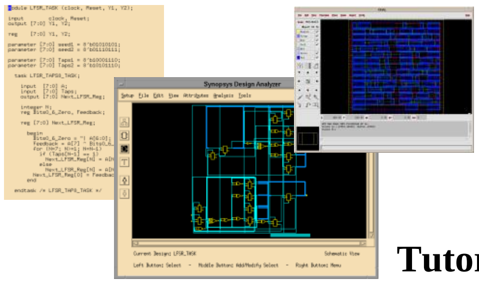
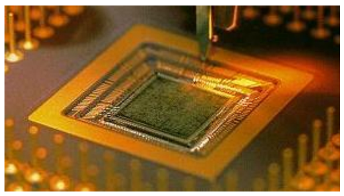




ECE 464 / ECE 520

Tutorial : Verilog Simulation And Synthesis

Revision History
July, 2002 : Rewritten from previous version (P. Franzon)
December, 2002 : Update for 2003 (P. Franzon)
December, 2003 : Update for Simvision (Ma, Franzon)
December, 2006 : General update (P. Franzon)
December, 2007 : Update for Modelsim (R. Jenkal /P. Franzon)
October, 2010 : Update for tool revisions (P. Franzon)
January, 2011: Update for tool/library revisions (W. Choi, P. Franzon)
January, 2012: Rewritten from previous version (W. Choi, P. Franzon)
February, 2013: Add Modelsim (Student Version) Guide (W. Zhao, P. Franzon)
January, 2015: Update for 2015(J.Park, P.Franzon)
September,2015: Update for NCSU computing environment (J.Park, P.Franzon)
September,2016:Updated for ssh command error and Mac environment(J.Park, P.Franzon)

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

Part A: VERILOG SIMULATION

1. Learning Objectives

- Get familiar with NCSU computing environment
- Get familiar with Mentor Graphic's Modelsim® 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 detail server information when you log-on any grendel server. For detail information on “ssh” command, put “ssh --help" on the shell

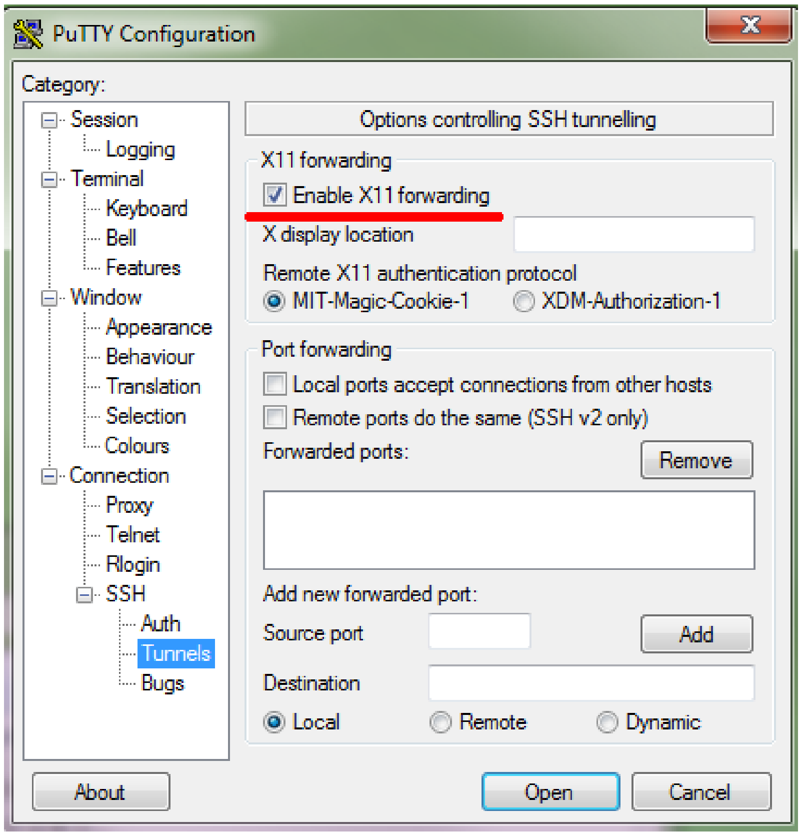
Home

- First, Run VPN for Windows and Mac OS.
  http://www.wolftech.ncsu.edu/support/support/Remote_Access

- Windows machine
  1. Install Putty and Xwin32
     http://www.eos.ncsu.edu/software/downloads/

  2. Run Xwin32
  3. Run Putty and connect on campus linux cluster, “grendel.ece.ncsu.edu”
     Please make sure you enable X11 forwarding option as a figure below.

     While you do some simulation, leave putty running.



Options controlling SSH tunnelling

X11 forwarding
[✓] Enable X11 forwarding
X display location
Remote X11 authentication protocol
[✓ MIT-Magic-Cookie-1] XDM-Authorization-1

Port forwarding
[ ] Local ports accept connections from other hosts
[ ] Remote ports do the same (SSH v2 only)
Forwarded ports:
Add new forwarded port:
Source port
Destination
[✓ Local] Remote Dynamic

- Mac

Mac is similar to on-campus linux machine. However, you may have to install "X Server" application such as XQuartz. Note that you must use -X instead of -x for ssh command.
https://www.xquartz.org/
http://www.eos.ncsu.edu/remoteaccess/mac.php

please make sure you put your unity ID as
shell> ssh -X UNITY_ID@grendel.ece.ncsu.edu

3. ModelSim Set-up

The Cadence environment in which you will run the Modelsim simulator and the waveform viewer is a complex one. I strongly recommend you do each design in a new directory. Let us get started with these two commands in the command prompt:

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

Before proceeding, copy the `modelsim.ini` file into your lab1 directory. This file sets up the necessary default environments for the Modelsim. Type the following two commands:

```
> add modelsim10.3b
> setenv MODELSIM modelsim.ini
```

You must type these commands every time you log in. Next:

```
> vlib mti_lib
```

This command creates the library into which all the design units (i.e. Verilog files) will be compiled. If the compilation crashes for some reason, you are advised to delete the mti_lib directory and re-create it as shown above.

Note: setenv is a tcsh command. If you are on a a machine running bash, you need to run "export" instead. i.e.

- *export MODELSIM = modelsim.ini*

4. Simulating a sample Verilog file

Download `counter.v` and `test.v` from the web page. 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 that the vlog compilation is compliant with Verilog2001, which has many advantages over Verilog-95.

