

ECE 551 Design Vision Tutorial

ECE 551 Staff

Dept of Electrical & Computer Engineering, UW-Madison

Last Updated: Spring 2013 (emphasis on script based)

Lesson 0 – Tutorial Setup	2
Lesson 1 – Code Input (Reading)	5
Lesson 2 - Simple Synthesis (with no constraints).....	9
Lesson 3 – GUI's Are For Children.....	11
Lesson 4 – Synthesis Script (why not save my work)	12
1) Create Shell Script (preferably with a .dc file extension).....	12
2) Running a Synthesis Script.....	12
<u>Method 1:</u> Sourcing the script from within dc_shell	12
<u>Method 2:</u> Invoking both dc_shell and the script from the Unix prompt	13
Lesson 5 – Constraining The Design	13
1) Constraining the clock	13
2) Setting Input Delay	13
3) Set drive strength of inputs.....	14
4) Setting Output Delay Constraints	14
5) Some Miscellaneous Constraints	14
Lesson 6 – Analyzing Results (reports)	15
1) Max Delays (Is the result fast enough?)	16
2) Min Delays (Is the result too fast...or do we have too much clock skew).....	16
3) How Big is the Design?	16
Lesson 7 – Further Constraints and Optimizations.....	17

Lesson 0 - Tutorial Setup

1. From your home directory, go to your ece551 directory (as should have been created in the ModelSim tutorial. Otherwise create a directory named “ece551”).

```
%> cd ece551
```

2. Copy all files from the Design Vision tutorial directory to your current directory

```
%> cp -r ~ece551/public/tutorials/designvision ./
```

3. Change directory to your tutorial directory

```
%> cd designvision
```

Choosing Your Tech Library:

This tutorial now comes with a **.synopsys_dc.setup** configuration file that allows you to choose between multiple different standard cell libraries.

Before beginning the tutorial, you should confirm that you have the correct Tech library set.

4. Using any text editor of your choice, open the **.synopsys_dc.setup** in the current directory (Note: files starting with a ‘.’ are hidden files in Unix.).

```
%> gedit .synopsys_dc.setup
```

At the beginning of the file you should see a list of different technology library names with descriptions, such as “TSMC 40nm High Performance”. Below this there are a series of commands that say `set TECH_LIB ____`. All but one of these commands should be preceded by a “#” which means that they are commented out. For the purpose of this tutorial we will enable the 40 nm general purpose library.

5. In the **.synopsys_dc.setup** file, make sure that the command
`set TECH_LIB tcbn40lpbwptc` is **not** commented out, but all other
`set TECH_LIB` commands *are* commented out (preceded by a #).
6. Save the **.synopsys_dc.setup** file and exit the text editor.

7. Copy the file from the current directory to your home directory.

```
%>cp .synopsys_dc.setup ~/
```

When you launch Design Vision, the tool will first check your current directory, then your home directory for a valid **.synopsys_dc.setup** file. If you need to alter the configuration file later, you should be aware of the location of each file.

8. Change to the main tutorial directory

```
%>cd dv_tutorial
```

It is important to start Design Vision from the directory where your project is so that environment variables that are automatically setup will be made correctly.

IMPORTANT

If you see that the design variables do not match later on in the tutorial do the following:

*In the directory you started Design Vision in remove the **.synopsys_dc.setup** file.

```
%> rm .synopsys_dc.setup
```

This will force design vision to use the one from your home directory.

9. Start Design Vision in GUI mode

```
%> /cae/apps/bin/design_vision
```

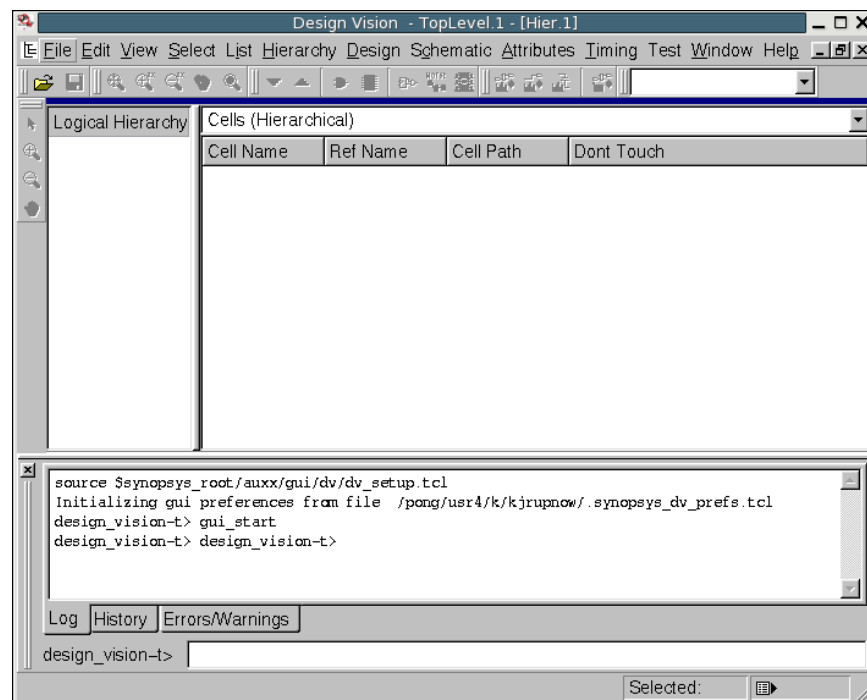


Figure 0-1 : Design Vision Default Window

We may look at the main window of Design Vision to see a few of the program's features. In the top panel, there are two panes. The left pane is a full Hierarchy pane; it will show the entire hierarchy of the current design (as selected from the drop-down box in the upper control panel).

The right panel is a context based panel which will display contents based upon the selection in the drop down box at the top of the panel.

The bottom panel has three different tabs: log, history and errors/warnings. The important thing to note about this panel is that every command you perform will appear in this panel, allowing you to learn the commands and create scripts of commands for future use. It is also the panel that you must monitor to determine the source of errors and warnings, allowing you to fix the code or correctly determine if a warning is expected.

