ASIC Design Laboratory
Lab 10 : IC Layout Tutorial and Area Tuning
Lab Manual

Spring 2023

The purpose of this lab is to provide you 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 Design Compiler®, 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.

In this lab, you will perform the following tasks:

- Generate Several cell-level Layouts (Physical Designs) of a provided design using Innovus.
- Generate a detailed transistor-level layout of the provided design using Virtuoso.

1 Lab Setup

In a UNIX terminal window, issue the following commands, to setup your Lab 10 workspace:

mkdir –p ~/ece337/Lab10
cd ~/ece337/Lab10
dirset
setup10

The setup10 command is an alias to a script file that will check your Lab 10 directory structure and give you file needed for starting the lab. If you have trouble with this step please ask for assistance from your TA.

IMPORTANT: Make sure to add this new workspace into your 337 Repository, like you did in Lab1.

This way, you will always have the original copy in storage.

2 Generating a Full Layout

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 insulated from each other by materials like 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.

2.1 Adding I/O pads

For this portion of the lab assignment, you must add pads to your synthesized design. Synthesize the design provided to you via the following command in your Lab10 directory:

make mapped_layout_lab_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 Lab10 directory:

pads <design_name>

After the pads script has finished the mapped file 'layout_lab_design.v' in your 'mapped' folder should have the pad-frame wrapper module definition described in Source Code 1 at the end of it.

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

```
module layout_lab_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;
    layout_lab_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) );

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

endmodule
```

Source Code 1: Pad-Frame Wrapper Module Definition for Layout Lab's Design

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 Lab10 directory. One of the files copied was the full_run.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 init.tcl that is utilized by the script to setup Innovus to handle place and routing of the design.

As mentioned before, the layout you are going to generate is going to include I/O pads. Inside the 'init.tcl' script you should have noticed that a 'innovus.io' file was configured for the layout design. This file contains the pad-frame definition that will used to generate the layout's actual pad-frame. It must contain all active pads as well as filler cells that are used to bridge between active cells for both padding and continuation of strong power and ground rings that connect to all of the active pads. Active pads must have labels/names that match their respective pad instance in the mapped file's pad-frame wrapper module. Filler pads must have labels/names that do not match any component instance within the mapped file. Now, examine the 'innovus.io' file that was copied by the setup script, it should look similar to the following text.

```
Orient: R0
Pad: U5 N
Orient: R0
Pad: P0 N PADVDD
Orient: R0
Pad: G0 N PADGND
Orient: R0
Pad: U6 N

Orient: R90
Pad: U1 W
Orient: R90
Pad: P1 W PADVDD
Orient: R90
Pad: G1 W PADGND
Orient: R90
Pad: U2 W

Orient: R180
Pad: U8 S
Orient: R180
Pad: U9 S
Orient: R180
Pad: U10 S
Orient: R180
Pad: U11 S
Orient: R180
Pad: P2 S PADVDD
Orient: R180
Pad: G2 S PADGND
Orient: R180
Pad: U12 S
Orient: R180
Pad: U13 S
Orient: R180
Pad: U14 S
Orient: R180
Pad: U15 S

Orient: R270
Pad: U3 E
Orient: R270
Pad: U4 E
Orient: R270
Pad: P3 E PADVDD
Orient: R270
Pad: G3 E PADGND
Orient: R270
Pad: U7 E
Orient: R270
Pad: U16 E
```

The format of the pad declaration for an signal/port pad in the .io file is:

`Pad: <Pad Name> <Direction>`

So for example, Pad: U2 W means that pad U2 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.).

Power and Ground pads don't have corresponding ports on the module and thus have a different syntax as follows:

`Pad: <Pad Name> <Direction> <PADGND for Ground or PADVDD for Power>`

The prior 'innovus.io' file defines pad locations for this labs layout design's ports as defined in Source Code 1, so you shouldn't need too modify those, but will need to add filler cells to finish out the pad-frame as discussed below.

