FINAL PROJECT

Layout & Verification

DECEMBER 1, 2024

MARK A. IVEY
UTD EECT 6325, FALL 2024

Introduction

For EECT 6325, I needed to take my digital design from concept to layout, and using the provided tools, show that the design worked. I was mostly successful with my creation; while it did meet and exceed the project requirements, the design provided does not meet timing requirements for it to operate at the 200 MHz I was hoping for, and I will cover this later in the appropriate section.

To briefly summarize, these are the requirements I had for my project this semester:

1) HDL code complex enough to generate at least 3000 cells in Design Vision
2) The code must use at least 1 D flip flop
3) Create layouts of at least 8 cells, of which NOR2, NAND2 and INVERTER were mandatory. These cells, along with my D flip flop, will be what I use to generate the liberty file.
4) The cells must have a gate width of 62 nm.
5) The height of each PMOS diffusion layer must accommodate 4 contacts, and for NMOS diffusion, it must accommodate 3 contacts.
6) Contact spacing within each cell must follow this equation, center to center:
    a. $D = 260x\ nm$, where x is a whole number
7) Contacts from the center to the edge of the cell must follow this formula:
    a. $D = 130\ nm + 260x\ nm$, where x is a whole number
8) Check each layout for DRC errors and ensure the layout matches the schematic using LVS
9) Using parasitic extraction, characterize each layout using Hspice to ensure proper functionality.

For the rest of this paper, I will break down each major step for the final project and highlight issues I ran into and how I addressed them.

Brief Overview of The Hub75 Controller

Large outdoor full color signage installations are often comprised of LED panels and operate from a VGA like standard called hub75. Each panel will have up to a 64 x 64-pixel array made up of RGB LED diodes. The outputs are the same pins from the Hub75 standard, while the inputs are basically a 24-bit bus, as shown in Figure 1.

As it stands, if we just set each pixel on or off, then we would only be able to produce 8 colors, black, white, red, green, blue, purple, orange, or teal. To get more colors, we need to add some form of PWM. The easiest way to implement this would be to use a binary coded modulation, or BCM. Each color is represented by 3*n bits of data, with each bit within n controlling how long the RGB LEDs are on or off.

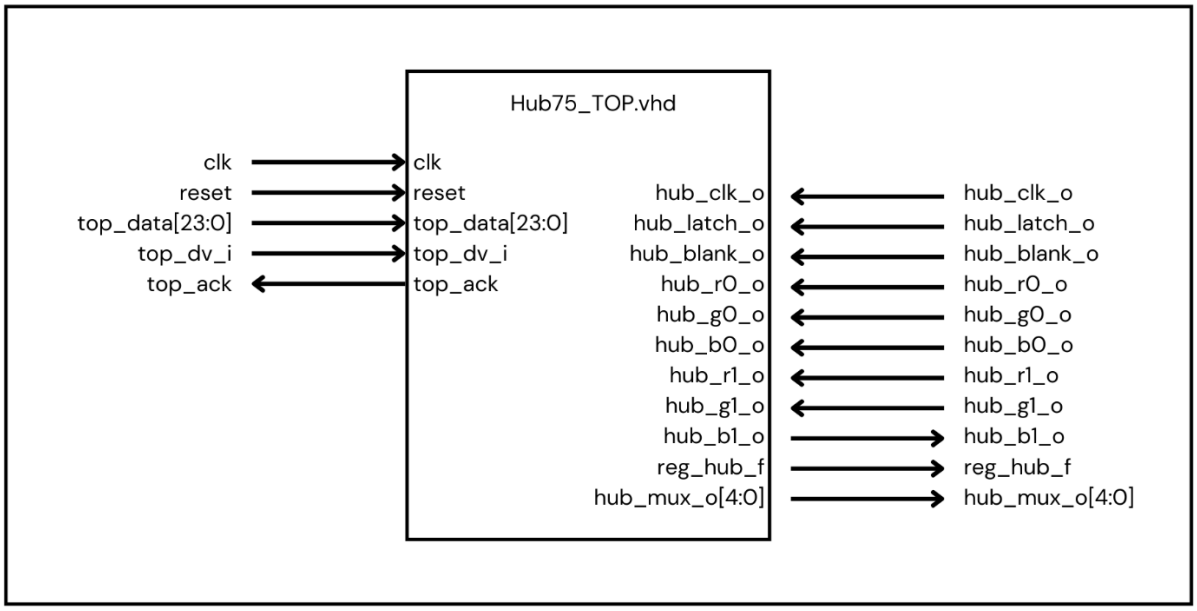

Figure 1: Block Diagram of the Real Deal

HDL Code & Simulation

For the first two projects, I needed to create my HDL code and then ensure that it was complex enough to generate at least 3000 cells. Then in project six, I used the cells I created to generate a new netlist for my IC layout. Comparing the last lines of project 2 to what I generated:

Last Lines of the Cell Report:

```
registers/s_mem_sipo_r0_reg[7]
                                                dff             library         7.000000  n
--------------------------------------------------------------------------------
Total 4321 cells                                          11668.000000
```

Mark Ivey EECT 6325 Project 2

Figure 2: Original Design Vision Netlist and Area from Provided Liberty File in Project 2

```
1   registers/s_mem_sipo_r0_reg[7]
2                                   DFF             mark_gf65       28.160000 n
3   --------------------------------------------------------------------------------
4   Total 5647 cells                                                52484.187705
5
```

Figure 3: Project 6 Design Vision Netlist and Area from Created Liberty File for My Cells

For the cells, both the total count and area increased appreciably. The cell count increasing is likely due to a couple of reasons; the fact that the DFF was falling edge instead of rising edge, using different generic values for the HDL generated, and my cell liberty file does not contain the AOI21 and OAI21 gates which were used a lot in the netlist in project two. As for the area, the database file clearly has smaller sized gates, while using 20% fewer cells.