Lesson 1 – Code Input (Reading)

1. File -> Read

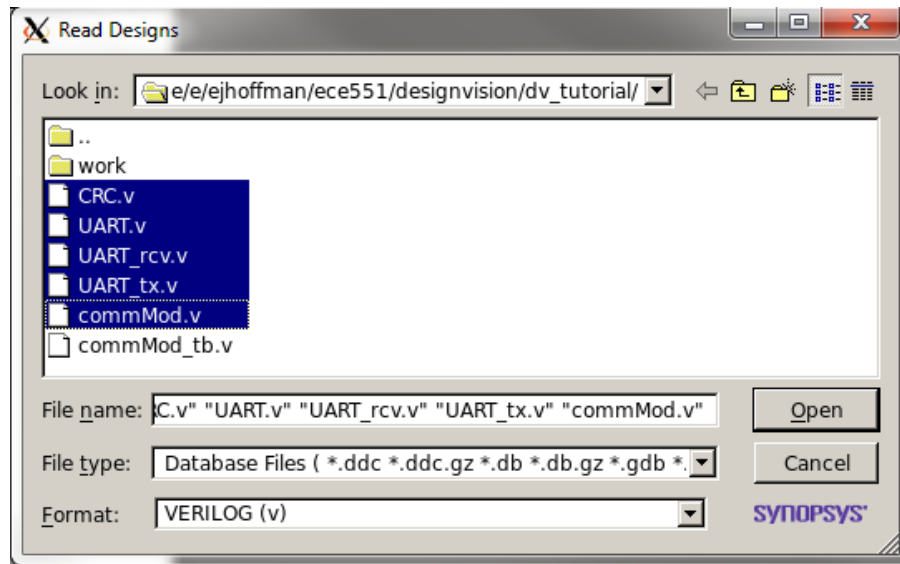


Figure 1-1 : Reading Verilog Files

Select all the Verilog files except commMod_tb.v. This module is the testbench, so is not synthesized. Set the format to be Verilog, or SVERILOG (if using system Verilog). Click Open

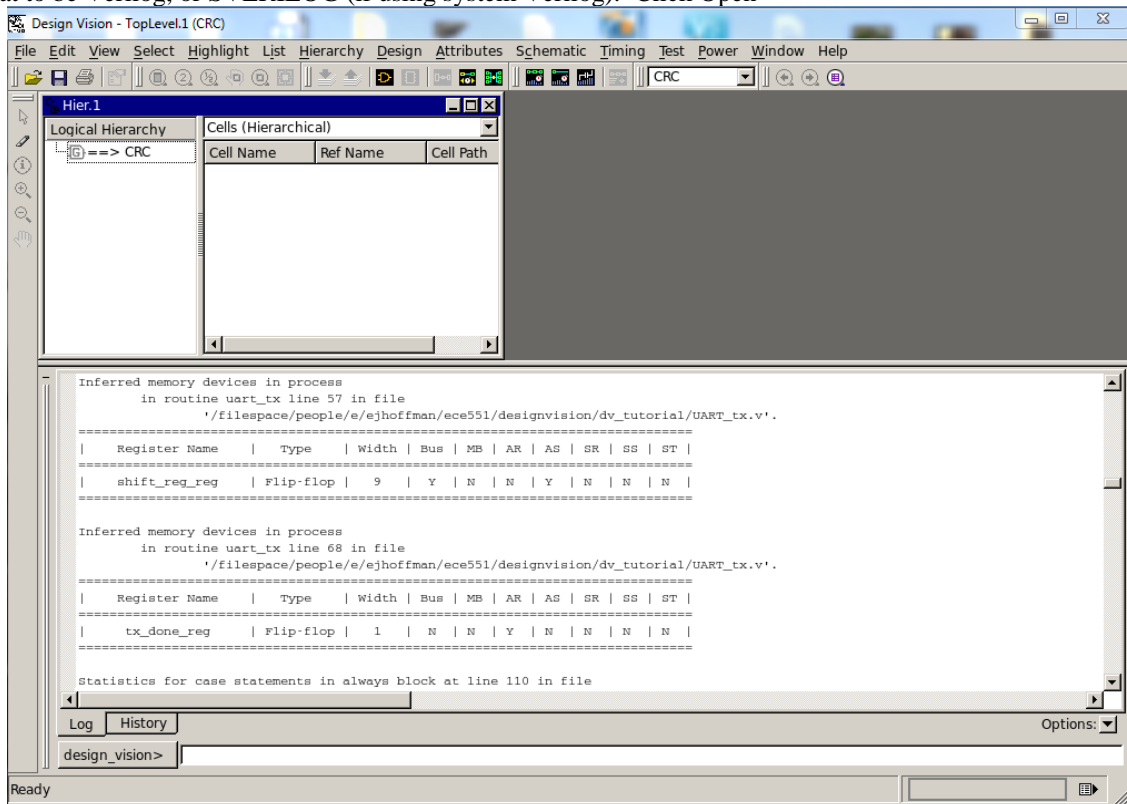


Figure 1-2 : Log Results of Verilog Read

If you expand the bottom panel and look at the log history you should see it read the Verilog and inferred several different flops. This is always a good section to look at and ensure that the flops inferred in your design are the one you think should be flops, and that there are no latches mistakenly inferred in your design.

2. Set Current Design to top level (commMod)

We read all these Verilog files in, but Synopsys does not know which one represents the top level design. In this case the top level design is a module called commMod. We need to set the current design to be commMod. Start by listing all the modules in the design.

