

EE 330

Laboratory Experiment Number 11

Design and Simulation of Digital Circuits using Hardware Description Languages

Spring 2019

Naming Matters: The name of your files *and Verilog modules* will be important as you go through this lab. We designed the lab to have them all be named "boolean_function". This does not need to be names Boolean_function, but it does need to be named the same *every place it is used*. If you name it something different the commands in this document will not work verbatim.

Contents

Purpose:	3
Background.....	3
Part 1: Boolean Function	4
1.1 Simulating Behavioral Code	4
1.2 Simulation of a Boolean System	4
Part 2: Verilog Synthesis with RTL Compiler	5
Introduction	5
RTL Compiler.....	5
Setup Standard Cell Library.....	6
Standard cell files for RTL Compiler and Encounter	6
Import Schematic to Cadence Virtuoso after Synthesis.....	9
Part 3: Layout of Digital Circuits with Encounter.....	11
Introduction	11
SoC Encounter.....	11
Running SoC Encounter.....	11
Floorplanning.....	15
Power Planning.....	16
Placement	17
Routing	17
Filler Cells.....	18
Verification.....	18
Part 4 Import the layout into Cadence.....	21
Possible Errors	24
Red Cells with no Layers	24
DRC and LVS error corrections	25
Deliverables:.....	26

Purpose: The purpose of this experiment is to develop methods for using Hardware Description Languages for the design of digital circuits.

Background

Hardware Description Languages (HDL) are widely used for representing digital systems at both the Behavioral and Structural levels. Full functionality of a digital system can ideally be captured in a HDL and the design of digital circuits is often accomplished within the confines of an HDL framework. At the appropriate level in the HDL, CAD tools can take a design down to the mask level for fabrication in silicon with minimal engineering intervention at the Physical Level in the design process. A system appropriately represented in a HDL can be simulated to predict not only the functionality of sequential and combinational logic circuits, but also timing information about how those circuits are predicted to perform when instantiated in silicon.

The two most widely used HDLs today are Verilog and VHDL. There is considerable similarity between these two languages and engineers are expected to be proficient in both. In this laboratory experiment we will limit our discussion to Verilog which has become more popular in US industry today. Specifically we will focus on how a HDL can be used for design and simulation of digital integrated circuits.

Appendix A of the Weste and Harris text has a brief discussion of Verilog and students should become familiar with the material in this appendix. There are also numerous books and WEB sites devoted to a discussion of Verilog. Beyond the basic introduction to Verilog discussed in this laboratory experiment, students will be expected to take the initiative to develop their own HDL skills to the level needed to support the digital design component of this course.

Part 1: Boolean Function

1.1 Simulating Behavioral Code

Before simulating, a test bench for the design under test needs to be written. The test bench is needed to specify the time and value properties of inputs. For given inputs, certain outputs are expected which can be plotted during simulation and compared to expected results.

Example Verilog code: inverter

```
module inverter (vout, vin);
    output vout;
    input  vin;
    assign vout = ~vin;
endmodule
```

Example test bench for inverter

```
module inverter_TB(vin, vout);
    input vout;
    output vin;
    reg vin;
    inverter name(.vin(vin), .vout(vout));
    initial
    begin
        vin = 1'b0;
    end
    always
        #20 vin = ~ vin;
endmodule
```

This TB will initially set *vin* at a (1 bit) value of 0, and after every 20 time units, will invert it.

1.2 Simulation of a Boolean System

We will now simulate the full Boolean Function created in lab 4. Create a Verilog behavioral file for the function you created in Lab 4 and a corresponding test bench to test its functionality. Name this Verilog file “**boolean_function.v**” and name the module “**boolean_function**”. Simulate and compare results with those found in Lab 4. They should agree, but one obviously took less time to create and simulate. Compare the time and effort it took to implement your Boolean function in Cadence vs in Verilog.

NOTE: To create the behavioral design for the Boolean function should require only one line of code after creating the inputs, outputs, wires, etc.

Part 2: Verilog Synthesis with RTL Compiler

Introduction

The purpose of this section is to develop the concept of synthesis of digital systems from a hardware language description of the system. This is the intermediate step in a three-step design process which starts with the development of a HDL description, followed by synthesis into a set of standard cells. The third step is placing and routing the cells to create a layout of the circuit which can then be fabricated. The generalized digital circuit design flow, with the topics discussed in this tutorial highlighted, can be found below:

1. Design in HDL (Verilog file)
2. **RTL Compiler** (Verilog file --> Synthesized Verilog file)
3. Encounter (Synthesized Verilog file --> Layout)
4. Cadence
 1. Layout Import (Encounter --> CIW Import Stream)
 2. Netlist Import (Synthesized Verilog file --> Verilog Import)
5. LVS Verification

RTL Compiler

RTL Compiler is a tool that synthesizes and then optimizes behavioral circuit elements.

Logic synthesis translates textual circuit descriptions like Verilog or VHDL into gate level representations. Optimization minimizes the area of the synthesized design and improves the design's performance. The HDL description can be synthesized into a gate level net-list composed of instances of standard cells.

Once the gate-level net-list is finished, it can be imported into Encounter and used to create a layout. Consistency between the layout and schematic can then be verified with an LVS within Cadence.

NOTE: if there are problems with the later steps check the ~/.software file for the line "EDI"

Setup Standard Cell Library

First, download the Standard Cell Library – TAR file and RTL Tar File from the course website. Then move them to the in ee330 folder for use. They can be extracted through the GUI or with,

```
tar -xvzf <Filename>
```

Now change the **cds.lib** file in your ee330 directory by adding the following line.

```
DEFINE OSU_stdcells_tsmc018 ~/ee330/OSU_stdcells_tsmc018
```

By opening up Virtuoso, the OSU018 standard cells should now appear (you may need to restart Cadence if it was open). Click around to the different cells to make sure the layout and schematic views can be viewed. The standard cell library should now be available to use.

Standard cell files for RTL Compiler and Encounter

Now, the necessary files for use in Genus and Encounter must be obtained to run the digital design flow. We will be using the OSU018 files in this lab. This cell library uses the same TSMC02 process that we should be familiar with from the earlier labs.

Download the rtl.tar.gz file which contains a folder called 'lab11_ee3300'. This folder contains the LIB, LEF, and Verilog files for the OSU standard cells. Move rtl.tar.gz into your ee330 folder and extract it.

Now the files are downloaded and extracted, which can be confirmed by looking for the *lab11_ee330* folder in the location used above. Throughout this lab, carefully watch for path names and make sure they are pointing to the correct directories and files.