The simulation of the new netlist produced the same results as with project two:

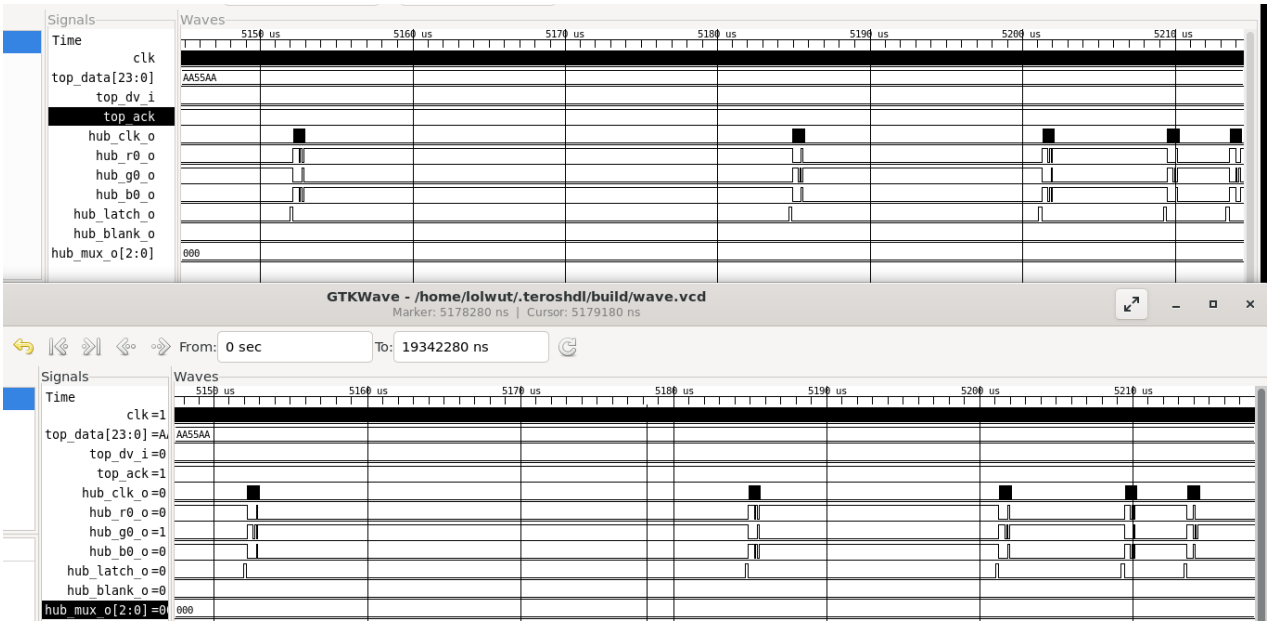

Figure 4: Zoom in on data transfer, Binary Coded Modulation currently being displayed. The wait time between clock and data activity is halved as expected.

Figure 5: Zoom in on one clock and data transfer.

An attempt was made to simulate the design in Hspice as required by point 3 under the “What to Turn In” sub header of project 6, but after one hour and having my account locked due to a storage quota violation, I was unable to get the same simulation to work:

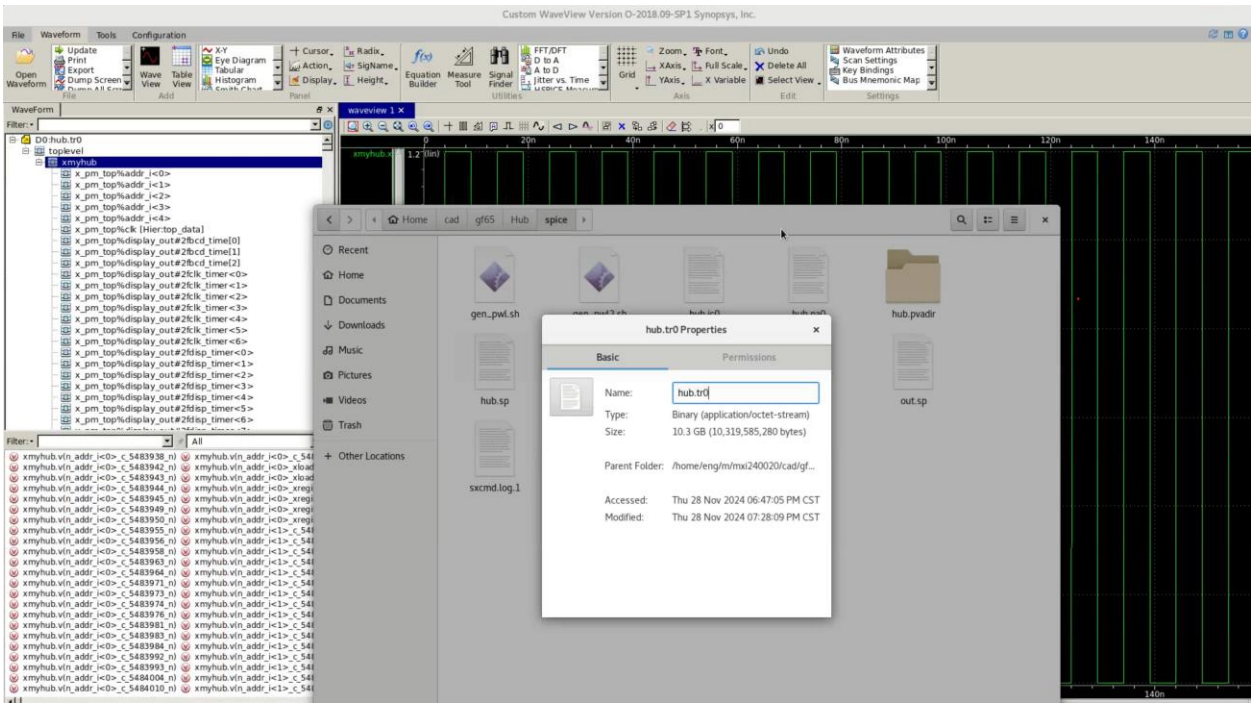

Figure 6: Hspice fiasco.

I think to remedy this, I would need to learn the tool better so I can tell it not to capture the inner workings of each cell and tell it to start capture at the relevant times.

Synopsys Library Complier

The steps here were generally straightforward, however I did have two issues that took some time to resolve:

1) Issue: Missing template definitions
    a. Cause: My D flip flop had different templates than the other cells.
    b. This was resolved by identifying which ones were used/ or different than what was already defined and adding the files to the cell.lib file.
2) Issue: Correctly adding the cells to the library files
    a. Cause: I had a total of 17 cells, so keeping track of each one was tedious and error prone
    b. Resolution: To manage the creation, modification, and appending the liberty files for each of the cells, I created several scripts to automate the process. I go into more detail under the *Other Tools, Resources and Conclusion* section.

LEF File Modifications

The steps to generate and modify this .lef file were mostly straightforward as well, but I did run into an issue here:

1) Issue: The “CatenaDesignType” error that prevented the design import during the Innovus steps.
    a. Cause: Modifying the .lef file as instructed in the project 6 guide removed a critical portion that appears to define macros.
    b. Resolution: This was resolved by adding back the “PROPERTYDEFINITIONS” block that was atop each of the generated .lef files.

Cell Layout & Innovus

I found myself redoing the Innovus steps several times due to errors in my cell. For one, I did have an issue with a few of my cells that I did not catch during project four. I will go into more detail when I discuss the DRC issues in Virtuoso. Once the issues in my cell.lef file and my Verilog netlist files were corrected in Virtuoso and Design Vision respectively, I was able to place and route my design. There were two things I learned the hard way, however, and they were the addition of pins, to tell Innovus where to connect to a net, and the width of the contacts/pins should be 140 nm. Both those issues lead to DRC errors and each time I had to go back and modify the cell layouts, effectively starting over.

