

Synthesis and APR Tools Tutorial

(Last updated: Oct. 26, 2008)

Introduction

This tutorial will get you familiarized with the design flow of synthesizing and place and routing a *Verilog* module. All the files needed to synthesize this module will be given to you for the purposes of this tutorial, but you will have to supply or modify the files in order to run your own modules in the future.

First, you will need to synthesize a *behavioral Verilog module* to generate a *synthesized Verilog netlist* using *Design Compiler*. Then you will have to auto place and route the *synthesized Verilog* into layout and finally import the auto placed and routed layout into *Cadence* and run DRC and LVS. For the purpose of this tutorial, you will be building a multiplier.

The Example Design

Before you start, please first make sure that you remove the `.synopsys_dc.setup` file from your home directory from any previous course, so it doesn't interfere with you EECS427 project. The example design for this tutorial is an 8x8 bit multiplier, which can be located in the following directory:

/afs/umich.edu/class/eecs427/ibm13/synth_tutorial

Please copy this directory over to your personal eeecs427 space.

```
% cd /afs/umich.edu/class/eecs427/ibm13
```

```
% cp -r synth_tutorial ../07/<unique name>/cad6/.
```

Synthesis (*Design Compiler dc_shell*)

The synthesis process is controlled by a script file that the *Synopsys* tool *dc_shell* reads. The newest version of *dc_shell* uses the *TCL* script language

For documentation, run the command *sold* (*Synopsys Online Documentation*). You should, at the very least, look up each command in the synthesis scripts.

To run *dc_shell* you must invoke the *TCL* mode of the tool. It is also very helpful to store the verbose output of the program into a log file.

```
% dc_shell -xg_mode -tcl_mode -f <yourscript.tcl> | tee <log_file>
```

For this tutorial's purposes, the *TCL* file is located at: ***synth_tutorial/tcl/mult.tcl***. You must run that command while in the *TCL* folder for the pointers within the *TCL* file to work. Please examine the *TCL* file and try and understand what is going on.

```
% cd synth_tutorial/tcl
% dc_shell -xg_mode -tcl_mode -f mult.tcl | tee mult.dc.log
```

It is a good practice to use a "Makefile" to execute complex unix commands, and there is one create for you in this tutorial. Type the following command in the ***synth_tutorial/tcl*** directory, which will do the same thing as the previous command.

```
% make synth_mult
```

The Tcl Files

Now that you have seen that the script actually works, it might be a good idea to take a look at all the *.tcl* files.

mult.tcl: the main tcl script which calls all other scripts.
common.tcl: the common tcl script that sets up the standard cell library and search path location.
timing.tcl: the tcl script that set the clock period and input and output delay.

The Output Files

The main output file, located at ***synth_tutorial/verilog/mult.nl.v***, is the synthesized Verilog netlist file which contains a gate-level (structural) netlist made up of standard library cells. From this point on the netlist is both process dependent and technology dependent (remember that, in contrast, behavioral Verilog files are typically process *independent*). The Synopsys synthesis program also creates a log file and a report file. The log file is located in the ***synth_tutorial/tcl*** directory and is named ***mult.dc.log***. The log file tracks the progress of the tcl script. After each run, you should first check to see if any command gives you an error message. The report file can be found in ***synth_tutorial/tcl/mult.dc.rpt***. The report file gives you a summary of the quality of the synthesized netlist. It reports parameters such as area, power, and timing, which might be essential to your design. Another file of interest is the sdf file which gives you a more accurate model of the delay for verilog simulations. The sdf file can be found in ***synth_tutorial/sdf/mult.dc.sdf***. The post-synthesis sdf gives you a better idea of the delay of your design but is not always accurate since it is only based on the gate-level netlist (there is no physical routing in the synthesized netlist). The post-APR sdf is much more accurate because it contains parasitics due to interconnect.

Automatic Place & Route (APR)

The main tool for placement and routing is *Encounter*. While this tool does have a graphical interface, it also has a textual, command line driven interface found in the same terminal you run *Encounter* from. The graphical interface merely presents forms for you to enter in the needed information which is then printed out as commands. Thus, by looking at the log file the tool generates, you can quickly learn the commands you need to script the tool.

