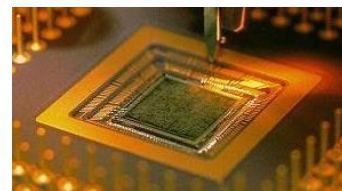


ECE 464 / ECE 564

Tutorial : Verilog Simulation And Synthesis



PLEASE DO NOT CUT AND PASTE COMMANDS FROM THE TUTORIAL. THESE LEAD TO ERRORS. YOU ARE ADVISED TO TYPE THEM OUT INSTEAD.

Part A: VERILOG SIMULATION

1. Learning Objectives

- Get familiar with NCSU computing environment
- Get familiar with Mentor Graphic's Modeslim® and the environment setup for simulation
- Learn how to debug a design using waveform invocation, viewing, formatting and storage.

2. NCSU Computing Environment

On Campus:

By using Linux machines, you can run all tools relative this course by accessing EB2 1014 or EB3

For your information, if you load any tools in any linux machine. However, in that case, tools are run on that machine. This could slow down your simulation time. So, if you want faster simulation, we recommend you “log-on” NCSU linux server cluster. Note that you must use -X instead of -x for ssh command.

```
shell> ssh -X UNITY_ID@grendel.ece.ncsu.edu
```

Since we have 18 servers, if you want specify the server, you could do

```
shell> ssh -X UNITY_ID@grendel1.ece.ncsu.edu
```

....

```
shell> ssh -x UNITY_ID@grendel18.ece.ncsu.edu
```

You could check detailed server information when you log-on any grendel server. For detail information on “ssh” command, put “ssh --help” on the shell

Please see

<https://go.ncsu.edu/ece-linux>

FastX and mobaxterm are the preferred options. A less capable option is Xwin32.

Top of Form

Home:

- First, Run VPN for Windows and Mac OS.

<https://vpn.ncsu.edu>

For more info: <https://oit.ncsu.edu/campus-it/campus-data-network/vpn/>

- Windows machine. Please install Fastx or Mobaxterm. Fastx can be found in the software catalog <https://it.engr.ncsu.edu/software/catalog/> Further instructions are found in the resources section of the class Moodle page and at <https://go.ncsu.edu/ece-linux>

If you have an older PC you can still use Putty and Xwin32 but I strongly urge you to try mobaxterm or fastx first.

If you prefer to use Putty and Xsin 32:

1. Install Putty and Xwin32

<https://it.engr.ncsu.edu/software/catalog/>

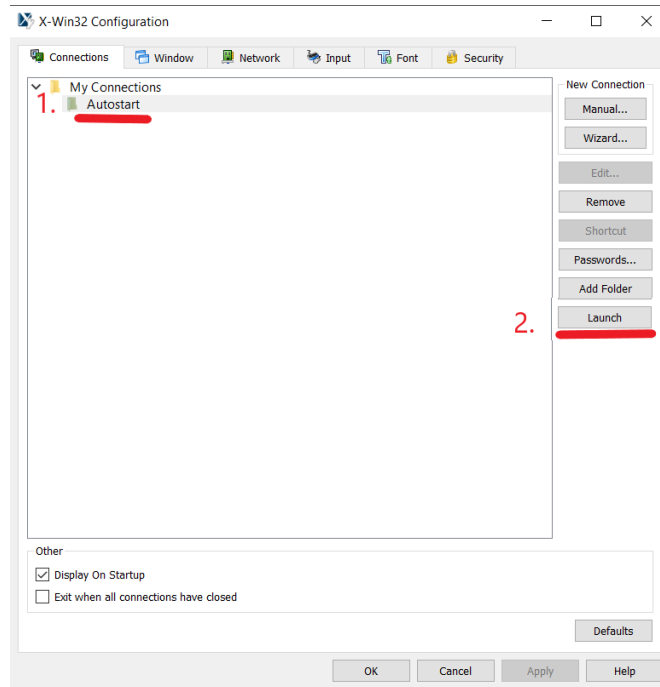
X-Win32 provides PuTTY with the GUI for things like “gedit”.

An easy method is to install FastX only.

<https://it.engr.ncsu.edu/software/catalog/fastx/>

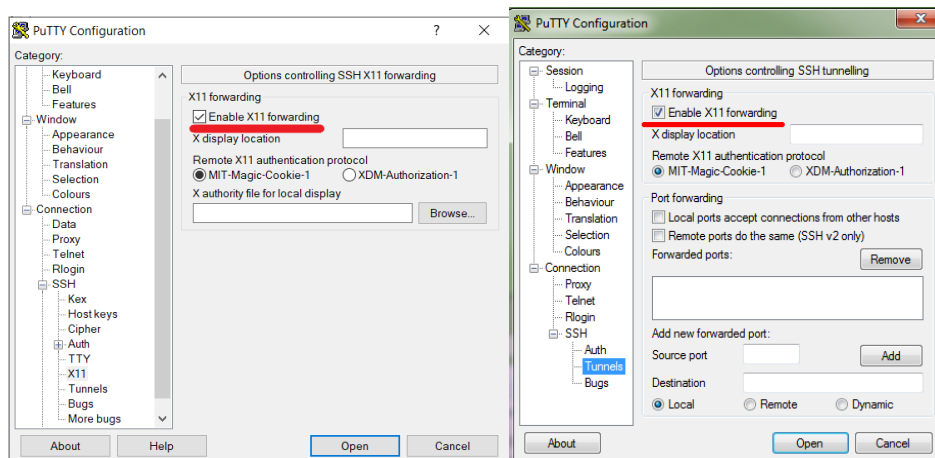
2. Run Xwin32

Click ‘Autostart’ under ‘My Connections’, then click the ‘Launch’ button on the right hand side of the window.



3. Run Putty and connect on campus linux cluster, “remote.eos.ncsu.edu”
Please make sure you enable X11 forwarding option as a figure below.

While you do some simulation, leave putty running.



For older versions of PuTTY X11 options are in the Tunnels branch of SSH, and for newer versions it has its own dedicated branch.

- Mac. Please install Fastx. Fastx can be found in the software catalog <https://it.engr.ncsu.edu/software/catalog/> Further instructions are found at <https://go.ncsu.edu/ece-linux>

The following alternative is left in the tutorial but fastx will be better.

Mac is similar to an on-campus linux machine. However, you may have to install an “X Server” application such as XQuartz. Note that you must use -X instead of -x for ssh command.

<https://www.xquartz.org/releases/XQuartz-2.7.11.html>
<https://it.engr.ncsu.edu/services/remote-access/>

please make sure you put your unity ID as
shell> ssh -X [UNITY_ID@remote.eos.ncsu.edu](https://it.engr.ncsu.edu/services/remote-access/)

For more information on remote access please check out:
https://tools.wolftech.ncsu.edu/support/index.php/Remote_Access

NOTE: As of 2022, we are phasing out the use of the “add” system as used in AFS in favor of “module” as used in NFS. While old practices may continue until the end of working year, use of module for the research is highly encouraged.