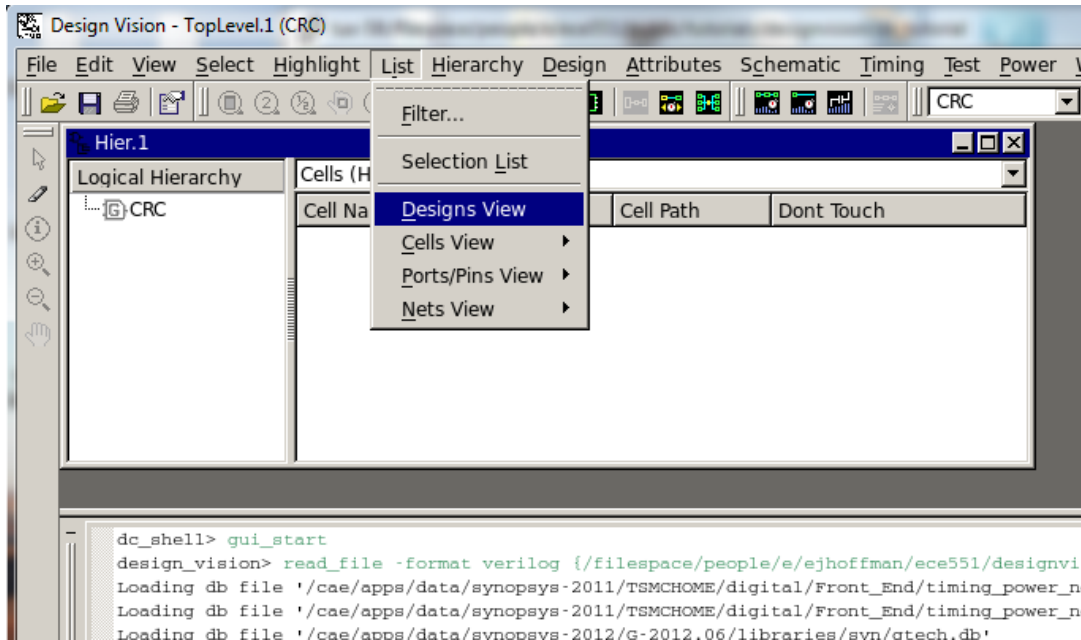


Figure 1-3 : Set Current Design (part I)

Now we set commMod to be the current design we are working on (top level). Click on commMod to select it then right click and choose “Set Current Design” from the context menu.

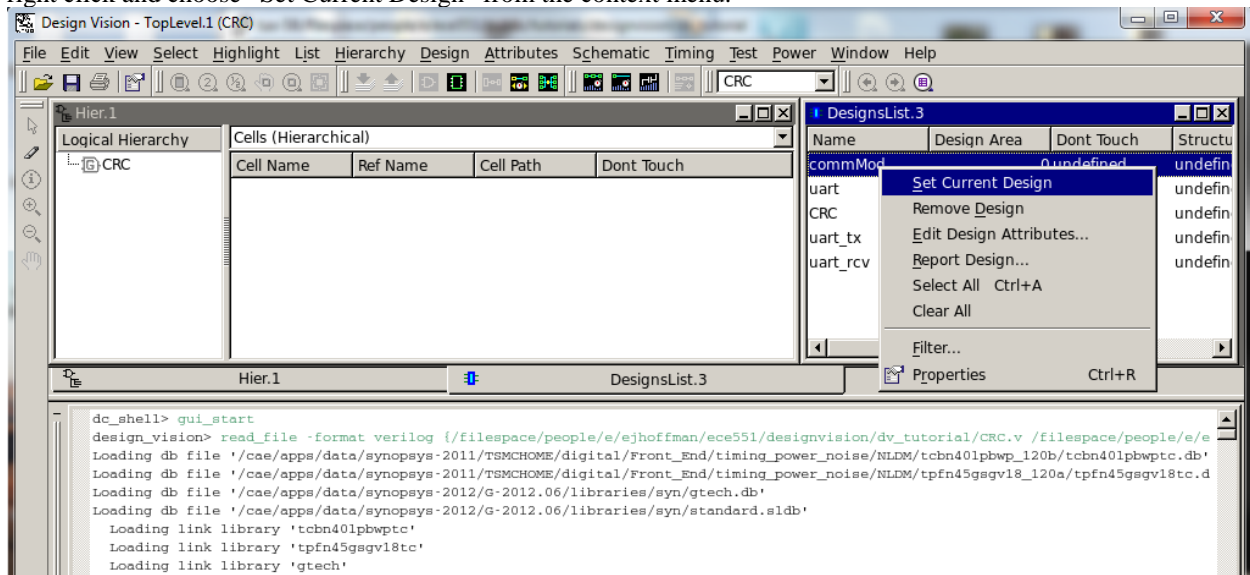


Figure 1-4 : Set Current Design (part II)

3. Viewing Pre-Synthesis Schematic

Once in a while it can be useful to view the schematic view of what Synopsys thinks the circuit is. These schematic views are auto generated and often hard to follow. Still if it is your design (you wrote the Verilog and are familiar with it) then sometimes it will make sense to you and give you some insight into what Synopsys is “thinking”.

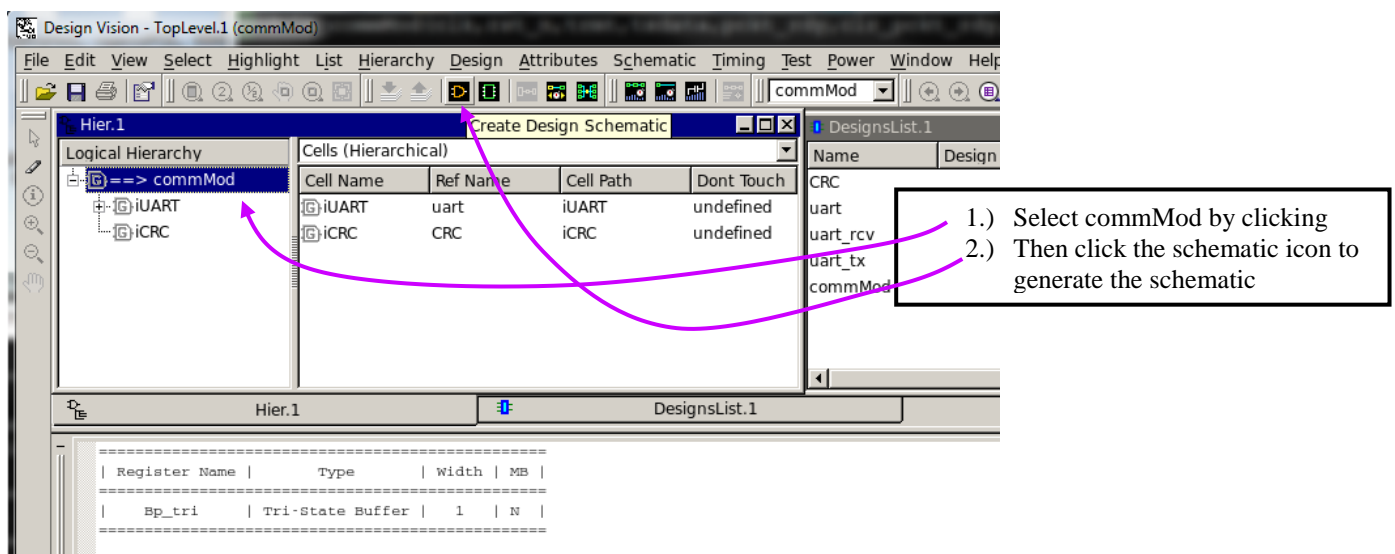


Figure 1-5 : Schematic Generation

Once the schematic is generated you can view it and zoom in and out and pan. Play around a bit with this. At this point we have not actually synthesized the design. We have read it in, and analyze and elaborate has been performed. The design is currently not mapped to our TSMC standard cell library, but rather just mapped to Synopsys GTECH cells (generic functional blocks).

