EE 5327 VLSI Design Laboratory

Lab 1 - Verilog Simulation

PURPOSE: The purpose of this lab is to introduce you to gate, behavioral and dataflow Verilog models. You will compile and simulate three models for a full adder and a model for a four-bit adder using the Synopsys Verilog tool suite – **VCS** [1] and **Verdi** [2].

Setup

1. These lab manuals will guide you step-by-step in learning how to use the CAD tools.
2. Each lab will be devoted to learning one or more of the tools.
3. This lab deals with simulation.
4. Type `mkdir labs` to make a directory labs in your home directory.
5. Type `cd labs` to enter the directory labs.
6. Type `mkdir lab1` to make a directory lab1 in the directory labs.
7. Type `cd lab1` to enter the directory lab1.
8. Download files to `lab1` directory.

Compile and Simulate Files Using Command Line

9. Use the following VCS commands to compile and simulate Verilog files. The VCS compiler will generate an executable file `simv` and two folders (`csrc`/ and `sim.daidir`/) which contain object files and libraries.

   vcs Is the command that starts the Verilog compiler.

   -Mupdate Is a compile-time option. Compile-time options control how VCS compiles your source code. There are also run-time options that control how VCS simulates your design. This compile-time option specifies incremental compilation and to update the makefile. Incremental compilation is compiling only the modules that have changed since you last compiled the source file. Using this compile-time option now, even though you have never compiled any of the design's modules before is worthwhile because this option also tells VCS to create a subdirectory in your current directory named csrc; in that subdirectory are object files, and in some cases C or assembly intermediate files, and files that VCS uses to determine if it needs to compile a module over again. The makefile contains commands that VCS uses to generate object files, sometimes C or assembly files, and build the executable that you simulate. Over-writing the makefile with a new one ensures that VCS does not use a makefile that is out of date for your design. We recommend that you always use the ***-Mupdate*** compile-time option.

9.1. Compile and simulate gate model

```
vcs –Mupdate fa_st.v fa_gate.v
./simv
```

9.2. Compile and simulate dataflow model

```
vcs –Mupdate fa_st.v fa_df.v
./simv
```

9.3. Compile and simulate behavioral model

```
vcs –Mupdate fa_st.v fa_bh.v
./simv
```

Compile and Simulate Files Interactively Using DVE

10. Use the following commands to start an interactive simulation.

vcs Is the command that starts the Verilog compiler.

-debug_access+all Is used at compile time to selectively enable the required debug capabilities in a simulation. The ***+all*** switch enables all debug options. This option is needed to generate the files supported by Verdi debug.

-kdb Is used by the Verdi platform in its internal synthesis technology to recognize and extract specific structural, logical, and functional information about the design and stores the resulting detailed design information in the KDB (Knowledge DataBase).

-gui This *runtime*-time option tells the VCS-created executable, simv, to start an interactive simulation via the Verdi gui.

Compile and simulate the gate model:

```
vcs –Mupdate -debug_access+all -kdb fa_st.v fa_gate.v
./simv -gui
```