2.1 MobaXterm: The preferred computing environment for Windows

I strongly suggest using MobaXterm for Windows users to connect to the remote Linux machine. Mac users could use FastX as Mobaxterm is not provided for MacOS. This could be downloaded from the following website:
<https://mobaxterm.mobatek.net/download.html>

2.2 Transferring files from your Local Machine to the Linux server

Windows:

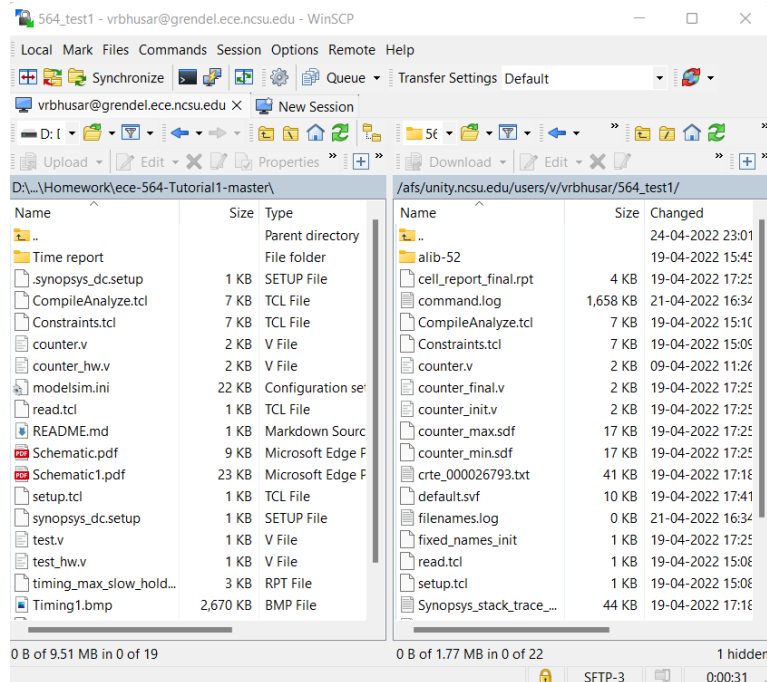
For transferring the files from your local machine and to the linux server and vice versa, you could use the following options:

1. Use WinScp:

Winscp is one of the most convenient ways to transfer files back and forth. This application is freely available on the following website where step-by-step guide for installation is provided:

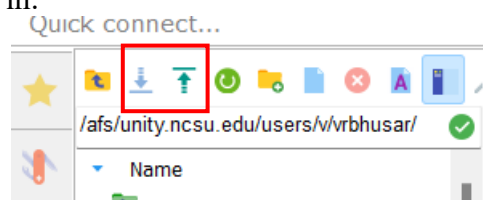
https://winscp.net/eng/docs/guide_install

Once you install it, you can simply drag and drop files back and forth and the interface would be as shown below:



2. Download and upload files via MobaXterm:

MobaXterm also allows you to transfer files between the local and Linux server. However, it does the process pretty slower than the other options. With larger files, it takes up too much time. In case, you choose to download or import files, you could find this option as shown below which appears on the left on your MobaXterm Interface when you log in:



3. Use remote server extensions on VS Code:

This is by far the most convenient and fastest option to edit your files on the remote server or even transfer the files from Linux to local and vice versa. If your favorite editor is VS Code, I would recommend you to install the following extension and follow the step-by-step guidelines to set up a connection between two environments. This certainly would help you to neatly oscillate between the environments! Following are couple of helpful extensions to make your coding in Verilog experience better.

Extension for sftp:

<https://marketplace.visualstudio.com/items?itemName=Natizyskunk.sftp>

Extension for Verilog:

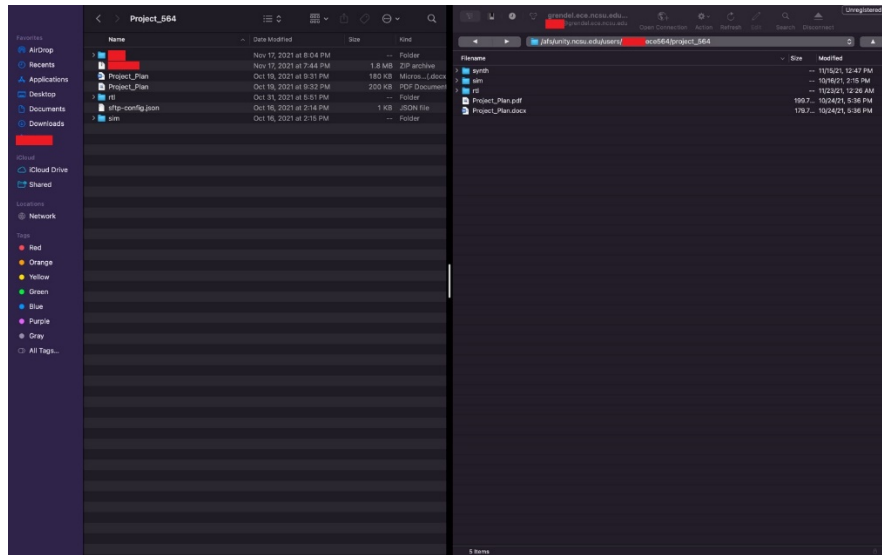
<https://marketplace.visualstudio.com/items?itemName=mshr-h.VerilogHDL>

(Also provides limited support for linting)

MAC:

1. Use CyberDuck:

Since WinScp service is not provided for Mac, CyberDuck is an alternative for transferring files between MacOS and Linux.



2. Use remote server extensions on VS Code:

VS Code also provides similar services on MacOS with the SFTP and various other extensions. You could refer to the information above provided in the Windows section.

3. Modelsim Set-up

The environment in which you will run the Modelsim simulator, which is called Design Sim and the waveform viewer is a complex one. It is strongly recommended that you do each design in a new directory. Let us get started with these two commands in the command prompt:

```
> mkdir lab1
> cd lab1
```

With the new practice of using **module**, it is much easier to add a particular application to our system:
Following commands would come in handy to play around with the module command:

```
> module avail // Loads all the modules that are available to you.
```

```
> module help modelsim/2021.2 // To obtain any further information on this.
```

```
> module load modelsim/2021.2 // Adds a module
```

```
> module list // Lists all the currently loaded modules
```

```
> module unload modelsim/2021.02 // Removes the modelsim application from the environment
```

Note: There are shortcuts available via `lmod` and it could save some time to type in the commands!

Note: You have to LOAD these applications every time you log into your Grendel server.

Functionality	Command	Shortcut
List available modules	<code>module avail</code>	<code>ml av</code>
Loads application	<code>module load modelsim/2021.2</code>	<code>ml modelsim/2021.2</code>
List loaded modules	<code>module list</code>	<code>ml list</code>
Info on the modules	<code>module help modelsim/2021.2</code>	<code>ml help</code>
Unloads application	<code>module unload modelsim/2021.2</code>	<code>ml - modelsim/2021.2</code>

- For any further information on the modules, you could refer to the following links:

<https://lmod.readthedocs.io/en/latest/>

<https://go.ncsu.edu/ece-linux>

<http://modules.sourceforge.net/>

4. Simulating a sample Verilog file

Download `counter.v` and `test.v` from the github tar vball. Another alternative is to download these files on local and transfer them to the linux remote machine via WinScp. Compile these files by typing:

```
> vlog counter.v
> vlog test.v
```

The result of this compilation is a note that shows `test_fixture` as the top-level module.

Note here that you can compile the source files for simulation in any order that you please. This is an advantage in that you can compile only the files that you modified. They can also be run within a GUI, which will be discussed in the next section.

5. Viewing the waveforms

To view the waveform, type `vsim` with gui access option.

```
> vsim -voptargs=+acc test_fixture &
```

This builds an instance of design hierarchy. You will see the appearance of the GUI as shown in figure 1. The GUI allows you to select signals for viewing. When you are simulating your own designs, make sure that there are no error messages shown in the transcript.