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 the name of your top-level module. –novopt option turns off the optimization flow, which can cause some tool conflicts. & allows you to use other applications while you are using vsim.

```
> vsim -novopt test_fixture &
```

[ASIDE]: Another point of note with Modelsim is the ability to do run-time waveform viewing without having to create often HUGE vcd/db files (i.e. you don't need to create the vcd/db files using the `$dumpvars()-$dumpfiles()` or `$shm_open()-$shm_probe()` commands).

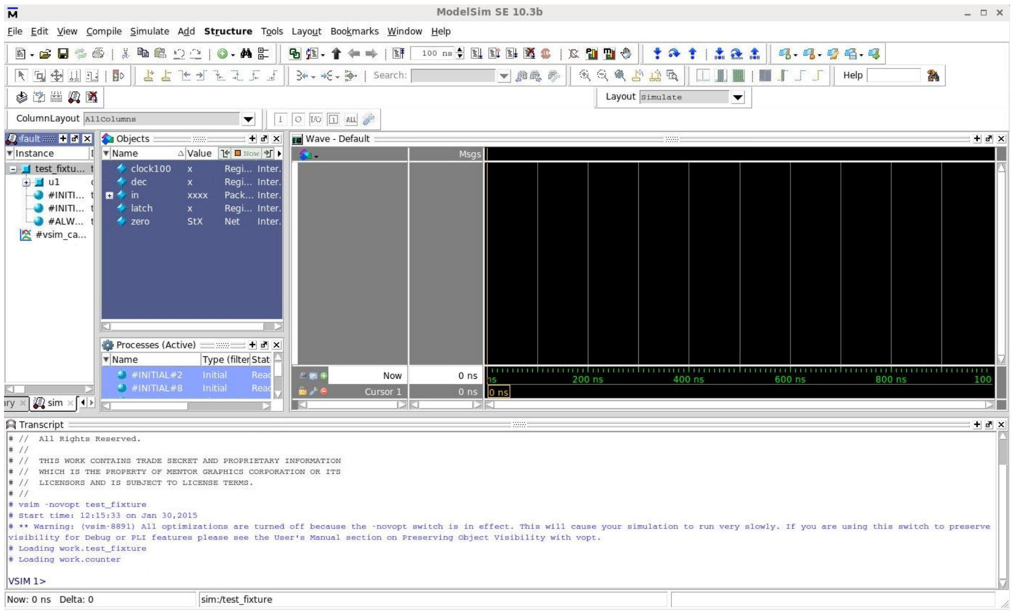
You will see the appearance of the GUI as shown in figure 1. The GUI allows you to select signals for viewing. When you are simulating your own designs, make sure that there are no error messages shown in the transcript.

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.

Referring to `test.v`:

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



```
# //  All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# //  WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# vsim -novopt test_fixture
# Start time: 12:15:33 on Jan 30,2015
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve
visibility for Debug or PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
# Loading work.test_fixture
# Loading work.counter

VSIM 1>
```

Now: 0 ns Delta: 0   sim:/test_fixture

Figure 1: VSIM (ready to simulate)

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`.

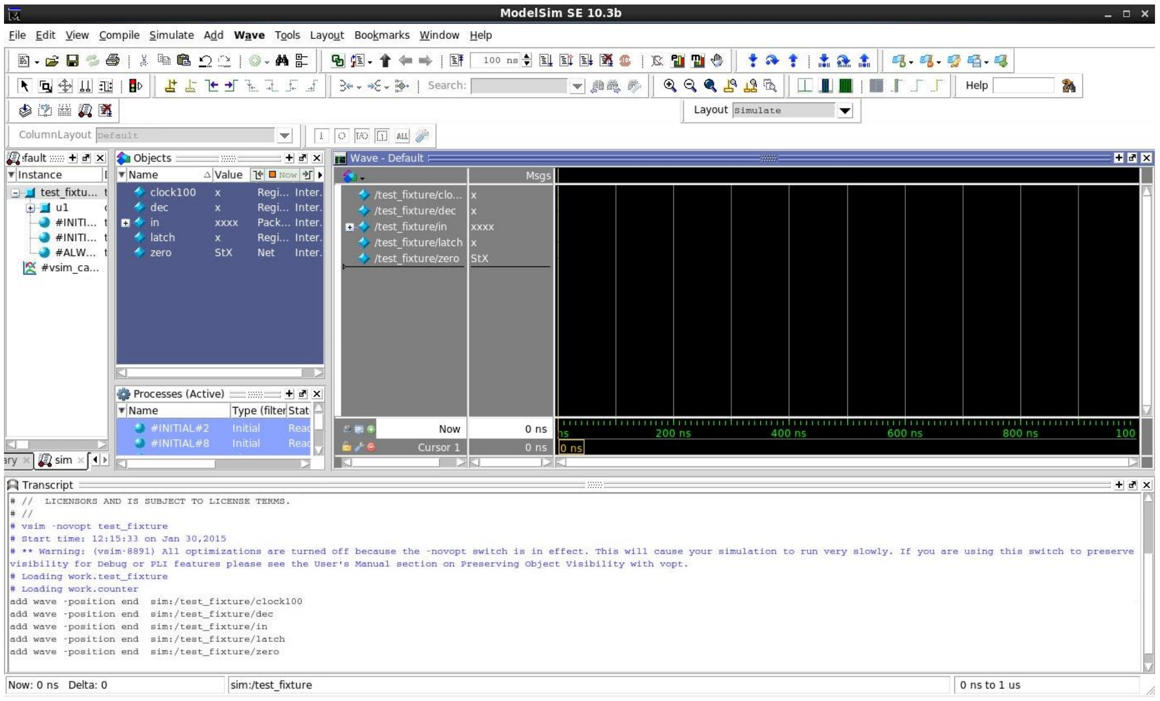
To this end, let us look at the contents of the GUI window. Note the presence of the workspace window and the objects window. The workspace contains all the design units within the `mti_lib` working library. At the top of the workspace window you will see the `test_fixture` design unit. This is the logical 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 `test_fixture` is highlighted we see `clock100, 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 have two options:

a. Display all the nets in the objects window by right clicking on the objects window and doing: `add to wave` → `Signals in Region` in the list that pops up.
b. Highlight all signals that are 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`.