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 Section 3.1). So if you want to create a square chip, distributing the number of pads evenly between sides really helps. Additionally, you should have noticed that there are four pairs of power(PADVDD) and ground(PADGND) in the .io file, with on pair roughly centered on each side of the chip. This may seem extraneous compared to only having one pair, but digital logic and I/O buffers tend to cause significant current spikes on the power and ground lines. To handle this behavior, it is common practice to create multiple power and ground taps to an IC in order to both help balance the power drawn through supply pins to the chip as well as help minimize voltage drops due to surge currents from activity by not aggregating them through one pair of pins. 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 'innovus.io' file. The format for adding a dummy pad is:

`Pad: <Name> <Direction> PADNC`

For example:

`Pad: FN1 N PADNC` This declares a dummy pad called FN1 that is located on the northern border of your chip.

NOTE: The pad name is arbitrary, but there better not be anything else called FN1 in the mapped file (for example, grep for FN1 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` This 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". For the sake of practice, you are being required to insert enough filler cells to produce a pad frame with 10 pads on each side. The USB Transmitter design uses significantly less than 40 I/O pads (24 to be precise), so filler pads will occupy the rest of the padding around the chip. Modify your 'innovus.io' file so each side of your chip has 10 pads (dummy or not) and has all four corner pads.

2.2 Generating a Cell-based Layout of the USB Transmitter

Once you have a complete pad-frame description in the 'innovus.io' file, then you are ready to proceed with running the place and route script for the layout. Issue the following command in your terminal to invoke Innovus.

innovus

In your Innovus command line, issue the following command to run the .tcl script:

source full_run.tcl

A successful run should produce a layout that is displayed in the Innovus user interface and should look similar to Figure 1.

Try to zoom in and out if the user interface is not focused in your layout. Once you have the 'layout' view of your layout_lab_designon screen, as in Figure 1, have a TA check off your work up to this point. Now you need to run a connectivity check on your design. From the Innovus User interface, select:

Verify $\rightarrow$ 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.