You can thus see the gui SimVision loaded and two windows will pop up: **Console and Design Browser 1**.

Make sure to run `vsim` from the shell (terminal) and in the same directory in which the files are stored. In Unix “&” means run in the background. If you type the `vsim` command into the Modelsim GUI drop the “&” – otherwise Modelsim will be looking for a file with that name.

6. Design Hierarchy

Verilog allows you to build modules that contain other modules. You might recall that we instantiated a copy of the module `counter` within the module `test_fixture` (look within `test.v`), to enable us to test it. Figure 1 and 2 show the gui representation.

Referring to `test.v`:

```
counter u1 (clock, in, latch, dec, zero);
```

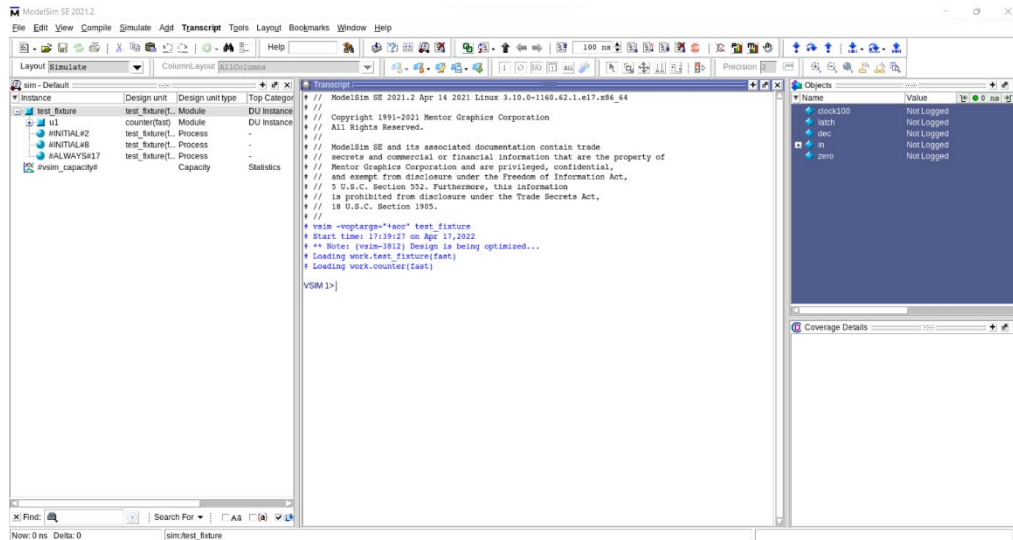



Figure 1: Modelsim SE 2021.2 (ready to simulate)

Name	Value	Kind	Mo	Now
clock100	1'hx	Regi...	Internal	
latch	1'hx	Regi...	Internal	
dec	1'hx	Regi...	Internal	
in	4'hx	Pack...	Internal	
zero	1'hx	Net	Internal	

Figure 2: Ports

You might recall that we had declared the following signals inside `test_fixture`:

```
reg          clock100;
reg          latch, dec;
reg  [3:0]   in;
wire         zero;
```

Thus within the instance `test_fixture`, we should be able to find the signals declared within that module and a reference to the `u1` instance of the module `counter`.

To this end, let us look at the contents of the GUI window. Note the presence of the design browser window and the objects window. At the top of the workspace window under Scope, you will see the `test_fixture` design unit. This is the logical top level of the simulation. Given that the counter is instantiated within the `test_fixture` as `u1`, you will notice the presence of the instance `u1` under `test_fixture`. The hierarchy of the design is captured by virtue of a series of buttons with + on them. To go

down the hierarchy you would need to click on the + to expand it and view the contents of the design unit.

The objects window lists the objects (inputs, outputs, internal nets, parameters, etc), corresponding to the design unit that is highlighted in the workspace window. The instance u1 has been highlighted by clicking on it (see figure 1). It is interesting to note that when the test_fixture is highlighted we see clock, latch, dec, in and zero, while on highlighting u1 we see clock, latch, dec, in, value and zero. This is along expected lines when it comes to the nets expected within each level of hierarchy.

To view waveforms, you could do the following:

- Right-click on the **Waveform** icon.

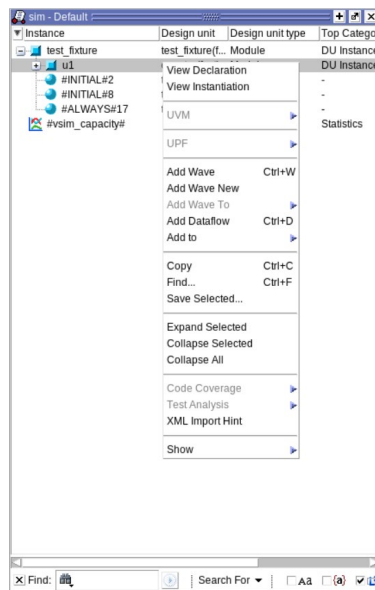


Figure 3: Waveform icon option

Figure 4 shows the waveform window before simulation.

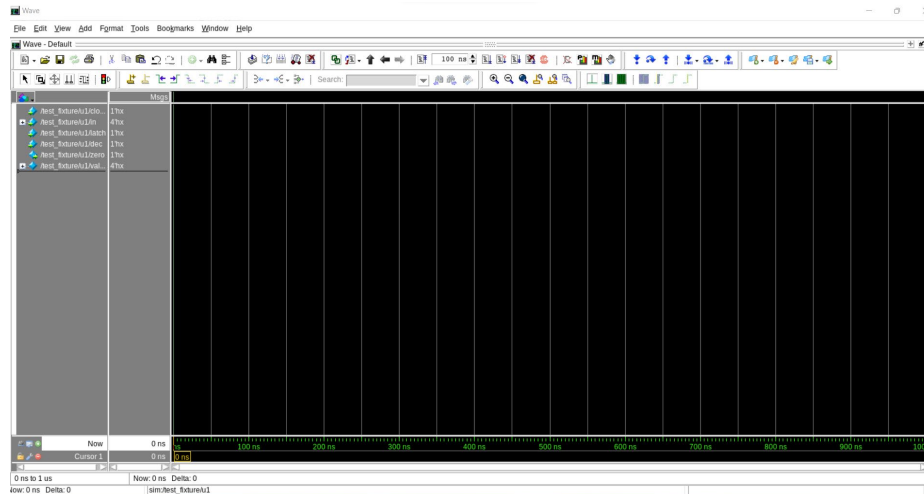



Figure 4: Just before the simulation

Further, to view waveforms, display all the nets in the objects window by right clicking on the objects window and doing right click as show above.
Or highlight all the signals of relevance to you by keeping the **ctrl** key pressed and clicking all the signals of interest once. Once this is done, you can drag the selected signals to wave pop-up.

NOTE: The window would generally result in a window that would be docked with the rest of the GUI. It is easiest to view the wave in a free moving window for the waves. To do this, “undock” the waveform window by hitting the  button at the top of the waveform window.

In addition to this, it would be noteworthy that the running of simulation leads to the logging of only the signals that have been sent to the waveform window. Thus, if you later wanted to view another signal, you would have to re-run the simulation after adding it to the waveform window. If you want to capture all the possible signals in the design, you would be best served by doing the following at the prompt of the GUI

```
VSIM #> log -r *
```

On the flip-side, this would lead to an increase in the time required for simulation.