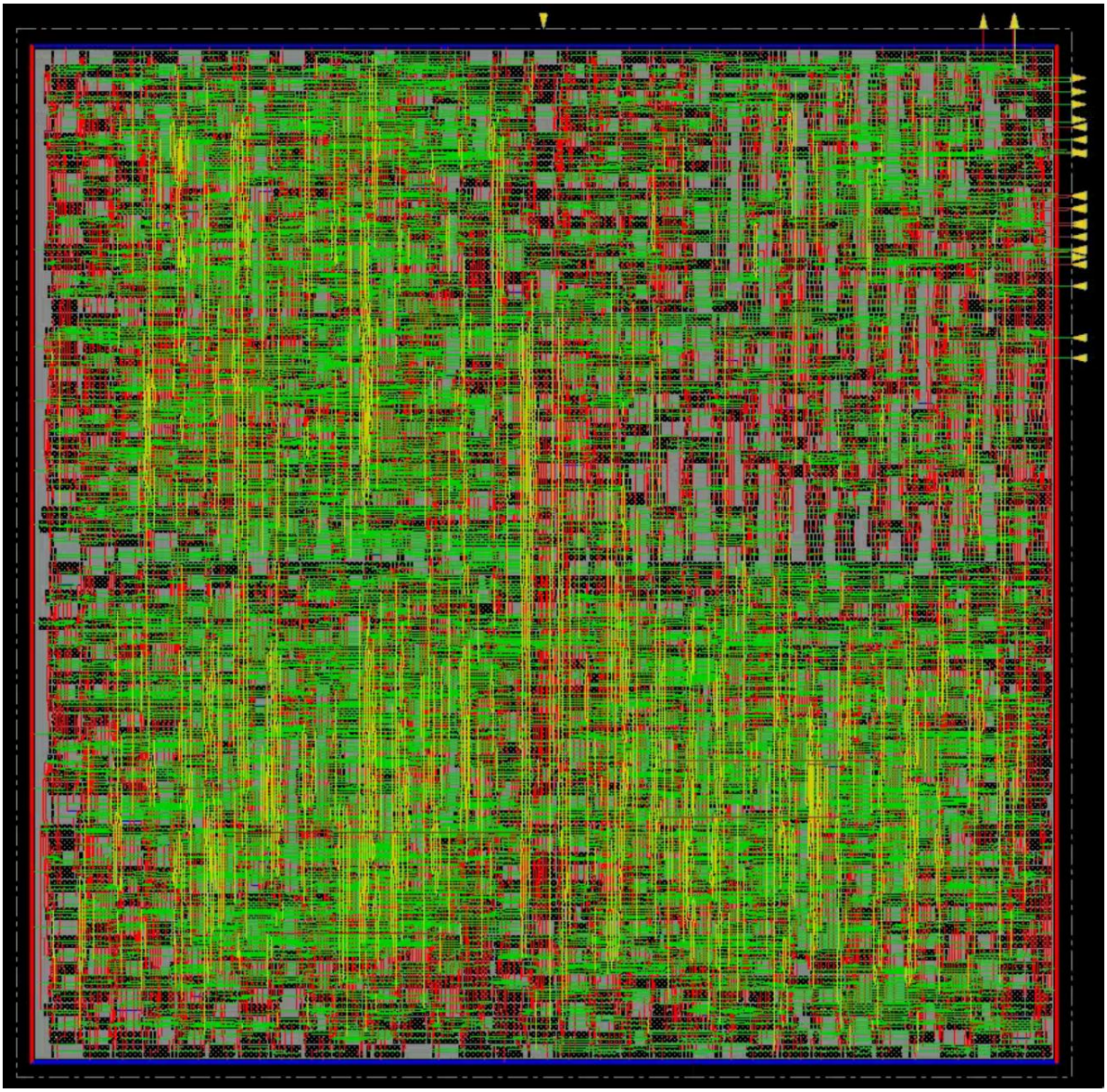

Figure 7: Innovus P&R output

Importing into Virtuoso

The steps for this phase can be summarized as follows:

1) Create a new library based on your previous one
2) Import the Innovus Verilog and DEF files
3) Remaster and flatten the design
4) Run DRC checks and fix issues
5) Run LVS checks and fix issues
6) Generate PEX files for Hspice

I was able to successfully get to step three with no issues, but I had to start over a few times due to steps four and five. The issues I saw, and lessons learned are listed below.

DRC issues

Initially, I had over 2000 errors generated by the tool, which was way too many for me to manually fix in the allotted time. After spending some time studying the types of errors and going back and looking at how Innovus generated its routing network, I realized that connections were being made to the M1 layer and not the M2 layer contacts specified. The other prominent error had to do with net width violations of the contacts when it did connect to them on M2. To resolve these issues, I did research on how to tell the routing tool where to connect and looked at ways to modify the routing trace width. Ultimately, I was unable to modify the trace width, though I think doing so would cause additional violations. I ended up adding the pin layers to each cell to specify where Innovus can connect to in each cell and widened each cell connection to 140 nm to eliminate 'short vertex' DRC errors similar to the one in Figure 9, but on the M2 layer below.

When the DRC errors caused by the cell errors were fixed, I ended up with 32 errors (see Figure 8). These were fixed by placing M1 over the affected area or stretching the metal.

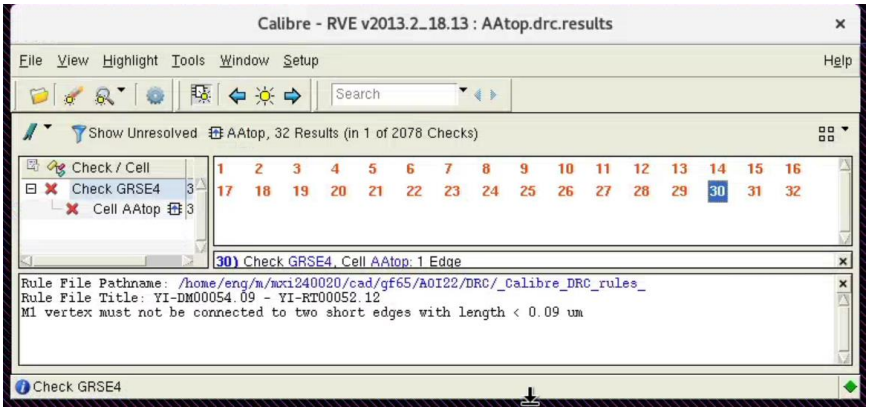

Figure 8: Remaining DRC Errors

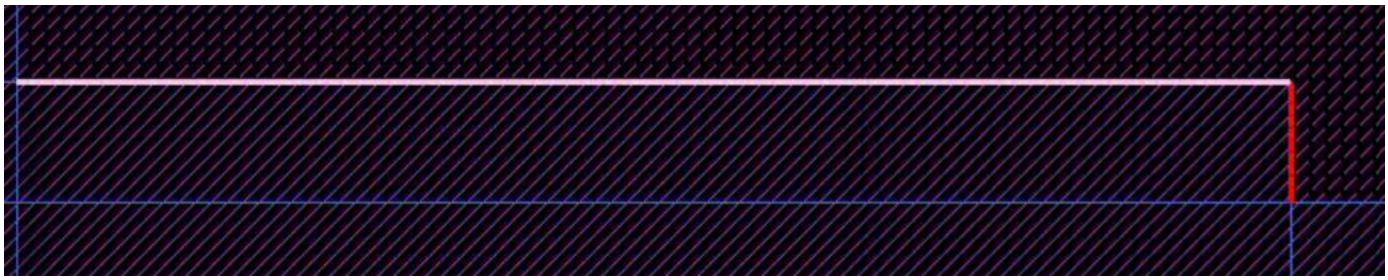

Figure 9: An example of the 'M1 vertex must not be connected to the short edge' error

