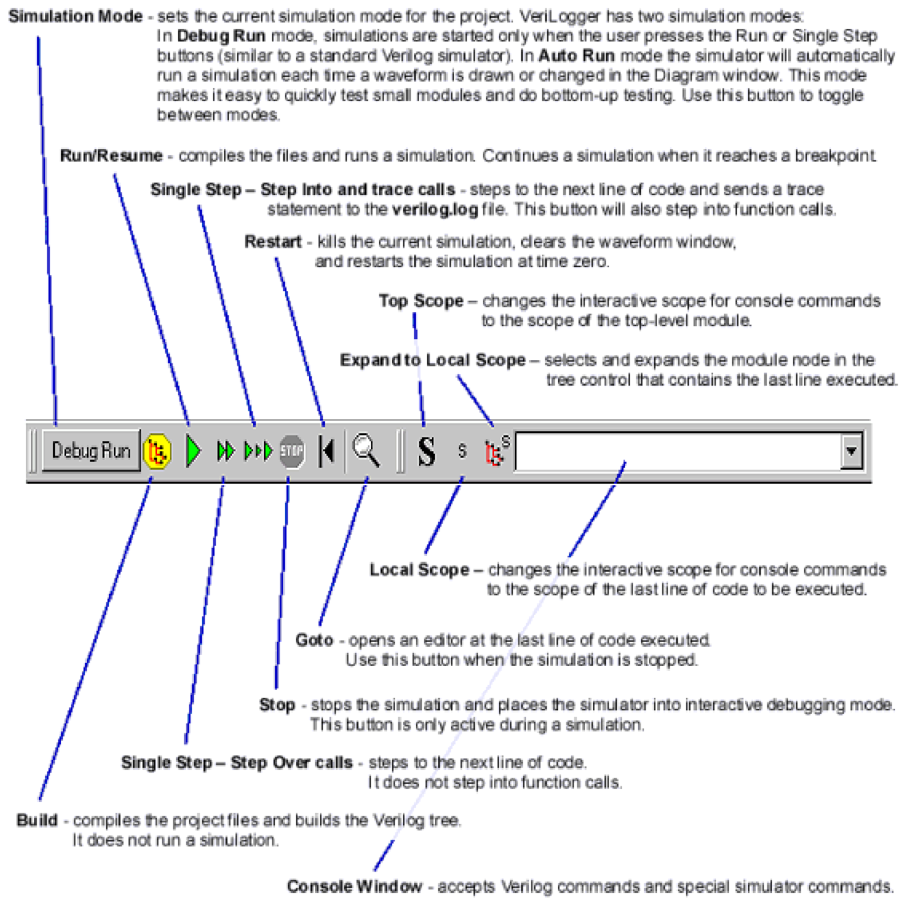
```
initial
  begin
    a_driver <= 16'h0320;
    b_driver <= 16'h0022;
    c_in_driver <= 1'b1;
  end

task Unclocked;
  begin
    #97;
    b_driver <= 16'h2345;
    #103;
    c_in_driver <= 1'b0;
```

```
Init ti
sim> st
sim> ru
Simulation time: 0.0 s (CPU time: 0.0 s)
sim> run -timepoint 100000 ps
Simulation time: 0.0 s (CPU time: 0.0 s)
sim>
```

Simulation Button Bar

The simulation bar allows you to control the simulation mode, run/resume your simulations, restart a project, change the interactive scope for console commands, or expand to local scope.

- **Simulation Mode** switches between normal debug/run mode, and a unique auto-run that restarts the simulation after graphical changes to the waveform.
- **Run/Resume** continues the simulation from the current time.
- **Single Step and Step Into trace calls** continues the simulation for one line of code.
- **Restart** stops the current simulation, and restarts at time zero.
- **Scoping Buttons** changes scope for console level commands.
- **Goto Button** opens an editor at the last line of code executed.
- **Stop** stops a Verilog simulation.
- **Build** runs the Verilog compiler and creates the Verilog tree, but does not start a simulation.



Simulation Mode - sets the current simulation mode for the project. VeriLogger has two simulation modes: In **Debug Run** mode, simulations are started only when the user presses the Run or Single Step buttons (similar to a standard Verilog simulator). In **Auto Run** mode the simulator will automatically run a simulation each time a waveform is drawn or changed in the Diagram window. This mode makes it easy to quickly test small modules and do bottom-up testing. Use this button to toggle between modes.

Run/Resume - compiles the files and runs a simulation. Continues a simulation when it reaches a breakpoint.

Single Step – Step Into and trace calls - steps to the next line of code and sends a trace statement to the **verilog.log** file. This button will also step into function calls.

Restart - kills the current simulation, clears the waveform window, and restarts the simulation at time zero.

Top Scope – changes the interactive scope for console commands to the scope of the top-level module.