Figure 2 shows the waveform window (with option a) before simulation.



```
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# vsim -novopt test_fixture
# Start time: 12:15:33 on Jan 30,2015
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve
visibility for Debug or PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
# Loading work.test_fixture
# Loading work.counter
add wave -position end  sim:/test_fixture/clock100
add wave -position end  sim:/test_fixture/dec
add wave -position end  sim:/test_fixture/in
add wave -position end  sim:/test_fixture/latch
add wave -position end  sim:/test_fixture/zero
```

Now: 0 ns Delta: 0 | sim:/test_fixture | 0 ns to 1 us

Figure 2: Just before the simulation

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.

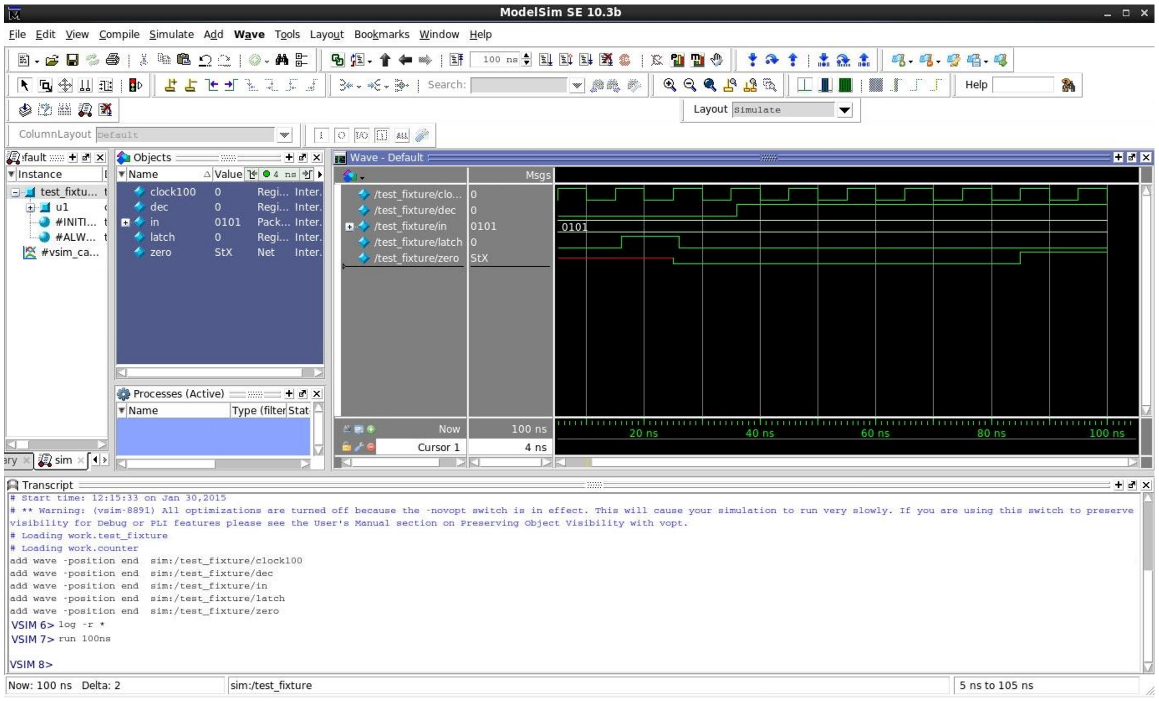
Before running the simulation it would be good to note that, by default, the running of the simulation leads to the logging of only the signals that have been sent to the waveform window. Thus, if you later wanted to view another signal, you would have to re-run the simulation after adding it to the 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.

We are now poised to run the simulation for this design. To this end, we would need to do the following at the GUI command prompt (say we are running only for 100ns). A preview of the waveform is shown in figure 3.

```
VSIM #> run 100ns
```



```
# Start time: 12:15:33 on Jan 30,2015
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve visibility for Debug or PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
# Loading work.test_fixture
# Loading work.counter
add wave -position end  sim:/test_fixture/clock100
add wave -position end  sim:/test_fixture/dec
add wave -position end  sim:/test_fixture/in
add wave -position end  sim:/test_fixture/latch
add wave -position end  sim:/test_fixture/zero
VSIM 6> log -r *
VSIM 7> run 100ns

VSIM 8>
```

Now: 100 ns Delta: 2 sim:/test_fixture 5 ns to 105 ns

Figure 3: After running 100ns

To view the waveform in its entirety you would need to fit the waveform in the window by wither hitting the ‘f’ key on the keyboard or by hitting the key to zoom to fit the waveform in the window.

[ASIDE]: To run this simulation to its completion we would need to do **`VSIM#> run -all`** where it will take the simulation onwards till a `$finish` is found. When a `$finish` is hit, the tool prompts a "`Are you sure you want to finish?`" to which you hit `No`. It is always advised to run the simulation to a controlled amount of time.

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 just a **`>vsim &`** to invoke the GUI and then running all the necessary commands**.** Coming back to the issue at hand, do the following:

a. 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.

b. 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.

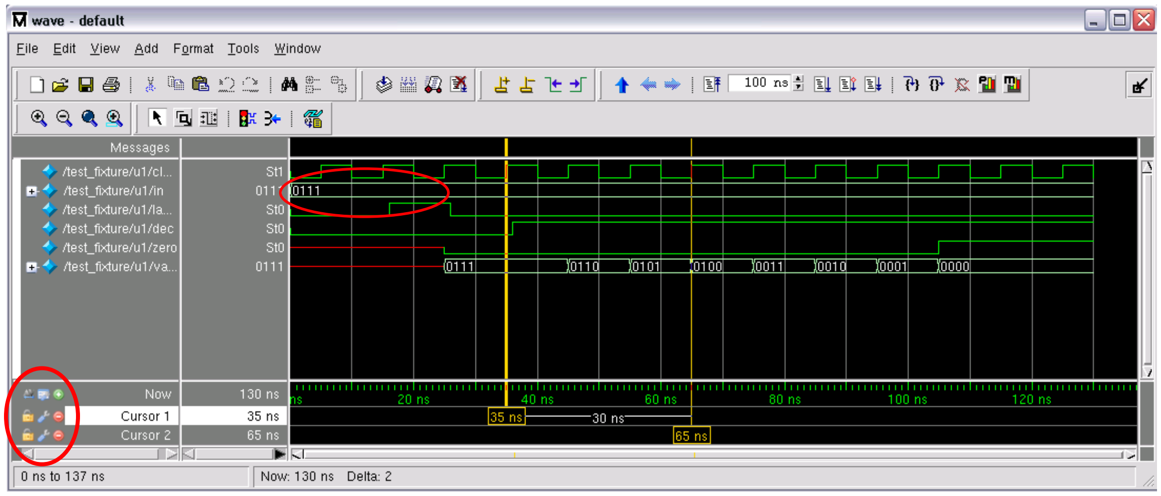


Figure 4: After modifying the value of register 7

Also note the presence of a set of tools (circled in figure 4), which can be used to create cursors (we have added a cursor in the above figure), lock the cursors, remove them, change the names of the signals from the full path to the short signal name etc.