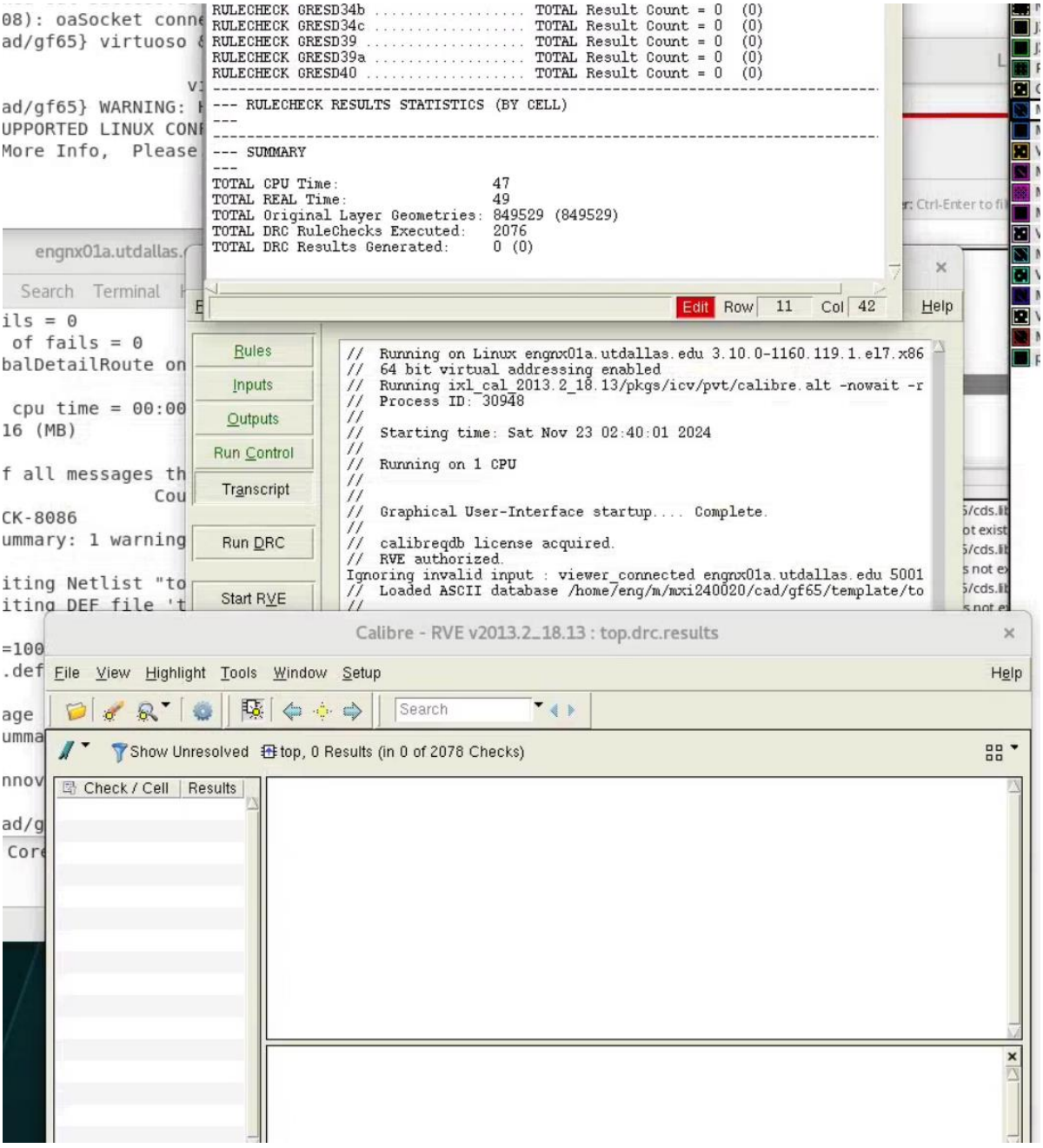

Figure 10: DRC Errors Fixed

LVS Errors

Similarly to the DRC errors, I had a large list of LVS errors. These errors basically fit into four types:

1) Unknown pins/labels on the layout
2) Unable to find labels from the schematic in the layout
3) XOR and OAI22 cell design flaw

The first two issues were easy to resolve; select all the pins and labels in the design and delete them, and for adding schematic layouts, after trial and error I found Create -> (L)abel -> Auto selecting the right layer would allow me to easily add the net names LVS expected to see.

The third error, however, took quite a while for me to debug. There were 10 or so errors, if I recall correctly. But after looking at Verilog file and the nets that were called out, I noticed that the nets all seemed to have the same OAI22 cell on them. Because of this issue, as well as a separate issue discovered when re-testing all my cells, I had to re-do the project steps.