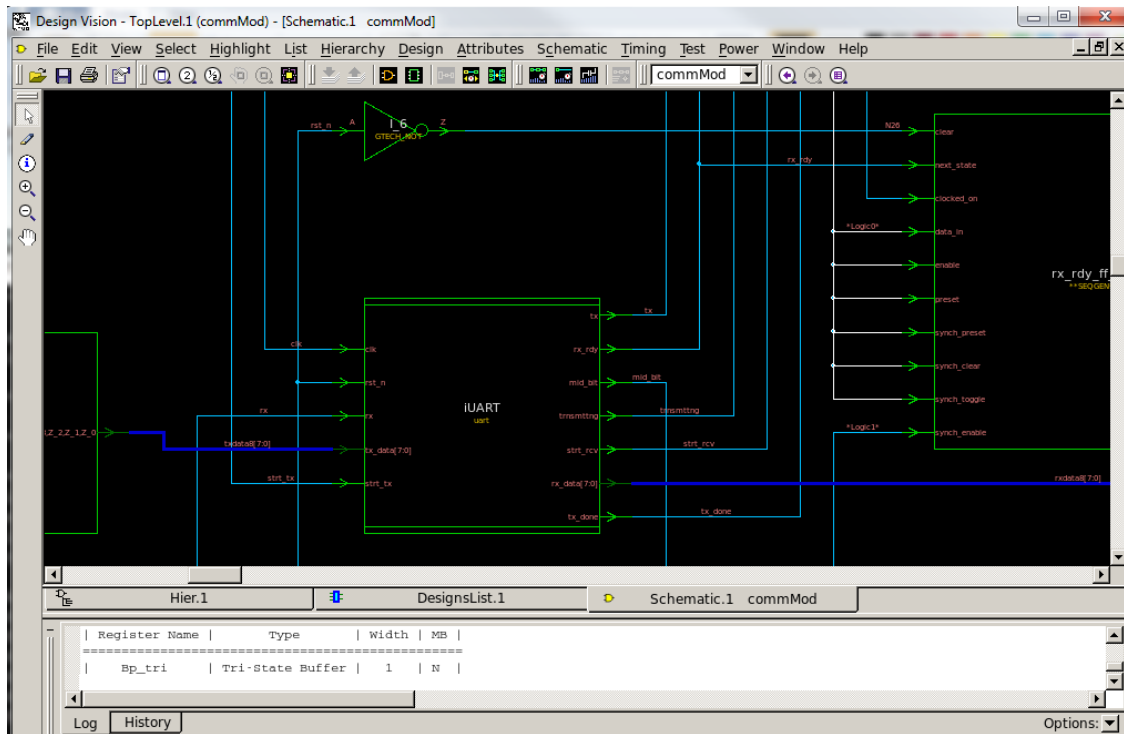


Figure 1-6 : Viewing a Schematic

Lesson 2 - Simple Synthesis (with no constraints)

Now that we have analyzed and elaborated the design, we can perform simple synthesis on our design. In ECE 551, we will require the use of design constraints which will be covered in Lesson 3, however we may see some information about the design by performing an unconstrained compilation.

In mapping options we may specify mapping and area effort. Map effort refers to how long the synthesizer will work to optimize the design in terms of mapping it to our TSMC standard cell library. Area effort refers to how long the synthesizer will try to reduce the area of the design before moving on to the next stage. Power effort indicates how long the synthesizer will try to optimize the power used by your circuit. Other compile options are:

1. Top Level - compiles only the top level of the design, leaving the rest uncompiled
2. Ungroup All - Ungroups the entire hierarchy so that all logic is compiled as one module
3. Scan - Refers to the insertion of scan chains for testing logic. This should not be used for this course.
4. Incremental mapping - specifies that the mapping should be done based upon only local information instead of global information. Furthermore, if some mapping information is already available, the synthesis will start from the previous map. This is especially useful when trying to improve upon an initial synthesis
5. Allow boundary conditions - allows that boundary conditions such as known input constants can be used to help optimize the design
6. Auto Ungroup - when enabled, you may choose either area or delay as the trigger, and the synthesizer will automatically ungroup designs to meet constraints if the constraints are not being met for the trigger

Design->Compile Design

Leave options as is and click OK

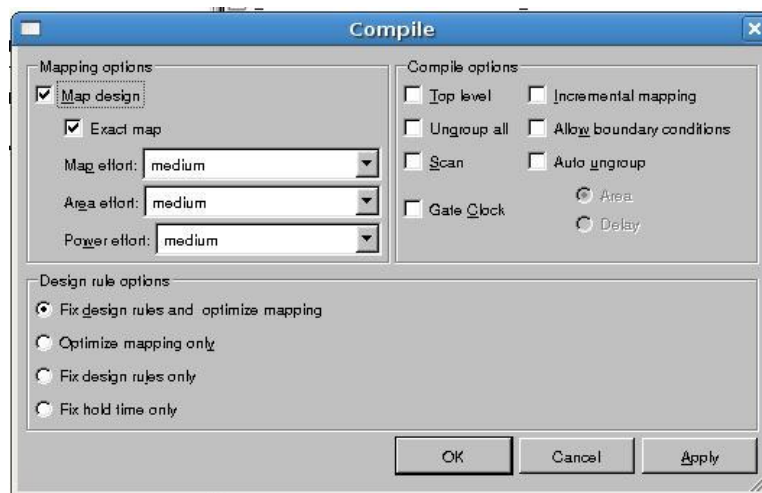


Figure 2-1 : Compile Design Dialog

Compile Ultra uses the same options, which have the same meaning as previously described. The difference is that all effort options are automatically set at their highest levels and the

As compilation occurs you can see the Synopsys spewing out text to the log area. One of the things it will output during this stage are tables that specify where it is currently at with regard to minimizing the synthesis cost function. Area, speed paths, and design rules.

