ECE 337 Lab 7: Layout of a USB Transmitter

In this lab, you will:

- Synthesize the provided simple USB Transmitter using Synopsys.
- Generate Several Layouts (Physical Designs) of the USB Transmitter using SOC Encounter.
- Utilizing timing analysis capabilities of SOC Encounter to analyze the USB Transmitter design.

1. Copying Setup Files

In a UNIX terminal window, issue the following command:

```
setup7
```

This command is actually an alias to a batch/script file in the ece337/Class0.5u directory that will copy the files that are necessary for the completion of this lab. **If you have trouble with this step, please ask for assistance from your TA.**

2. General Comments Regarding Lab 7

The purpose of this lab is to provide the student with exposure to layouts, physical designs that are constructed for designs based upon STANDARD CELLS. Throughout this course, you have been applying a standard cell design approach to your lab assignments and your projects. The Verilog source code that you write is mapped, via Synopsys, to a standard cell library. This library is comprised of cells that have previously been laid out in such a fashion that all the cells have the same height. This allows tools known as 'Place and Route' tools to take a design in a Hardware Description Language, such as Verilog or VHDL, and create a Layout for that design without the designer having to layout the individual cells or connect the cells together to generate the design. At this point, some of you may be thinking: "This sounds like the perfect way to design chips. Why do companies, such as AMD, IBM, and Intel not employ this scheme to design their respective microprocessors?" This is such a good question, that it deserves a place on the Lab 7 Check-Off sheet.

Why do you think that companies do not employ a fully Standard Cell, Place and Route approach to the high performance portions of their microprocessor designs? (Actually, these companies do use the approach for less critical portions of their microprocessor designs.)

The reason that you are concerned about layout in this course is due to the fact that layout is the actual representation of what a design will look like in silicon. The familiar symbols for NAND and NOR gates are ideal for working in the schematic arena; however, you would find it difficult to locate a NAND gate in an actual layout if you were looking for the schematic symbol. In actuality, gates in silicon are constructed from the interaction of various layers in the final design. Therefore, fabricated devices are not simply 2-dimensional constructions. They are in fact 3-dimensional. During the fabrication process for a chip, it is built in stages. Each stage utilizes a different “mask”.

This “mask” is essentially a glass plate that is transparent in certain areas and opaque in others. The “mask” defines where certain layers are allowed to be located in the fabricated design. Thus after one layer has finished its “masking” stage, the next “mask” layer is begun. However, the different layers are actually insulated from each other by materials similar to Silicon Dioxide, the “oxide” in Metal Oxide Semiconductor. If the layers are insulated from each other, how does one connect them if they need to be connected? This connection is accomplished by elements known as “vias” or “contacts”.

In most cases, “vias” refer to connections for metal interconnect layers and “contacts” refer to connections between active/diffusion and metal to form the source/drain connections on a transistor.

Also, it should be noted that the layouts you will be producing in this lab will contain Input/Output (I/O) pads. When a chip is completely fabricated, I/O pads are necessary in order to solder wires to the chip. These wires are then connected to the actual pins on the package. The pins on the package are what your chip uses to interface to the outside world. I/O pads serve several functions. The main function that is of importance to this class is that I/O pads act as buffers to drive signals into and out from the chip.

Although we will be adding I/O pads for this lab, not all designs need I/O pads. You may be wondering: “So, if a layout does not have I/O Pads, what good is it?” To answer this question, one can think about large microprocessor designs. In these designs, not every block interfaces to the outside world, so it is advantageous to have a program that can produce a layout for an internal block that simply has pins, but no I/O pads. These pins provide a means by which other blocks can connect to the inputs and outputs of the current block. In addition, I/O pads are typically much larger than the cells that are used to create the block; therefore, if every block in a design required I/O pads, the area of the overall design would grow quite rapidly. Designs with a larger area than necessary are not cost-efficient. In actuality this type of automated layout, layout having pins but no I/O pads, is done quite often in microprocessor design for blocks that are not timing critical, blocks that will not affect the performance of the design. This is done in an effort to shorten the design cycle as manual layout is much more time-consuming than having a tool do the layout for you.

3. Adding I/O pads

For this portion of the lab assignment, you must add pads to your synthesized design.

Design compiler will not add pads for you, so you must add them manually. Fortunately, we have provided a script that will automate the process. After you have synthesized the design you can add pads by running the following command in your Lab7 directory:

```
pads <design_name>
```

You might want to observe how the VDD and GND pads will appear after synthesis by examining the mapped version.

Once you have an error-free run of Synopsys and have added pads to your design, **have a TA check off your work up to this point.**

4. First Pass Layout of the USB Transmitter

Once you have synthesized your design, you are now ready to generate the layout for your design. The setup script should have copied files needed for this to your Lab 7 directory. One of the files copied was the encounter.tcl script. This is the script that automates the generation of the layout. Examine this script with your favorite text editor and see how the layout process flows. The commands needed to produce the layout are explained by the comments in the .tcl script. Also examine the configuration file encounter.conf that is utilized by the script to define your design name, path, etc.

As mentioned before, the layout you are going to generate is going to include I/O pads. As you might have read from the configuration file, you have to match the pad name in your synthesized Verilog code (inside your mapped directory with the .v extension) with the pad names defined inside the encounter.io file. So open the mapped file, look for the lab7_layout_designmodule and find the I/O pad definitions (the keyword for VDD is PADVDD, GND IS PADGND, output pads are PADOUT and PADINC is for input pads). Be advised that your mapped file may contain multiple modules, but you need to look for the module declaration of your **top level**. This usually is the last module in the file. Now, examine the encounter.io file that was copied by the setup script, it should look similar to the text on the following page.

```
# This file specifies how the pads are placed
# The name of each pad here has to match the
# name in the verilog code
# The Mosis padframe has 4 corners and 40 pads

Version: 2

Pad: U3 E
Pad: U4 E
Pad: U5 E
Pad: U6 E
Pad: U7 E
Pad: U8 E
Pad: U9 E
Pad: U10 E
Pad: U11 E
Pad: U12 E

Pad: U1 N

Pad: U13 W
Pad: U14 W
Pad: U15 W
Pad: U16 W
Pad: U17 W
Pad: U18 W

Pad: U2 S
```

The format of the pad declaration in the .io file is:

`Pad: PadName Direction`

So for example, Pad: U9 W means that pad U9 will be located in the western border of the chip. Direction can be N, S, W, E, NW, etc. (North, South, West, East, Northwest, etc.).