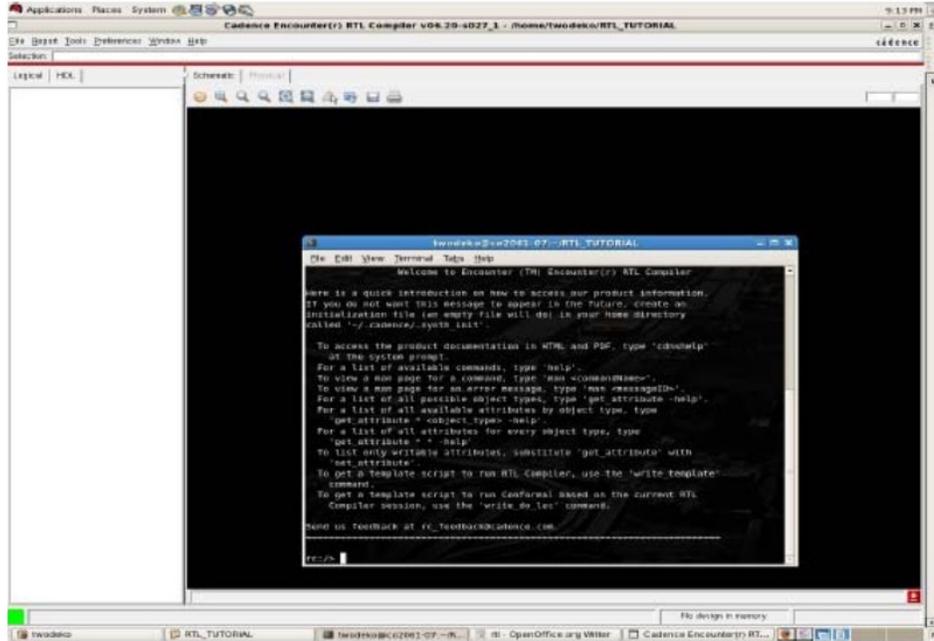
Now, copy over the Verilog file (boolean_function.v) which is created and tested in Part 1 into the lab11_ee330/rtl/ folder. You can do it through GUI or by using the following commands:

```
cp ~/<project_dir>/boolean_function.v  
~/<project_dir>/lab11_ee330/rtl
```

You should now have a folder named *lab11_ee330* that contains folders and files needed for the rest of the tutorial. Source the Genus program, then change directories to *lab11_ee330* and type the following to begin RTL Compiler:

```
source /etc/software/genus  
cd ~/ee330/lab11_ee330/syn  
genus -gui
```

A GUI will show and your terminal will change to the genus command line.



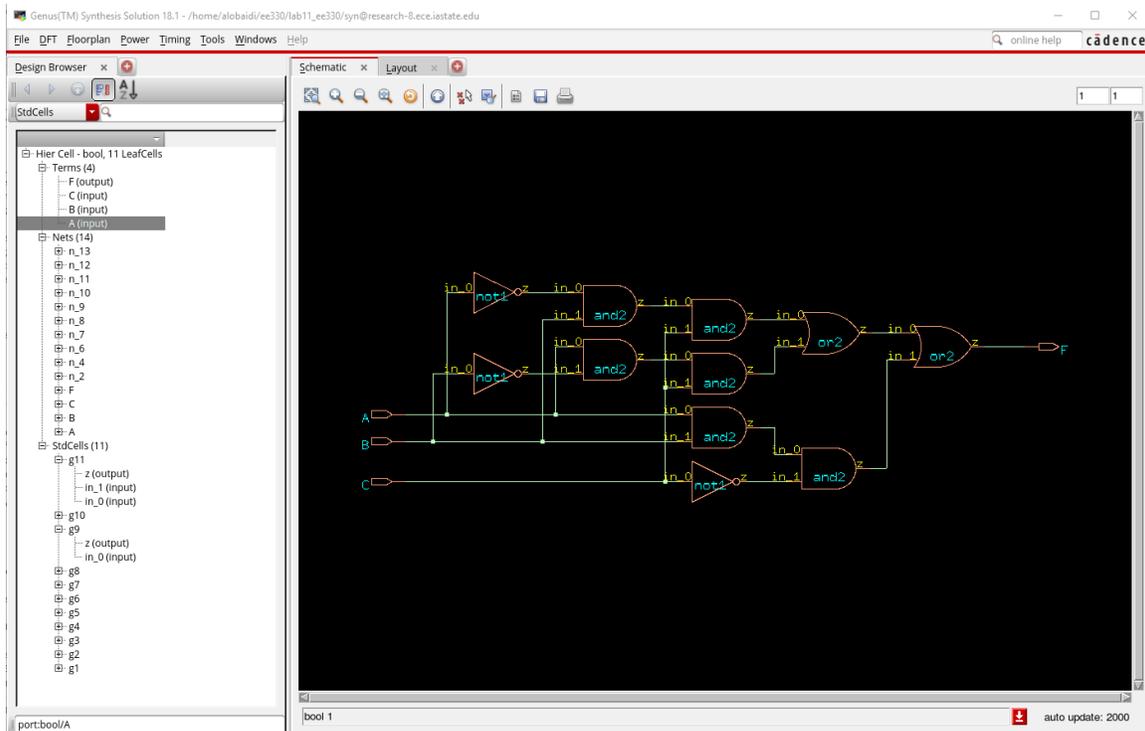
In the command window that has a “genus:/>” prompt, type the following to load the correct standard cell libraries and run the synthesis script:

```
set OUTPUT_DIR ./run_dir
set_db lib_search_path ../libdir
set_db library {osu018_stdcells.lib}
```

You will get multiple warnings, but should have no errors. Once the library is set, you need to load and elaborate your Verilog file. In this case, it was placed in the rtl folder and can be processed with the following commands:

```
load -v2001 ../rtl/boolean_function.v
set DESIGN "boolean_function"
elaborate $DESIGN
```

When Verilog is elaborated, each line is evaluated and replaced with a combination of logic gates. Go to your GUI window, there should be an empty “Layout” view, add a “Schematic” view. You should get a window with circuit blocks that represent the input Verilog, which was the following for our Boolean function:



At this point, RTL Compiler has generated an unoptimized version of the Verilog you provided. The final step is “synthesizing” the elaborated design, which optimizes the schematic already created and integrates your design with the TSMC standard cell library. While there are many optimization options available to you, we will skip them for this tutorial. The design can be synthesized with the following command:

```
synthesize -to_mapped -eff high -no_incr
```

The GUI window will not update to show the optimized design, but the design file was updated. Exporting the synthesized Verilog file is done with these two commands:

```
write -mapped > ${DESIGN}_synth.v  
write_sdc > ${DESIGN}.sdc
```

Import Schematic to Cadence Virtuoso after Synthesis

After synthesis, we want to import the gate-level netlist created in RTL Compiler into Cadence Virtuoso. Open Virtuoso and in the library manager create a new library with the name `boolean_function` (or whatever name you want to use for the library for this lab). When the technology file options pop up select “Attach existing technology library” and then select `NCSU_TechLib_tsmc02`.