10.1. Compile and simulate the dataflow model:

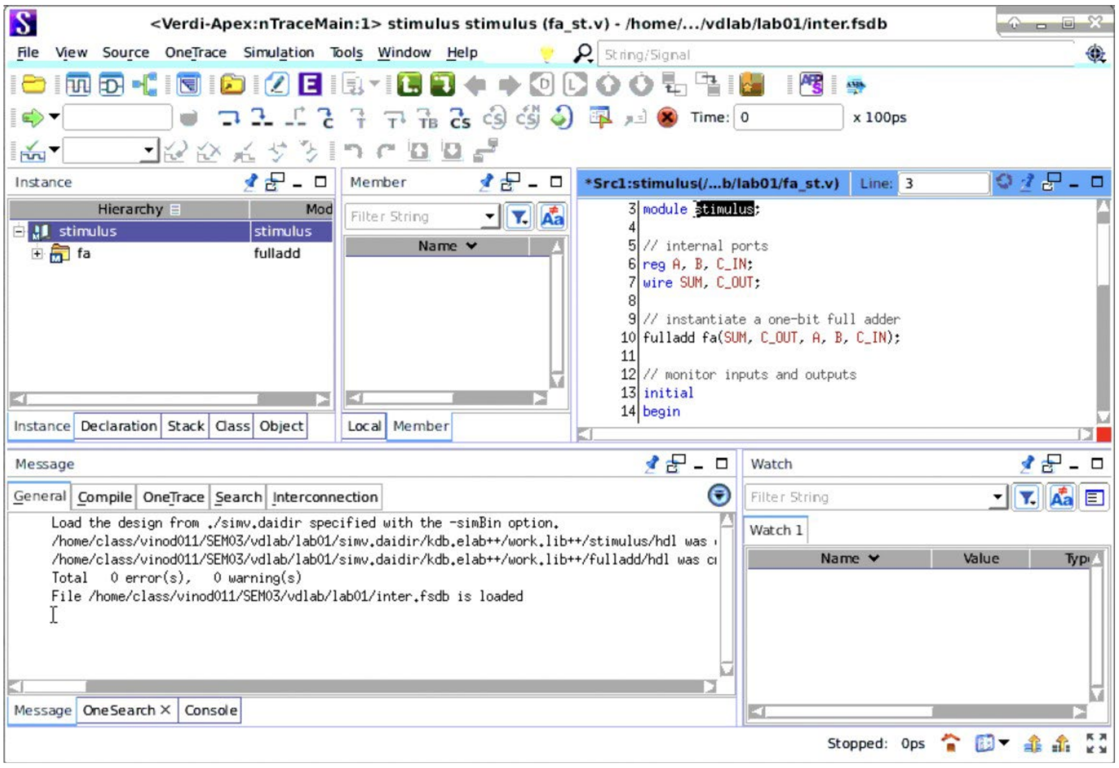
```
vcs –Mupdate -debug_access+all -kdb fa_st.v fa_df.v
./simv -gui
```

10.2. Compile and simulate the behavioral model:

```
vcs –Mupdate -debug_access+all -kdb fa_st.v fa_bh.v
./simv –gui
```

Note: Steps `11->16 below` are to be applied to each of the steps `10.1->10.3`.