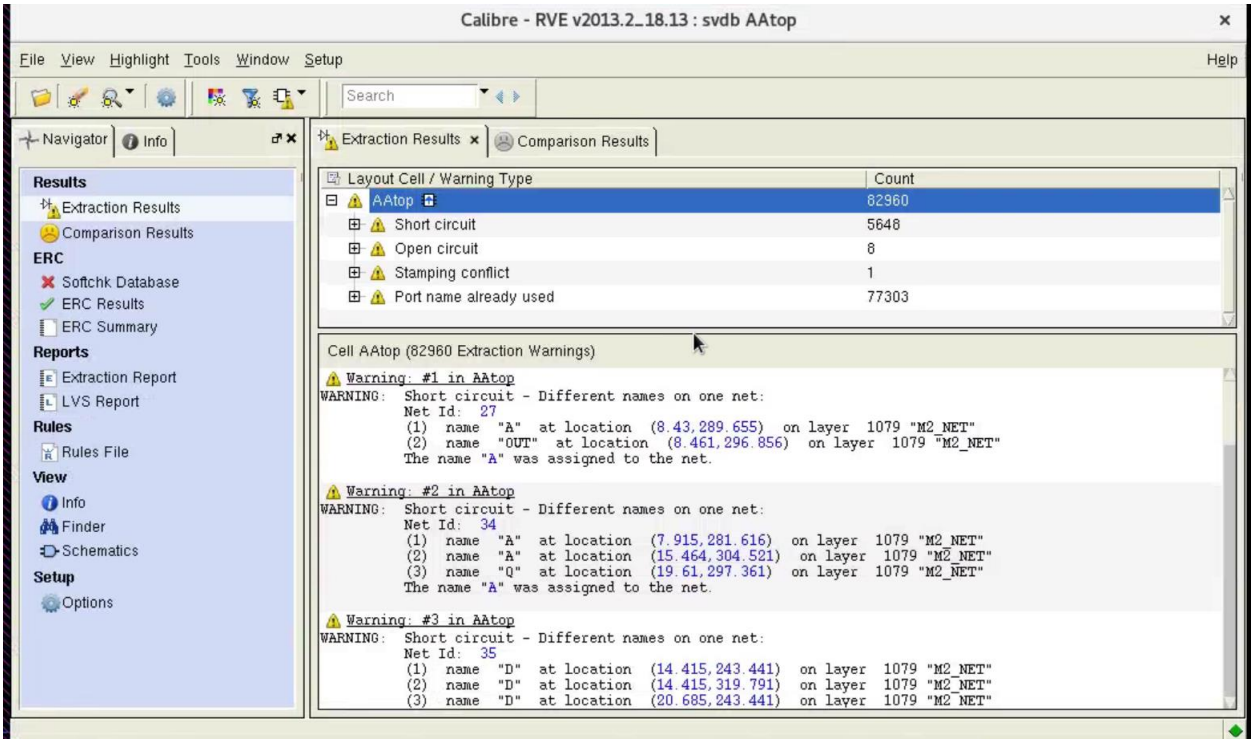

Figure 11: Re-creation of Some of the Errors LVS Generated

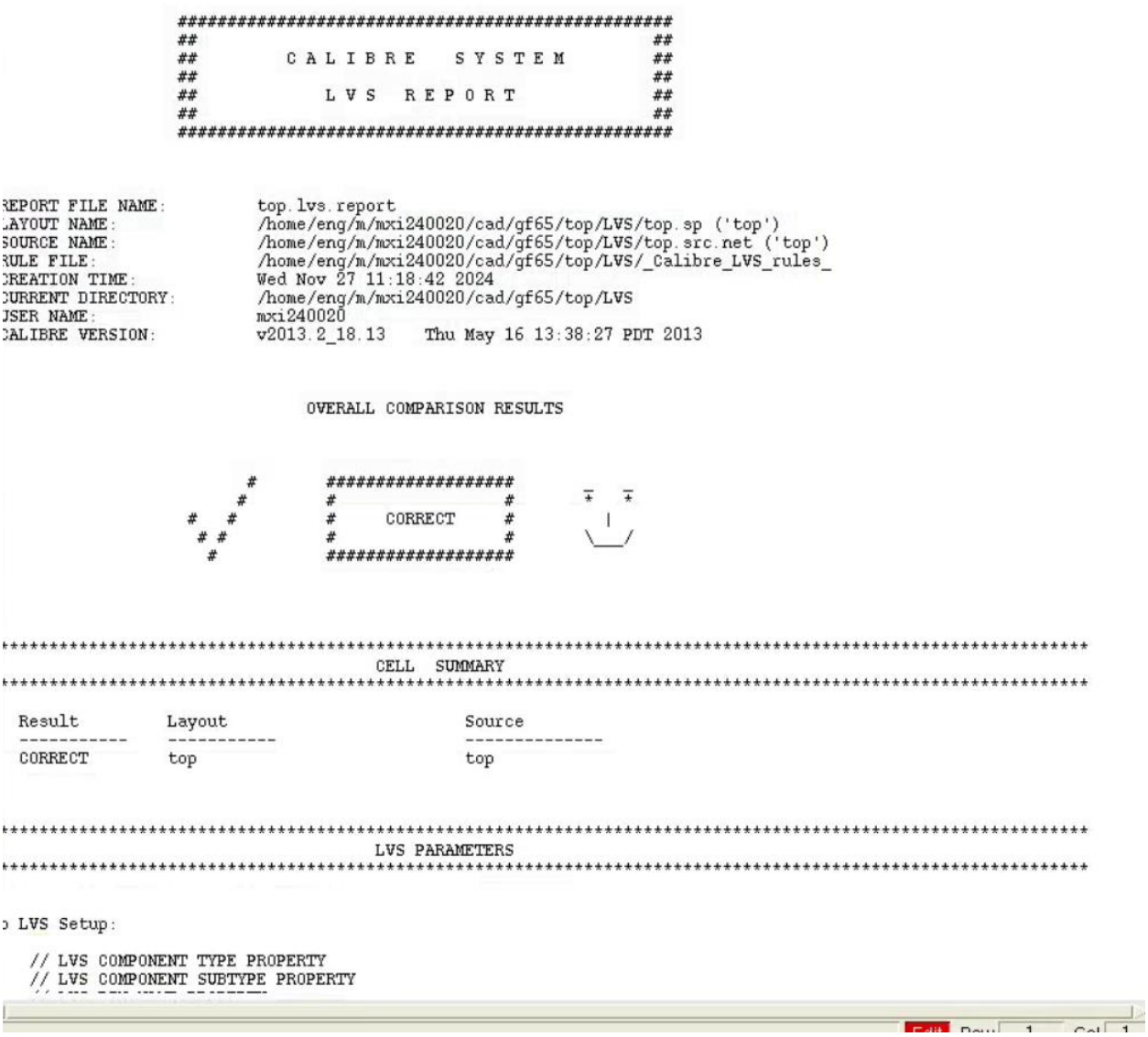

Figure 12: LVS Errors Resolved

Static Timing Analysis

Static Timing Analysis was done using *Primetime*. Using the provided shell script, I modified the 'variables1' file as shown in Figure 13. Figure 14 shows the minimum path delay; Figure 15 shows the maximum path delay; and Figure 16 shows the power estimation of the circuit. Based on a 30 ps rise and fall edge rate on the inputs, and 7.5 ns clock period, I was able to get a simulated clock speed of $133.\overline{3}\ MHz$.

```
6
7   set library_file "mark_gf65.db"
8   set verilog_file "top_innv.v"
9   set input_transition 0.03
10  set load 2
11  set driving_cell "INV"
12  set clock_pin_name "clk"
13  set clock_period 7.5
14  set reset_pin_name "reset"
```

Figure 13: 'Variable1' file used in 'Primetime.script' to navigate pt_shell.

```
Design : top
Version: O-2018.06-SP1
Date   : Sun Nov 24 12:15:55 2024
****************************************


  Startpoint: load_n_store/RGB_Array_0/data_out_o_reg[0]
              (falling edge-triggered flip-flop clocked by clk')
  Endpoint: load_n_store/mem_out_b0_reg[0]
              (falling edge-triggered flip-flop clocked by clk')
  Path Group: clk
  Path Type: min

  Point                                   Cap      Trans       Incr       Path
  -----------------------------------------------------------------------------
  clock clk' (fall edge)                            0.00       0.00       0.00
  clock network delay (ideal)                                  0.00       0.00
  load_n_store/RGB_Array_0/data_out_o_reg[0]/CLK (DFF)
                                                    0.00       0.00       0.00 f
  load_n_store/RGB_Array_0/data_out_o_reg[0]/Q (DFF)
                                          0.00      0.04       0.10       0.10 r
  load_n_store/U23/A (AOI22)                        0.04       0.00       0.10 r
  load_n_store/U23/OUT (AOI22)            0.00      0.04       0.03       0.14 f
  load_n_store/U22/A (INV)                          0.04       0.00       0.14 f
  load_n_store/U22/OUT (INV)              0.00      0.03       0.03       0.17 r
  load_n_store/mem_out_b0_reg[0]/D (DFF)            0.03       0.00       0.17 r
  data arrival time                                                       0.17

  clock clk' (fall edge)                            0.03       0.00       0.00
  clock network delay (ideal)                                  0.00       0.00
  clock reconvergence pessimism                                0.00       0.00
  load_n_store/mem_out_b0_reg[0]/CLK (DFF)                                0.00 f
  library hold time                                            0.03       0.03
  data required time                                                      0.03
  -----------------------------------------------------------------------------
  data required time                                                      0.03