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 SYNOPSYS

1. Learning Objectives

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

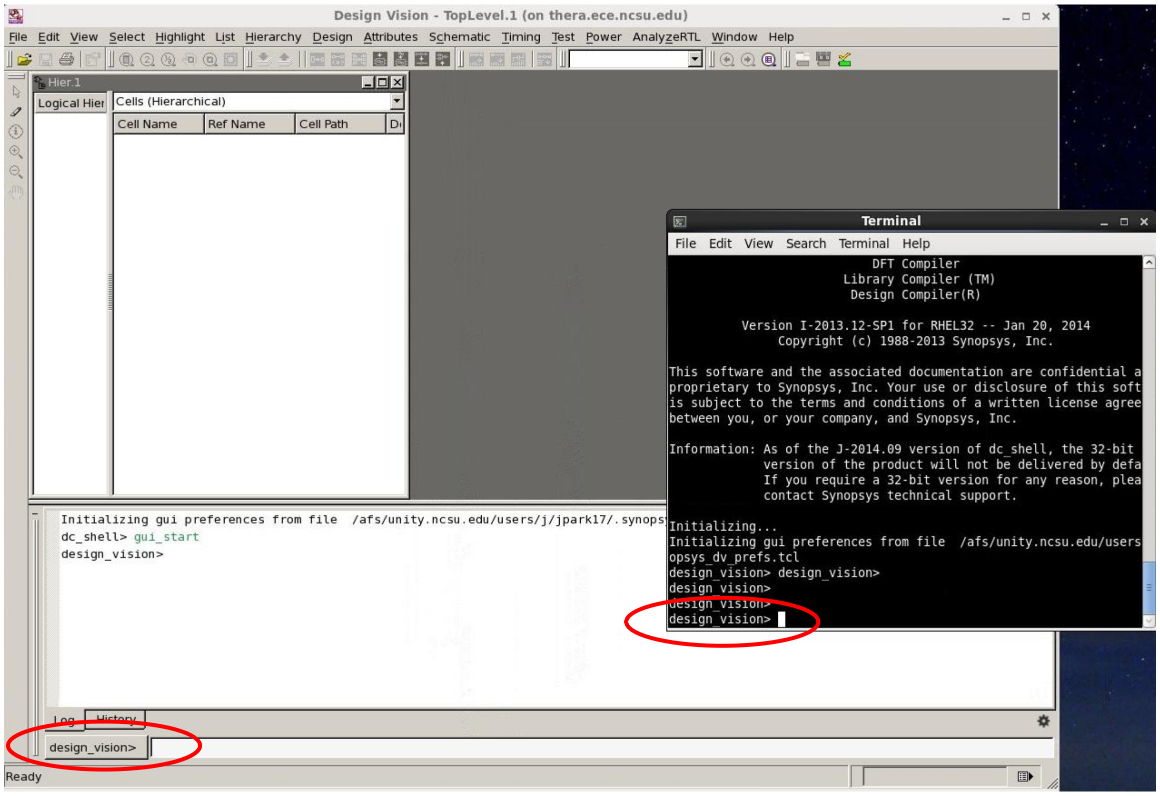
2. Establishing the Correct Environment and Launching Design Vision

I strongly recommend that you do each lab or design in a new directory (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 setups up environment, tool and user variables for a given synthesis run. **It is very important to create a copy of `.synopsys_dc.setup` in every directory from which you plan to run synopsys**. Otherwise, it will not be able to find the libraries needed for compilation. **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:

```
> add synopsys2015
> design_vision (Or > design_vision -64bit)
```

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. (the '-64bit' option would give you to run appropriate design_vision on 64bit machines)

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 (see figure 5).


Design Vision - TopLevel.1 (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.1
Logical Hier
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Log
History
design_vision>
Ready
Terminal
File Edit View Search Terminal Help

```
Initializing gui preferences from file  /afs/unity.ncsu.edu/users/j/jpark17/.synops
dc_shell> gui_start
design_vision>
```

```
                     DFT Compiler
                 Library Compiler (TM)
                   Design Compiler(R)

      Version I-2013.12-SP1 for RHEL32 -- Jan 20, 2014
           Copyright (c) 1988-2013 Synopsys, Inc.

This software and the associated documentation are confidential a
proprietary to Synopsys, Inc. Your use or disclosure of this soft
is subject to the terms and conditions of a written license agree
between you, or your company, and Synopsys, Inc.

Information: As of the J-2014.09 version of dc_shell, the 32-bit
             version of the product will not be delivered by defa
             If you require a 32-bit version for any reason, plea
             contact Synopsys technical support.

Initializing...
Initializing gui preferences from file  /afs/unity.ncsu.edu/users
opsys_dv_prefs.tcl
design_vision> design_vision>
design_vision>
design_vision>
design_vision>
```