Expand to Local Scope – selects and expands the module node in the tree control that contains the last line executed.

Local Scope – changes the interactive scope for console commands to the scope of the last line of code to be executed.

Goto - opens an editor at the last line of code executed. Use this button when the simulation is stopped.

Stop - stops the simulation and places the simulator into interactive debugging mode. This button is only active during a simulation.

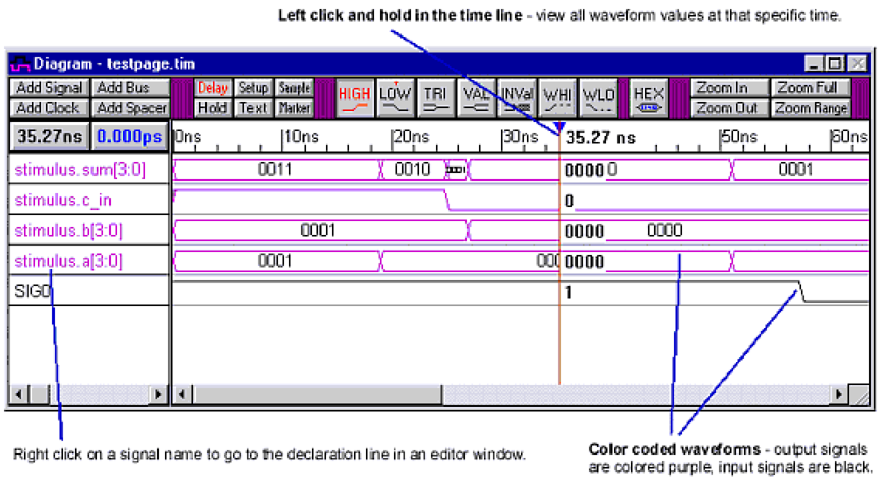
Single Step – Step Over calls - steps to the next line of code. It does not step into function calls.

Build - compiles the project files and builds the Verilog tree. It does not run a simulation.

Console Window - accepts Verilog commands and special simulator commands.

Diagram Window

Color coded waveforms help you distinguish between graphical test bench waveforms and simulated result waveforms. Left clicking in the time line, displays a marker showing the exact waveform value at a particular time. Right clicking on a signal name will take you to where the signal is declared in the Verilog source code.



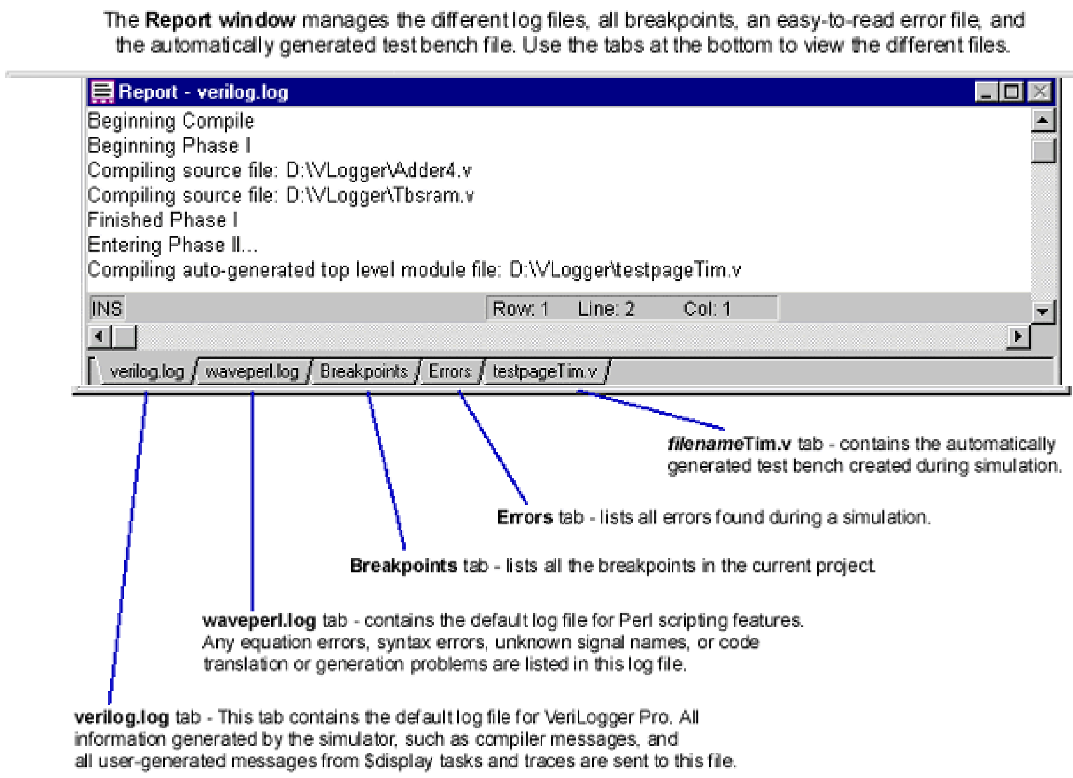
Left click and hold in the time line - view all waveform values at that specific time.