11. The popped-up window is main Verdi Interactive window that looks like this.

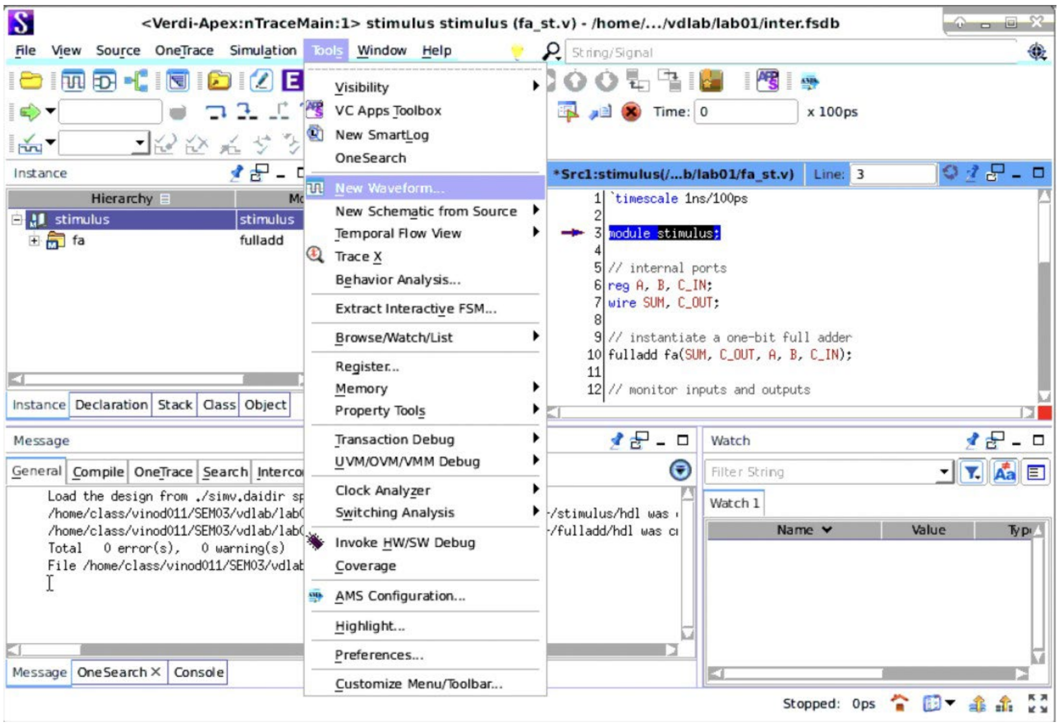


<Verdi-Apex:nTraceMain:1> stimulus stimulus (fa_st.v) - /home/.../vdlab/lab01/inter.fsdb
File View Source OneTrace Simulation Tools Window Help
Time: 0 x 100ps
Instance
Hierarchy
Mod
stimulus stimulus
fa fulladd
Member
Filter String
Name
*Src1:stimulus(/...b/lab01/fa_st.v) Line: 3
Instance Declaration Stack Class Object
Local Member
Message
General Compile OneTrace Search Interconnection
Watch
Filter String
Watch 1
Name Value Type
Message OneSearch Console
Stopped: 0ps

```
module stimulus;

// internal ports
reg A, B, C_IN;
wire SUM, C_OUT;

// instantiate a one-bit full adder
fulladd fa(SUM, C_OUT, A, B, C_IN);

// monitor inputs and outputs
initial
begin
```

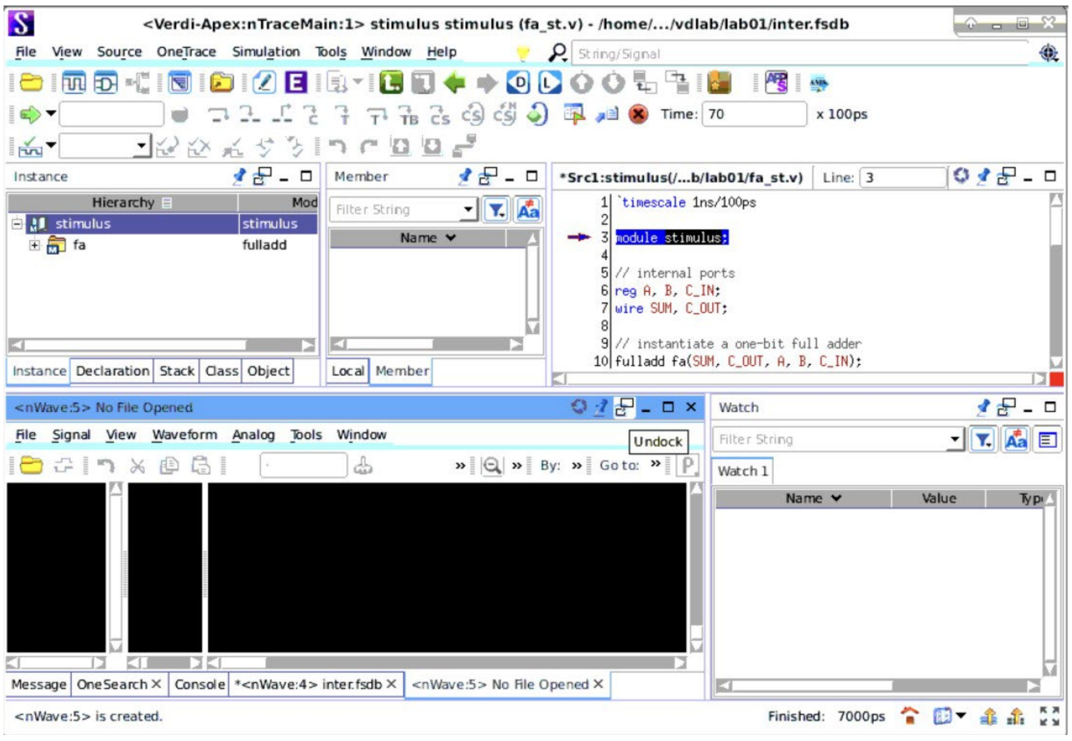
```
Load the design from ./simv.daidir specified with the -simBin option.
/home/class/vinod011/SEM03/vdlab/lab01/simv.daidir/kdb.elab++/work.lib++/stimulus/hdl was
/home/class/vinod011/SEM03/vdlab/lab01/simv.daidir/kdb.elab++/work.lib++/fulladd/hdl was c
Total    0 error(s),    0 warning(s)
File /home/class/vinod011/SEM03/vdlab/lab01/inter.fsdb is loaded
```