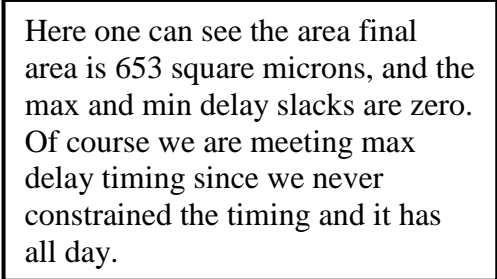


Figure 2-2 : Compile Results in Log Area

Lesson 3 - GUI's Are For Children

OK perhaps GUI's are not all bad. In many CAD tools the GUI interface more intuitive one than the command based one. However, with Synopsys Design Compiler/Design Vision this is not the case. Design Compiler was originally a command line tool, and the GUI was bolted around it in later years. It shows. This tool is much better run in command line mode. If you run it command line you develop a script and then can easily reproduce your results later by running the script.

Quit the GUI and just start with a Unix terminal in the same directory as the Verilog files reside.

1. Invoking Design Compiler in shell mode

```
%>design_vision -shell dc_shell
```

Now a Design Compiler shell (dc_shell) should be up with a prompt like: "**dc_shell>**"

2. Reading in the Verilog

```
dc_shell> read_file -format verilog { UART_tx.v UART_rcv.v UART.v \
CRC.v commMod.v }
```

The "\ " at the end of the line means the command continues on the next line. Lists of design files are included in {}.

3. Setting Current Design to Top level (to commMod)

```
dc_shell> set current_design commMod
```

4. Compiling (actually synthesizing)

```
dc_shell> compile -map_effort medium
```

Now we are already back to where we were when using the GUI, and it was actually much more direct to do it with the command line interface.

Now we will write out our resulting gate level netlist as a .vg file.

```
dc_shell> write -format verilog commMod -output commMod.vg
```

Now you can quit dc_shell using the 'quit' command and do a 'more' on commMod.vg to see the Verilog gate level netlist that was produced.

```
%>more commMod.vg
```

Note that the result is structural Verilog that instantiates standard cells from our TSMC library.

Lesson 4 – Synthesis Script (why not save my work)

It should be obvious that a command line driven CAD tool should be driven with a script as opposed to just typing the same commands over and over again. In this section we will develop a synthesis script.

A synthesis script is simply a text file with all the dc_shell command used typed out. Of course as with any script/program some commenting is always a good idea.

1) Create Shell Script (preferably with a .dc file extension)

Using a text editor of your choice create a file called: `commMod.dc`

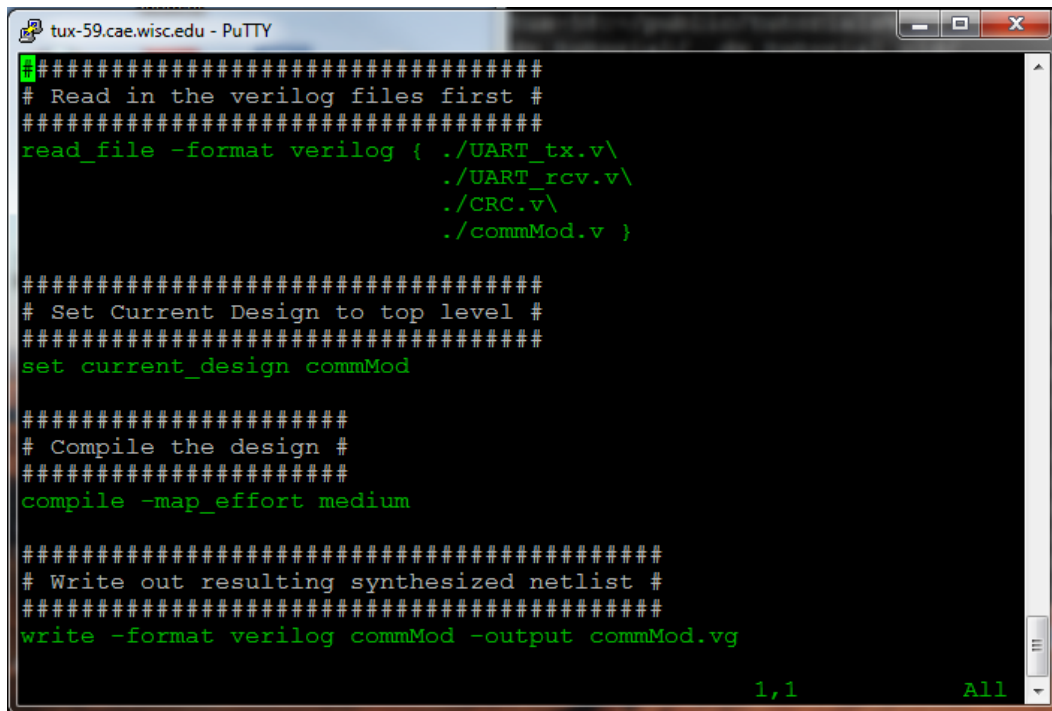
A screenshot of a PuTTY terminal window titled 'tux-59.cae.wisc.edu - PuTTY'. The terminal displays a synthesis script for 'commMod.vg'. The script is enclosed in multi-line comments and contains the following commands: 'read_file -format verilog { ./UART_tx.v\ ./UART_rcv.v\ ./CRC.v\ ./commMod.v }', 'set current_design commMod', 'compile -map_effort medium', and 'write -format verilog commMod -output commMod.vg'. The terminal shows the script being executed line by line, with the cursor at the end of the last line. The status bar at the bottom right shows '1,1' and 'All'.

Figure 4-1 : Simple Synthesis Script

The above script is the same as what we did in Lesson 3. It is a simple synthesis script with no timing constraints.

2) Running a Synthesis Script

Method 1: Sourcing the script from within dc_shell

Kick off a dc_shell from a Unix terminal

```
%>design_vision -shell dc_shell
```

Now source your synthesis script from within dc_shell

```
dc_shell> source commMod.dc
```

Method 2: Invoking both dc_shell and the script from the Unix prompt

```
%>design_vision -shell dc_shell -f commMod.dc
```

Try one of these two methods.