Right click on a signal name to go to the declaration line in an editor window.

Color coded waveforms - output signals are colored purple, input signals are black.

Report Window

The Report window manages your different log files, breakpoints, error files, and source code files for the Verilog simulator project. Each tab can also be opened in a different window if code needs to be viewed side-by-side.


The **Report window** manages the different log files, all breakpoints, an easy-to-read error file, and the automatically generated testbench file. Use the tabs at the bottom to view the different files.

Report - verilog.log

```
Beginning Compile
Beginning Phase I
Compiling source file: D:\VLogger\Adder4.v
Compiling source file: D:\VLogger\Tbsram.v
Finished Phase I
Entering Phase II...
Compiling auto-generated top level module file: D:\VLogger\testpageTim.v
```

INS   Row: 1   Line: 2   Col: 1

verilog.log / waveperl.log / Breakpoints / Errors / testpageTim.v

filenameTim.v tab - contains the automatically generated test bench created during simulation.

Errors tab - lists all errors found during a simulation.

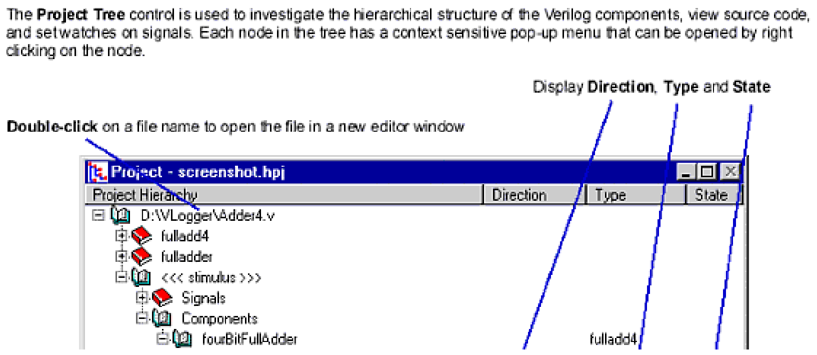
Breakpoints tab - lists all the breakpoints in the current project

waveperl.log tab - contains the default log file for Perl scripting features. Any equation errors, syntax errors, unknown signal names, or code translation or generation problems are listed in this log file.

verilog.log tab - This tab contains the default log file for VeriLogger Pro. All information generated by the simulator, such as compiler messages, and all user-generated messages from $display tasks and traces are sent to this file.

Project window

Whether you are working on a single project, or many at a time, with the project window, you will be able to easily manage and keep track of as many Verilog files as you need. Once the Project is built by the Verilog compiler, the Project will display a tree of the design.

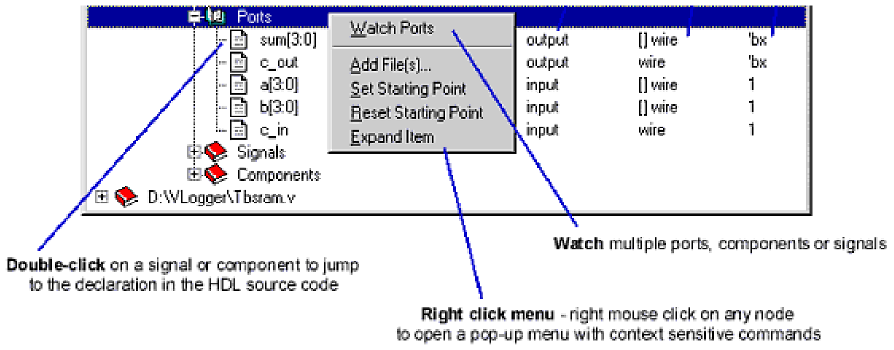

The **Project Tree** control is used to investigate the hierarchical structure of the Verilog components, view source code, and set watches on signals. Each node in the tree has a context sensitive pop-up menu that can be opened by right clicking on the node.