TCL Script

The *TCL* script sets different parameters for floorplanning, cell placement, power routing, clock generation, and I/O pins. Change the parameters as you see fit especially for the floorplan and the power stripes. For the purpose of this tutorial, the *TCL* file is provided at:

synth_tutorial/encounter/mult.tcl

Configuration File

The configuration file specifies the technology file locations. The files required by *Encounter* are the *.lef*, *.lib*, and *.v*. The *LEF* file contains metal information of the standard cells used for routing. The *LIB* file has timing information for placement and clock distribution network generation. The gate-level netlist with a *.v* file extension has the connectivity information (this will be a previously synthesized, or custom structural netlist, e.g., the ***synth_tutorial/verilog/mult.nl.v*** created earlier). For the purpose of this tutorial, the configuration file is provided at:

synth_tutorial/encounter/mult.conf

Pin Placement

In addition, you will need to create a pin-placement constraint file. By looking at the floorplan diagram, you have to decide the pin placement such that the congestion is minimized during global routing. There are two ways to create this file. One is to create them by hand and the other way is to create them using the “Pin Editor” under “Edit”. One thing to note is that *encounter* is a grid-based router, so any pins that are not on grid will not be routed. For the purpose of this tutorial, the pin I/O file is provided at:

synth_tutorial/encounter/mult.save.io

Timing Constraint

The file, “*mult.sdc*” creates the clock for the *encounter* tool. Make sure that the clock period and uncertainty match those in the synthesis script.

Running *Encounter*:

You can run *Encounter* in graphical mode until you become familiar with the tool and automate the

scripts to run in text mode.

Here is how you would invoke the executable:

```
% cd /afs/umich.edu/class/eecs427/f08/<unique name>/cad6/synth_tutorial/encounter
% encounter
```

Do not run *Encounter* with a “&” to run it in background mode because the current terminal will be used for entering the commands. If you accidentally run it in the background, you can bring the process to the foreground with the “fg” command.

For future CADs, do not forget to edit the *.conf file to change the pathname to your own files. You may also have to change the aspect ratio, the size of your floorplan, the number of power stripes, etc. in order to get a compact layout. Most likely you have to hit **View→Redraw** (hotkey ctrl+R) to see updated results of the commands you run. For the purposes of this tutorial, we will simply execute the files that have been provided with the following command while in **synth_tutorial/encounter/**:

```
% cd /afs/umich.edu/class/eecs427/f08/<unique name>/cad6/synth_tutorial/encounter
% encounter -init mult.tcl
```

Once everything is working, you should get several output files. The main files of interest are *.apr.sdf, *.def (which will be in: **synth_tutorial/def**), and *.apr.v (which will be in: **synth_tutorial/verilog**). The *.apr.sdf is an extracted timing file that lets you back annotate the parasitic for your *Verilog* simulations. Most of you should have noticed that the SDF file generated by *Design Compiler* has no information about wire delays and all the delay values corresponding to the wires is zero. Note that your placed design could be different from the synthesized structural netlist because of the insertion of clock tree. Hence it is important to verify the functionality of the *Verilog* file generated by *Encounter* with proper back-annotation of the corresponding sdf file.

The above command should simply run *Encounter* with the given *mult.tcl*, which calls *mult.conf* and *mult.save.io*. If all is successful, you should end with an “*encounter* >” prompt. Just enter “win” at the prompt and the graphical interface should show up with the placed and routed layout of the multiply module.

For reference, *Encounter* is a *Cadence* software so in order to get the manual, you need to run

```
% /usr/caen/ic-5.141_usr4/share/bin/cdsdoc &
```

to pull up the manual. To get access to all the cadence software that is used in EECS427, please copy the .cdsdoc_path file in the **ibm13/setup** directory to your home directory.

Importing Layout (Cadence)

Open *ICFB* and create a new cadence library called “mult” in cad6 using the instructions from Tutorial 1. Don’t forget to attach it to the *cmr8sf* technology library as you have for your previous libraries.

On the *CIW* (Command Window), click **File**→**Import**→**Stream**. The window shown below in Figure 1 will open up. Fill in the form as shown (also fill in the user-defined data and options sub-forms).