  data arrival time                                                      -0.17
  -----------------------------------------------------------------------------
  slack (MET)                                                             0.14
```

Figure 14: Minimum Path Delay Constraint

```
Startpoint: display_out/hub_b0_o_reg
            (falling edge-triggered flip-flop clocked by clk')
Endpoint: display_out/hub_b0_o_reg
            (falling edge-triggered flip-flop clocked by clk')
Path Group: clk
Path Type: max

Point                                   Cap      Trans       Incr       Path
-----------------------------------------------------------------------------
clock clk' (fall edge)                             0.03       0.00       0.00
clock network delay (ideal)                                   0.00       0.00
display_out/hub_b0_o_reg/CLK (DFF)                 0.03       0.00       0.00 f
display_out/hub_b0_o_reg/Q (DFF)        2.00      14.02       6.85       6.85 r
display_out/U84/D (AOI22)                         14.02       0.00       6.85 r
display_out/U84/OUT (AOI22)             0.00       1.58       0.30       7.15 f
display_out/U83/A (INV)                            1.58       0.00       7.15 f
display_out/U83/OUT (INV)               0.00       0.32       0.25       7.40 r
display_out/hub_b0_o_reg/D (DFF)                   0.32       0.00       7.40 r
data arrival time                                                         7.40

clock clk' (fall edge)                             0.00       7.50       7.50
clock network delay (ideal)                                   0.00       7.50
clock reconvergence pessimism                                 0.00       7.50
display_out/hub_b0_o_reg/CLK (DFF)                                       7.50 f
library setup time                                           -0.09       7.41
data required time                                                        7.41
-----------------------------------------------------------------------------
data required time                                                        7.41
data arrival time                                                        -7.40
-----------------------------------------------------------------------------
slack (MET)                                                               0.01
```

Figure 15: Slack Timing for Critical Path at 7.5ns

```
report_power
****************************************
Report : Averaged Power
Design : top
Version: O-2018.06-SP1
Date   : Sun Nov 24 12:15:56 2024
****************************************


  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           3.019e-05 4.260e-04 2.209e-09 4.562e-04 (80.90%)  i
register                1.991e-05 1.235e-05 1.563e-07 3.242e-05 ( 5.75%)
combinational           2.281e-05 5.242e-05 8.539e-08 7.532e-05 (13.36%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 4.908e-04   (87.03%)
  Cell Internal Power  = 7.291e-05   (12.93%)
  Cell Leakage Power   = 2.439e-07   ( 0.04%)
                         ---------
Total Power            = 5.639e-04  (100.00%)
```