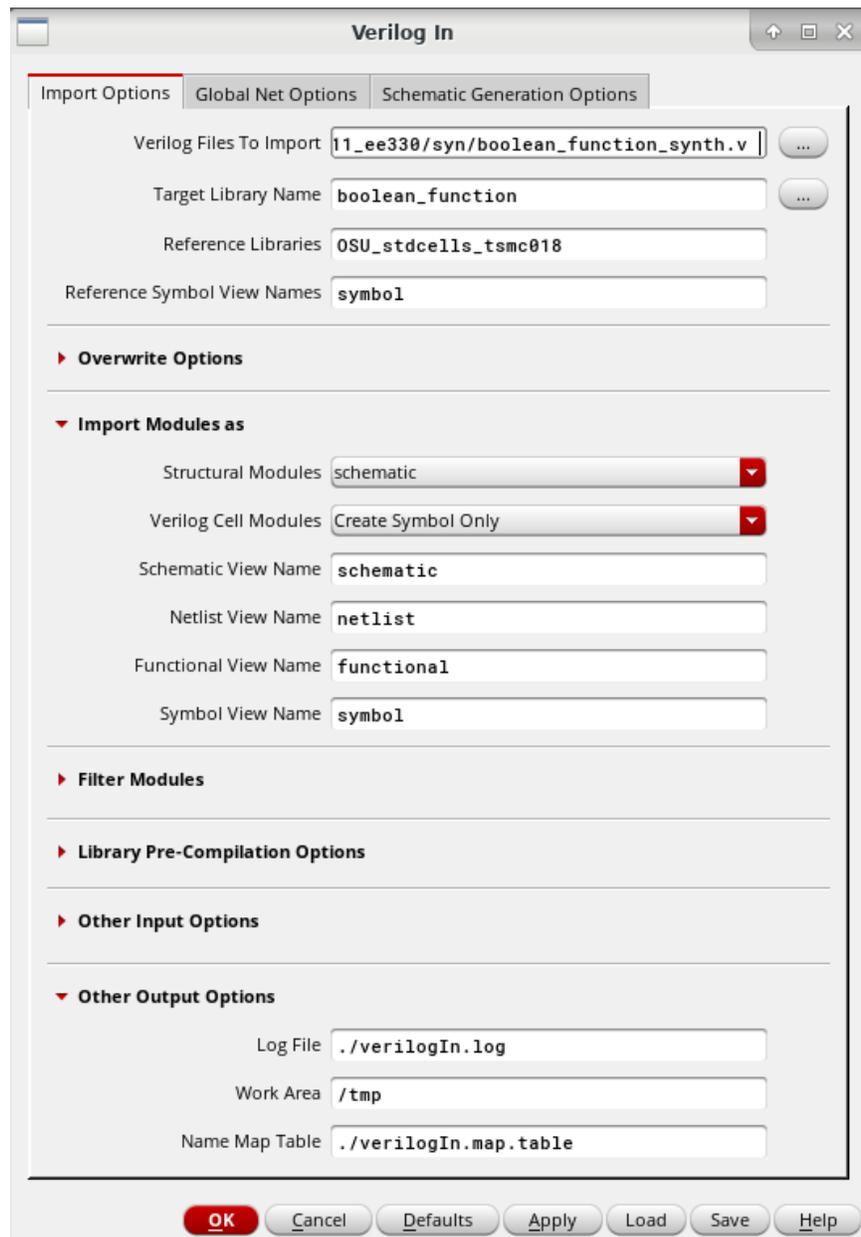
Once the library is created, change to the CIW (where the errors and other messages appear) select *File->Import -> Verilog*.

The following screen will appear. Fill it in as is shown in the screen shot except replace the Verilog file with your synthesized Verilog file created by Genus and your own library name. This file will end with `_synth.v` if following the above steps.

(*Make sure to use the OSU library name that you defined above in the Reference Library field – for example “*OSU_stdcells_tsmc018*”)

Note: An image of what to put in is on the next page

Make sure you filled everything right



The import process should complete and may have a few errors or warnings. Ignore for now. Go to the schematic cell view created by the import process and open in Virtuoso. The synthesized optimized design should appear with the lowest level of hierarchy being the standard digital blocks (gates) made of transistors in the TSMC02 process. If not something is wrong. Retry the import process or possibly synthesis before asking for help from the TAs.

Part 3: Layout of Digital Circuits with Encounter

Introduction

In a typical digital design flow, a hardware description language is used to model a design and verify the desired behavior. Once the desired functionality is verified, the HDL description is then taken to a synthesis tool such as RTL compiler. The output of RTL compiler is a gate-level description of the desired circuit. The next step is to take this gate-level description to a place-and-route tool that can convert it to a layout level representation. In this tutorial, you will be introduced to a place-and-route tool called SoC Encounter. For the purposes of this tutorial, it will be assumed that you have a synthesized Verilog file ready to be placed and routed.

The generalized digital circuit design flow, with the topics discussed in this tutorial highlighted, can be found below:

1. Design in HDL (Verilog file)
2. RTL Compiler (Verilog file --> Synthesized Verilog file)
3. **Encounter (Synthesized Verilog file --> Layout)**
4. Cadence
 1. Layout Import (Encounter --> CIW Import Stream)
 2. Netlist Import (Synthesized Verilog file --> Verilog Import)
5. LVS Verification

SoC Encounter

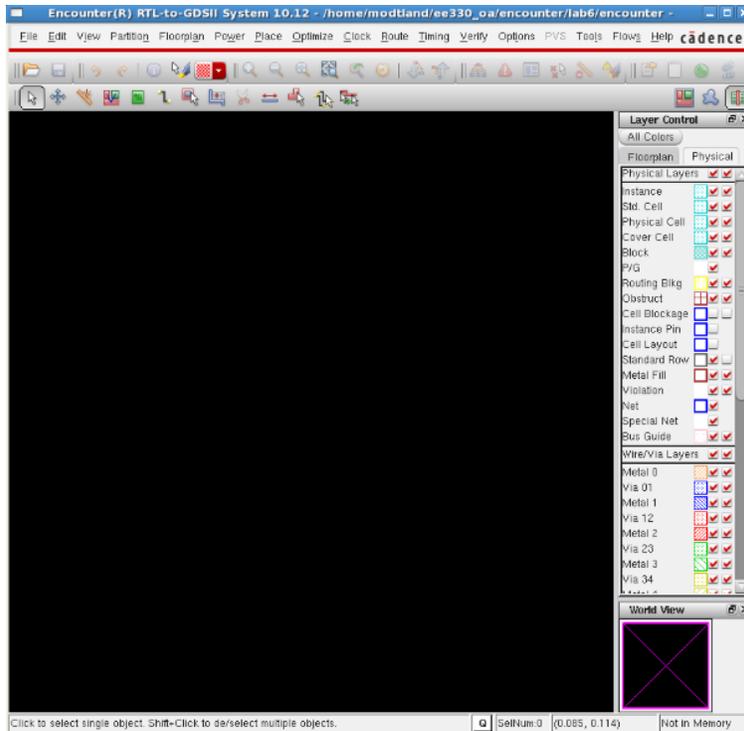
SoC Encounter, in its simplest form, allows a user to create a layout from a synthesized HDL file. Once the synthesized file is imported, Encounter provides a variety of tools to create a floorplan for your design, place standard cells, and automatically connect power rails and local routing. While this tutorial provides a brief overview of Encounter, there are many other optimization and layout options available.

Running SoC Encounter

In the lab11_ee330 folder, there should be a folder named encounter, enter that folder. Encounter can be started from here with the following command:

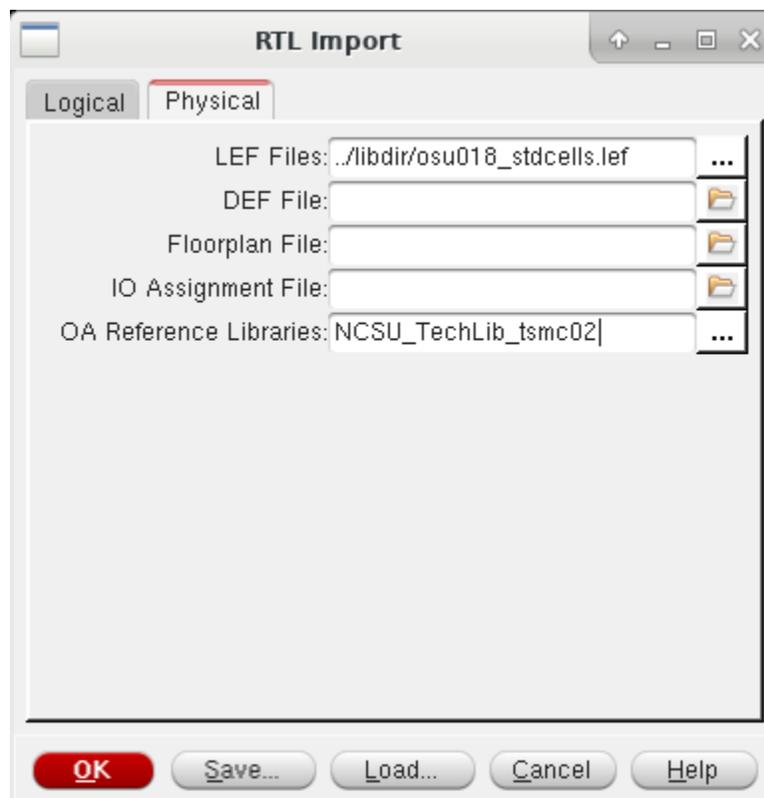
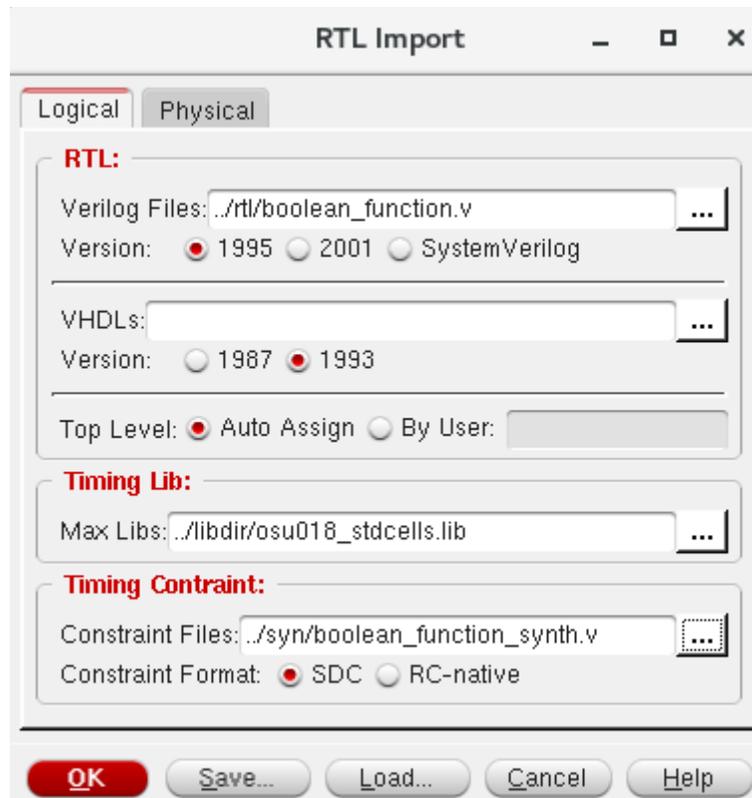
```
encounter
```

The command prompt will change to encounter> and you will be presented with a GUI, which has been shown below.



Importing Synthesized Verilog

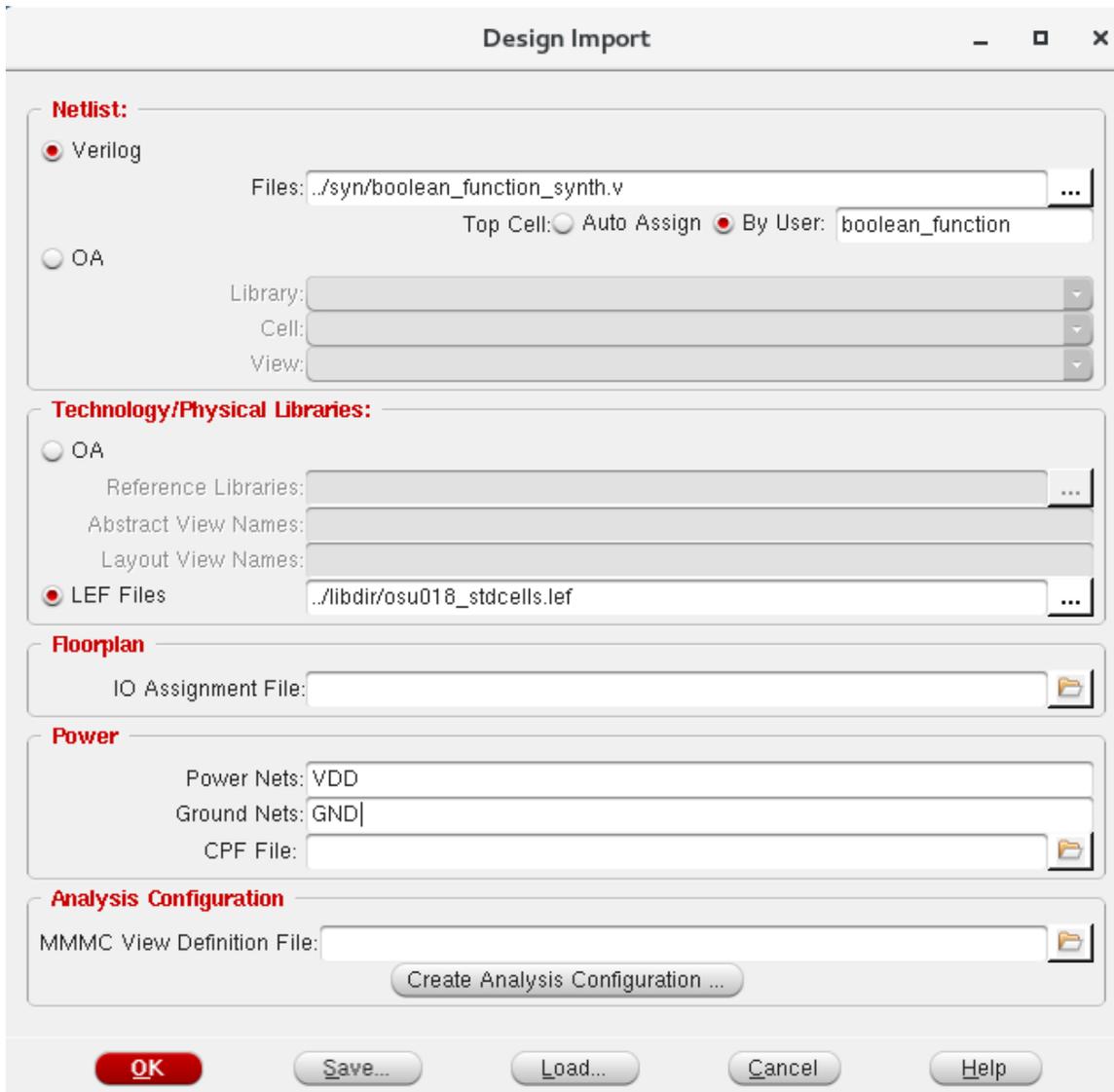
Once Encounter is initialized, you need to import your synthesized HDL description file. Go to *File -> Import RTL*. Fill in the Verilog Files field with the Verilog files used in the design. Then give a top level name to the design. (Images of what to do are on the next page) Under the max timing library field, load the *osu018_stdcells.lib*. Next, under Timing Constraints, load the SDC file created by RTL Compiler. Move to the Physical tab and load the LEF file for the OSU standard cells called *osu018_stdcells.lef* which was included in the *libdir* directory. Finally, in the OA Reference Libraries, add the *NCSU_TechLib_tsmc02* tech library. This will allow the tool to reference the tsmc02 library for layer information. Click OK and make sure the RTL is successfully imported by looking at the terminal for errors.