There are two options with which you can run the simulation:

- i. Click on the **Run** icon or click on **Simulate** → **Run** to generate the signal waveforms for our example Verilog counter circuit as shown in Figure 5.
- ii. Or type “run” on the console, and you could also provide a specific time period through which you want to run the simulation.

e.g: **VSIM > run 130ns**

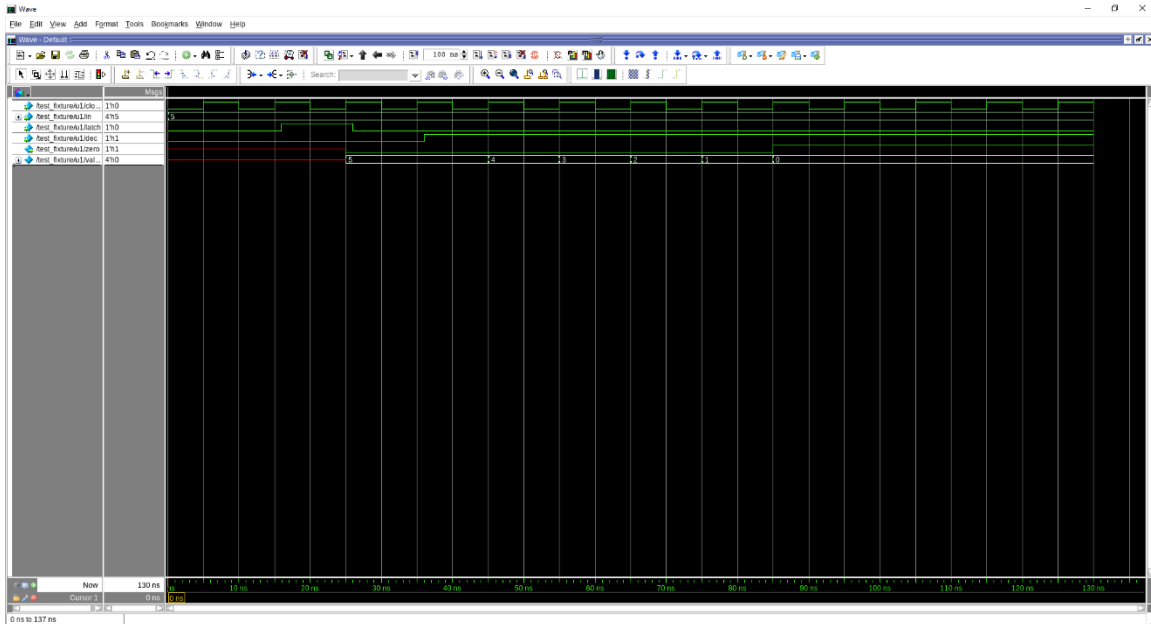


Figure 5: After running

To scale the waveform so that it fits entirely in the window, use the “Zoom out fully on x-axis” icon. It looks like an equals (=) sign. You can also play around other zoom options to analyze the waveform better.



Figure 6: Zoom options

Let us now attempt to make a change in the testbench to make the initial load of the value register 7 and then re-run the simulation. The idea here is to note that we do not need to leave the GUI based environment to do the necessary compilation. In fact, all of the previous `vlog` and `vlib` commands could have been run within the GUI by doing **>vsim &** to re-invoke the GUI and then running all the necessary commands. Coming back to the issue at hand, do the following:

- Open `test.v` and change the value assigned to `in` from `4'b0101` to `4'b0111` and save the file. This is going to be the value loaded initially into the `value` register.
- In the `vsim` GUI, type the following **VSIM#> vlog test.v**. This compiles the edited file within the GUI environment.

c. Restart the simulation that is presently running by doing a

```
> restart -f  
> run 130ns
```

At this point, you should be able to note the modified initial loading of the value register and the time when zero goes high.

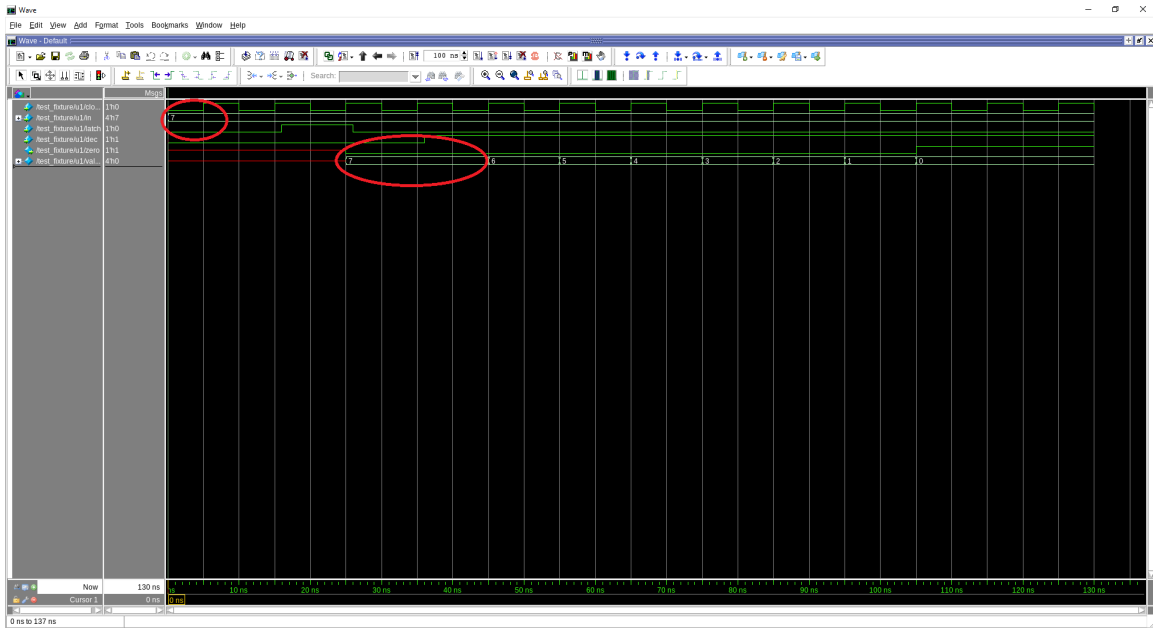


Figure 7: After modifying the value of register 7

For more information useful for design simulation, tool usage and work optimization, refer to Appendix A and http://www.eda.ncsu.edu/wiki/Tutorial:Modelsim_Tutorial.

[ASIDE]: For your information, you can type `>vsim -novopt -c <your top module>` if you wish to run your simulation without using waveforms (i.e. in console mode).

Part B: Synthesis with SYNOPSISYS

1. Learning Objectives

- Learn how to use the basics of the Synopsys synthesis tools (Design Vision™)
- Learn how Synopsys can be used to identify common coding problems.
- Practice the complete design cycle, from specifications through synthesis.

2. Establishing the Correct Environment and Launching Design Vision

It is recommended that you do each lab or design in a new directory (preferably different from the directory you used for Part A).

First copy `.synopsys_dc.setup` to this directory from the class website. This file is the basic setup file (as the name suggests) that sets up the environment, tool and user variables for a given synthesis run. **It is important to create a copy of `.synopsys_dc.setup` in every directory from which you plane to run synopsys.** Otherwise, it will not be able to find the libraries needed for compilation and you will encounter UIO-3 error (which will be discussed further in the appendix).