Figure 16: Estimated Power Requirements for the Hub75 IC

Final Project 6 Statistics

Number of Cells: $5647\ Cells$

Cell Types Used

- AOI22
- AOI221
- INV
- NOR2
- NOR3
- NOR4
- XOR2
- NAND2
- NAND3
- NAND4
- OAI22
- OAI221
- OAI41
- DFF

Cell Types Not Used

- AOI41
- AOI32
- OAI32

Integrated Circuit Dimension: $400.56 * 388.245\ \mu m^2$

Worst case Slack: $0.01\ ns$

Estimated Power usage: $563.9\ \mu W$

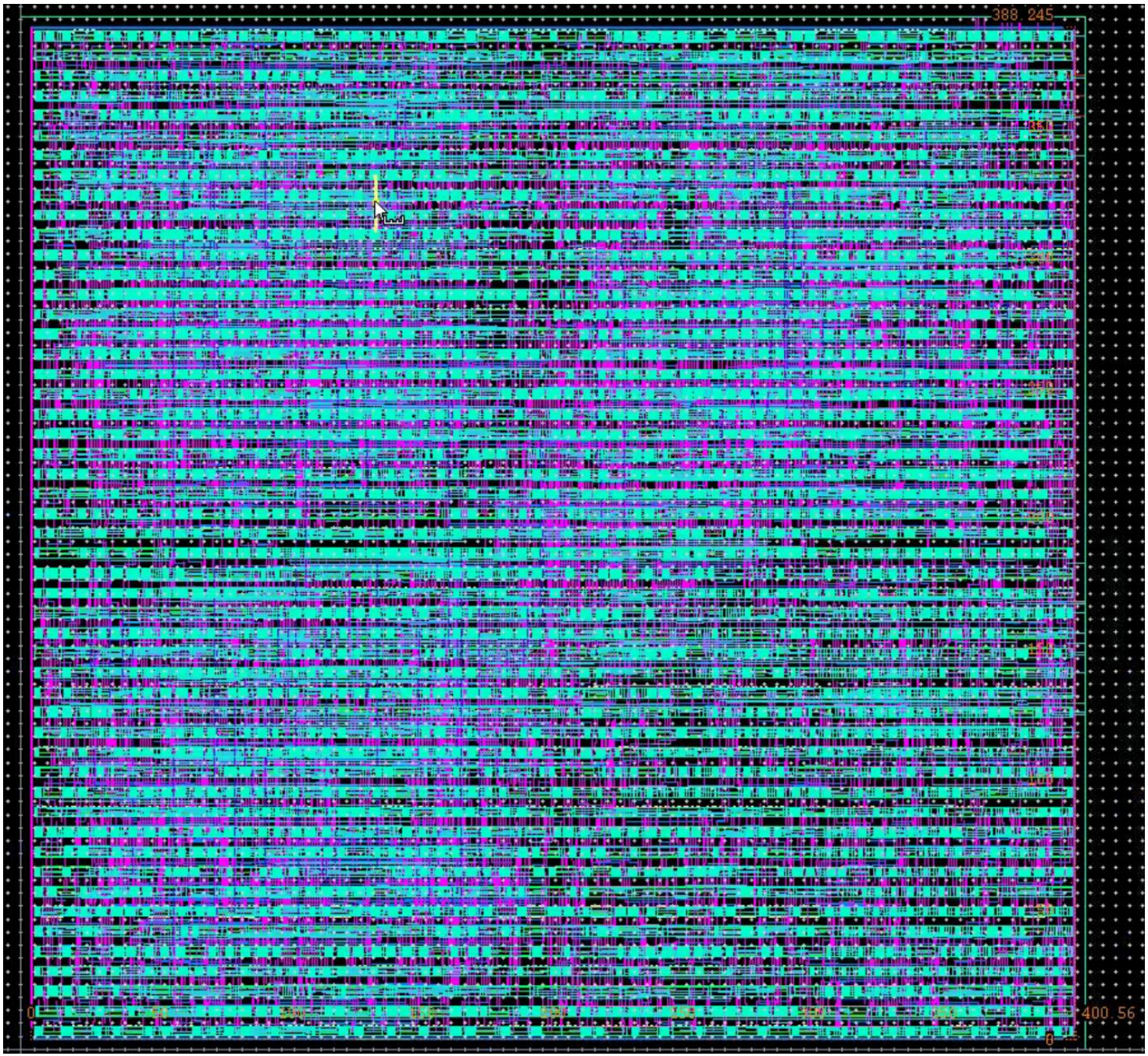

Figure 17:Imported DEF into Virtuoso

Other Tools, Resources and Conclusion

For projects four, five and six, I was able to make use of two other tools to help me manage the cells: AI and scripting. I used the AI in two ways, the first of which was to learn more about the tools I was using and how to do certain things, particularly in Hspice. Secondly, and more importantly, I used AI to help me generate scripts. For example, by investing a few hours early on and creating a structured library file, I was able to successfully automate my liberty file generation. I was also able to automate the Hspice testing of the DFF, by creating a csv from the .mt0 file Hspice created from multiple runs.

With more time, there are a few things I would have done differently. First, the use of the powerful Virtuoso tools like SKILL and Pcells to create the cells of my library. Secondly, in class we learned much later about logical effort; and with it, it became clear that some gates don't make much sense to create, like the OAI41, and other larger five input gates. Third, I would have implemented several cells, like SRAM, decoders, and the mirror sum/mirror carry logic. Finally, I would have spent more time in Design Vision to add timing constraints and conducted more testing with PrimeTime to ensure I got the 200MHz.