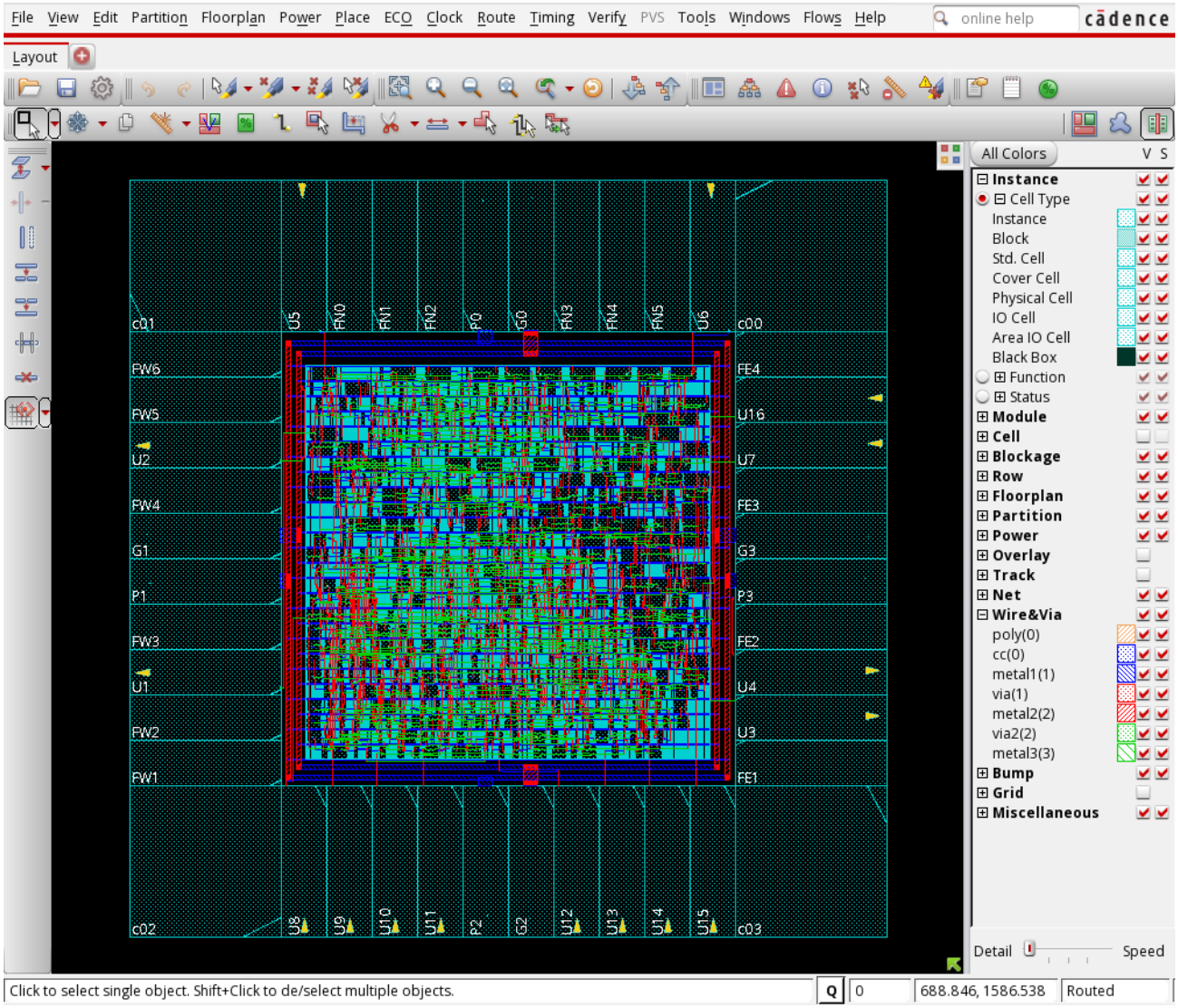

Figure 1: Layout in Innovus with Complete Pad-Frame

2.3 Generating a Transistor-Level Layout of the USB Transmitter

The layout you are viewing in Innovus 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/Lab10 directory before issuing this command:

virtuoso

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

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.

Fill in the following values:

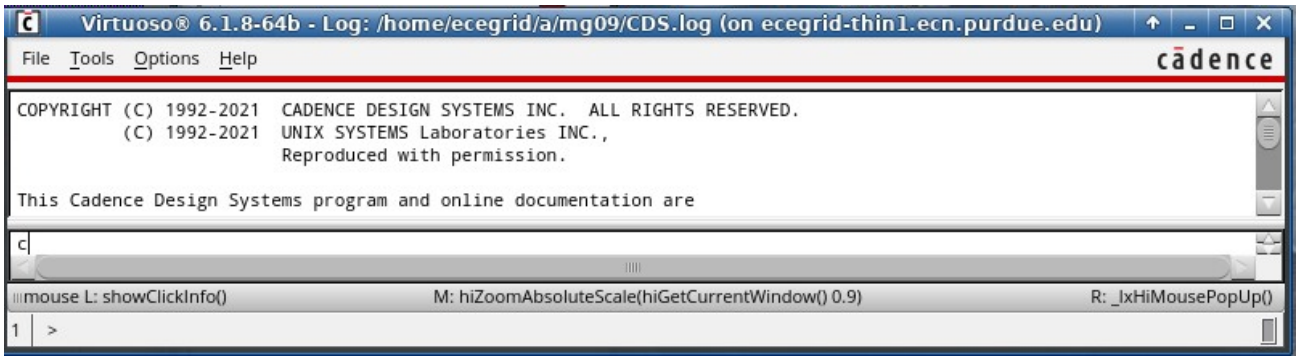

Figure 2: CIW Window

Name My_Layout

Path ~ /ece337/

Technology Library Attach to existing tech library → AMI 0.60u C5N

I/O Pad Type Perimeter

Click OK

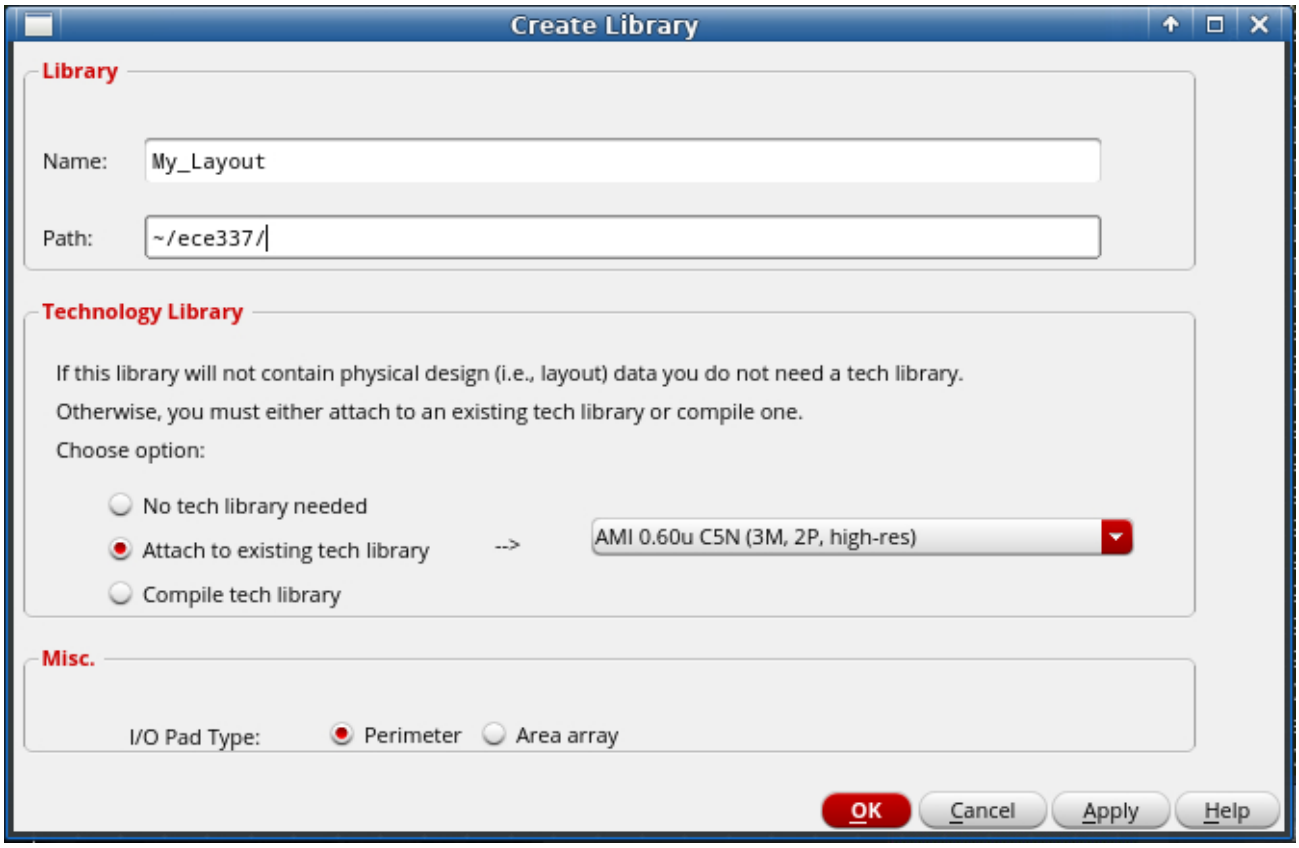

Figure 3: Filled in Create Library Window

Next you will need to import the layout stream file that was generated by Innovus 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.