12. Click on `Tools -> New Waveform` as shown. The nWave waveform viewer window will open up.



```
`timescale 1ns/100ps

module stimulus;

// internal ports
reg A, B, C_IN;
wire SUM, C_OUT;

// instantiate a one-bit full adder
fulladd fa(SUM, C_OUT, A, B, C_IN);

// monitor inputs and outputs
```

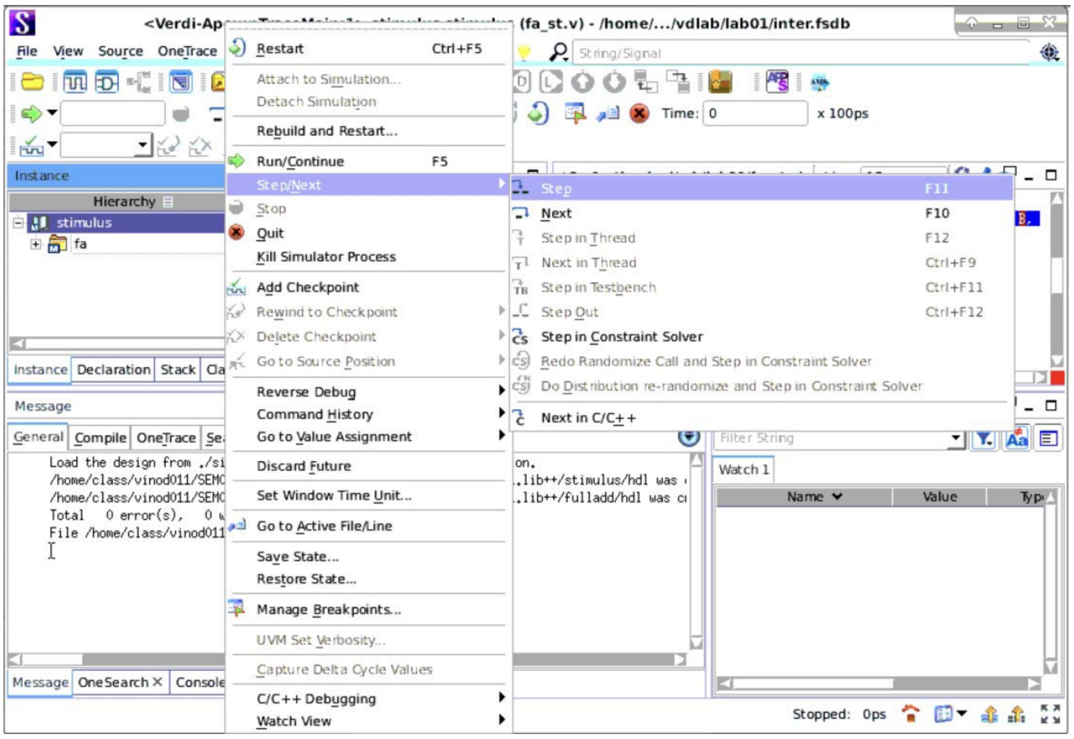
```
Load the design from ./simv.daidir sp
/home/class/vinod011/SEM03/vdlab/lab
/home/class/vinod011/SEM03/vdlab/lab
Total    0 error(s),    0 warning(s)
File /home/class/vinod011/SEM03/vdlab
```