Additionally, please make sure that there is no `.synopsys_dc.setup` file in your home directory. This is important given that this file is called along with the file in the present directory. Finally, you will see that `.synopsys_dc.setup` is renamed into `synopsys_dc.setup` (without `.`) if you copied the file from Windows PC. Choose your favorite text editor and rename the file correctly.

Now type:

```
> module load synopsys/2021
> design_vision
```

At the time of tool invocation, this file is sourced to setup. Therefore, if you copy the setup file to the directory after invocation of the synthesis tool, you will have to exit and restart Design Vision.

You should see the Design Vision GUI appear on the screen. You will also note that the prompt changes to `design_vision >` on the command prompt. Following new and improved GUI for Design Vision will be opened:

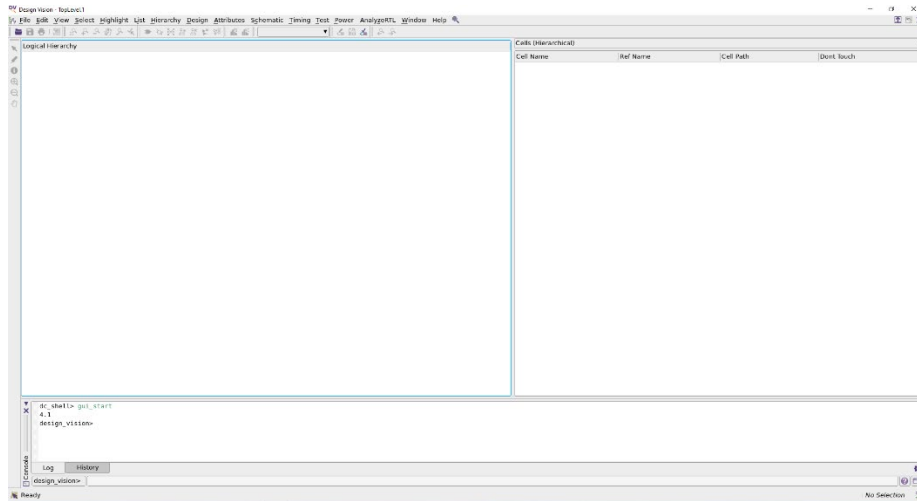


Figure 8: DesignWare Interface

3. Synthesizing a sample Verilog File

Download `counter.v`, `setup.tcl`, `read.tcl`, `Constraints.tcl` and `CompileAnalyze.tcl` from the web page. Open `counter.v`, `setup.tcl`, `read.tcl`, `Constraints.tcl` and `CompileAnalyze.tcl` in your favorite editor. Review what is written there, especially in the comments. Then in the command line, enter the following:

```
design_vision> source setup.tcl
design_vision> source read.tcl
```

- i. `setup.tcl`: Sets up the variables that will be used to distinguish different runs of synthesis.
- ii. `read.tcl`: Reads and compiles all the Verilog files to load the design.

You could check the information that loads after running the commands on logs of your terminal as well as the DesignVision GUI.

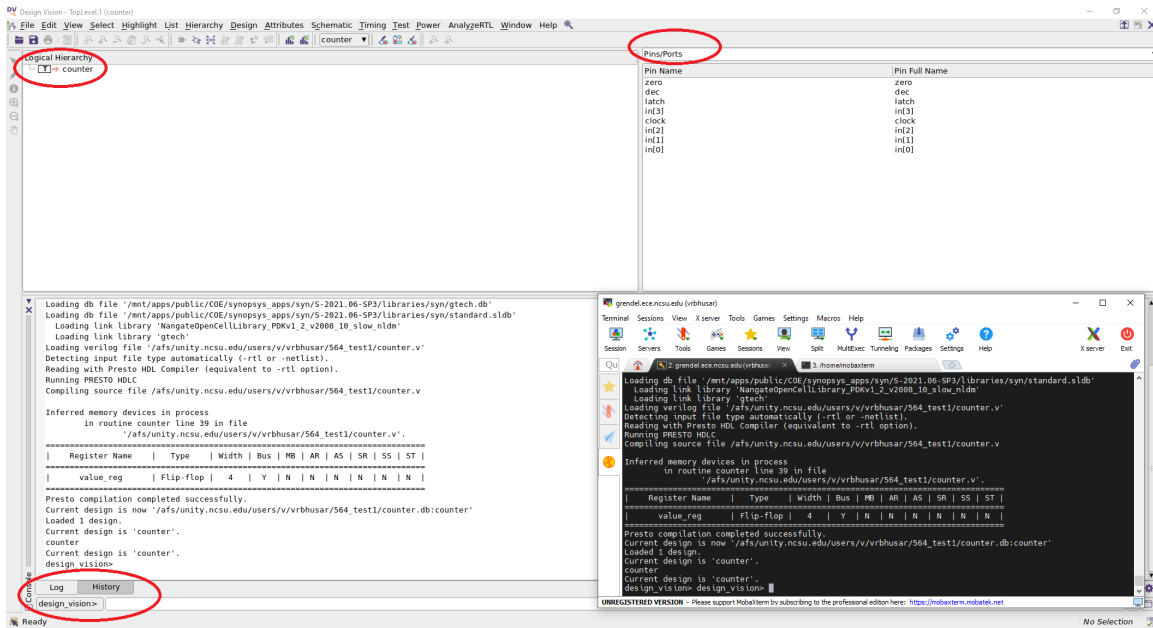


Figure 9: After sourcing setup.tcl and read.tcl

Now, we will try to analyze some of the aspects pertaining to your homework or project and playing around Design Compiler.

i. Flip flops that are specified in this design according to your Verilog Code.

At this time it would be apropos to note the presence of the History and Log tabs (Point A: circled in the figure above). The History tab would enumerate all the previous commands executed and provides the ability to re-execute any of them using the "Execute" tabs that appears the History tab when it is clicked.

Also, note the presence of the top-level instance counter at the top left corner which corresponds to the hierarchy window (Point B circled in the Figure). Finally, note the window corresponding to Point C. The options in the list at the top of this window allow you to look at the different objects (nets, ports, pins, instances, etc). In this case we have enabled the viewing of the ports/pins of the counter. Following are the additional options in the list:

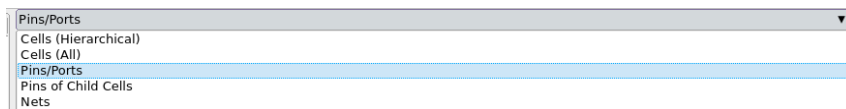


Figure 10: Options for objects selection

NOTE: Make sure to note the type of information that appears on the screen. Please play around with the tool and the information given to you. It makes issues much easier to debug down the line. **Any problems or unintended latches should be fixed before proceeding. These should also appear in the Log window.**

Next, let us run the synthesis on this design by typing:

```
design_vision> check_design
design_vision> source Constraints.tcl
design_vision> source CompileAnalyze.tcl
```

- a. The **check_design** command is useful to check for consistencies in the design and will produce a report of warnings and errors that could potentially prevent your design from compiling.
- b. **Constraints.tcl**: sets up the necessary constraints (input and output loads, clock period and skew, area constraint etc) on the design for valid synthesis.
- c. **CompileAnalyze.tcl**: runs the compilation for synthesis, fixes hold issues if any and generates all the necessary area and timing reports for this design.