Figure 5: GUI of Synopsys Design Vision

3. Synthesizing a sample Verilog file

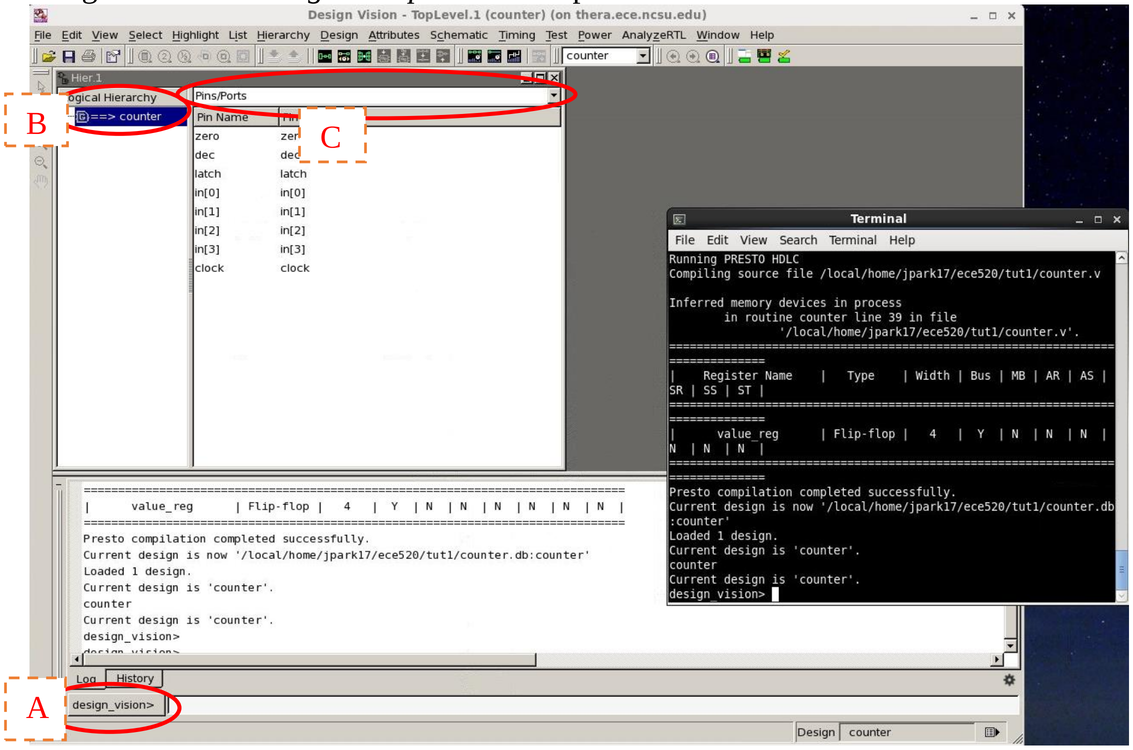
Download `counter.v`, `setup.tcl`, `read.tcl`, `Constraints.tcl` and `CompileAnalyze.tcl` from the web page. Open `counter.v`, `setup.tcl`, `read.tcl`, `Constraints.tcl` and `CompileAnalyze.tcl` in your favorite editor. Review what is written there, especially in the comments. Then in the command line,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

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

1) setup.tcl: Sets up variables that will be used to distinguish different runs of synthesis.
2) read.tcl: Reads and compiles all verilog files to load the design. Now:

Q1. According to the Verilog code, what flip-flops are specified in this design? (Put these in your Homework Answer Sheet)

Figure 6 shows the result of the above two commands. You should see your command echoed in the large message frame of log window. After a moment, you will see messages from the *Design Compiler* scroll up the screen.


Design Vision - TopLevel.1 (counter) (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
counter
Hier.1
Logical Hierarchy
==> counter
Pins/Ports
Pin Name
zero
dec
latch
in[0]
in[1]
in[2]
in[3]
clock
B
C
A
Log
History
design_vision>
Design
counter
Terminal
File Edit View Search Terminal Help

```
Running PRESTO HDLC
Compiling source file /local/home/jpark17/ece520/tut1/counter.v

Inferred memory devices in process
        in routine counter line 39 in file
                '/local/home/jpark17/ece520/tut1/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 '/local/home/jpark17/ece520/tut1/counter.db
:counter'
Loaded 1 design.
Current design is 'counter'.
counter
Current design is 'counter'.
design_vision>
```

```
=================================================================================
|      value_reg      | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  | N  |
=================================================================================
Presto compilation completed successfully.
Current design is now '/local/home/jpark17/ece520/tut1/counter.db:counter'
Loaded 1 design.
Current design is 'counter'.
counter
Current design is 'counter'.
design_vision>
design_vision>
```

Figure 6: After sourcing setup.tcl and read.tcl

At this time would good 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`" tab that appears above the `History` tab when it is clicked.

Also note the presence of the top level instance `counter` at top left corner which corresponds to the hierarchy window(Point B: circled in the figure above). 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) of interest within for the object highlighted in the hierarchy window. In this case, we have enabled the viewing of the ports/pins of the counter.

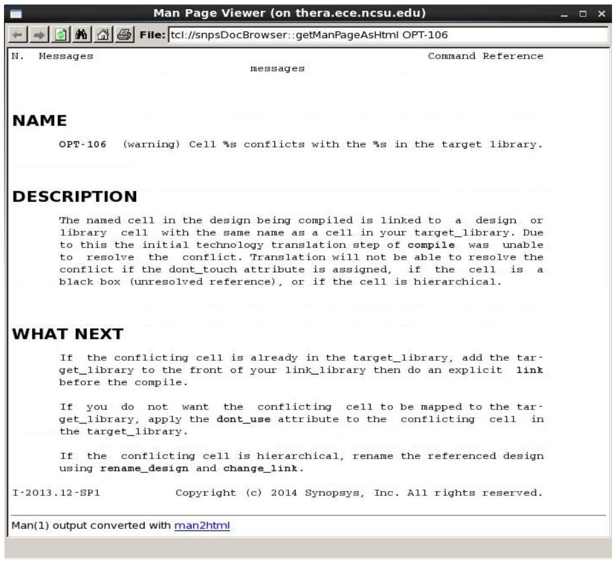
Make sure to note the type of information that appears on the screen. Please play around with the tool and the information given to you. It makes issues much easier to debug down the line. **Any problems or unintended latches should be fixed before proceeding. These should also appear in the `Log` window.**

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

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

1) Constraints.tcl: sets up the necessary constraints (input and output loads, clock period and skew, area constraint etc) on the design for valid synthesis.
2) CompileAnalyze.tcl: runs the compilation for synthesis, fix 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. Click the blue colored link at the end of warning message at the `Log message`. `The design_vision` will show a pop-up message window explains 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)


Man Page Viewer (on thera.ece.ncsu.edu)
File: tcl://snpsDocBrowser::getManPageAsHtml OPT-106

```
N.  Messages                                          Command Reference
                              messages

NAME
       OPT-106   (warning)  Cell %s conflicts with the %s in the target library.

DESCRIPTION
       The named cell in the design being compiled is linked to  a  design  or
       library  cell  with the same name as a cell in your target_library. Due
       to this the initial technology translation step of compile  was  unable
       to  resolve  the  conflict. Translation will not be able to resolve the
       conflict if the dont_touch attribute is assigned,  if  the  cell  is  a
       black box (unresolved reference), or if the cell is hierarchical.

WHAT NEXT
       If  the conflicting cell is already in the target_library, add the tar-
       get_library to the front of your link_library then do an explicit  link
       before the compile.

       If  you  do  not  want  the  conflicting  cell to be mapped to the tar-
       get_library, apply the dont_use attribute to the  conflicting  cell  in
       the target_library.

       If  the  conflicting cell is hierarchical, rename the referenced design
       using rename_design and change_link.

I-2013.12-SP1          Copyright (c) 2014 Synopsys, Inc. All rights reserved.

Man(1) output converted with man2html
```

Figure 7: Warning Pop-up (OPT-106)

.