The prior encounter.io file defines pad locations for the following Verilog description.

```
module lab7_layout_design ( clk, rst_n, d_plus, d_minus, transmit,
write_enable, write_data, fifo_empty, fifo_full);

input  [7:0] write_data;
input  clk, rst_n, transmit, write_enable;
output d_plus, d_minus, fifo_empty, fifo_full;
wire  nclk, nrst_n, ntransmit, nwrite_enable, nd_plus, nd_minus,
nfifo_empty, nfifo_full;

wire  [7:0] nwrite_data;
    lab7_layout_design_t I0 ( .clk(nclk), .rst_n(nrst_n),
.d_plus(nd_plus),
      .d_minus(nd_minus), .transmit(ntransmit),
.write_enable(nwrite_enable), .write_data(nwrite_data),
      .fifo_empty(nfifo_empty), .fifo_full(nfifo_full) );

PADVDD U1 ( );
PADGND U2 ( );
PADOUT U3 (.DO(nd_minus), .YPAD(d_minus));
PADOUT U4 (.DO(nd_plus), .YPAD(d_plus));
PADOUT U5 (.DO(nfifo_empty), .YPAD(fifo_empty));
PADOUT U6 (.DO(nfifo_full), .YPAD(fifo_full));
PADINC U7 (.DI(nclk), .YPAD(clk));
PADINC U8 (.DI(nrst_n), .YPAD(rst_n));
PADINC U9 (.DI(ntransmit), .YPAD(transmit));
PADINC U10 (.DI(nwrite_data[0]), .YPAD(write_data[0]));
PADINC U11 (.DI(nwrite_data[1]), .YPAD(write_data[1]));
PADINC U12 (.DI(nwrite_data[2]), .YPAD(write_data[2]));
PADINC U13 (.DI(nwrite_data[3]), .YPAD(write_data[3]));
PADINC U14 (.DI(nwrite_data[4]), .YPAD(write_data[4]));
PADINC U15 (.DI(nwrite_data[5]), .YPAD(write_data[5]));
PADINC U16 (.DI(nwrite_data[6]), .YPAD(write_data[6]));
PADINC U17 (.DI(nwrite_data[7]), .YPAD(write_data[7]));
PADINC U18 (.DI(nwrite_enable), .YPAD(write_enable));

endmodule
```

You have some freedom on where you want to locate the I/O pads, but in a real world design, some considerations that must be made include signal delay and noise. You don't want to put sensitive I/O pads of your chip close to a noise source, such as power, ground, and rapidly/commonly changing signals like clocks or serial data. Also, you don't want your I/O pads to be far away from the chip block that is using/producing them because that adds delay to the signal. For your design however, it is recommended for you to group the related inputs/outputs together. Keep in mind that the number of pads you have on one side of the chip can alter your chip's aspect ratio (what is this? See next section). So if you want to create a square chip, distributing the number of pads evenly between sides really helps.

You will also need to make your chip layout such that it includes filler pads and corner pads. Without them it may result in power and ground not being distributed inside your chip and I/O pads. The I/O pads also need power because there are buffers inside the I/O pads.

To include the filler pads, you will need to declare "dummy" pads in your encounter.io file. The format for adding a dummy pad is:

```
Pad: <Name> <Direction> PADNC
```

For example:

```
Pad: U102 N PADNC
```

declares a dummy pad called U102 that is located on the northern border of your chip. ***Note: The pad name is arbitrary, but there better not be anything else called U102 in the mapped file (for example, grep for U102 in the mapped file if in doubt).***

The syntax for declaring a corner pad is:

```
Orient: R<degree>
Pad: <Name> <Corner direction> PADFC
```

For example:

```
Orient: R270
Pad: c02 NE PADFC
```

declares a corner pad called c02 in the North East corner of your chip with an orientation of 270 degrees. Each corner pad should be oriented so that the bottom right corner of it is the "inside corner", that is it is the corner closest to the chips core. Thus the North West corner pad should not be rotated (aka R0) and the South West and South East corner pads should be rotated 90 and 180 degrees respectively, and the North East corner pad should be as done in the example. *Note: the "Orient:" syntax is case-sensitive so make sure to always use 'R' instead of 'r'. Also, there cannot be spacing between the 'R' and the angle value.*

The entire chip must be "padded". This is what the filler pads are used for. MOSIS allows us a maximum of 40 I/O pads (filler pads are not included in this count). The USB Transmitter design uses significantly less than 40 I/O pads (18 to be precise), so filler pads will occupy the rest of the padding around the chip. Modify your encounter.io file so each side of your chip has 10 pads (dummy or not) and has all four corner pads (note: for a core bound design you might need to insert even more filler cells. For the final project, if you use a 3mm x 3mm area, you will need more than 10 I/O pads or filler pads on each side).