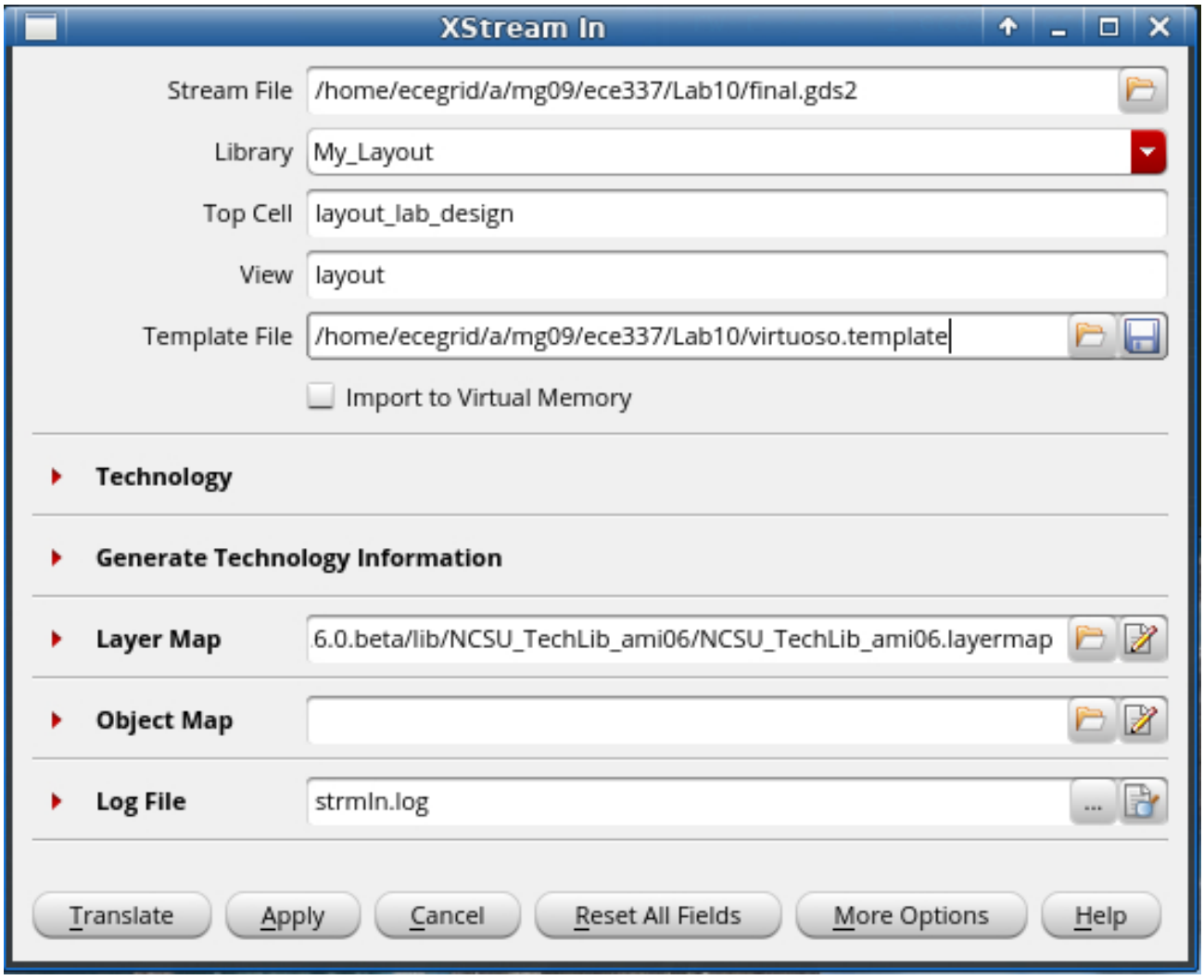

Figure 4: Stream In Window

A template for this window has been provided for you. Browse and find the file virtuoso.template in your Lab10 directory and click Open. The other values should be automatically filled in.

Click Apply.

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.

NOTE: Warning about date format is not destructive, also ignore the warning about the techfile.

Now browse for a different Stream File to find osu05_stdcells.gds in your project directory. Make sure that "Top Cell" is empty. Then click Apply This should cause the standard cells to be loaded into your library.

Repeat the Stream In process yet again time, but this time browse to find osu05_pads.gds in your project directory. Make sure that "Top Cell" is empty. Then click Translate. This should cause the IO pad cells to be loaded into your library.

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.