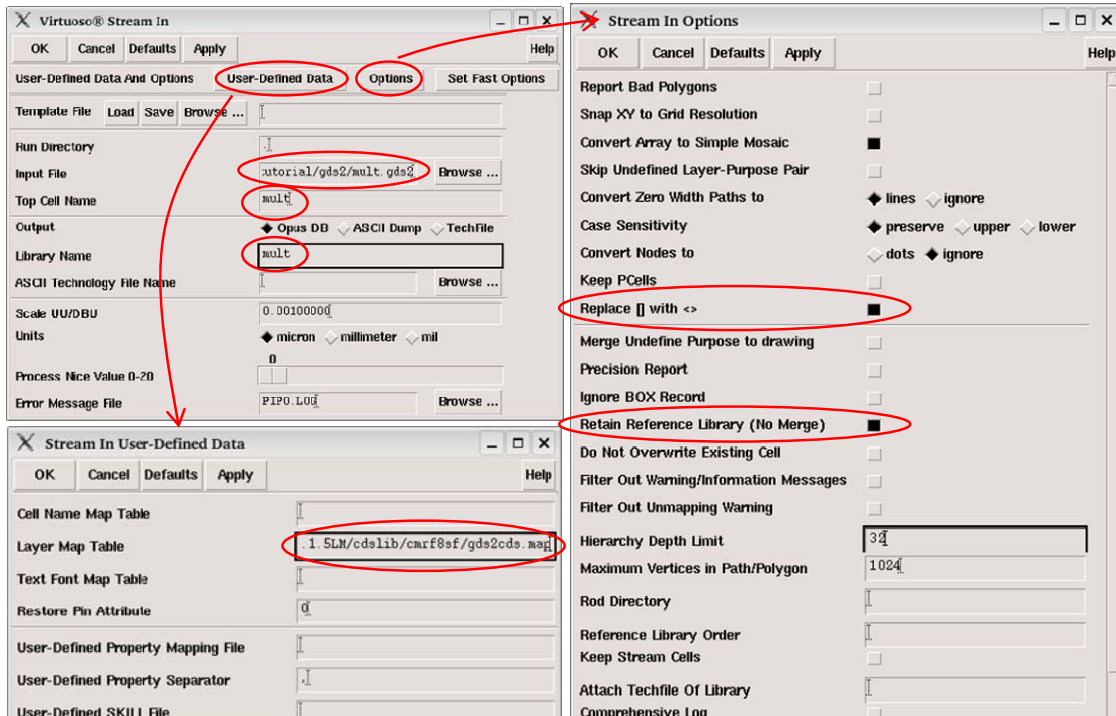


Figure 1: Stream in Forms.

Input File:

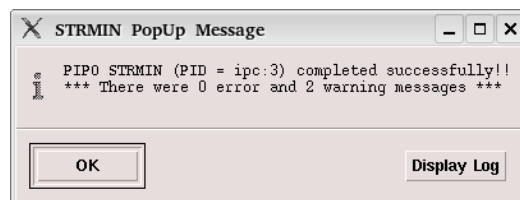
/afs/umich.edu/class/eecs427/w08/<unique name>/cad6/synth_tutorial/gds2/mult.gds2

Layer Map Table:

/afs/umich.edu/class/eecs427/ibm13/cmr8sf/reILM/cdslib/cmr8sf/gds2cds.map

Then Click OK.

You will see a pop-up message at the end of the process.



Next, you need to add the GRLOGIC box around the whole design before the DRC check. Finally, save the design and move on to DRC and LVS.

Design Rule Check (DRC)

DRC check should be the same as any previous DRC.

Layout versus Schematic (LVS)

Please make sure that you are using the LVS rules from the *cmrf8sf* tech library. In order to run LVS, it is slightly different from what you may be used to. First, because we only have a placed and routed verilog structural netlist, you will have to convert it to a spice netlist. This can be done using the program *v2lvs* (which stands for *Verilog* to LVS). If you are not familiar with it, there is an example of it in the *synth_tutorial/encounter/Makefile*, or you can just type “% *make cdl_mult*” in that directory and a cdl file would be made in the *synth_tutorial/cdl* folder. On top of that, you have to include the *Spice CDL* file with all the standard cell definitions which is located at:

/afs/umich.edu/class/eecs427/ibm13/UMSTD13/lvs_netlist/UMSTD13.cdl

It would be easier if you make a link to the standard cell cdl file in the *cad6/Calibre/LVS* folder. In order to fit the two files, you have to click the bottom arrow beside the input box. Don't forget to tell to hit “add at end” when prompted when you add the *CDL* file.