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 um^2 along with the total area of your design.

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

```
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)                      0.6185     0.6185 f
  U17/ZN (NOR4_X2)                             1.1363     1.7547 r
  U16/ZN (NOR3_X2)                             0.2291     1.9838 f
  U21/ZN (INV_X4)                              0.1361     2.1199 r
  U14/ZN (NOR2_X2)                             0.0676     2.1875 f
  U30/ZN (AOI22_X2)                            0.3008     2.4883 r
  U20/ZN (INV_X4)                              0.0419     2.5303 f
  U29/ZN (AOI21_X2)                            0.3224     2.8527 r
  U19/ZN (INV_X4)                              0.0409     2.8936 f
  value_reg[2]/D (DFF_X1)                      0.0000     2.8936 f
  data arrival time                                       2.8936

  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                          -0.3114     9.6386
  data required time                                      9.6386
  -----------------------------------------------------------
  data required time                                      9.6386
  data arrival time                                      -2.8936
  -----------------------------------------------------------
  slack (MET)                                             6.7450
```

The values of importance in this case is the data arrival time (2.8936) 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.

[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

The 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. 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.

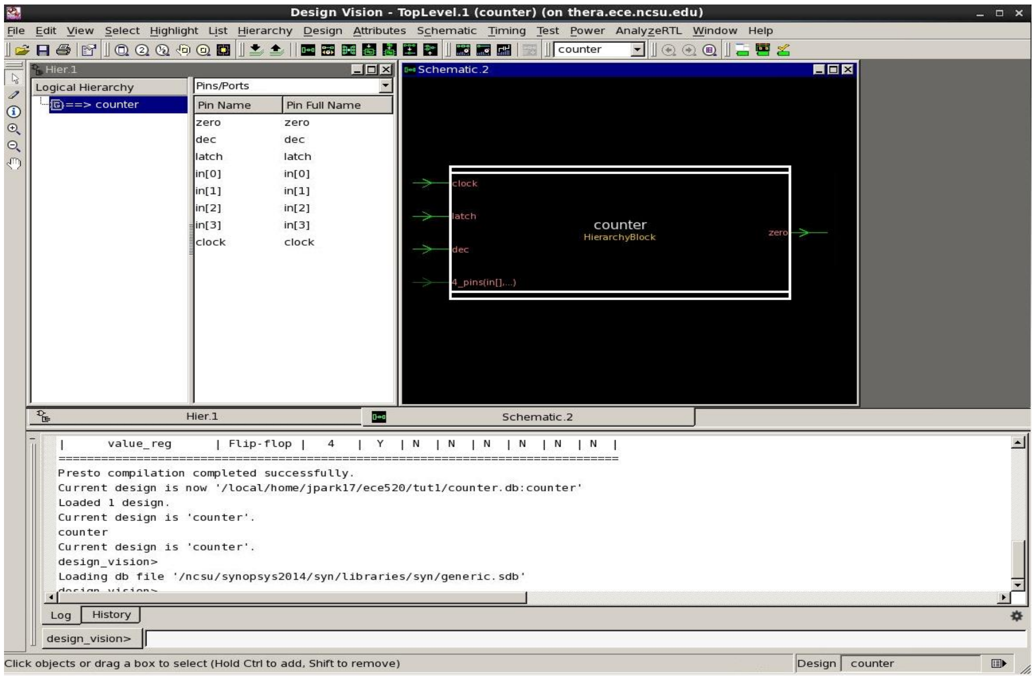
```
Warning: Variable ‘foo’ is driven in more than one process or
block in file ‘counter.v’.  This may cause mismatch between
simulation and synthesis.  (HDL-220)
```

  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 Design Schematic button in the toolbar. When you double click “Counter” box in figure 8, you can see the inside of the box as figure 9. You can look into it more detail by clicking

This results in the appearance of 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.

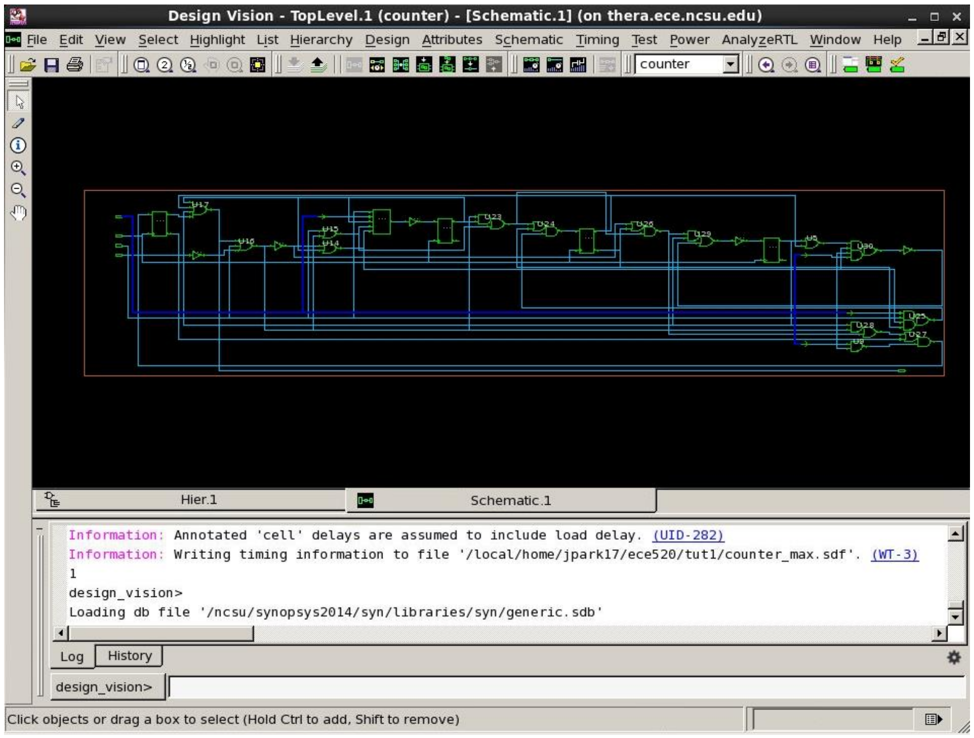

Design Vision - TopLevel.1 (counter) (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.1
Logical Hierarchy
==> counter
Pins/Ports
Schematic.2
clock
latch
dec
4_pins(in[],...)
counter
HierarchyBlock
zero
Hier.1
Schematic.2
Log
History
design_vision>
Design
counter