Display **Direction**, **Type** and **State**

Double-click on a file name to open the file in a new editor window

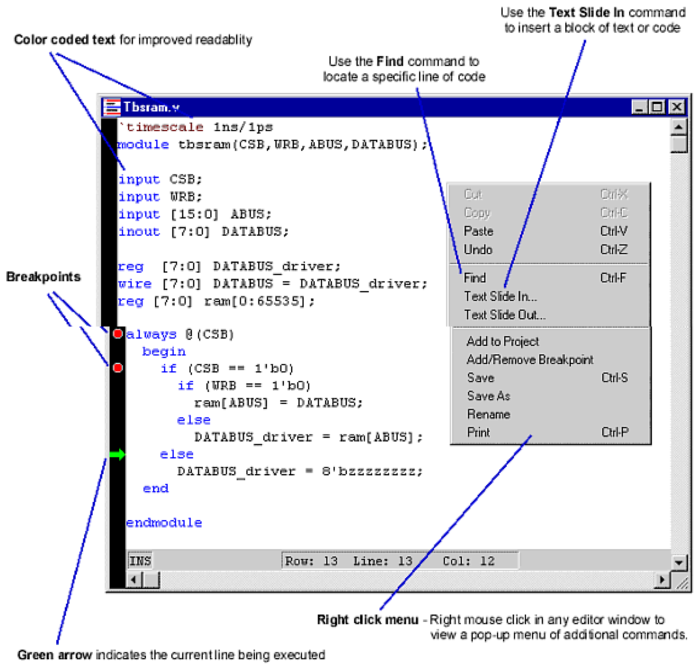
Project - screenshot.hpj

Project Hierarchy	Direction	Type	State
D:\VLogger\Adder4.v			
fulladd4			
fulladder			
<<< stimulus >>>			
Signals			
Components			
fourBitFullAdder		fulladd4	


Ports
sum[3:0]
c_out
a[3:0]
b[3:0]
c_in
Signals
Components
D:\VLogger\Tbsram.v
Watch Ports
Add File(s)...
Set Starting Point
Reset Starting Point
Expand Item
output [] wire 'bx
output wire 'bx
input [] wire 1
input [] wire 1
input wire 1
Double-click on a signal or component to jump to the declaration in the HDL source code
Watch multiple ports, components or signals
Right click menu - right mouse click on any node to open a pop-up menu with context sensitive commands

Editor window

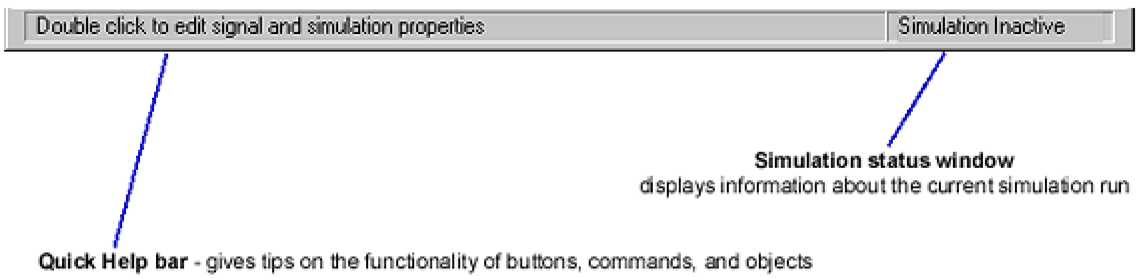
The editor window offers extremely useful features to ensure that you get the most out of your Verilog simulation and debug experience. You have the ability to watch multiple signals, ports, or components. You can also hover over variable names to see their value, and move quickly between the tree and the editors to locate definitions.


Use the **Text Slide In** command to insert a block of text or code
Color coded text for improved readablity
Use the **Find** command to locate a specific line of code
Tbsram.v
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Undo Ctrl-Z
Find Ctrl-F
Text Slide In...
Text Slide Out...
Add to Project
Add/Remove Breakpoint
Save Ctrl-S
Save As
Rename
Print Ctrl-P
Breakpoints
INS
Row: 13 Line: 13 Col: 12
Right click menu - Right mouse click in any editor window to view a pop-up menu of additional commands.
Green arrow indicates the current line being executed

```
`timescale 1ns/1ps
module tbsram(CSB,WRB,ABUS,DATABUS);

input CSB;
input WRB;
input [15:0] ABUS;
inout [7:0] DATABUS;

reg  [7:0] DATABUS_driver;
wire [7:0] DATABUS = DATABUS_driver;
reg [7:0] ram[0:65535];

always @(CSB)
  begin
    if (CSB == 1'b0)
      if (WRB == 1'b0)
        ram[ABUS] = DATABUS;
      else
        DATABUS_driver = ram[ABUS];
    else
      DATABUS_driver = 8'bzzzzzzzz;
  end

endmodule
```

Status Bar

The status bar on the VeriLogger is easy to access, and will ensure that you always know what state your Verilog simulation is in.


Double click to edit signal and simulation properties
Simulation Inactive
Simulation status window displays information about the current simulation run
Quick Help bar - gives tips on the functionality of buttons, commands, and objects

Task:

1. Implementation of Basic Logic Gates(i.e. AND, OR,XOR) Using Verilog
    a. Draw the circuit diagram
    b. Paste the code or screenshot of the code
    c. Paste the screenshot of timing diagram