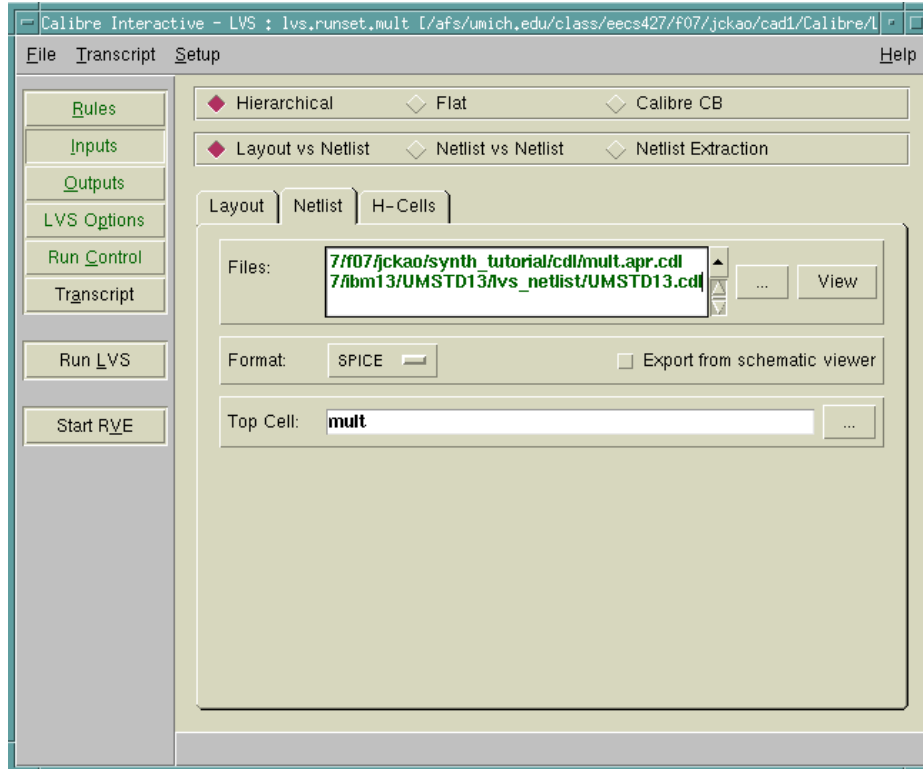


Figure 2: LVS Netlist Input.

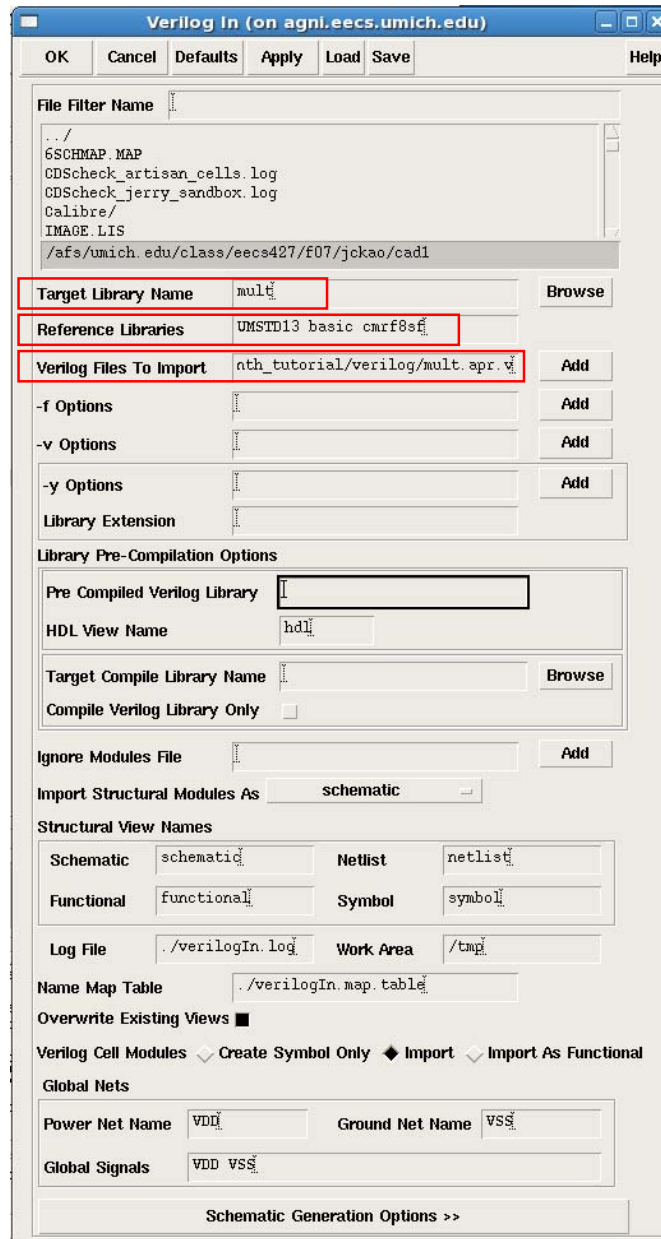
Now change the input format to *Spice*. Before you start the LVS run, make sure that “Recognize all gates” is selected in the “LVS options” under the “Gate” tab. Remember to save the runset file before you quit so you don’t have to change the setup every time. If all goes well then you should get your smiley face!