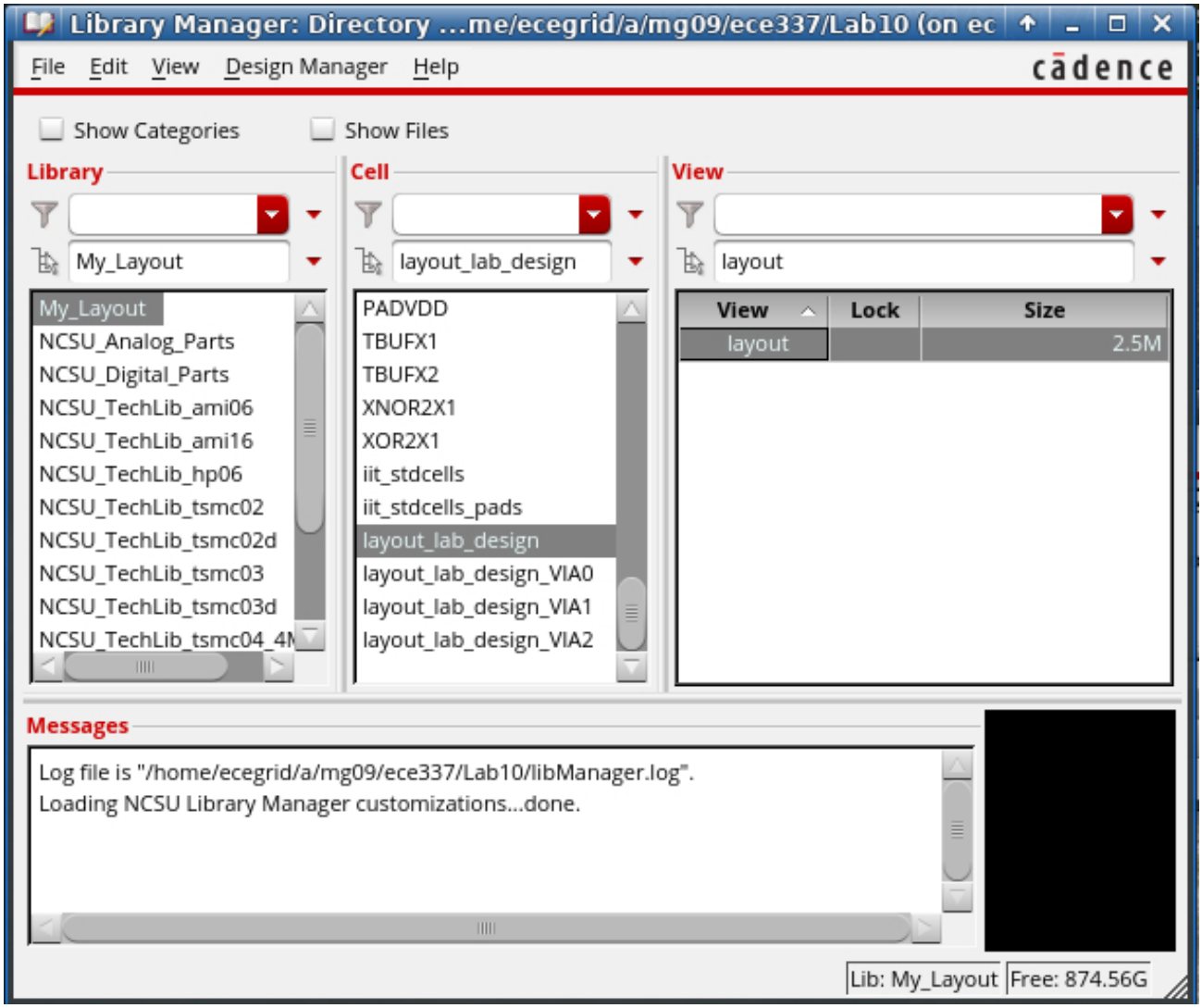

Figure 5: Library Browswer Window

You can now select the library in which the layout was placed. From here you should be able to select your 'layout_lab_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. It is possible that the red outlines won't be displayed, but as long as Figure 7 is correct, then there is no problem.