Lesson 5 – Constraining The Design

Synopsys is cost function based, and tries to optimize the design by minimizing its “cost”. The cost function of a design is based on meeting design rules, meeting timing, and minimizing area & power. To have a properly synthesized design one needs to have a properly constrained design.

1) Constraining the clock

The clock frequency at which your design has to run is one of the most fundamental of timing constraints. Clock period specification, and the pin it is sourced to is done as follows.

```
dc_shell> create_clock -name "clk" -period 2 -waveform {0 1} {clk}
```

The command above is creating a clock with period of 2 time units (nano seconds in our case). The waveform has 50% duty cycle since it starts high at 0, and falls at 1 time unit. The clock has a name of “clk” (this is a handle). It is sourced to a pin **clk**.

The clock goes to a lot of flops in your design. It is a high fanout net. We don’t want Synopsys doing anything stupid with it like buffering it. Clock distribution is typically handled in a separate tool. We want to tell Synopsys not to mess with it.

```
dc_shell> set_dont_touch_network [find port clk]
```

Add the above two lines to your synthesis script, after the: “**set current_design**”, and before the: “**compile -map_effort medium**” lines.

2) Setting Input Delay

Input delay needs to be specified for all inputs (except the clock). In our case we will use the same timing for all inputs. This would not be the case in a “real” design. To aid us in setting a uniform input delay across all inputs we will setup a variable “**prim_inputs**” that holds the name of all our input ports except clock.

```
dc_shell> set prim_inputs [remove_from_collection [all_inputs]\  
[find port clk]]
```

Now prim_inputs is a variable that holds the name of all input ports except the clock. We can use this variable in our input delay statement.

```
dc_shell> set_input_delay -clock clk 0.5 $prim_inputs
```

Add the above two lines to your synthesis script.

3) Set drive strength of inputs

Synopsys needs to know how strongly driven input are so it can calculate the slope of the rise/fall transitions. This is necessary to get accurate timing assessment. There are two commonly used commands to set drive strength. One command "set_driving_cell" tells Synopsys that the input is driven with the same strength as if it was driven by a specified cell in the cell library. The other method just specifies a drive strength in ohms of the driver of the input.

The "set_driving_cell" method will be demonstrated for all inputs. We will use the set_drive command to specify the drive strength for reset separately.

```
dc_shell> set_driving_cell -lib_cell AO33D0BWP -from_pin A1 -library\
          tcbn40lpbwptc $prim_inputs
```

Reset is another special signal like clock, and also has a very high fanout. We will use a set_drive command to tell Synopsys that rst_n is very strongly driven, so it does not think it needs to buffer it.

```
dc_shell> set_drive 0.1 rst_n
```

Add the two lines above to your synthesis script.

4) Setting Output Delay Constraints

For all outputs we need to tell Synopsys two things:

- 1.) When the output needs to be valid.
- 2.) How much capacitive load the output has to drive.

```
dc_shell> set_output_delay -clock clk 0.5 [all_outputs]
```

With the above line we are telling Synopsys that we need all outputs valid 0.5ns prior to the next rising edge of clock.

Now we will tell Synopsys how much load the output has to drive. The more load an output has to drive the slower it will be. Synopsys has to know that it needs to upsize the output driver to deal with the large capacitive load.

```
dc_shell> set_load 0.1 [all_outputs]
```

This command informs Synopsys that all outputs have to drive a 0.1pF load.

Add the two commands above to your synthesis script.

5) Some Miscellaneous Constraints

For internal nodes in the design there are parasitic routing capacitances. These parasitic routing capacitances can adversely affect timing. We need to have Synopsys at least make a guess as to what these parasitic capacitances might be so it can design the logic to be fast enough to overcome them.

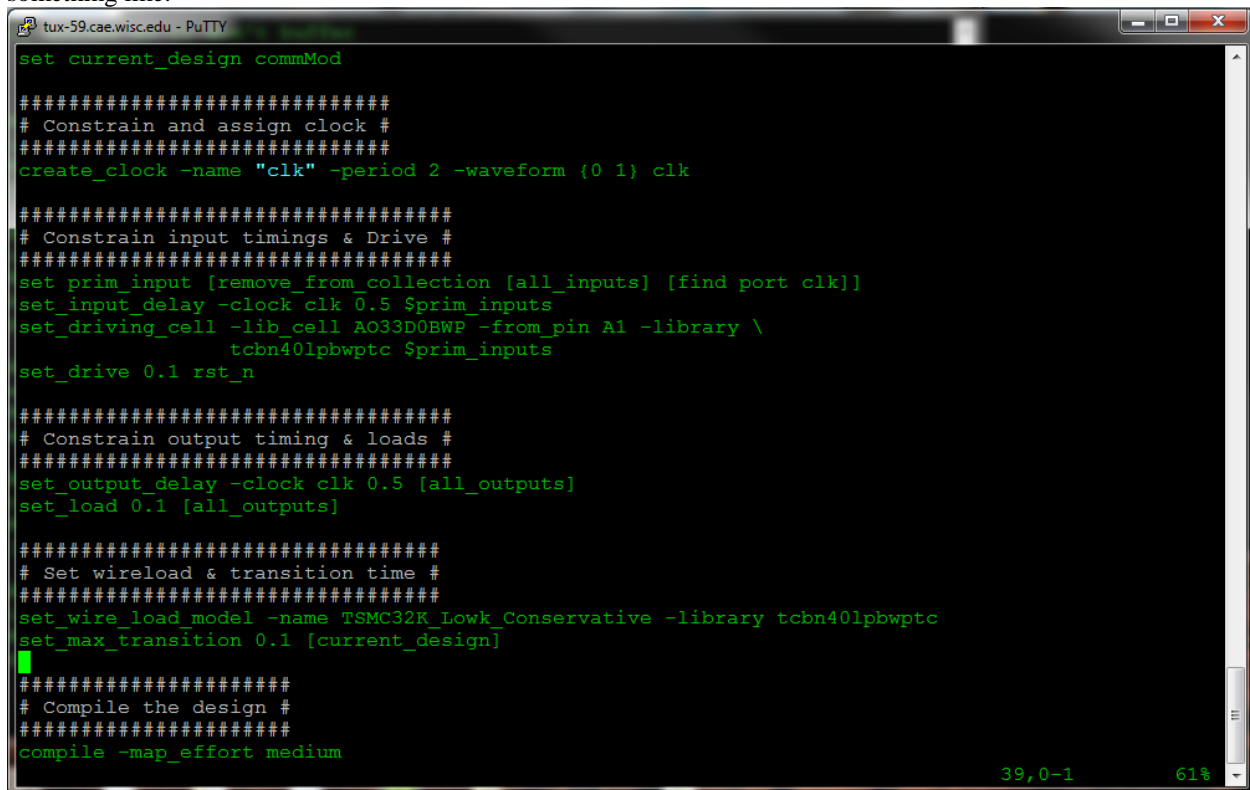
```
dc_shell> set_wire_load_model -name TSMC32K_Lowk_Conservative \  
-library tcbn40lpbwptc
```