<Verdi-Apex:nTraceMain:1> stimulus stimulus (fa_st.v) - /home/.../vdlab/lab01/inter.fsdb
File View Source OneTrace Simulation Tools Window Help
Time: 70 x 100ps
Instance
Hierarchy
Mod
stimulus
stimulus
fa
fulladd
Instance Declaration Stack Class Object
Member
Name
Local Member
*Src1:stimulus(/...b/lab01/fa_st.v)
Line: 3
<nWave:5> No File Opened
File Signal View Waveform Analog Tools Window
Undock
By:
Go to:
Watch
Watch 1
Name
Value
Message OneSearch Console *<nWave:4> inter.fsdb <nWave:5> No File Opened
Finished: 7000ps

```
`timescale 1ns/100ps

module stimulus;

// internal ports
reg A, B, C_IN;
wire SUM, C_OUT;

// instantiate a one-bit full adder
fulladd fa(SUM, C_OUT, A, B, C_IN);
```

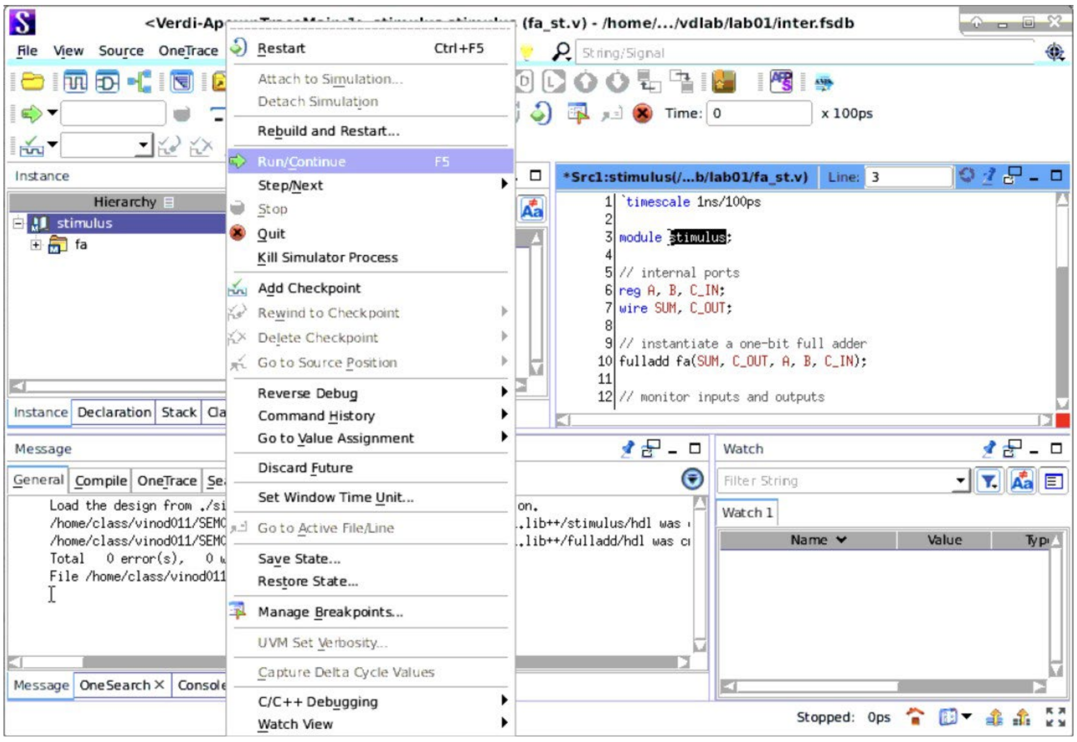
<nWave:5> is created.