Congratulations! You’ve just synthesized & placed and routed your first *Verilog* module!

Import the Verilog Netlist to Create a Schematic View

One way to start of the verilog simulation in the familiar NCverilog environment is to import the verilog netlist back to the cadence as a schematic view. You can also use this view to do LVS as well. To start the import process, go to the “CIW” Window and select **File** → **Import** → **Verilog**, and then the following window will show up:



You need to change the following 3 fields:

Target Library Name

Reference Libraries

Verilog Files to Import

(Note: You might see a handful of warnings saying that the “Verilog definition for module...” was not found. It should follow by saying that the symbol from UMSTD13 is being used. This is expected and acceptable. If you view the resulting schematic, the correct symbols from the reference library, UMSTD13, should be used.)

After you complete the import you can run the *Verilog* simulation based on your imported schematic.

Post-Layout Simulation (NC-Verilog + SDF Back-annotation)

Next, we use *.apr.v and *.apr.sdf (generated from *Encounter*) to do the post-layout simulation. The reason that *Verilog* simulation is preferred over *Spice* simulation is that *Verilog* with sdf annotation is much faster and also provides enough timing accuracy.

The steps are much the same as what you did in Tutorial 1, except that you need to turn on some additional options, and add more lines in your *testfixture.template*.

1. Initialize Design (the same as Tutorial 1).
2. Go to **Setup**→**Netlist** and click “Support Escape Names,” “Netlist Explicitly,” and “Declare Global Locally.”