An internal design rule that Synopsys has to make regards the transition time of nodes. If nodes are too slow to transition they will spend too much time in a region with both high Vds and high Vgs. This can damage the transistors and slow them down over time. We inform Synopsys that all nodes in the design must have transition times of 0.1ns or faster.

```
dc_shell> set_max_transition 0.1 [current_design]
```

Add the above two lines to your synthesis script.

That covers the common timing and design rule constraints. The middle of your synthesis script should look something like:



```
set current_design commMod  
  
#####  
# Constrain and assign clock #  
#####  
create_clock -name "clk" -period 2 -waveform {0 1} clk  
  
#####  
# Constrain input timings & Drive #  
#####  
set_prim_input [remove_from_collection [all_inputs] [find port clk]]  
set_input_delay -clock clk 0.5 $prim_inputs  
set_driving_cell -lib_cell AO33D0BWP -from_pin A1 -library \  
tcbn40lpbwptc $prim_inputs  
set_drive 0.1 rst_n  
  
#####  
# Constrain output timing & loads #  
#####  
set_output_delay -clock clk 0.5 [all_outputs]  
set_load 0.1 [all_outputs]  
  
#####  
# Set wireload & transition time #  
#####  
set_wire_load_model -name TSMC32K_Lowk_Conservative -library tcbn40lpbwptc  
set_max_transition 0.1 [current_design]  
#####  
# Compile the design #  
#####  
compile -map_effort medium  
  
39,0-1 61%
```

Figure 5-1 : Input Delay

Lesson 6 – Analyzing Results (reports)

Now start with a fresh design_vision session and execute your new synthesis script. One thing you may notice is the synthesized area is now larger that it was the last time. This is because the design has constraints (both timing and design rule) and Synopsys has to work harder and use more gates to meet the constraints.

How do we know our result is good. One handy command to execute after synthesis completes is “check_design”.

After running the synthesis script you should execute the **check_design** command. You may get a warning about a signal *Bp* which is an input, but is also driven by a tri-state driver. This is expected for our design, however the output of **check_design** is a good place to look to make sure there are no strange unexpected topologies that are part of your resulting circuit.

1) Max Delays (Is the result fast enough?)

One of the main things to check is did your design meet timing from a max delay point of view. Meaning, is it fast enough? Does it meet setup times of internal flops, and output delay constraints of primary outputs. To check this we use the “**report_timing -delay max**” command.

```
dc_shell> report_timing -delay max
```

The report that results from this command shows that we are missing timing by a considerable amount. The problem seems to stem from port *Bp*. There is a large delay (1.12ns in my report) early on at the *Bp* inout port.

This means we are not meeting time (or at least Synopsys does not think so). We will look into fixing this later.

2) Min Delays (Is the result too fast...or do we have too much clock skew)

Meeting hold time of our flops is often more important than making setup times. If the design is too slow we can slow down the clock, and still test the chip. If we have a hold time problem we are in trouble at any clock speed. To check for hold time problems we use the “**report_timing -delay min**” command.

```
dc_shell> report_timing -delay min
```

The results from this report show we are meeting timing. The clock to Q timing of the flops is greater than the hold time requirements of the flops.

3) How Big is the Design?

To determine the size of your design use the “**report_area**” command. This will tell you the size of the design in square microns. It also gives reports about the number of cells used, and a breakdown of whether the cells are combinational or sequential. If the wireload model is complete it will also give an estimate of routing area. Our wireload model does not give area estimate.

```
dc_shell> report_area
```

Another handy thing one can do with reports is to re-direct the output to a file. This way you have a permanent record of your reports. This applies to timing reports as well as area reports. An example with an area report is shown next.

```
dc_shell> report_area > area.rpt
```

Add lines for timing and area reports to your script. These lines should be added after the “**compile -map_effort medium**” command.

Lesson 7 – Further Constraints and Optimizations

There are other constraints that have not been covered so far. On big one is clock skew. Modern digital designs will have huge numbers of flops. The clock cannot be distributed to all these flops uniformly (we try, but it is impossible). Therefore, one must specify a clock uncertainty so that Synopsys can ensure min delay timings (hold times) even in the presence of an imperfect clock. I can never remember what the command is, but I know it has the word “**uncertainty**” in it.

```
dc_shell> help *uncertainty*
```

This command returns a list of all Synopsys commands that have “**uncertainty**” as part of the command name. I see a command for “**set_clock_uncertainty**” and this makes me say: “Ohh yeah that was it”!

So now I need to know the details of the “set_clock_uncertainty” command. I can type:

```
dc_shell> set_clock_uncertainty -help
```

That returns a more detailed help of the command. Now I can see how I want to use it.

```
dc_shell> set_clock_uncertainty 0.15 clk
```

Now that we have applied a clock uncertainty what does our min delay timing look like? Lets do a report again.

```
dc_shell> report_timing -delay min
```

If you look at the report you can see there is now a line stating the clock uncertainty of 0.15ns, and now we are no longer making it (we have negative slack on min delay).

So how do we fix it? Well we informed Synopsys about the clock uncertainty after it had compiled the design, so all we need to do is compile again, and it should fix the hold times right?

```
dc_shell> compile -map_effort medium
```

```
dc_shell> report_timing -delay min
```

What? It didn’t fix it. Even after compiling again Synopsys didn’t fix our hold time problem? Yes, it is bizarre, but we specifically have to request Synopsys to fix hold time problems. This is off by default.

```
dc_shell> set_fix_hold clk
```

```
dc_shell> compile -map_effort medium
```

```
dc_shell> report_timing -delay min
```

Good! Now it is fixed. Add the following lines to your synthesis script. Add them after the “**compile -map_effort medium**”, but before the timing and area reports.