13. In the main Verdi window, the right pane has the source Code view currently associated with the selected Hierarchy. If you expand the '+' symbol on the 'stimulus' in the Hierarchy and select the adder to be simulated, then the code contents for adder module will be seen. You can select the signals form the source code and drag to the WaveForm Viewer.
14. To begin the simulation, set the Time Step to 10 ps (i.e., the drop-down menu just below '`Simulation`', upper-left hand side) then click on `Step/Next` (while noting the associated symbol for '`Step`') and then click on `Step`. This will advance the simulation 10 ps. You may modify this step number as needed. Now hit the same '`Step`' symbol on the top tool bar of the Verdi window to continue stepping the simulation. Finally, if you have included code in your simulation testbench to halt (or break) the simulation you may hit the '`Start/Continue`' button to reach the end without further stepping. Without a simulation break (typically a dedicated `initial begin end` that waits a predetermined amount of time before invoking a $finish), if you happen to encounter a code race condition then your simulation could run for hours and never advance a single time step. So, it is best to include one, especially as your coding becomes more complex.



```
Load the design from ./si
/home/class/vinod011/SEMC
/home/class/vinod011/SEMC
Total    0 error(s),    0
File /home/class/vinod011
```

```
on,
.lib++/stimulus/hdl was
.lib++/fulladd/hdl was c
```

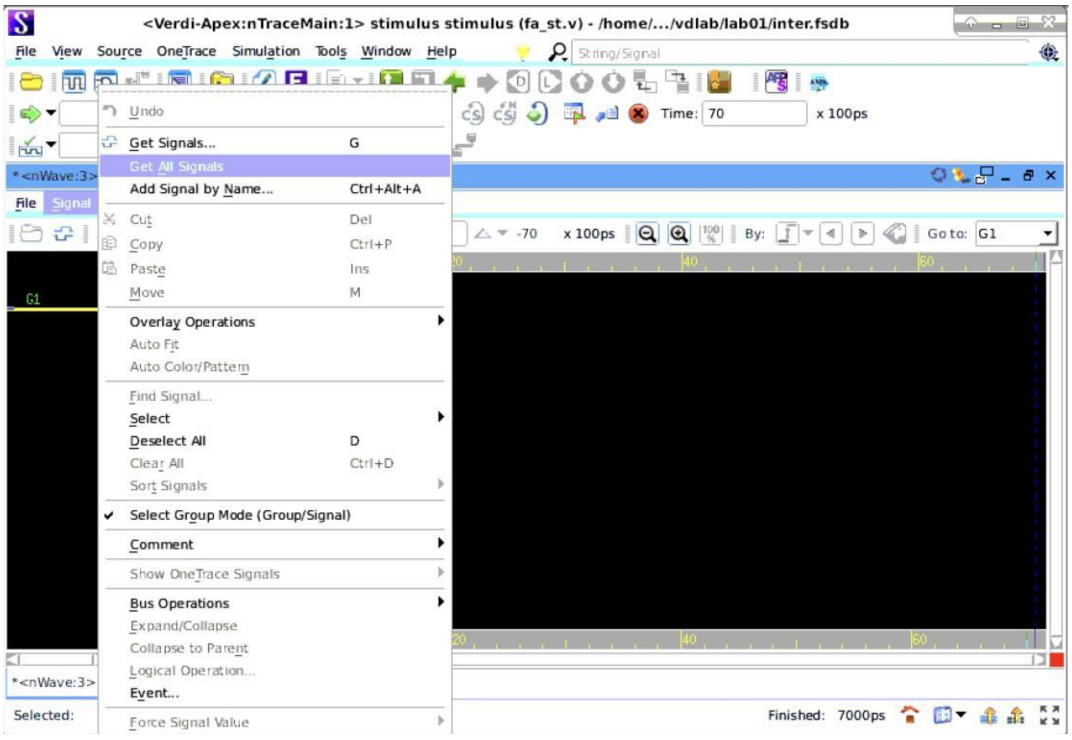


```
`timescale 1ns/100ps

module stimulus;

// internal ports
reg A, B, C_IN;
wire SUM, C_OUT;

// instantiate a one-bit full adder
fulladd fa(SUM, C_OUT, A, B, C_IN);

// monitor inputs and outputs
```