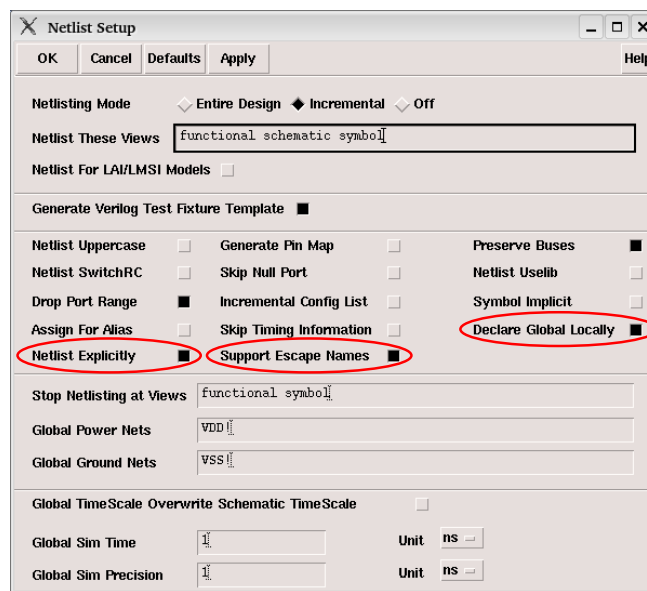
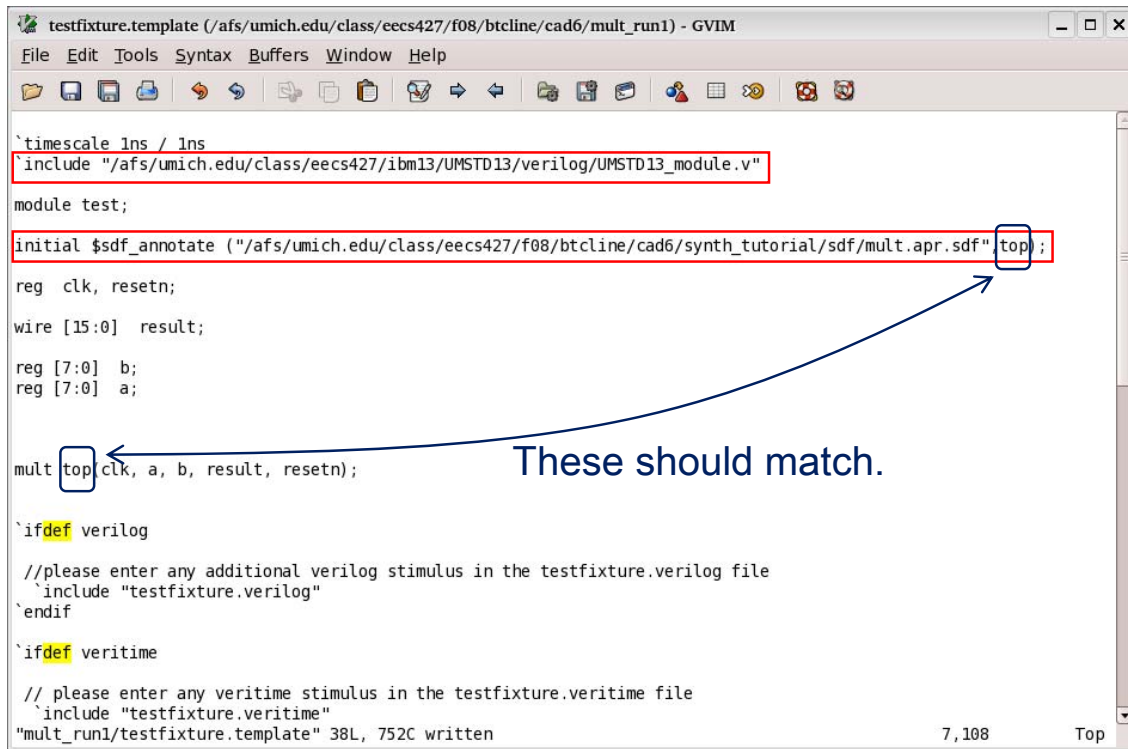


Figure 3: NC-Verilog Netlist Setup.

3. Generate Netlist (the same as Tutorial 1).
4. Go to the mult_run1 directory, open “testfixture.verilog”, and give the test signals you want (the same as Tutorial 1).
5. Go to the mult_run1 directory, open “testfixture.template”, and add the following two lines (shown on next page in Figure 4) to include “UMSTD13_module.v” and “mult.apr.sdf”.
6. Simulate (the same as Tutorial 1).
7. Open the waveform viewer to see the simulation result (the same as Tutorial 1).



```
testfixture.template (/afs/umich.edu/class/eecs427/f08/btcline/cad6/mult_run1) - GVIM
File Edit Tools Syntax Buffers Window Help

`timescale 1ns / 1ns
`include "/afs/umich.edu/class/eecs427/ibm13/UMSTD13/verilog/UMSTD13_module.v"

module test;
`initial $sdf_annotate ("/afs/umich.edu/class/eecs427/f08/btcline/cad6/synth_tutorial/sdf/mult.apr.sdf" top);
reg clk, resetn;
wire [15:0] result;
reg [7:0] b;
reg [7:0] a;

mult top clk, a, b, result, resetn);

`ifdef verilog
//please enter any additional verilog stimulus in the testfixture.verilog file
`include "testfixture.verilog"
`endif

`ifdef veritime
// please enter any veritime stimulus in the testfixture.veritime file
`include "testfixture.veritime"
"mult_run1/testfixture.template" 38L, 752C written
7,108 Top
```

Figure 4: Modifying *testfixture.template*.