You might encounter warnings (i.e. OPT-106 etc.). You should also check and take care of these warnings or errors in this stage. You can do this by running the **check_design** command again to corroborate the intended structure. Click the blue colored link at the end of the warning message at the Log message. The design_vision will show a pop-up message window explaining the warning or the error. You could ignore OPT-106 and UID-401 warnings, since the tool can handle these within current design flow. (see appendix A)

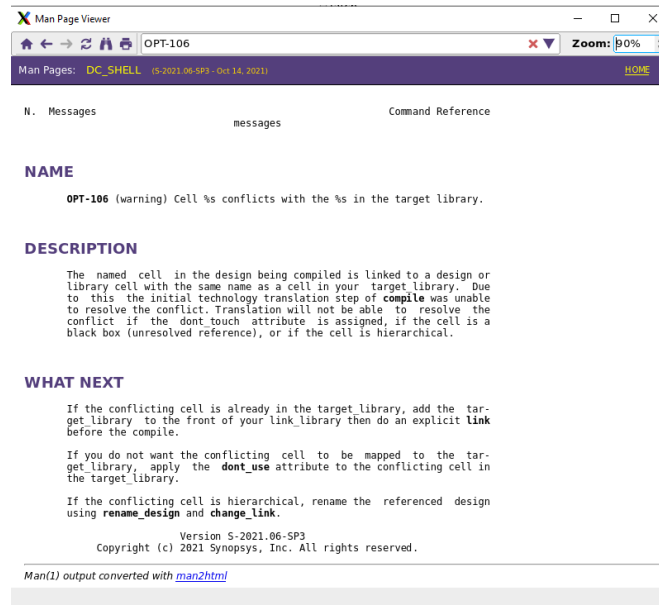


Figure 11: Help Tab

Pay special attention to the timing reports generated and check that there are no timing problems:

- timing_max_slow.rpt: the critical path delay of the circuit for the slowest process corner after the first compilation.
- timing_min_fast_holdcheck_tut1.rpt: The minimum delay through the circuit after the hold problems are fixed. The slack has to be met at the fastest corner of the design.
- timing_max_slow_holdfixed_tut1.rpt: The critical path delay of the circuit at the slowest process corner after hold problems have been fixed. This is of importance because there is a remote possibility of the logic inserted to fix the hold issues in the circuit actually changing the critical path delay of the circuit.
- cell_report_final.rpt: Contains all the top-level modules of your design and their respective area contributions in μm^2 along with the total area of your design.

The timing report timing_max_slow_holdfixed_tut1.rpt would look something like this:

```
Operating Conditions: slow Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm
Wire Load Model Mode: top

Startpoint: value_reg[2]
            (rising edge-triggered flip-flop clocked by clock)
Endpoint: value_reg[2]
          (rising edge-triggered flip-flop clocked by clock)
Path Group: clock
```

Path Type: max

Point	Incr	Path

clock clock (rise edge)	0.0000	0.0000
clock network delay (ideal)	0.0000	0.0000
value_reg[2]/CK (DFF_X1)	0.0000	0.0000 r
value_reg[2]/Q (DFF_X1)	0.5632	0.5632 f
U24/ZN (NOR4_X2)	1.1182	1.6814 r
U28/ZN (NOR2_X1)	0.1580	1.8394 f
U29/ZN (INV_X1)	0.2081	2.0475 r
U37/ZN (NOR3_X1)	0.1179	2.1654 f
U38/ZN (AOI22_X1)	0.3816	2.5469 r
U39/ZN (OAI21_X1)	0.1941	2.7410 f
value_reg[2]/D (DFF_X1)	0.0000	2.7410 f
data arrival time		2.7410
clock clock (rise edge)	10.0000	10.0000
clock network delay (ideal)	0.0000	10.0000
clock uncertainty	-0.0500	9.9500
value_reg[2]/CK (DFF_X1)	0.0000	9.9500 r
library setup time	-0.3352	9.6148
data required time		9.6148

data required time		9.6148
data arrival time		-2.7410

slack (MET)		6.8738

The values of importance in this case is the data arrival time (2.7410) and the slack available. The slack is the delay available after the setup time (if applicable), the clock uncertainty, external delays and the data delay is subtracted from the clock signal. **IT MUST BE MET**. Note that the setup time would come into the picture if both the start and end points of the critical path are on registers. For our projects or homework, it is always advisable to adjust the clock period in setup.tcl in such a way that the slack is **MET** and minimum as well.

[NOTE]You should not synthesize two designs in succession without exiting and restarting Synopsys between designs (two different synthesis runs of the counter design for example). It is allowed to read in multiple modules at the same time corresponding to hierarchy, if any, in a given design.

4. Using Synopsys to discover common coding problems

Synopsys read_verilog command does more than just parse the Verilog file. It checks the file for existence of flip-flops, latches, and common coding problems. Please refer to the class notes for more information.

```

design_vision> read_verilog counter.v
Loading verilog file '/afs/unity.ncsu.edu/users/v/vrbhusar/564_test1/counter.v'
Detecting input file type automatically (-rtl or -netlist).
Warning: Overwriting design file '/afs/unity.ncsu.edu/users/v/vrbhusar/564_test1/counter'. (DOB-24)
Reading with Presto HDL Compiler (equivalent to -rtl option).
Running PRESTO HDLC
Compiling source file /afs/unity.ncsu.edu/users/v/vrbhusar/564_test1/counter.v

Inferred memory devices in process
in routine counter line 39 in file
'/afs/unity.ncsu.edu/users/v/vrbhusar/564_test1/counter.v'.
=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
=====
| value_reg | Flip-flop | 4 | Y | N | N | N | N | N | N |
=====
Presto compilation completed successfully.
Current design is now '/afs/unity.ncsu.edu/users/v/vrbhusar/564_test1/counter.db:counter'
Loaded 1 design.
Current design is 'counter'.
counter
Current design is 'counter'.

```

Figure 12: read_verilog command

Common problems are identified by Synopsys as follows:

- **Unintentional latches.** Synopsys identifies all latches synthesized. E.g.

```

=====
| Register Name | Type |
=====
| D_reg | Latch | ...

```

Remove the latch as described in class and the text.

- **Incomplete Sensitivity Lists.** Synopsys identifies incomplete sensitivity lists, referring to them as “Timing Control Blocks”. E.g.

Warning: Variable 'D' is being read in routine counter line 24 in file 'counter.v', but does not occur in the timing control of the block that begins there. (HDL-180).

Complete the sensitivity list. If you use Verilog2001 and start all combinational logic with `always@(*)`, then this problem does not occur.


- **Unintentional Wired-OR logic.** Synopsys identifies situations where the same variable is assigned in two separate procedural blocks or continuous assign statements. E.g.

Warning: Variable 'foo' is driven in more than one process or block in file 'counter.v'. This may cause mismatch between simulation and synthesis. (HDL-220)

This warning identifies unintentional wired-OR logic.

5. Plotting schematic of the synthesized design

We can now preview the circuit that has resulted from synthesis. Right click on the counter instance in the hierarchy window and hit Schematic View on the list that pops up.

Alternatively, you can hit the Create Schematic of Selected Objects button  in the toolbar.

When you double click the “Counter” box in the Figure, you can see the inside of the box as shown in Figure . You can look into it with more details by clicking further on each aspect of schematic.

This results in the appearance of a preview of the schematic in another window, which can be maximized to a full window. To fit in the window, you can zoom either by hitting “f” on the keyboard or by using the zoom icons in the top toolbar.