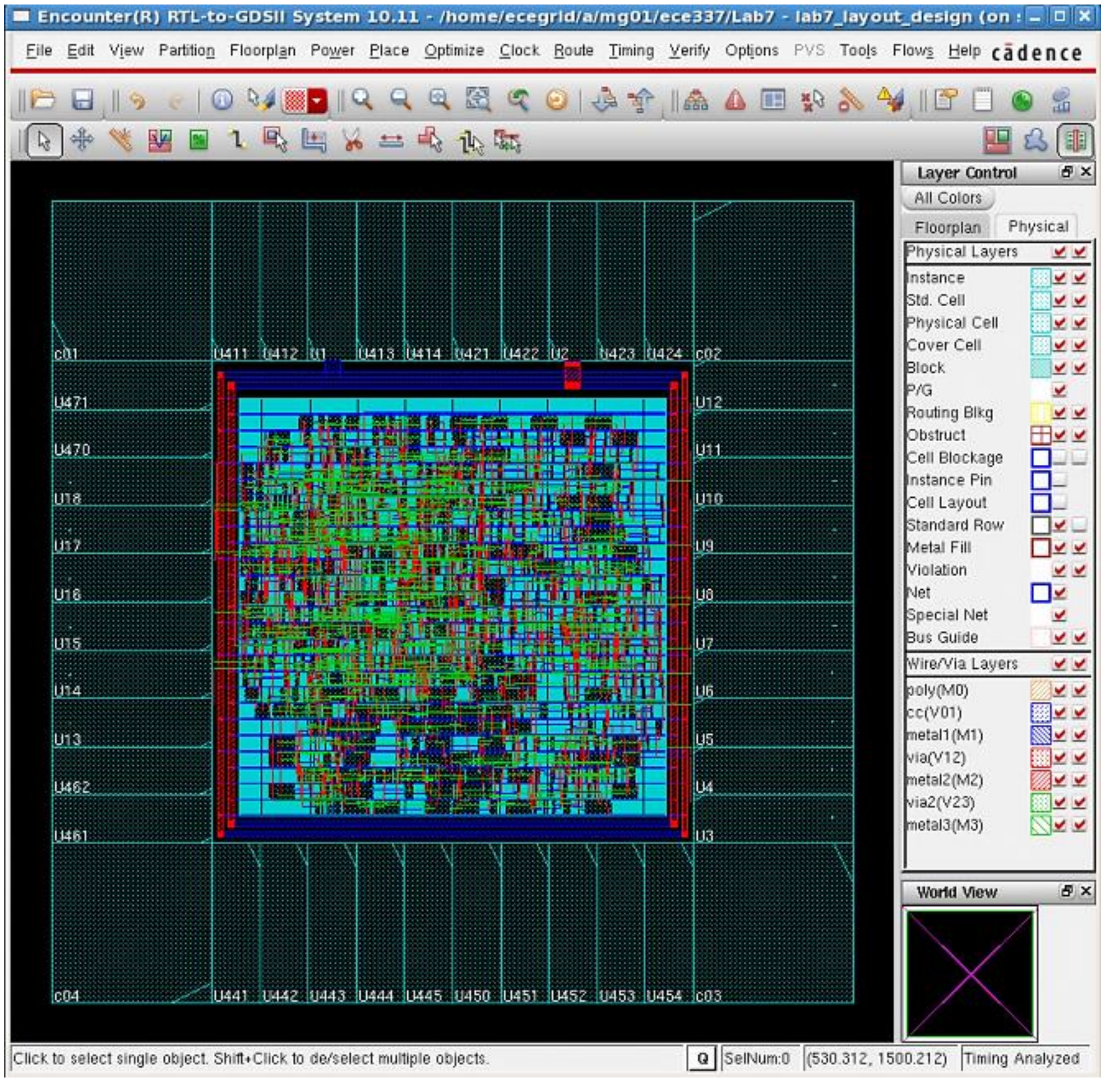
After you have done the needed modifications, issue the command:

```
encounter
```

in your terminal to invoke SOC Encounter. In your Encounter command line, issue the following command to run the .tcl script:

```
encounter 1> source encounter.tcl
```

A successful run should produce a layout that is displayed in the encounter user interface.



Encounter(R) RTL-to-GDSII System 10.11 - /home/ecegrid/a/mg01/ece337/Lab7 - lab7_layout_design (on :

Click to select single object. Shift+Click to de/select multiple objects. | SelNum:0 | (530.312, 1500.212) | Timing Analyzed

Figure 1: Layout in SOC Encounter

Try to zoom in and out if the user interface is not focused in your layout. Once you have the 'layout' view of your lab7_layout_design on screen, as in Figure 1, **have a TA check off your work up to this point.**

STOP

Now you need to run a connectivity check on your design. From the SOC Encounter User interface, select:

Verify > Verify Connectivity

This will create a new popup menu. In the Verify connectivity menu, **select "All" for Net Type and Nets**. For this lab, you are asked to do the **connectivity check on open connections, unconnected pin, connectivity loop, antenna and geometry loop. Note: you cannot perform geometry loop checks with any of these other options checked so you will need to do as separate run. Click OK**. Examine the report generated by the connectivity check and make sure there are no violations or error. If you encounter any violations or errors, you can fix them by modifying your layout generation floor plan parameters such as aspect ratio and row density and generating a new layout.

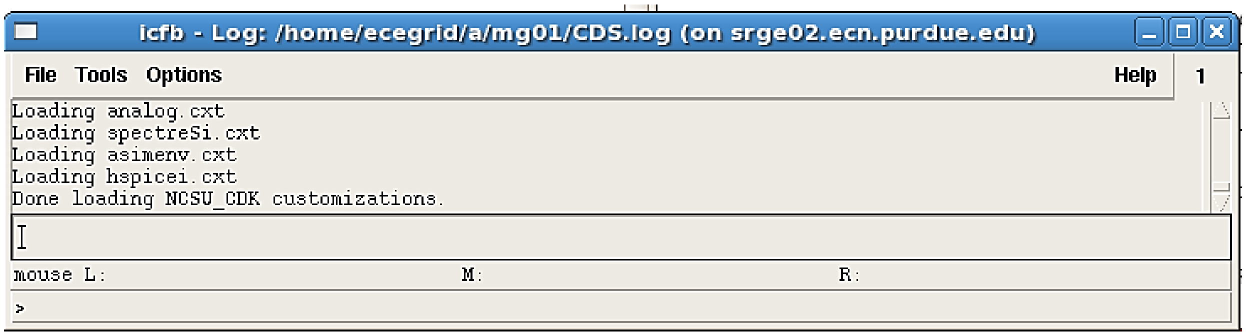
Once you have an errors and violations free connectivity check, have a TA check off your work up to this point.

5. Setting Up Virtuoso for Layout

The layout you are viewing in Encounter only shows the interconnects. In order to show the transistors in your layout, you need to use a new tool, Virtuoso. In order to begin using Composer, you will need to bring up Virtuoso's Command Interface Window (CIW). In order to invoke the CIW, type the following command at your UNIX prompt, **make sure that you are in your ~/ece337/Lab7 directory before issuing this command:**

```
grid icfb
```

The CIW will appear at the bottom of your screen, and will look like Figure 2 below.


icfb - Log: /home/ecegrid/a/mg01/CDS.log (on srge02.ecn.purdue.edu)
File Tools Options
Help
1
mouse L:
M:
R:
>