An error might occur, ignore it and repeat the process and it should work. Next go to File-> Import Design and under Files select ../syn/Boolean_function_synth.v

Do not import the OA under Technology/Physical Libraries, instead click the LEF Files button and choose ../libdir/osu018_stdcells.lef

Under Power Nets put VDD and for Ground Nets put GND



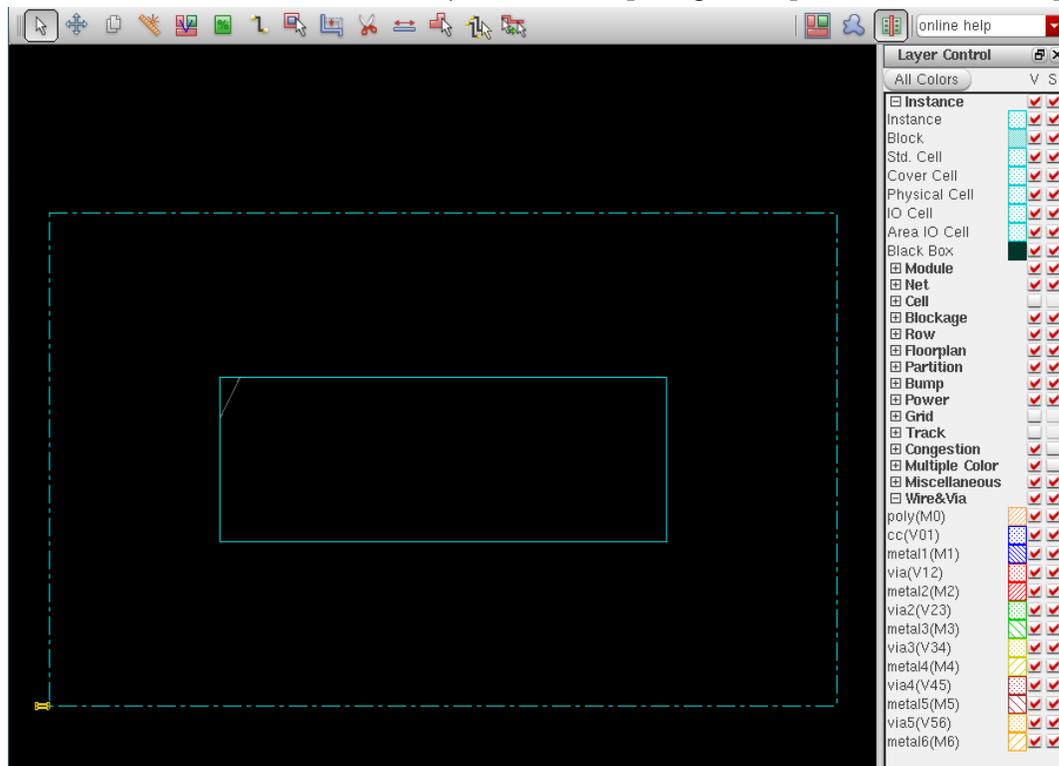
Floorplanning

The Encounter GUI should now display an empty “die” and the information box in the bottom-right corner should read “Design is: In Memory.” The next step is to specify a floorplan for your layout. Floorplanning is done to specify the dimensions of your layout and spacing between the core (or area where the standard cells are placed) and power/signal routing. Open up the floorplanning options by clicking on the *Floorplan* menu and selecting *Specify Floorplan*.

Options that you will need to pay attention to are:

- Aspect ratio – height/width ratio of the core.
 - Suggested: 1.0 (this will change to the closest workable number)
- Core Utilization – the amount of core area used for placing standard cells. A larger value means that your design will be fairly compact but routing may be difficult, if not impossible. Smaller values will drastically increase the design area.
 - Suggested: 0.5 (this will change to the closest workable number)
- Core Margins by – specifies the distance between the core and edge of the die. The margin area needed is proportional to the number of input/output (I/O) pins.
 - Suggested: 10 for core to left, right, top, and bottom

Once these values are entered, the layout die/core spacing and aspect ratio should be updated.



Power Planning

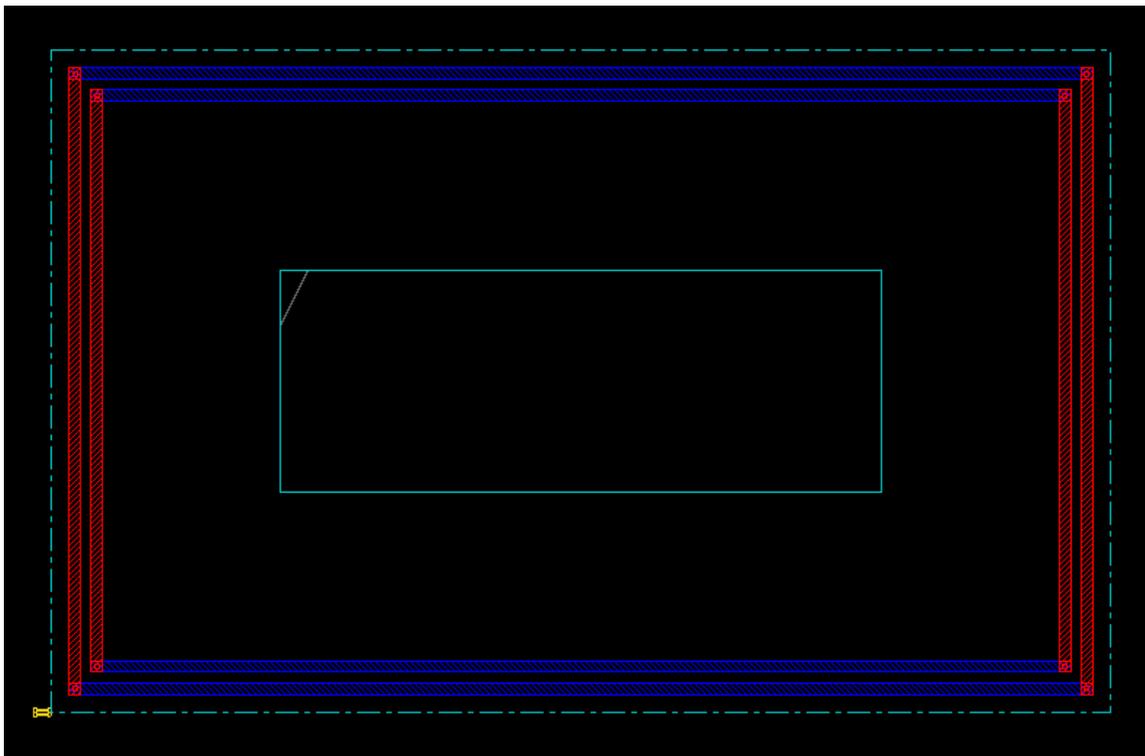
In order to connect your supply voltages to the standard cells, power and ground need to be available on all sides of the die. This is done by adding power rings and specifying global net connections.

Select the *Power* menu and click on *Power Planning > Add Rings*.

A dialog box will pop up and allow you to specify the ring type and configuration. You can plan power in various modes to facilitate various design styles. Since it is preferable to have the routing done inside the power rings, it is a better idea to have them around the I/O boundary so choose this option. If it is not already entered, add “GND VDD” (without quotes) to the Net(s) field.

In the Ring Configuration section, verify that the width and spacing comply with the process design rules and are a multiple of lambda.