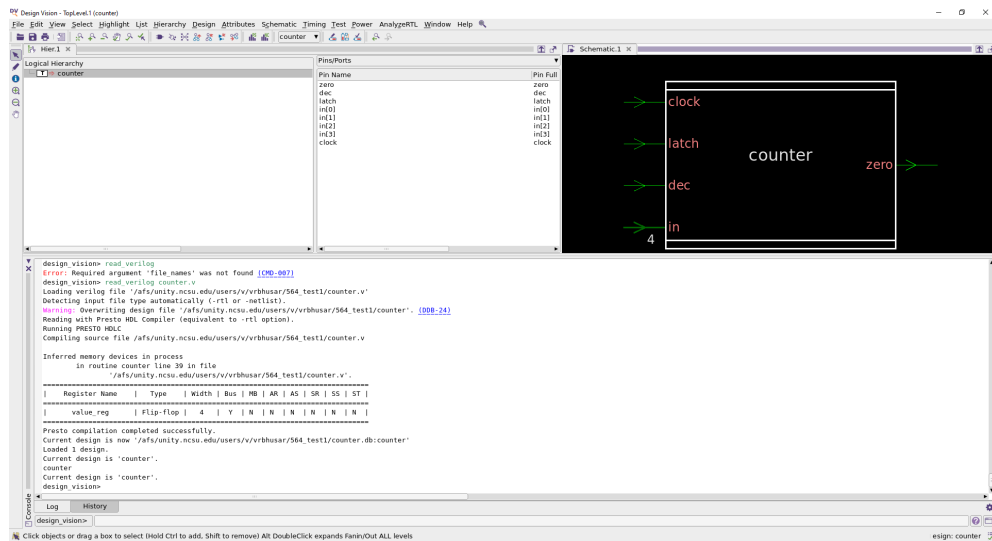


Figure 13: Schematic View

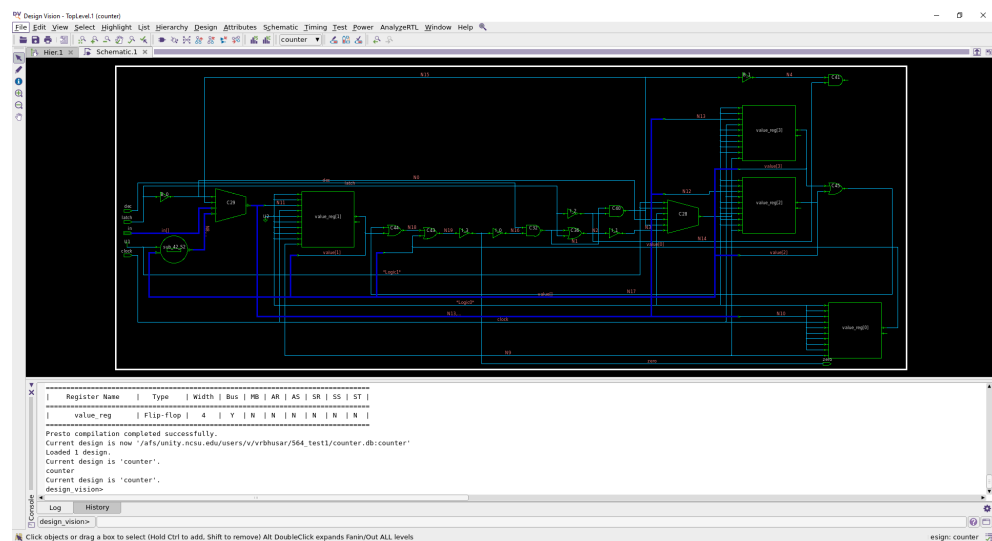


Figure 14: Detailed Schematic View

You must experiment a bit with this tool at this point. In particular, it is suggested that you familiarize yourself with the toolbar available to explore your design which is shown below. These can be used to determine the worst-case path, the distribution of delay along a path, the distribution of capacitance and load along a path etc. The results can be seen diagrammatically and/or in a histogram fashion. Click on one of the button for exploring further options:



6. Textual Interface

Synopsys also provides a text only interface. After adding synopsys, enter the command

```
dc_shell
```

In `dc_shell` you can enter commands and get textual responses. Ie. it gives you everything you get in the lower panes of `design_vision` as shown below:

```
[vrbbhusar@grendel29 564_test1]$ module load synopsys
[vrbbhusar@grendel29 564_test1]$ dc_shell

      Design Compiler Graphical
      DC Ultra (TM)
      DFTMAX (TM)
      Power Compiler (TM)
      DesignWare (R)
      DC Expert (TM)
      Design Vision (TM)
      HDL Compiler (TM)
      VHDL Compiler (TM)
      DFT Compiler
      Design Compiler(R)

      Version S-2021.06-SP3 for linux64 - Oct 14, 2021

      Copyright (c) 1988 - 2021 Synopsys, Inc.
      This software and the associated documentation are proprietary to Synopsys,
      Inc. This software may only be used in accordance with the terms and conditions
      of a written license agreement with Synopsys, Inc. All other use, reproduction,
      or distribution of this software is strictly prohibited. Licensed Products
      communicate with Synopsys servers for the purpose of providing software
      updates, detecting software piracy and verifying that customers are using
      Licensed Products in conformity with the applicable License Key for such
      Licensed Products. Synopsys will use information gathered in connection with
      this process to deliver software updates and pursue software pirates and
      infringers.

      Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
      Inclusivity and Diversity" (Refer to article 000036315 at
      https://solvnetplus.synopsys.com)

      Initializing...
      dc_shell> █
```

Figure 15: dc_shell interface

6. Synopsys Help

The necessary help for `design_vision` can be invoked using the Help tab. There are some excellent user guides and tutorials there. You might need to do an “add acread” to be able to view some of the pdf manuals that come with the tool. I suggest becoming familiar with them. If you want to access the pdf documents directly you can do so by invoking “sold” (Synopsys OnLine Documentation) on the command prompt.

APPENDIX A: ADDITIONAL TOOL INFORMATION

Working with Modelsim:

1. Working on the command prompt: It is useful to note that, if the GUI is to be avoided (debug using only text commands/generation of files/checking for compilation correctness.), the simulation can be invoked in the non-GUI mode by doing a `vsim -c`. This is sometimes a good alternative when X-windows proves troublesome or you do not have a connection strong enough to support a GUI based operation.

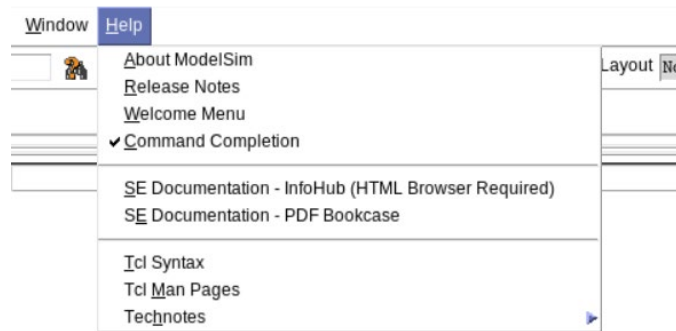
2. Compiling selectively: If you make a modification in just a few files, then you can compile just those files with a `vlog` (instead of the entire set of files you are working with), and just do a `VSIM#> restart -f` to restart the simulation. There is no need to quit the GUI.