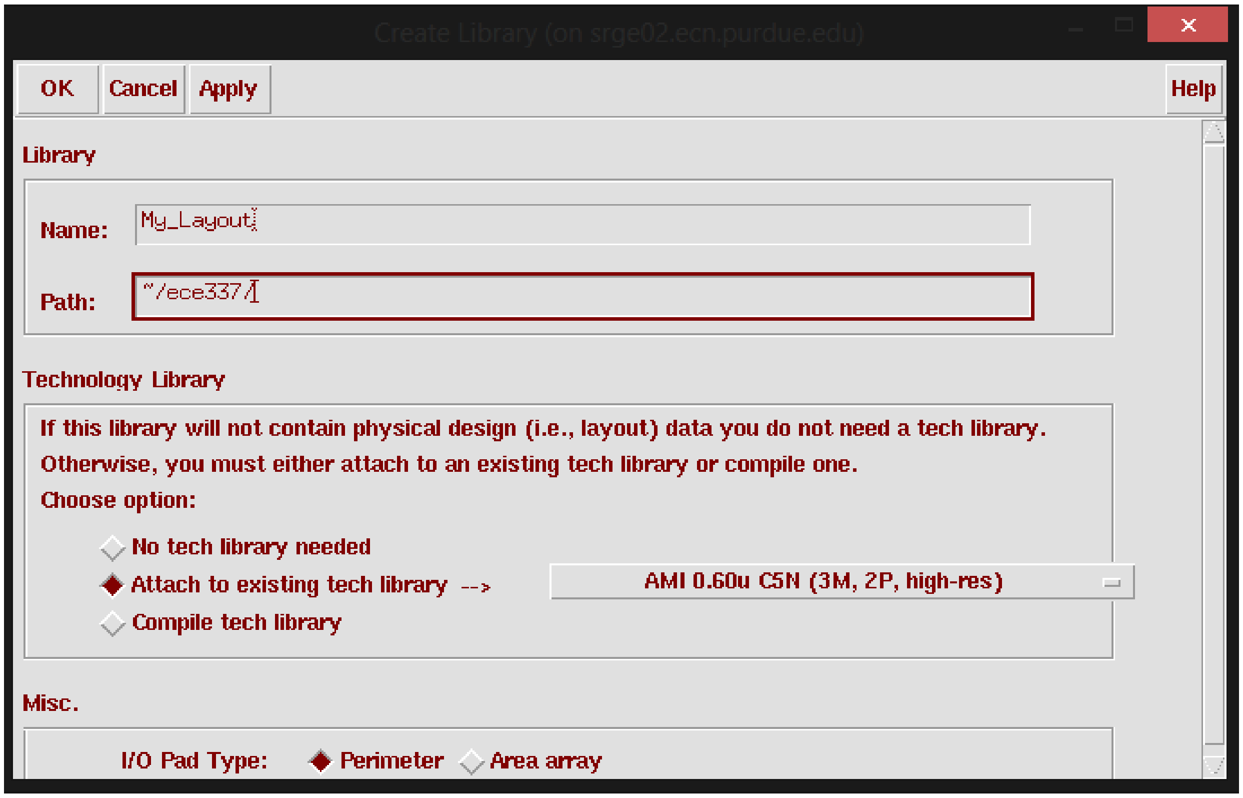
```
Loading analog.cxt
Loading spectreSi.cxt
Loading asimenv.cxt
Loading hspicei.cxt
Done loading NCSU_CDK customizations.
```

Figure 2: CIW Window

Now you need to create your library by selecting the following Menu option from the CIW window:

File > New > Library...

This will bring up a window that will look like Figure 3 below.



Library

Name: My_Layout

Path: ~/ece337/

Technology Library

If this library will not contain physical design (i.e., layout) data you do not need a tech library.
Otherwise, you must either attach to an existing tech library or compile one.
Choose option:

◇ No tech library needed
◆ Attach to existing tech library --> AMI 0.60u C5N (3M, 2P, high-res)
◇ Compile tech library

Misc.

I/O Pad Type: [✓ Perimeter] Area array

Figure 3: Filled in Create Library Window

Fill in the following values:

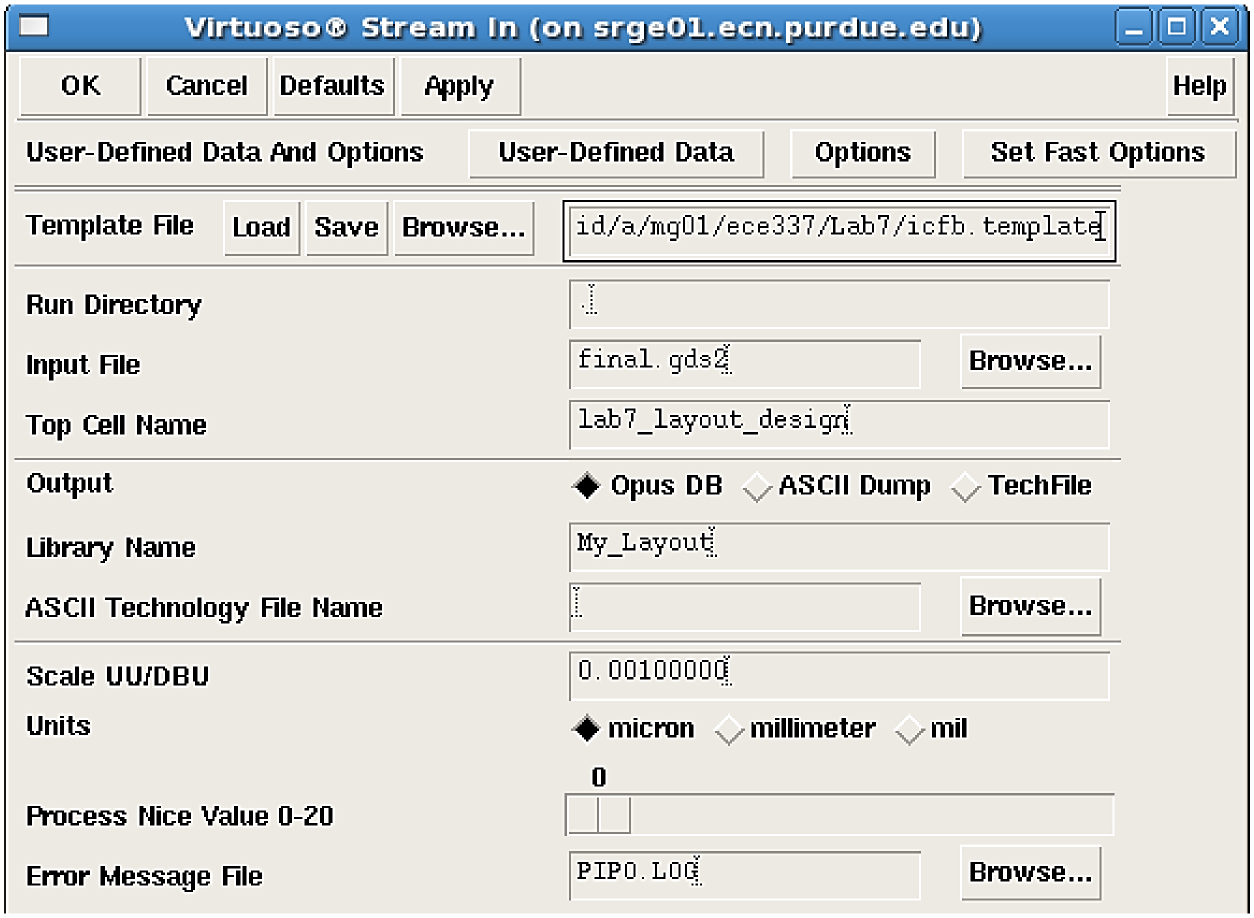
- Name: My_Layout
- Path: ~/ece337/
- Technology Library: Attach to existing tech library → AMI 0.60u C5N
- I/O Pad Type: Perimeter