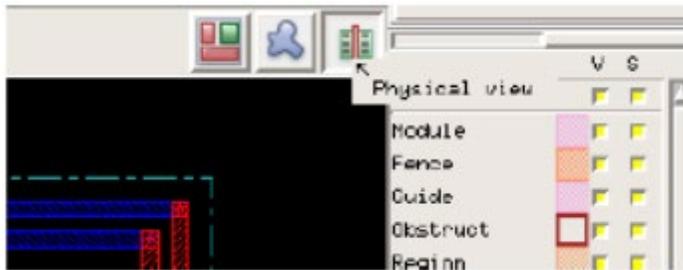
Once finished, you should see two rings that run inside the I/O boundary. It should look like the following:



Connection to the global nets is done automatically if the nets are given the same names as was done in the setting of the power nets above. If all was done correctly, power planning should be finished.

Placement

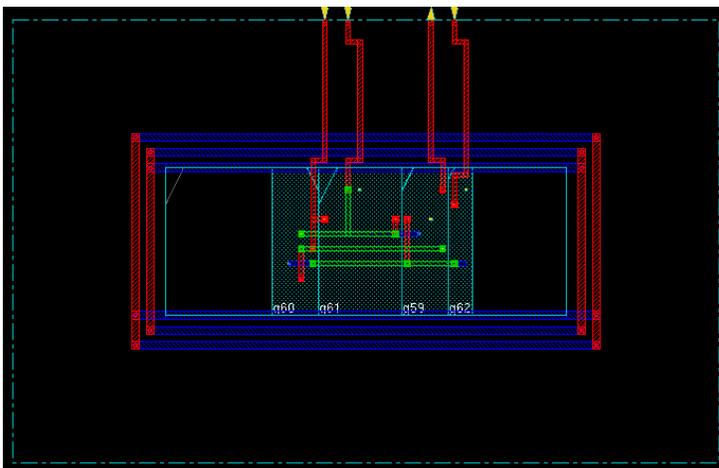
Now we are ready to perform the placement. At the end the cells used in the design will be placed so that they don't overlap and there is enough space for routing in between them. Select the *Place* menu and then *Place Standard Cells*. In the dialog box that appears, leave the default settings and hit OK. In the main GUI window, you can verify that the standard cells were placed by selecting the Physical View Icon in the upper-right:



You should see the cells placed in the middle of the core. The IO pins have shifted position accordingly as well.

If placement fails, it is most likely because Encounter cannot place the cells in the given area without causing gate overlaps. To solve this problem, you must start over by reinitializing the floorplan with relaxed constraints. The best constraint to change is the row utilization factor. By lowering this factor, you can increase the likelihood of placement without overlaps.

The final placement should look similar to the following:



Routing

The built-in routing options within Encounter can now be used to connect the supply nets and create interconnects between the standard cells.

Power and ground routing is performed with the “Special Route” tool. This can be found

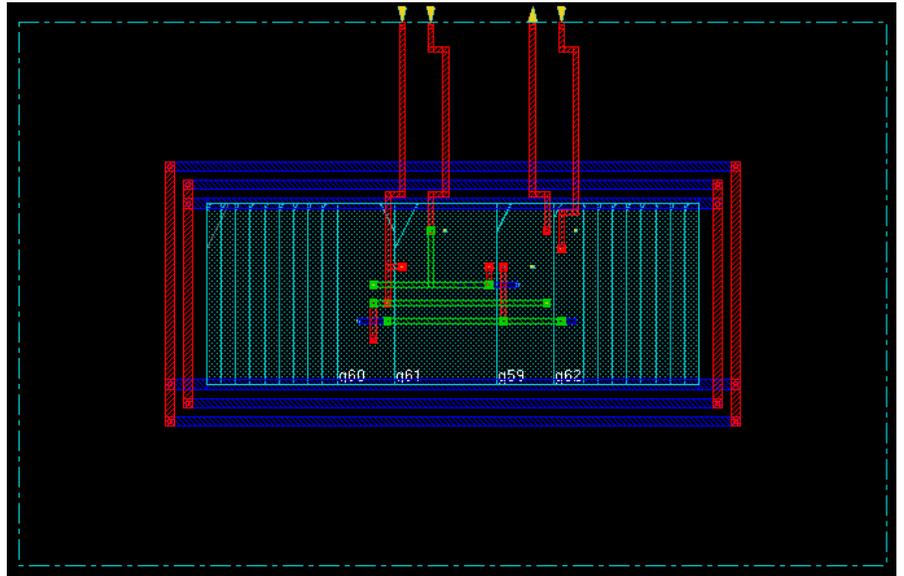
by selecting the *Route* menu and then *Special Route*. In the dialog box that appears, unselect *Block pins*, *Pad pins*, and *Pad rings*. After clicking OK, you should see horizontal and/or vertical metal power routing added to your design.

Once power routing is complete, the remaining connections between standard cells and the die I/O can be routed using the “NanoRoute” tool. Select *Route* and then *NanoRoute* and finally *Route...* to enter the dialog box. You do not have to make any changes and can click OK. Your design should now resemble the following:

Filler Cells

If you import the complete layout from Encounter into Cadence at this point, you may have DRC errors due to nwell layers not being spaced far enough. You can fix these errors by hand in Cadence or you can add pre-designed filler cells with Encounter. These cells provide continuity for nwell layers and power rails between placed cells.

Filler cells are placed by selecting the *Place* menu and then *Physical Cell -> Add Filler*. In the dialog box that appears, click “Select” in the Cell Name(s) field and select the appropriate fill layers. FILLER CELLS IN THE OSU018 LIBRARY ARE JUST CALLED ‘FILL’ AND THERE IS ONLY ONE OF THEM. You can now click OK to fill in the blank regions in the core of your layout.



Verification

Before exporting your layout, you need to verify that there are not any geometric errors that occurred during placement or connectivity errors from routing.

This can be done by selecting *Verify Geometry* and *Verify Connectivity* in the *Verify* menu. Reports will be generated and placed in the directory that you started Encounter from. You can also look at the terminal used to start Encounter and see the results of the verification.

If no errors exist in either geometry or connectivity, the layout is good to fabricate. Save the design by selecting *File -> Save Design*. This may open to save in OA format, change to encounter format, give it a name and save.



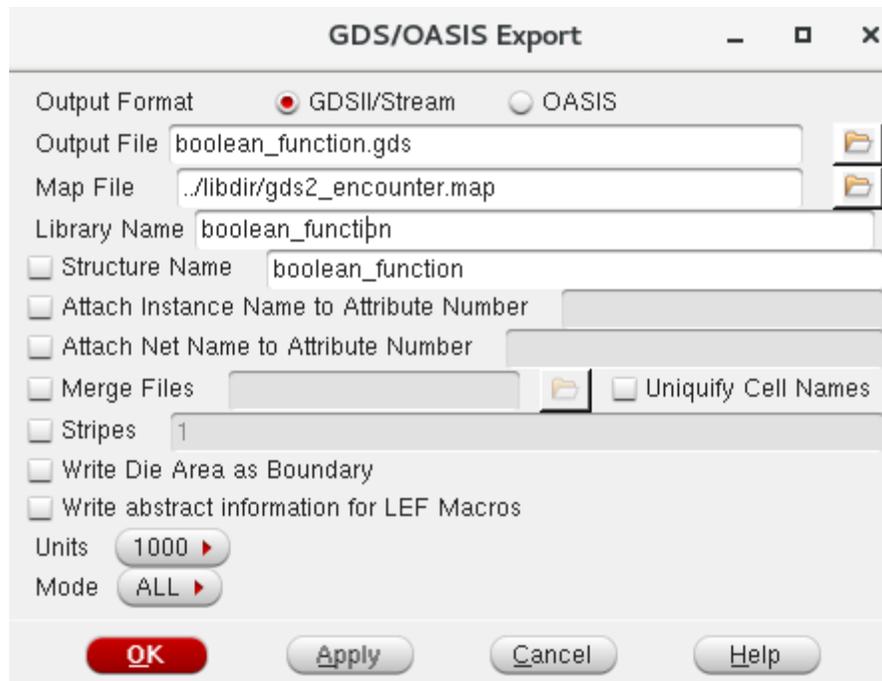
GDS Export of Layout from Encounter

Once the layout is complete in Encounter (post-routing and post-filler cells), then you need to perform an export of the layout. This is done in Encounter by going to *File -> Save -> GDS/OASIS...*

In the window that appears, select the GDSII radio button. Then type an output file name in 'Output File' field such as *boolean_function.gds*. In the map field select the 'gds2_encounter.map' file located in the *libdir* directory.