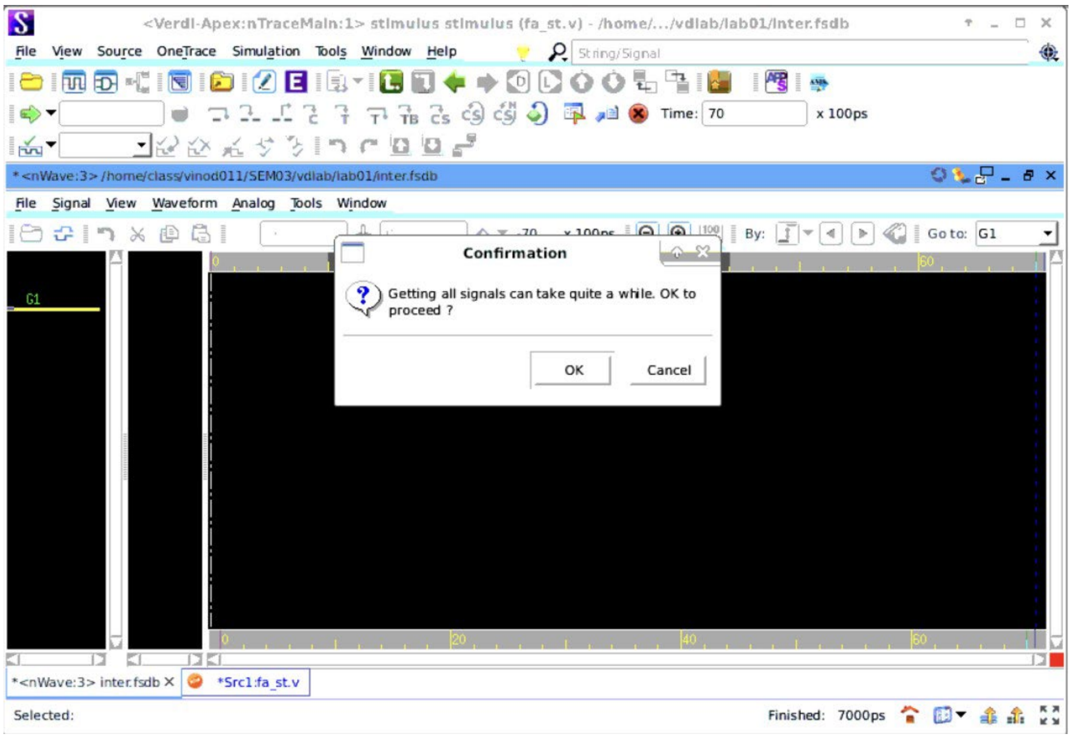
```
Load the design from ./si
/home/class/vinod011/SEMC
/home/class/vinod011/SEMC
Total    0 error(s),    0
File /home/class/vinod011
```