Pin Name	Pin Full Name
zero	zero
dec	dec
latch	latch
in[0]	in[0]
in[1]	in[1]
in[2]	in[2]
in[3]	in[3]
clock	clock

```
|      value_reg      | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
Current design is now '/local/home/jpark17/ece520/tut1/counter.db:counter'
Loaded 1 design.
Current design is 'counter'.
counter
Current design is 'counter'.
design_vision>
Loading db file '/ncsu/synopsys2014/syn/libraries/syn/generic.sdb'
design_vision>
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

Figure 8: Schematic of the synthesized design


Design Vision - TopLevel.1 (counter) - [Schematic.1] (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
counter
U17
U15
U16
U14
U23
U24
U26
U29
U5
U30
U28
U25
U27
U8
Hier.1
Schematic.1
Log
History
design_vision>

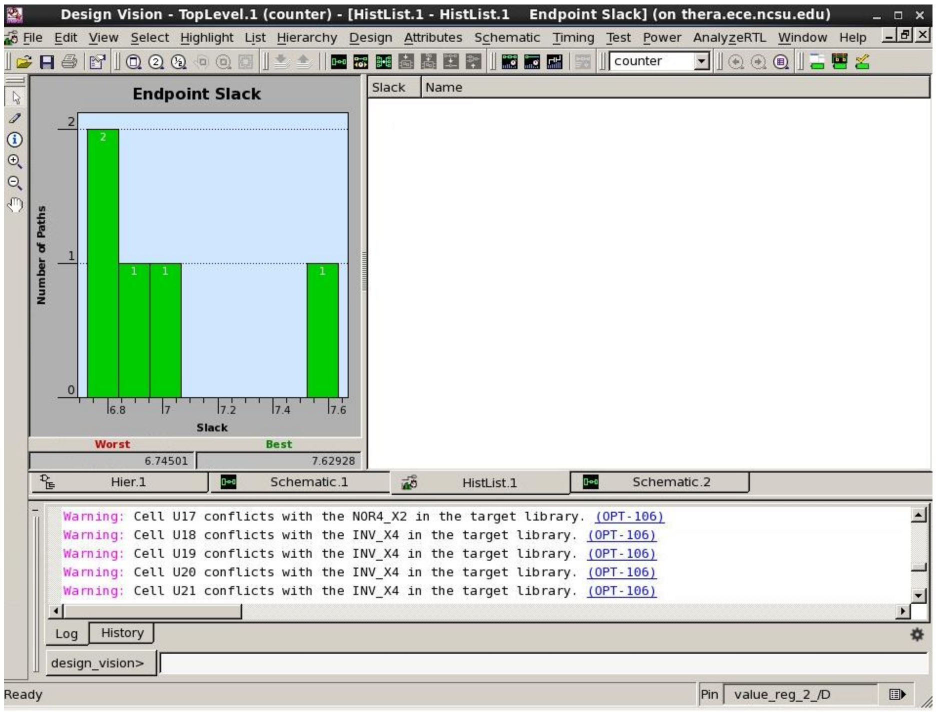
```
Information: Annotated 'cell' delays are assumed to include load delay. (UID-282)
Information: Writing timing information to file '/local/home/jpark17/ece520/tut1/counter_max.sdf'. (WT-3)
1
design_vision>
Loading db file '/ncsu/synopsys2014/syn/libraries/syn/generic.sdb'
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

Figure 9: Detailed Schematic of the synthesized design

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.

Of particular use is the worst case path tool (the middle icon in the above toolbar). This tool helps to determine if there is a possibility of further optimization in your design, determine benefits of pipelining, the un-even nature of pipelining, the disparity between the top X number of worst-case paths, etc. When no values are input into the menu that pops up (just click OK), the result is seen below, figure 10. You will see the bar graph which is shown worst to best slack paths.

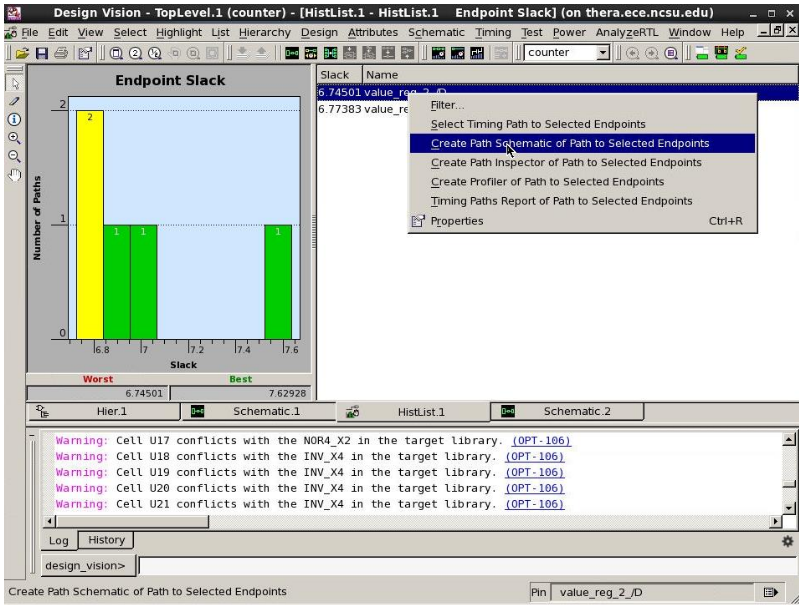


Design Vision - TopLevel.1 (counter) - [HistList.1 - HistList.1 Endpoint Slack] (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
counter
Endpoint Slack
Number of Paths
2
1
0
6.8
7
7.2
7.4
7.6
Slack
Worst
Best
6.74501
7.62928
Slack
Name
Hier.1
Schematic.1
HistList.1
Schematic.2
Log
History
design_vision>
Ready
Pin
value_reg_2_/D

```
Warning: Cell U17 conflicts with the NOR4_X2 in the target library. (OPT-106)
Warning: Cell U18 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U19 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U20 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U21 conflicts with the INV_X4 in the target library. (OPT-106)
```