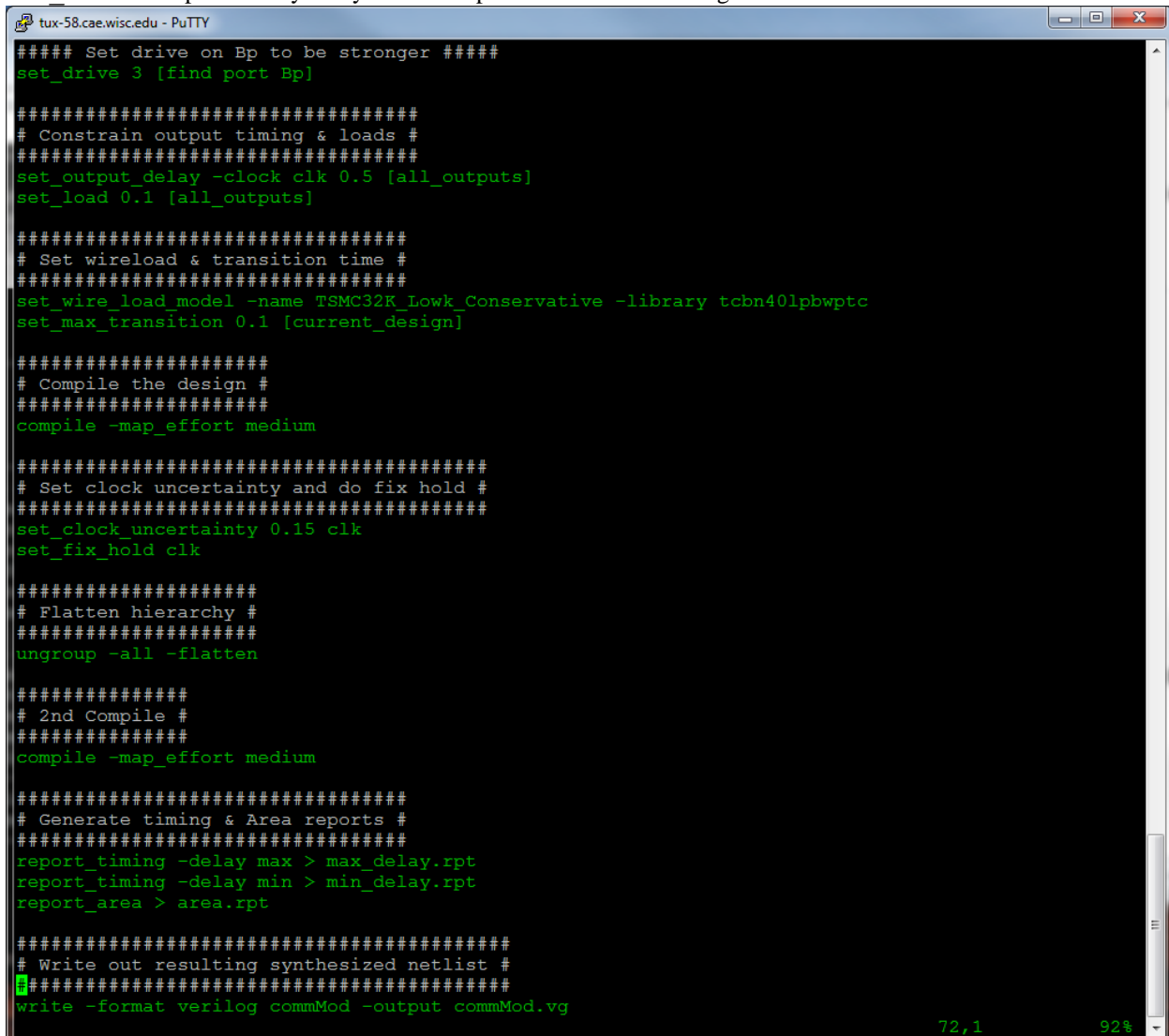
```
set_clock_uncertainty 0.15 clk
set_fix_hold clk
compile -map_effort medium    # 2nd compile
```

Our design is looking pretty good now, except for the issue with max delay timing. We have this strange problem with the *Bp* input/output having this large mysterious delay at the input port. This stems from the fact that *Bp* is an

inout, and is subject to both the **set_load** command (0.1pF) and the **set_driving_cell** command. The cell chosen (AO33D0BWP) does not have the drive strength to drive a 0.1pF load with decent timing. We will fix this issue by overriding the driving strength of the *Bp* input with a higher drive.

```
dc_shell> set_drive 3 [find port Bp]
```

Now we are telling Synopsys that as an input the *Bp* signal is driven with a gate with an equivalent output impedance of 3 ohms (pretty strong). Add this line to your synthesis script after the line that sets the drive on *rst_n*. The last portion of your synthesis script should look something like:



```
tux-58.cae.wisc.edu - PuTTY
#### Set drive on Bp to be stronger ####
set_drive 3 [find port Bp]

#####
# Constrain output timing & loads #
#####
set_output_delay -clock clk 0.5 [all_outputs]
set_load 0.1 [all_outputs]

#####
# Set wireload & transition time #
#####
set_wire_load_model -name TSMC32K_Lowk_Conervative -library tcbn40lpbwpdc
set_max_transition 0.1 [current_design]

#####
# Compile the design #
#####
compile -map_effort medium

#####
# Set clock uncertainty and do fix hold #
#####
set_clock_uncertainty 0.15 clk
set_fix_hold clk

#####
# Flatten hierarchy #
#####
ungroup -all -flatten

#####
# 2nd Compile #
#####
compile -map_effort medium

#####
# Generate timing & Area reports #
#####
report_timing -delay max > max_delay.rpt
report_timing -delay min > min_delay.rpt
report_area > area.rpt

#####
# Write out resulting synthesized netlist #
#####
write -format verilog commMod -output commMod.vg
72,1 92%
```

Now write your synthesis script to disk and remove the entire design from Synopsys, and run the synthesis script from scratch. Check out the results.

```
dc_shell> remove_design -all
dc_shell> source commMod.dc
```

Synopsys is a large CAD tool that has been around for over 25 years. There are many more commands available to constrain and optimize. Pay attention in class, and we will cover a few more tricks. This tutorial, however, should be enough to get you on your way.