In the Library Name, type the Library in Cadence you would like to add the design to. Use the same library you used to import the Verilog schematic.

Leave all other fields as default making sure the units are '1000' and mode is 'ALL'. Click OK. Look at the terminal and make sure Stream out has finished successfully.



Part 4: Import the layout into Cadence.

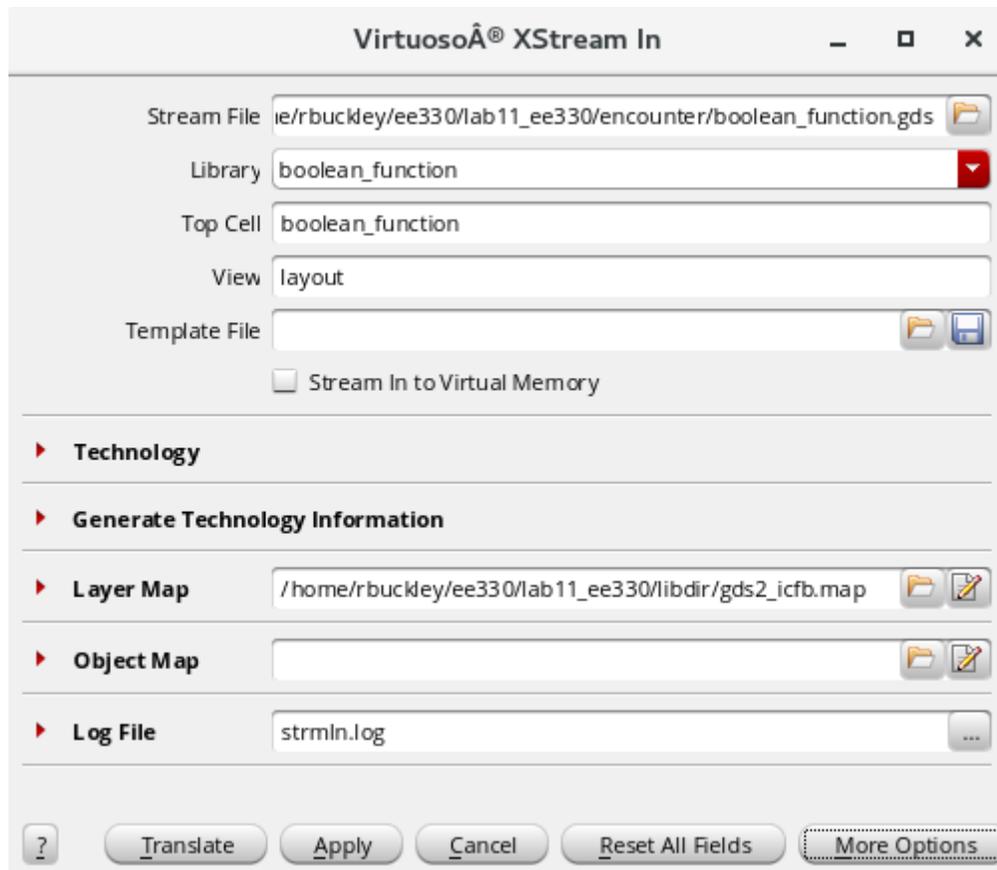
6. Design in HDL (Verilog file)
7. RTL Compiler (Verilog file --> Synthesized Verilog file)
8. Encounter (Synthesized Verilog file --> Layout)
9. Cadence
 3. Layout Import (Encounter --> CIW Import Stream)
 4. Netlist Import (Synthesized Verilog file --> Verilog Import)
10. LVS Verification

GDS Import of Layout into Virtuoso

After creating a layout in Encounter and creating the GDS file, open Virtuoso. Click on the CIW (Virtuoso window where errors and messages appear) and click *File -> Import -> Stream*. This may produce an error the first time. Try again and it should work and bring up an import window.

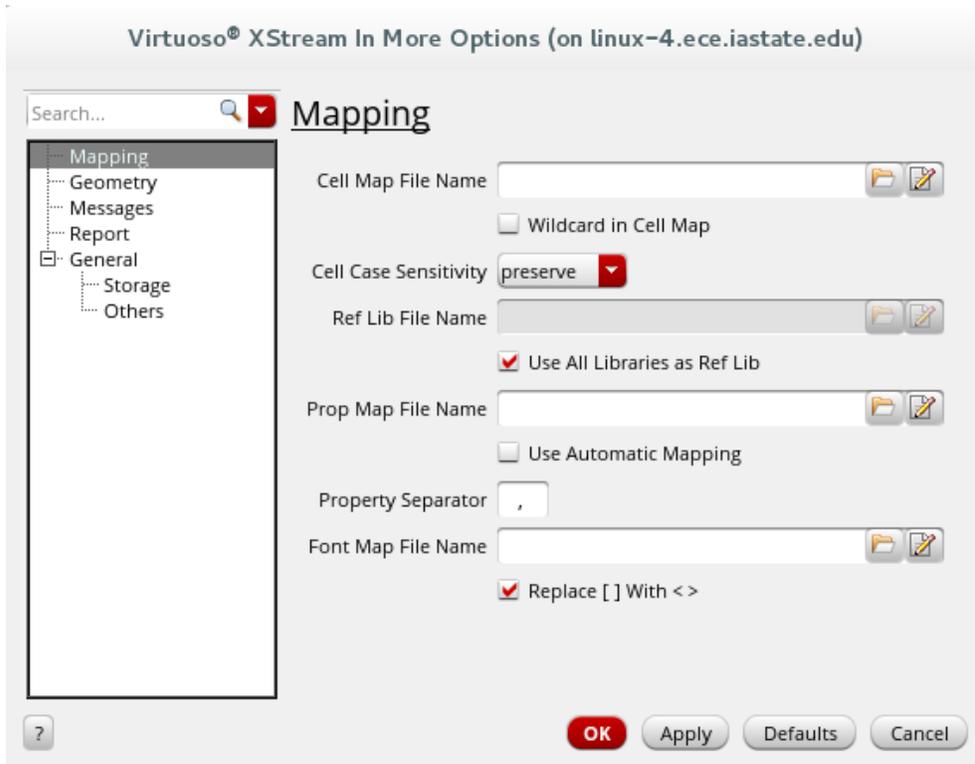
On the stream in window choose the GDS file created in Encounter as the 'Stream In' file.

Under library, type the name of the library you would like to save the module in. Note, this does need to be the same as the name of the library given in the stream out in Encounter. Then give a name to the top level cell once it is imported.