Figure 10: End Point Slack

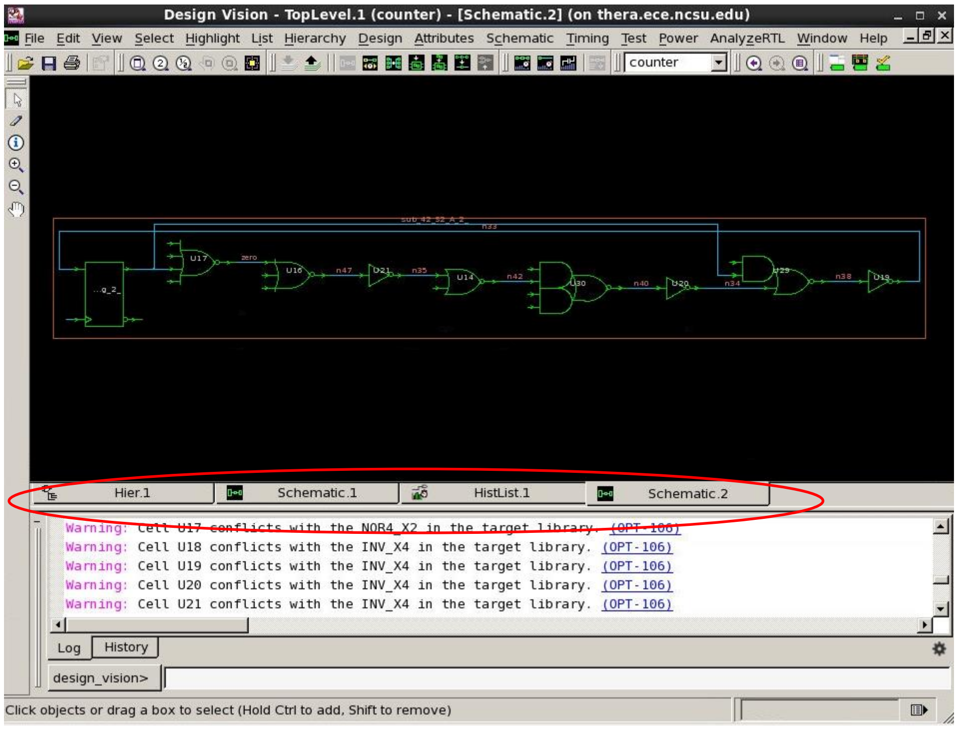
Once you select the most left side bar (the worst case), you will see the two candidates. Let's right click the most top sided one and select the third option("Create Path Schemetic of Path to Selected Endpoints").


Design Vision - TopLevel.1 (counter) - [HistList.1 - HistList.1 Endpoint Slack] (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
counter
Endpoint Slack
Number of Paths
Slack
Worst
Best
6.74501
7.62928
Slack Name
6.74501 value_reg_2_/D
6.77383 value_re
Filter...
Select Timing Path to Selected Endpoints
Create Path Schematic of Path to Selected Endpoints
Create Path Inspector of Path to Selected Endpoints
Create Profiler of Path to Selected Endpoints
Timing Paths Report of Path to Selected Endpoints
Properties Ctrl+R
Hier.1
Schematic.1
HistList.1
Schematic.2
Log
History
design_vision>
Create Path Schematic of Path to Selected Endpoints
Pin value_reg_2_/D

```
Warning: Cell U17 conflicts with the NOR4_X2 in the target library. (OPT-106)
Warning: Cell U18 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U19 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U20 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U21 conflicts with the INV_X4 in the target library. (OPT-106)
```

Figure 11: End Point Slack

You will see that the path shown in the figure corresponds with the previously generated timing report (holdfixed timing). Please note that all the previous windows can still be reached using the tabs at the bottom of the schematic window (circled in figure 12).


Design Vision - TopLevel.1 (counter) - [Schematic.2] (on thera.ece.ncsu.edu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
counter
U17
zero
U16
n47
U21
n35
U14
n42
U30
n40
U20
n34
U29
n38
U19
n33
Hier.1
Schematic.1
HistList.1
Schematic.2
Log
History
design_vision>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

```
Warning: Cell U17 conflicts with the NOR4_X2 in the target library. (OPT-106)
Warning: Cell U18 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U19 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U20 conflicts with the INV_X4 in the target library. (OPT-106)
Warning: Cell U21 conflicts with the INV_X4 in the target library. (OPT-106)
```

Figure 12: Worst case path of the design

To plot the full design schematic, you would first need to go to the full schematic from the worst case path view using the tabs at the bottom of the schematic window. Then, do the following:

a. Go to the file menu and do: `File` → `Print Schematic`.

b. In the printer view menu, choose the print to file option. You could print schemetic as a file (ps or pdf format) that will be written to in your home directory. It is best if you save it in a file and space of your choice using the browse button.

c. The options (color v/s black and white; layout options etc) can be catered to your choice. Then hit OK.

d. You can view the resulting ps file by doing `> gv <filename>.ps`. Note, you have to do this from a linux prompt, not from within synopsys. You will note that the schematic created will have empty fields for `Designer` and `Company`. This can be modified using the `.synopsys_dc.setup` file.

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 acroread`" 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 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.

4. **Modesim Help:** `/afs/bp.ncsu.edu/dist/modelsim/docs` where a choice of pdf or html documentation exists

5. Student Version Setup:

a) Download the student version Modelsim SE at http://model.com/content/modelsim-pe-student-edition-hdl-simulation
b) Install it on your local machine.
c) Follow the instructions on website and provide your email address, license will be sent to you via email.
d) Copy the `modelsim.ini` file into the following directory:
   "Installation directory"/examples
   This is the default directory when you run modelsim. If you change the running directory, please remember to save `modelsim.ini` file in your running directory.
e) Copy all your design files and testbench into the same directory:
   "Installation directory"/examples
   Again, you can change working directory if needed.
f) Launch modelsim, type exact same commands as on linux machine:
   setenv MODELSIM modelsim.ini
   vlib mti_lib
g) Compile design and testbench files with "vlog"
h) Looking for the top level testbench module at subwindow "library → mti_lib"
   Double click the top level testbench module there. Modelsim is all set and ready to run simulation

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:

```
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 commend 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.