Click OK.

Next you will need to import the layout stream file that was generated by SOC encounter to Cadence Virtuoso. To do this, from the CIW window menu, select:

File > Import > Stream...

This will bring up a window that will look like Figure 4 below.



Template File: id/a/mg01/ece337/Lab7/icfb.template
Run Directory: .
Input File: final.gds2
Top Cell Name: lab7_layout_design
Output: [✓ Opus DB] ASCII Dump TechFile
Library Name: My_Layout
ASCII Technology File Name:
Scale UU/DBU: 0.00100000
Units: [✓ micron] millimeter mil
Process Nice Value 0-20: 0
Error Message File: PIPO.LOG

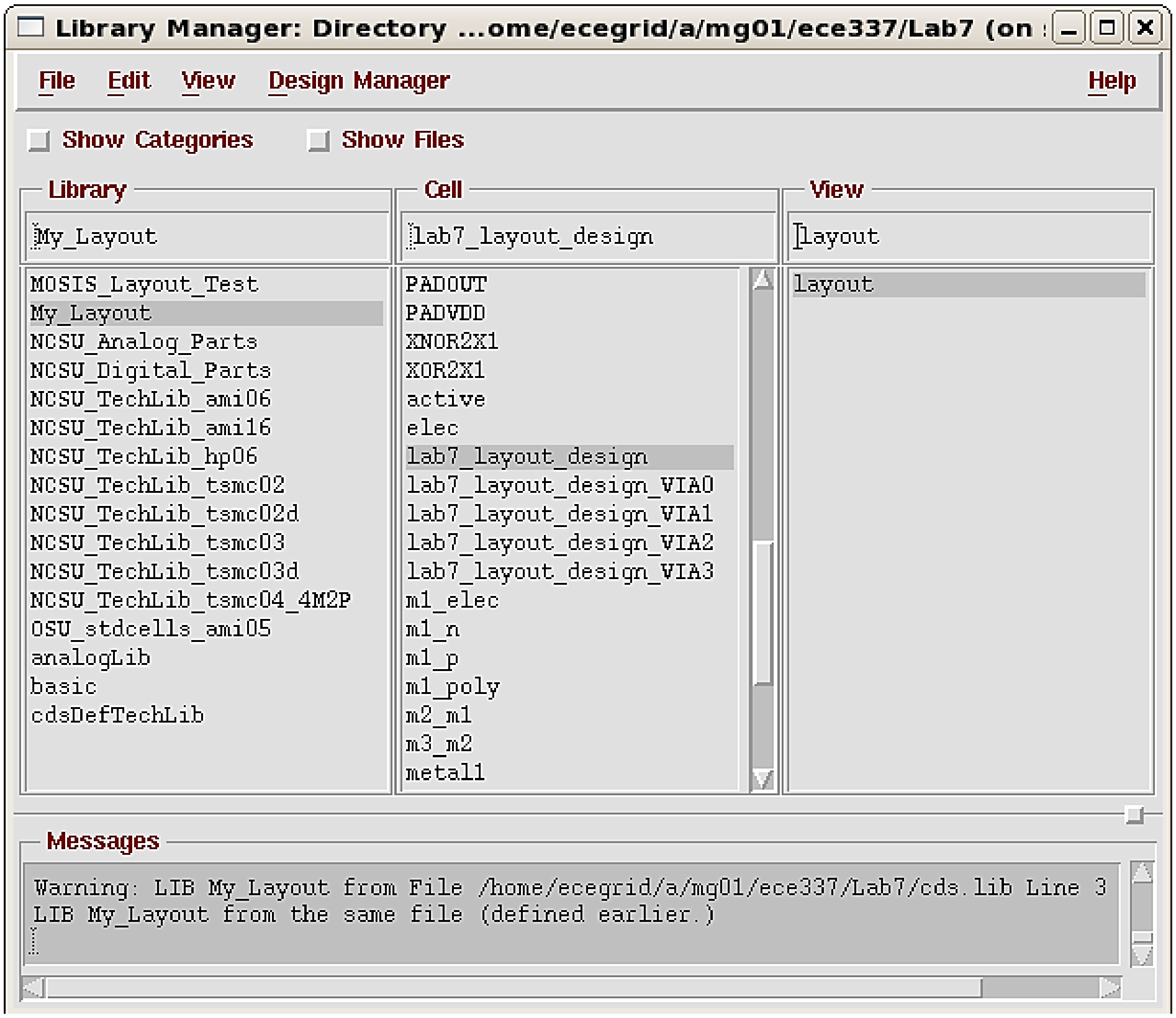
Figure 4: Stream In Window

A template for this window has been provided for you. Just browse and find the file icfb.template in your Lab7 directory and click Load. The values should be automatically filled in. **Click OK**. Once the process is done, a pop up box will notify you about how many errors or warning you may have. If you receive some errors, display the log file and seek what your error is. Notify your TA about your error. If you received warnings, then display the log file and make sure the warnings are non-destructive (warning about date format is not destructive, also ignore the warning about the techfile).

Once you have completed the layout and import process you can open up the layout. If the Library Manager window did not automatically popup or you closed it you may open it by selecting the following Menu option from the CIW window:

Tools > Library Manager...

This will bring up a window that will look like Figure 5 below.