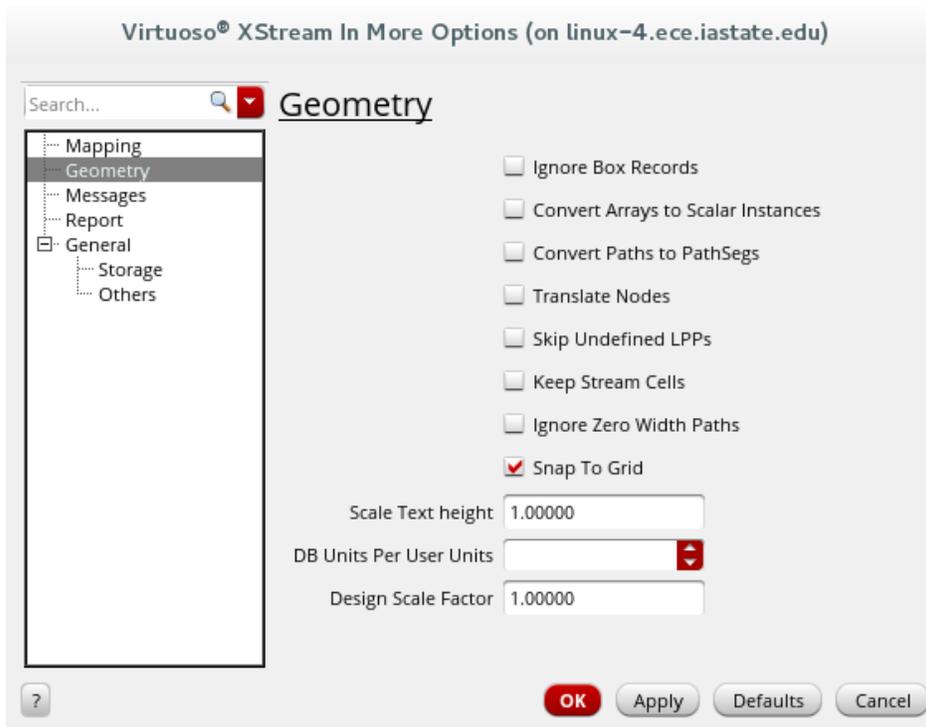


Note: The above assumes you are streaming in to a folder that does not have an attached technology file. If one is attached already do not include anything in the Attach Technology File section. (Also, there is more to do on the next page, under the More Options button)

Go to More Options, then in the Mapping tab check the “Replace [] with <>” box and “Use All Libraries as Ref Lib” box.



Go to the Geometry tab. Select Snap To Grid.



Now we are ready to translate the file and import the design. Click *Translate*. If a box to save the libraries comes up make sure to save the libraries as something. The process will take a little bit. When complete there should be a log file and a message box that appears. This should have 0 errors and only a few warnings if not 0.

Go to the library created in the Library Manager and try to open the layout of the design. The standard cells should appear. Hit *Ctrl+F* to see the layout view.

Possible Errors

Red Cells with no Layers

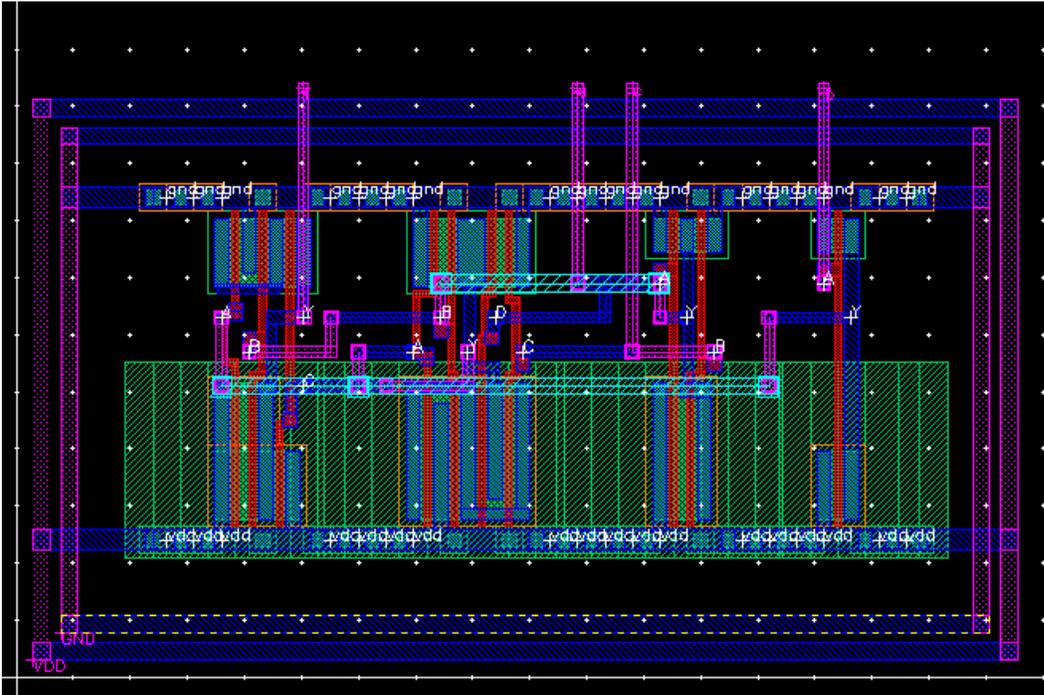
If the cells appear but are red and disappear when you press Shift+F then there is a problem with the technology file in the folder. You should try copying the layout into a library with a known good technology file. If that does not work delete the library and attempt to stream it in again.

This can be solved through a slightly complicated process.

1. Copy the current library. This is just good practice.
2. Delete the original library.
3. Create the library from in Cadence (with the proper name, normally `boolean_function`)
4. Import the schematic created from RC
5. Stream in the file from Encounter, but do not check (or uncheck) the box “Use all libraries as ref. Lib” in the More Options -> Mapping area.
6. Steam in the file again, this time with the “Use all libraries as ref. Lib” checked
 - a. This should give 0 errors and 7 warnings
7. Check the Layout and see if it worked.

DRC and LVS error corrections

At this point, the synthesized schematic and layout from Encounter have been imported into Virtuoso. DRC will not pass due to “dubiousData” errors, the Encounter stream is creating metal layer text that should have been text layer. These can be deleted, but before doing so you should add pins. Encounter is adding the name of what the pin should be where the pin should be placed, but does not actually create the pins. We advise creating the pins with the label name then deleting the label.



Congrats on completing this tutorial. Please provide a summary of the steps required for synthesizing and performing layout with Cadence RTL Compiler and SoC Encounter. Explain what the main steps do and provide screen shots of major steps as well as the two verification steps at the end.

Deliverables:

- Verification sheet
- Introduction:
 - Discuss what is done in lab
 - Discuss why is it useful
 - Think about how making digital circuits is easier via Verilog rather than Cadence
- Part 1 –Boolean Function:
 - Boolean function Verilog code
 - Boolean function waveform output
 - Comparison between Boolean function implementation in Cadence and in Verilog
- Part 2 - Verilog Synthesis with RTL Compiler:
 - What is an RTL compiler?
 - Picture of your 1st synthesis (from GUI)
 - Picture of your optimized synthesis (from Cadence)
 - Picture of transistor level implementation of optimized block
 - Discuss the synthesis process
 - Discuss how is this tool helpful
 - Analyze how optimization cut down number of transistors need to implement your design
- Part 3 & 4 - Layout of Digital Circuits with Encounter:
 - What is SoC encounter ?
 - Briefly discuss the layout process
 - Talk about the different steps (Floorspacing etc)
 - Picture of layout (from Cadence)
 - Discuss how is this tool helpful
 - Think about how much time the software need vs how much you needed when you did the layout
- Conclusion:
 - Discuss what you learned in lab
 - Add any comments on what you liked or what would you want to see changed