3. Using .do files: An extremely useful feature in Modelsim is the presence of `.do` files. These are the equivalent of shell scripts for Modelsim. All the necessary compilation commands, simulation commands, waveforms etc can be stored within a `.do` file. An example `.do` file is provided on the EDA page. This allows you to avoid typing in all the commands manually. Instead, you can call this file within the `vsim` environment (GUI / no GUI) by doing a `VSIM #> do <filename>.do`. Of particular importance in working with `.do` files is working with waveforms.

a. Saving Waveforms: Once you have set up all the relevant waves on the waveform window, it is feasible in Modelsim to store the format of these waves in a `.do` file and restore the same set of waves the next time you come back to the simulator. This is done by saving all the waveforms by doing a `File□Save □` and saving the format as a `.do` file. The next time you invoke the simulator, MAKE SURE THAT THE DESIGN HAS BEEN LOADED using a `vsim <modulename>` for eg, `vsim -novopt test_fixture`,

after which you can open the waveform window and do a File □ Load □ <filename>.do to get the same set of waveforms as before.

3. **Modesim Help:** /afs/bp.ncsu.edu/dist/modelsim/docs where a choice of pdf or html documentation exists. You can also find the documentation under help tab on your ModelSim GUI:



Working with design_vision:

1. Reading and Writing intermediate designs: To be able to store the design with constraints at any given time it is a good idea to store it as a ddc file. This can be done by doing the following for reads and writes of the design

- a. Write: `> write -f ddc -hierarchy -o counter.ddc`
- b. Read: `> read_ddc counter.ddc`

Remember: you must set current_design to the top level module by doing (in this case): `> current_design counter`

2. Auto-completion: An excellent feature available in design_vision is the availability of command completion. Assume you are in the process of typing a command, say report_timing, and you are not too sure what the command is. You can type report_ and hit a tab and the command options for this beginning are listed. Moreover, if you want to know the options that come with this command, like report_timing -to <> -from <>, you can type "report_timing -" and hit the tab button and the options like from, to, through etc, would appear.

3. Ignorable Synthesis Warnings (See Appendix C):

Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)

Warning: Cell xxxx conflicts with xxxx in the target library. (OPT-106)

4. Man pages: All the necessary documentation needed for understanding the various commands that are used in the .tcl scripts can be found by


```
design_vision-xg-t> man <command> for example
design_vision-xg-t> man read_verilog
```

Help ☐ Man Pages on the GUI and searching for the command of interest

APPENDIX B: FREQUENTLY ENCOUNTERED Q & As

Problem #1:

I get the following two errors/warnings when I run "source Constraints.tcl":

```
Error: Could not read the following target libraries:
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db (UIO-3)
Warning: Can't read link_library file
'NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db'. (UID-3)
```

Solution:

.synopsys_dc.setup file not present in the directory that you are synthesizing in. Make sure that there is a "." at the beginning of the file.

Problem #2:

When I type the command >gv .ps (with the filename I specified) into the prompt in design_vision as shown in the tutorial, it tells me that 'gv' is not a recognized command. I also could not find 'gv' in the man pages.

Solution:

The gv command needs to be run on the UNIX/Linux terminal/prompt and not on the design_vision prompt.

Problem #3:

I have the .synopsys_dc.setup file in the synthesis directory but on calling design_vision, I get

```
Error: unknown command 'designer' (CMD-005)
Error: unknown command 'company' (CMD-005)
..
...
..
Error: unknown command 'edifout_netlist_only' (CMD-005)
Error: unknown command 'edifout_target_system' (CMD-005)
Error: unknown command 'edifout_instantiate_ports' (CMD-005)
..
```

Solution:

There is a .synopsys_dc.setup file in your home (K:/) directory that is over-riding the file present in the current directory. Please remove it.

Problem #4:

No matter which server I'm using, after I type the "> add synopsys" and "> design_vision" as the tutorial said, the command line just show "Initializing...", then "design_vision-xg-t> design_vision-xg-t>" and "+ Suspended (tty input) design_vision". And the design_vision does not start all the time.

Solution:

The reason you are getting the error is because you are invoking the tool by doing ">design_vision &". Please remove the "&" at the end.

Problem #5:

While doing the iterative compilation with different values of clock period, should we create each design in a separate directory? (Which implies that one has to restart design_vision in each directory.). If running each iteration in the same directory do we have to shut down and restart design_vision between iterations in the same directory?

Solution:

There is no need to do either. You must remain in the same directory and in the same session of design_vision while working on an incremental compilation with modified clocks. This is because the base design does not change. You attempt to compile with increasingly greater constraints on the compilation tool in an attempt to improve the performance of the synthesized netlist. This is done best by carrying over optimizations from one incremental compile to the next, which requires you to keep the same session going until you are satisfied with your results or see no further improvement.

APPENDIX C: Major vs. Minor Synthesis errors and what Warnings can be safely ignored

The most important thing is to run check_design after the compile step. Most synthesis issues flagged at this point have to be fixed. The check_design runs before the compile tends to flag more issues, though the major ones tend to stay.

This covers just a few of the errors and warnings Synopsys can report on. The full list can be found in alphabetical order in the following 8,000 page manual

<https://www.wolftech.ncsu.edu/manuals/synopsys/synopsys2015/Synthesis/synn.pdf>

Major Synthesis Issues:

Major synthesis issues include: Unintentional Latch, Timing Arcs, Wired-Or, Incomplete Sensitive Lists, No connection to Input Port(LINT-34, typically). Note, this is not a complete list but these are the most common ones.

Some less common major issues include

- LINT-34 (related to improper use of tri-state drivers);
- LINT-3 and LINT-58 undriven inputs (indicating a poorly structured design)
- LINK-3 about port mismatches are usually indicating bugs in your code
- LINT-38 multiple drivers is a warning about "wired-or" logic
- LINT-6 "input clock and reset drives wired logic" implies you are connecting clock and/or reset to something besides flip-flop ports

Minor Synthesis Issues:

- Unreachable default case
- Undeclared logic
- No reset

Ignorable Warnings:

- Output Port not connected or has no loads, e.g. LINT-2
- Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)
- Warning: Cell xxxx conflicts with xxxx in the target library. (OPT-106) . This usually happens on translation.
- A similar warning is (OPT-187) Warning: There are conflicts between cells in libraries
- Warning: Design 'hist_calc' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134) (This is usually the reset net which is not in a timing path).
- Warning: /afs/bp/dist/synopsys_syn/dw/sim_ver/DW02_cos_function.inc:360: Function 'DWF_cos' with non-empty body is mapped to 'COS_TC_OP'; body will be ignored. (VER-136)
- Warning: /afs/bp/dist/synopsys_syn/dw/sim_ver/DW02_cos_function.inc:374: Function 'cos' with non-empty body is mapped to 'COS_TC_OP'; body will be ignored. (VER-136)

- Warning: Design 'counter' inherited license information from design 'DW02_cos_8_8'. (DDB-74)
- Warning: Design 'counter' is being converted to a limited design. (DDB-75)
- Warning: Current design named counter is hidden. (UID-408)
- Warning: Verilog 'assign' or 'tran' statements are written out. (VO-4)
- Warning: signed to unsigned conversion occurs (VER_318)

- When using DesignWare, Lint 32 "pin connected to logic 0 or 1" might appear. This is OK, if this is going to an input signal in the DW module that it is safe to tie to a logic value permanently

LINT-1, LINT-2, LINT-28, LINT-29, LINT-31, LINT-52, LINT-99 are usually OK

LINT-3 and LINT-58 can be caused also by certain cells in the cell library and designware. However, you cannot assume that if you see these warnings - you still need to verify that your design is NOT causing them.

When using DesignWare, Lint 32 "pin connected to logic 0 or 1" might appear. This is OK, if this is going to an input signal in the DW module that it is safe to tie to a logic value permanently