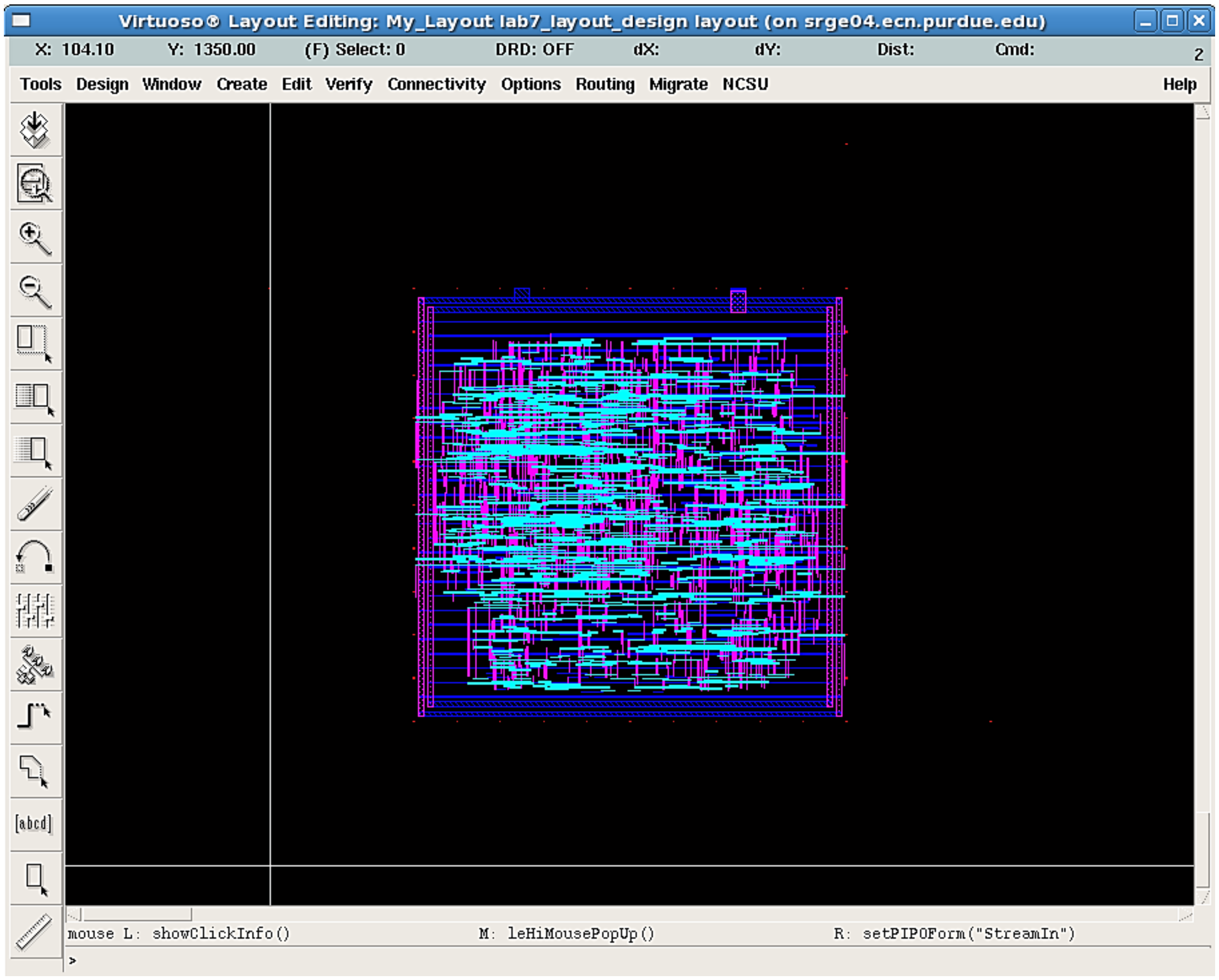
```
My_Layout              lab7_layout_design          layout

MOSIS_Layout_Test      PADOUT                      layout
My_Layout              PADVDD
NCSU_Analog_Parts      XNOR2X1
NCSU_Digital_Parts     XOR2X1
NCSU_TechLib_ami06     active
NCSU_TechLib_ami16     elec
NCSU_TechLib_hp06      lab7_layout_design
NCSU_TechLib_tsmc02    lab7_layout_design_VIA0
NCSU_TechLib_tsmc02d   lab7_layout_design_VIA1
NCSU_TechLib_tsmc03    lab7_layout_design_VIA2
NCSU_TechLib_tsmc03d   lab7_layout_design_VIA3
NCSU_TechLib_tsmc04_4M2P m1_elec
OSU_stdcells_ami05     m1_n
analogLib              m1_p
basic                  m1_poly
cdsDefTechLib          m2_m1
                       m3_m2
                       metal1
```

```
Warning: LIB My_Layout from File /home/ecegrid/a/mg01/ece337/Lab7/cds.lib Line 3
LIB My_Layout from the same file (defined earlier.)
```

Figure 5: Library Browser Window

You can now select the library in which the layout was placed. From here you should be able to select your lab7_layout_design design inside the My_Layout library and finally double-click on the “layout”. This should open up your layout which should be similar to Figure 6 on the next page.



```
mouse L: showClickInfo()          M: leHiMousePopUp()          R: setPIPOForm("StreamIn")
>
```

Figure 6: USB Transmitter Layout

Now, you will need to swap the “blank” cellview in your current layout with the real standard cell layout. First, you need to change your display option so the layout editor will display all the layers. To do this, in your Virtuoso Layout Editor window click:

Options > Display...

or you may use the hotkey “e”. Now it will open a new window for display options. Change the display levels stop to 20. You may save this new setting by clicking the “Save To” button in the bottom of the window to save this to your .cdsenv file. Click OK.