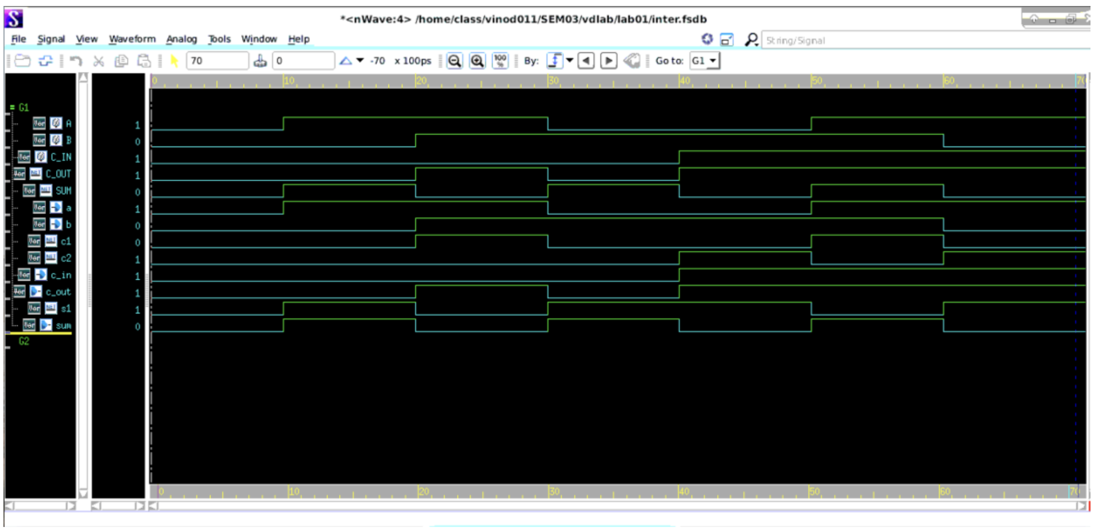
```
on,
.lib++/stimulus/hdl was
.lib++/fulladd/hdl was c
```

15. Aso, all the signals can be selected by clicking on `Signal -> Get All Signals.`



16. Press OK on the confirmation page that comes next. The signals will be loaded onto the window.


<Verdi-Apex:nTraceMain:1> stimulus stimulus (fa_st.v) - /home/.../vdlab/lab01/inter.fsdb
*<nWave:3> /home/class/vinod011/SEM03/vdlab/lab01/inter.fsdb
File Signal View Waveform Analog Tools Window
Confirmation
Getting all signals can take quite a while. OK to proceed ?
OK
Cancel
*<nWave:3> inter.fsdb
*Src1:fa_st.v
Finished: 7000ps


*<nWave:4> /home/class/vinod011/SEM03/vdlab/lab01/inter.fsdb
File Signal View Waveform Analog Tools Window Help
G1
A
B
C_IN
C_OUT
SUM
a
b
c1
c2
c_in
c_out
s1
sum
G2

17. Printing a wave. In the waveform window, you can click on `File/Print` and specify the begin time and end time of the print-out in the `nWave Options` tab of the dialogue box that opened. You can print to the VLSI Lab printer (set up to be `vlsilab`) or you can print to a file as a Postscript or Image file. If your waveforms appear congested, then you may also need to modify the Time Step before you print them.

As noted above, Steps `11->16` are to be applied to each step `10.1->10.3` before moving on to step 17.

18. **Task: First design and then compile and simulate your own four-bit adder (either a ripple or carry look-ahead adder). Ensure that your testbench will adequately check your design.**
    Note: A two-bit adder Verilog file is included for reference on the class web page.
    The command to compile the file is:
    vcs –Mupdate –debug_access+all -kdb twobitfa.v fa_xx.v (ie: 'xx' for any 1-bit full adder module).
    Then type:
    ```
    ./simv -gui
    ```
    to start the interactive simulation.

    Provide: The code for your adder and testbench, as well as snapshots of the beginning and end of your simulation (w/count values legible).

    Also, answer the following questions:

    1. Give the names of one command and three switches that have been used in Lab 1.

2. For the names you listed in question 1, explain what each command or switch does.

Optional: Use the VCS/Verdi documentation to determine how to extract the line coverage your testbench is providing and give the line coverage report.

References

[1] VCS MX/VCS MXi User Guide from Synopsys Inc.

[2] Verdi User Guide from Synopsys Inc.

NOTE: Three methods to obtain the VCS/DVE documentation are listed below:

a) Locate all non-synthesis Synopsys PDF documents via
```
ls $VCS_HOME/doc/UserGuide/pdf
```
You will note many useful documents here, not only on VCS/DVE but others addressing coverage (line/functional) and assertions (SystemVerilog /Vera).

b) Invoking `vcs –doc` gets you to VCS-only documentation in HTML browser.

c) Invoking `dve` then clicking '`Help/Help Contents`' in the resulting DVE gui gets you to both VCS and DVE documentation in HTML browser.