# Synthesis and APR Tools Tutorial

(Last updated: Oct. 13, 2009)

## 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 eecs427 space. % cd /afs/umich.edu/class/eecs427/ibm13 % cp –r synth tutorial ../f09/<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>

#### EECS427

For this tutorial's purposes, the *TCL* file is located at: **synth\_tutorial/syn/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/syn % 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/syn* 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/syn* 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/syn* /*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.

2

## 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/f09/<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/f09/<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, \*.lef (which will be in: **synth\_tutorial/lef**), 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 functionally of the V*erilog* 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 % acroread /afs/umich.edu/class/eecs427/f09/resources/Encounter\_Manual.pdf &

to pull up the manual.

## 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).

🗙 Virtuoso® Stream In 📃 🗆	×	✤ Stream In Options	_ 🗆 🗙
OK Cancel Defaults Apply	Help	OK Cancel Defaults Apply	Help
User-Defined Data And Options User-Defined Data Options Set Fast Option	ons	Report Bad Polygons	
Template File Load Save Browse		Snap XY to Grid Resolution	
Run Directory		Convert Array to Simple Mosaic	<b>=</b>
Input File Rowse		Skip Undefined Layer-Purpose Pair	
Top Cell Name		Convert Zero Width Paths to	🔶 lines  ignore
Output Opus DB 🔷 ASCII Dump 🔷 TechFile		Case Sensitivity	♦ preserve ◇upper ◇lower
Library Name		Convert Nodes to	⇔dots ♦ ignore
ASCII Technology File Name Browse		Keep PCells	
Scale UU/DBU 0.001000000	•	Replace [] with <>	
Units		Merge Undefine Purpose to drawing	
Process Nice Value 0-20		Precision Report	
Error Message File PIP0.L00 Browse		Ignore BOX Record	
V Starson In User Defined Date	<b>T</b> VI	Retain Reference Library (No Merge)	$\triangleright$
A stream in oser-beined bata		Do Not Overwrite Existing Cell	
OK Cancel Defaults Appry	Help	Filter Out Warning/Information Messages	-
Cell Name Map Table		Filter Out Unmapping Warning	
Layer Map Table	nop	Hierarchy Depth Limit	32[
Text Font Map Table		Maximum Vertices in Path/Polygon	1024
Restore Pin Attribute	-	Rod Directory	¥.
	_	Reference Library Order	I.
User-Defined Property Mapping File		Keep Stream Cells	
User-Defined Property Separator		Attach Techfile Of Library	<u>I</u>
User-Defined SKILL File		Comprehensive Log	

Figure 1: Stream in Forms.

Input File:

/afs/umich.edu/class/eecs427/f09/<unique name>/cad6/synth\_tutorial/gds2/mult.gds2

Layer Map Table:

#### /afs/umich.edu/class/eecs427/f09/resource/gds2cds.map

Then Click OK.

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



# **Design Rule Check (DRC)**

DRC check should be the same as any previous DRC.

# 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*  $\rightarrow$ *Import*  $\rightarrow$ *Verilog*, and then the following window will show up:

		hinderid	Veril	og In		
OK Can	cel Default	ts Apply	Load Save			I
File Filter Nar	ne 1					
/						
mult.apr.v						
mult.v						
(afa (ami ah	odu (olaco	(0000497 (	-00 (mana (arm	th tutorial (	rorilog(	
	. euu/ciass.	/0003427/1	vuo/wsma/syn	un_cucorrar/	Desuga	
larget Librar	y Name	Caug			Browse	
Reference Lil	oraries	eecs427a	rtisan basic	cmrf8sf		
Verilog Files <sup>-</sup>	To Import	th_tutor	ial/verilog/	mult.apr.v	Add	
-f Options		Ĭ.			Add	
-v Options		Ĭ			Add	
-y Options		Ĭ.			Add	
Library Exte	nsion	Ĭ.				
Library Pre-C	ompilation C	options				
Due Caranila	l Varilan Lik					
Pre Complie	a veniog Lib	rary ≞				
HDL View N	ame	hdl	1.			
Tarriet Com	nilo Library I	Name			Browse	
Compile Ver	log Library I	Only				
compile for		отпу				
Ignore Modul	es File	μ.			Add	
Import Struct	ural Module	s As	schematic	-		
Structural Vie	w Names		-			
		iď	Netlist	netlist		
Schematic	schemat:					
Schematic Functional	schemat:	nalį	Symbol	symbol		
Schematic Functional Log File	schemat:	nalį́ ogIn. logį́	Symbol Work Area	symbol <u>i</u>		
Schematic Functional Log File Name Map Ta	schemat: function	nalį́ ogIn.logį́ ./verilo	Symbol Work Area gIn. map. tabl	symbol <u>i</u> /tmpi		
Schematic Functional Log File Name Map Ta Overwrite Ex	schemat: function ./verilo ble isting Views	mal <u>ĭ</u> ogIn.logĭ ./verilo 5∎	Symbol Work Area gIn.map.tabl	symbol <u>i</u> /tm <u>pi</u>		
Schematic Functional Log File Name Map Ta Overwrite Ex Verilog Cell M	schemat: function ./verild uble isting Views odules $\blacklozenge$ C	malĭ ogIn.logĭ ./verilo ;∎ Yreate Sym	Symbol Work Area gIn.map.tabl bol Only 今 h	symboli /tmpi ei	rt As Functiona	
Schematic Functional Log File Name Map Ta Overwrite Ex Verilog Cell M Global Nets	schemat: function ./verild bble isting Views odules $\blacklozenge$ C	nalį́ ogIn.loǵ ./verilo ; <b>—</b> Yreate Sym	Symbol Work Area gIn. map. tabl bol Only 🔷 h	symboli /tmgi .ei	rt As Functiona	1
Schematic Functional Log File Name Map Ta Overwrite Ex Verilog Cell M Global Nets Power Net N	schemat: function ./verild bble isting Views odules $\blacklozenge$ C	mali ogIn. logi . ∕verilo : ■ Create Sym	Symbol Work Area gIn.map.tabl bol Only $\diamondsuit$ h Ground Ne	symbol /tmp e t Name VS	rt As Functiona	1
Schematic Functional Log File Name Map Ta Overwrite Ex Verilog Cell M Global Nets Power Net M	schemat: function ./verild bble isting Views odules ◆ C lame VDDI	malĭ ogIn.logĭ ./verilo s∎ Dreate Sym Vsš	Symbol Work Area gIn. map. tabl bol Only $\sim$ I Ground Ne	symbol] /tauğ ď mport ↓ Impo tt Name VSď	rt As Functiona	1
Schematic Functional Log File Name Map Ta Overwrite Ex Verilog Cell M Global Nets Power Net N Global Signa	schemat: function ./verild bble isting Views odules $\blacklozenge$ C lame VDD Is VDD	mal[ ogIn.logĭ √verilo :■ reate Sym	Symbol Work Area gIn. map. tabl bol Only  In Ground Ne	symbol] /tnuğ di mport ↓ Impo tt Name VSğ	rt As Functiona	I

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 eecs427artisan is being used. This is expected and acceptable. If you view the resulting schematic, the correct symbols from the reference library, eecs427artisan, should be used.)

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

# Layout versus Schematic (LVS)

LVS check should be the same as tutorial1.



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

## 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).
- Go to Setup →Netlist and click "Support Escape Names," "Netlist Explicitly," and "Declare Global Locally."

🗙 Netlist Setup			_								
OK Cancel Defau	ilts Apply			Help							
Netlisting Mode	🔾 Entire Design 🔶 Inc	remental 🔷 Off	r								
Netlist These Views functional schematic symbol											
Netlist For LAI/LMSI Models											
Generate Verilog Test Fixture Template											
Netlist Uppercase	Generate Pin Ma	p 🗌	Preserve Buses								
Netlist SwitchRC	Skip Null Port		Netlist Uselib								
Drop Port Range	Incremental Conf	īg List 🔄	Symbol Implicit								
Assign For Alias	Skip Timing Infor	mation	Declare Global Locali	у 🔳							
Netlist Explicitly	Support Escape	Names 🔳									
Stop Netlisting at Views	s functional symbol										
Global Power Nets	VDD I <u>ľ</u>										
Global Ground Nets	VSSI										
Global TimeScale Overwrite Schematic TimeScale											
Global Sim Time	1.	Unit	ns 🖃								
Global Sim Precision	1 <u>ĭ</u>	Unit	ns 💷								

Figure 2: 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 line (shown on next page in Figure 3) to include "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).



Figure 3: Modifying testfixture.template.