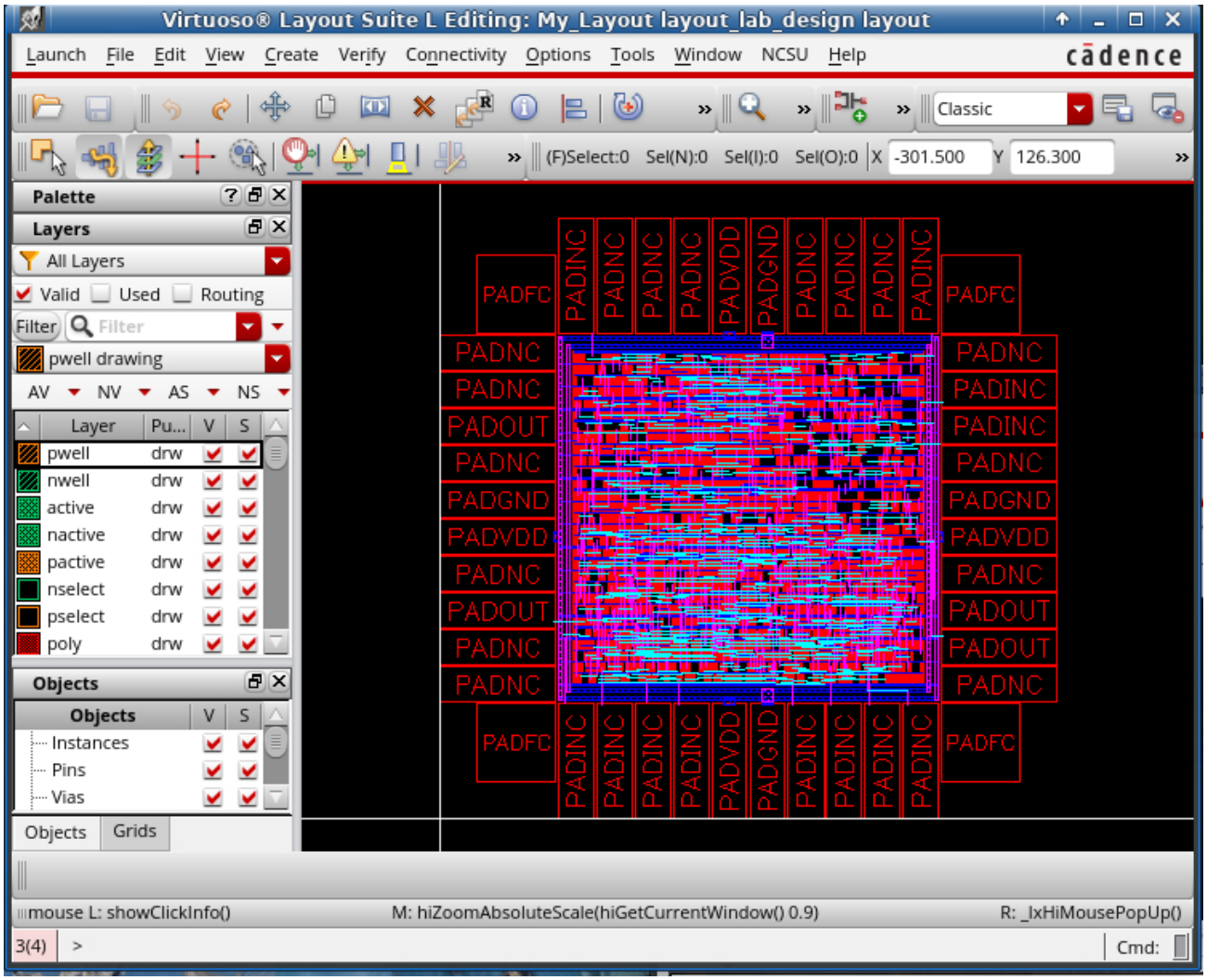

Figure 6: Initial Virtuoso Layout of USB Transmitter Design

Right now the layout only displays only the wiring of your design along with outlines of the standard cells and I/O pads. To see what is inside of each cell, 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 You should fnow be able to see your complete chip, including the I/O pads, in your layout editor, similar to Figure 7.

Notice that in the process of opening the 'layout' view of the "layout_lab_design", a new panel, titled "Layers" also opened. This is the layer selection panel. Examine this panel 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 layout panel answer the following question:

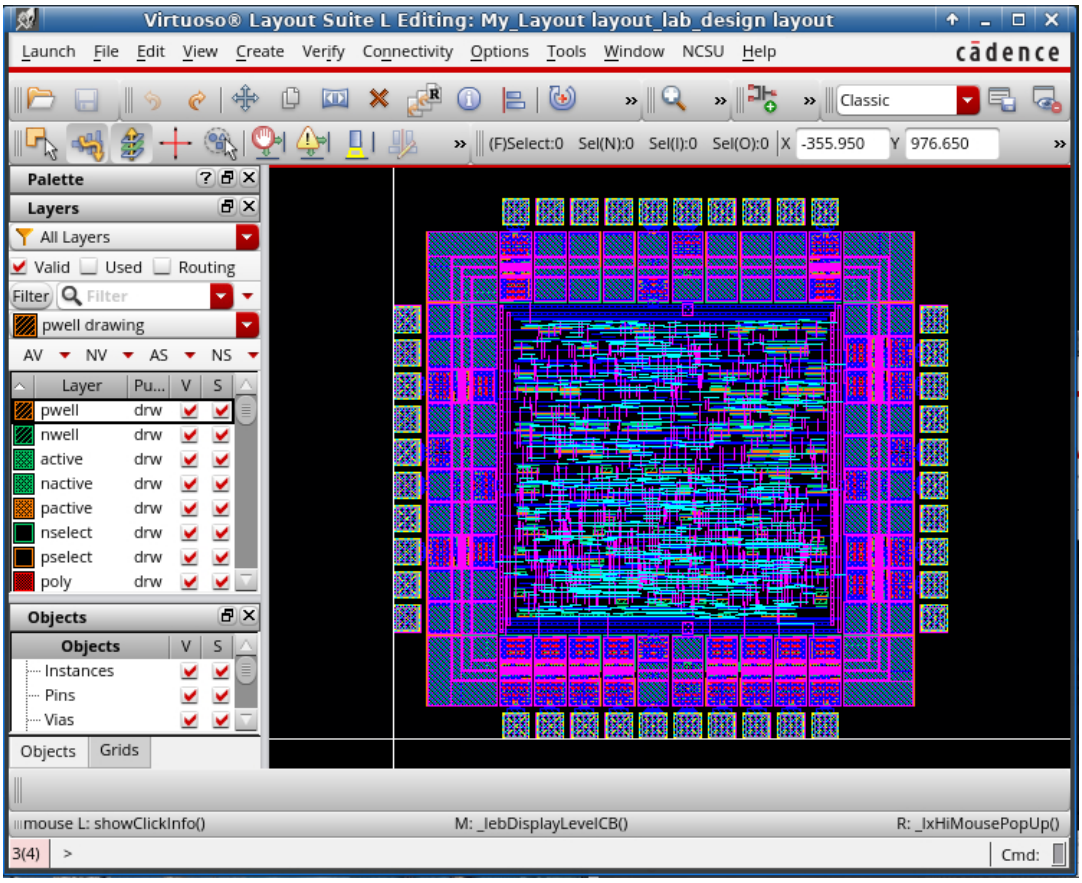

Figure 7: Complete USB Transmitter Layout in Virtuoso

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:

View → 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:

View → > Zoom to Fit All

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

File → Save

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 including the bonding pads 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.

3 Adjusting the Area and Shape of a Design Layout

3.1 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 'full_run.tcl' so that Innovus will produce a layout with aspect ratio of 2. To find out how, open the Innovus help file from the Innovus User Interface by clicking Help. This will bring up the user manual for Innovus 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 Innovus main help page and clicking again on the "Innovus 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.

NOTE: 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 with Innovus (no need to rerun through with Virtuoso), examine it and have a TA check off your work up to this point.**

Question: 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?

3.2 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.

NOTE: Examine the syntax of the floorplan command in Innovus 's help/user guide

Follow the same procedure that you did in previous sections to bring up your modified USB Layout. **Once a new layout has been produced with Innovus (no need to rerun through with Virtuoso), 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 teal box that encompasses the layout of the USB Transmitter.

Question: What is your design's area for this layout?

Question: 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?

Question: Do you think you should be able to generate a layout that achieves a Row utilization of 100%?

Layout the design with a row utilization target of 100% and briefly describe how the layout looks, its similarities, and its differences to the previous layouts generated.

Once you have the coordinates, calculated area, and have answered the above questions, have a TA check your values and sign-off your check-off sheet.

4 Conclusions

4.1 Adding Metal Fill to Layouts

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 Innovus, select:

Verify $\rightarrow$ Metal Density ...

Select OK and examine the report file generated to see if there are any maximum or minimum density violations. Normally one would need to eliminate density violations in order to ensure successful fabrication of your chip. **Have a TA check off your work up to this point.**

4.2 Follow-up Question

Question: 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.)

5 Closing Remarks

- Turn in your check-off sheet at the beginning of Lab Lab 11 .