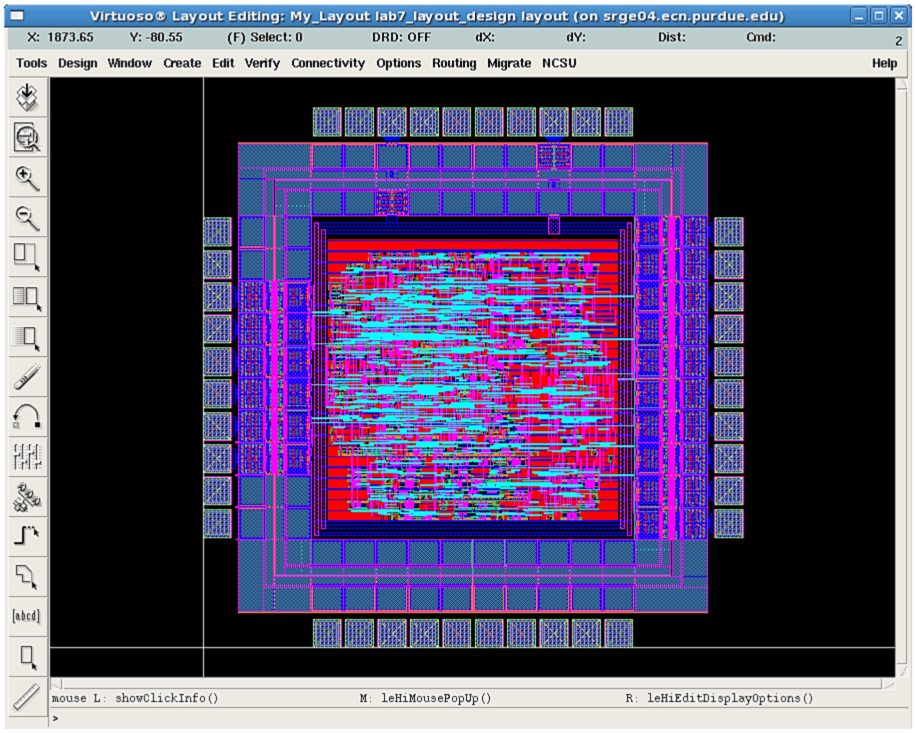
Now, you need to swap the cellviews. Again, in the Virtuoso Layout Editor select:

Edit > Search...

or you may use the hotkey "S". This will bring up the search window. **You want to search for "inst" in "current cellView"**. Now you want to **add a several criteria** to your search **by clicking the "Add Criteria" button eight times**. The **criteria are:**

- **lib name = = My_Layout**
- **view name = = layout**
- **cell name != M3_M2**
- **cell name != M2_M1**
- **cell name != lab7_layout_design_VIA0**
- **cell name != lab7_layout_design_VIA1**
- **cell name != lab7_layout_design_VIA2**
- **cell name != lab7_layout_design_VIA3**

Now at the bottom of the search window, there is a replace drop box. You want to replace your blank cellview with the standard cellview from the OSU_stdcells_ami05 library. To do this, **select "lib name" from the replace drop box and type OSU_stdcells_ami05 in the box next to it. Click Apply.** You might have to wait while Virtuoso is searching for the cellviews. When the "Current Figure" and "Figure Count" numbers in your search window changes from 0, then you know that it is done searching. The next step is to replace the cellviews by clicking the "Replace All" button. You should be able to see your complete chip, including the I/O pads, in your layout editor, similar to Figure 7.


Virtuoso® Layout Editing: My_Layout lab7_layout_design layout (on srge04.ecn.purdue.edu)

X: 1873.65 Y: -80.55 (F) Select: 0 DRD: OFF dX: dY: Dist: Cmd: 2

mouse L: showClickInfo() M: leHiMousePopUp() R: leHiEditDisplayOptions()

Figure 7: Complete USB Transmitter Layout

Notice that in the process of opening the 'layout' view of the "lab7_layout_design", a new window, titled "LSW" also opened. "LSW" stands for Layer Selection Window. Examine this window and you will see that it is partially a legend of the different layers in the layout of the USB Transmitter. Essentially, it is showing you what layer each color or colored pattern in the layout view corresponds to. Utilizing the "LSW" answer the following question:

What colors in the Layout correspond to Metal1 and Poly1?

Now, examine the window that contains the 'layout' view for the USB Transmitter. Are you able to distinguish any NAND or NOR gates? How about D Flip-Flops? In fact, when examining layouts of designs even as small as ~200 gates, it is difficult to distinguish individual transistors. Examine the layout more closely by zooming into any region of the layout. **In order to zoom in, select the following option:**

Window > Zoom > In

Next, **Click the LMB on a location in the layout, then move the mouse to create a rectangle, when you feel like you will zoom in on an area of interest click the LMB again.** Notice how you have zoomed in on the design. Can you identify an individual transistor? Zoom in until you can identify an individual transistor. If you are having trouble identifying a transistor, please ask a TA for assistance.

Now, zoom back out until you can see the entire design. **In order to see the entire design again, select the following option:**

Window > Fit All

Now you need to save your layout by selecting in the layout editor:

Design > Save (hotkey is f2)

Now, move the mouse cursor around and look at the upper left corner of the window. Notice how the values next to the X and the Y are changing as you move the mouse across the design. These are the coordinates at which the mouse is currently located. The origin of these coordinates is located at the intersection of the two bright white lines that are in the layout view.

Now determine the X and Y coordinates of the lower left corner of the full design <u>including the bonding pads</u> to 2 decimal places. Record this value on your Check-off Sheet. Now do the exact same procedure to determine the X and Y coordinates of the upper right corner. In addition, it should be noted that the units of the X and Y coordinates are micrometers (um).

Once you have the coordinates for both the lower left and the upper right corner of the box and have calculated the area, have a TA check your values and sign-off your check-off sheet.

6. Alter the Aspect Ratio of the Layout

In this exercise you will alter the Aspect Ratio of the USB Transmitter to see what impact it has on your layout. **Modify your encounter.tcl so that SOC Encounter will produce a layout with aspect ratio of 2**. To find out how, open the SOC Encounter help file from the Encounter User Interface by clicking `Help`. This will bring up the user manual for SOC Encounter in your web browser. Basically you will need to supply the right argument to the floorplan command in your script. Therefore, in the manual, look at how the argument of floorplan is supposed to be supplied. A quick reference for this can be found by clicking on the “Related Documents” link on the Encounter main help page and clicking again on the “Encounter Text Command Reference” link. Look under “Floorplan Commands” for the command “floorplan”. From here you can figure out what parameters you need to change for the floorplan command in your .tcl script. (Hint: the current aspect ratio of your chip is 1.0)

After you are done modifying your .tcl script, run the script to generate a new layout. Once a new layout has been produced, **examine it and have a TA check off your work up to this point.**

How does altering the Aspect Ratio of a design affect the design’s Layout? In essence, what mathematical formula could you derive for the Aspect Ratio?

7. Row Density/Utilization Investigation

The final layout parameter that is going to be investigated is that of Row Density/Utilization. The first step in our investigation of the Row Density parameter is to **produce a layout of the design that has a Row Density equal to 40% and aspect ratio of 1.** (hint: examine the syntax of the floorplan command in encounter)

Follow the same procedure that you did in previous sections to bring up your modified USB Layout. Once you have the 'layout' view of the USB Transmitter on screen, **have a TA check off your work up to this point.**

Obtain the coordinates of the LOWER LEFT and UPPER RIGHT corners of the DASHED PURPLE BOX that encompasses the layout of the USB Transmitter. (You can obtain the coordinates of the corners from Encounter. No need to import it to icfb)

Once you have the coordinates for BOTH the LOWER LEFT and the UPPER RIGHT corner of the box, **have a TA check your values and sign-off your check-off sheet.**

What is your design's area for this layout?

How does this new area compare with the area that was previously calculated for the layout utilizing only default values? Does this make sense given the two values for Row Utilization?

Do you think you should be able to generate a layout that has a Row utilization of 100%? If so, do it and briefly describe how the layout looks, its similarities, and its differences to the previous layouts generated. Also, does it have many design rule violations (marked with an "X" in your layout and can you see how many from the log in the command line)? If you think you should not be able to generate a layout, briefly explain why?

8. Metal Fill

When manufacturing a semiconductor, extra steps need to be taken to ensure a good yield. One such step is to add extra dummy metal to ensure that there is an even distribution of metal across the chip. A uniform metal density helps maintain planarity of the wafer, which aides in processing. Typically, a process will require that the average metal density of the chip be within a certain range. You generated your first layouts without a metal fill because the extra metal layers make it more difficult to identify components of the design.

Find and uncomment the line in the .tcl script that adds a metal fill to your design. Once you have generated a layout in Encounter, select:

Verify > Metal Density ...

Select OK and examine the report file generated to ensure that there are no maximum density violations. Normally one would need to also work to make sure there are now minimum density violations if they were actually fabricate the chip but for this lab we won't require that. **Have a TA check off your work up to this point.**

9. Timing Analysis

To run the timing analysis, you will need to specify the timing constraints. SOC Encounter takes timing constraints in the format of Synopsys PrimeTime. Luckily a template has been provided for you in the file encounter.pt in your Lab 7 directory.

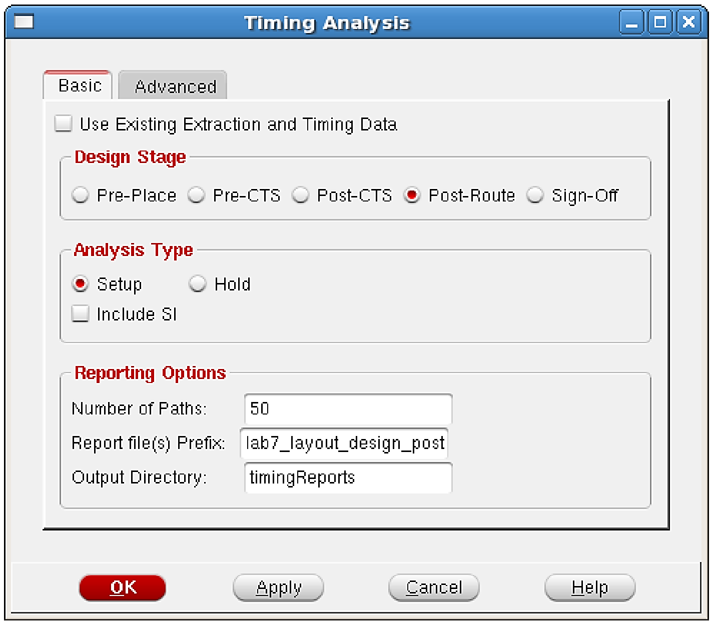
Examine the format. You may open the SOC Encounter help file if you need more information on the format of the file. The pin names in the file should already match your design's pin names. From the Encounter user interface, select:

Timing > Load Timing Constraint

Specify "encounter.pt" as the timing constraint file and click OK. Now select:

Timing > Report Timing

This will bring up another window as shown in Figure 8. Set the settings appropriately for your design and click OK. You may want to examine the Slack report file (.slk extension) and the Detailed Violation File (.tarpt extension) to see if your design violates the timing constraints.



[ ] Use Existing Extraction and Timing Data

Design Stage: Pre-Place, Pre-CTS, Post-CTS, [✓ Post-Route], Sign-Off

Analysis Type: [✓ Setup], Hold
[ ] Include SI

Reporting Options

Number of Paths:	50
Report file(s) Prefix:	lab7_layout_design_post
Output Directory:	timingReports

Figure 8: Timing Analysis

You can view each timing analysis path in the **lab7_layout_design_postRoute_all.tarpt** file in your **timingReports folder.**

Write down in your check off sheet the starting and ending components of your critical path and the number of combinational logic blocks in between.

What block does your critical path start from? What block does it end at?

Also, open your Synopsys synthesis report file and compare the critical path listed there with the critical path from your layout. Are they the same path? Do they have the same delay?

It is possible for you to modify the synthesis script so it will show more paths and a few more options in the report. To do this, you have to find out the options you want by opening the manual for "report_timing" command in dc_shell-t and then modifying your .fcr file accordingly.

Once you written down the answers to the questions above, have a TA check off your work up to this point.

Now, we want to investigate the maximum clock speed of the USB Transmitter design. Most likely with the default timing constraint file (with a clock period of 100 ns), you will have enough slack in your critical path.

We want you to modify the timing constraint file so that the clock period is reduced enough to produce a negative slack for your critical path.

To do this, change the clock period in the timing constraint file. The command in your timing constraint file should be self-explanatory. One thing you need to know is that you will need to modify your clock waveform so it will create a clock with a 50% duty cycle. You will need to change the rising and falling time in the waveform tuple.

Reload the timing constraint file and run the timing analysis again and write down what clock period you have determined in your checkoff sheet.

Note: Once you have generated a negative slack for your critical path you will be able to view the timing analysis for each of the paths with a graphical viewer in encounter by:

Timing > Timing Debug

Click OK

This will allow you to select a path by double clicking on the path and the selected path will be highlighted in your layout. It will also create a window focused on the timing analysis of that path, which has a useful feature that shows the path in schematic mode if you select the Schematic tab.

Have a TA check off your work up to this point.