Synopsys ASIC Tutorial

Version 11.4 Updated December 14th, 2015

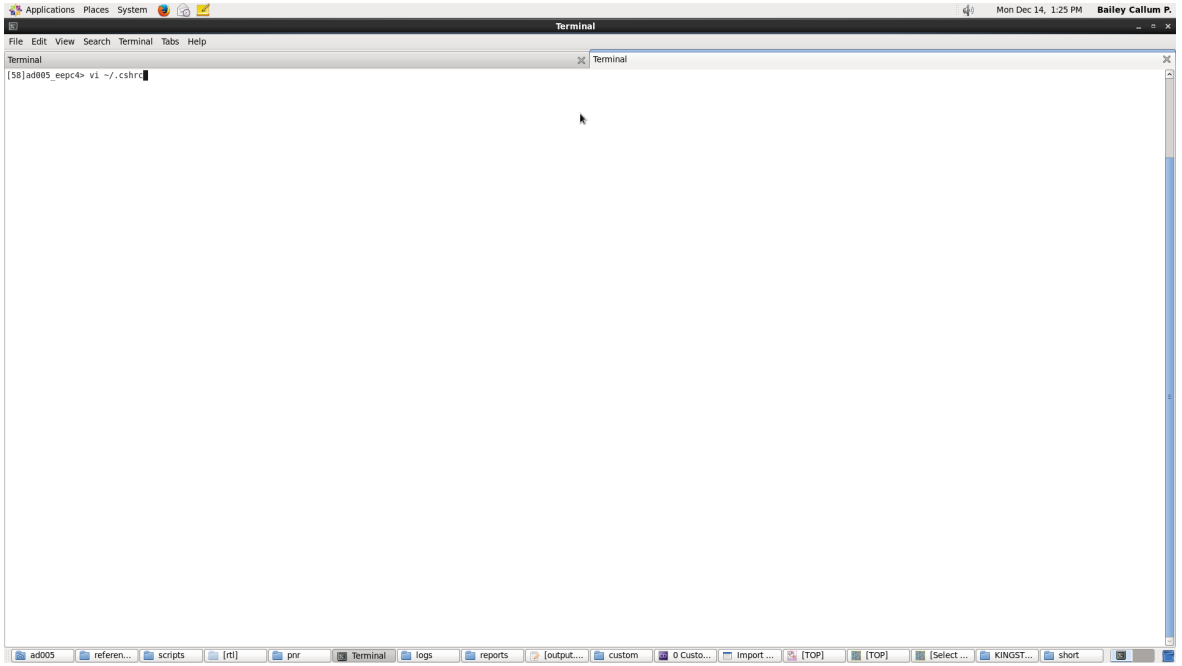
- Linux log in and tutorial
- Synthesis with ***dc_shell***
  - Timing
  - Area
- Chip implementation with ***icc_shell***
  - Placement
  - Routing
  - Clock tree
  - Finishing
  - Verify design with Design Rules Checks (DRC) and
  - Layout vs. Schematic (LVS).
- Chip testing and verification with ***cdesigner***
  - Final Design Rules Checks (DRC)
  - Add art to metal 3 (optional)
- MOSIS submission

LINUX Configuration

Before beginning you need to add the path for Synopsys tools to your environment so that you have the correct tools loaded in your environment. This is university specific. USM should be configured correctly and this step is unnecessary. UTEP has to source a config file and adding it to your .cshrc file causes this to happen automatically when you start the shell.

Open terminal and write the following command from your home directory.

vi ~/.cshrc



LINUX Configuration

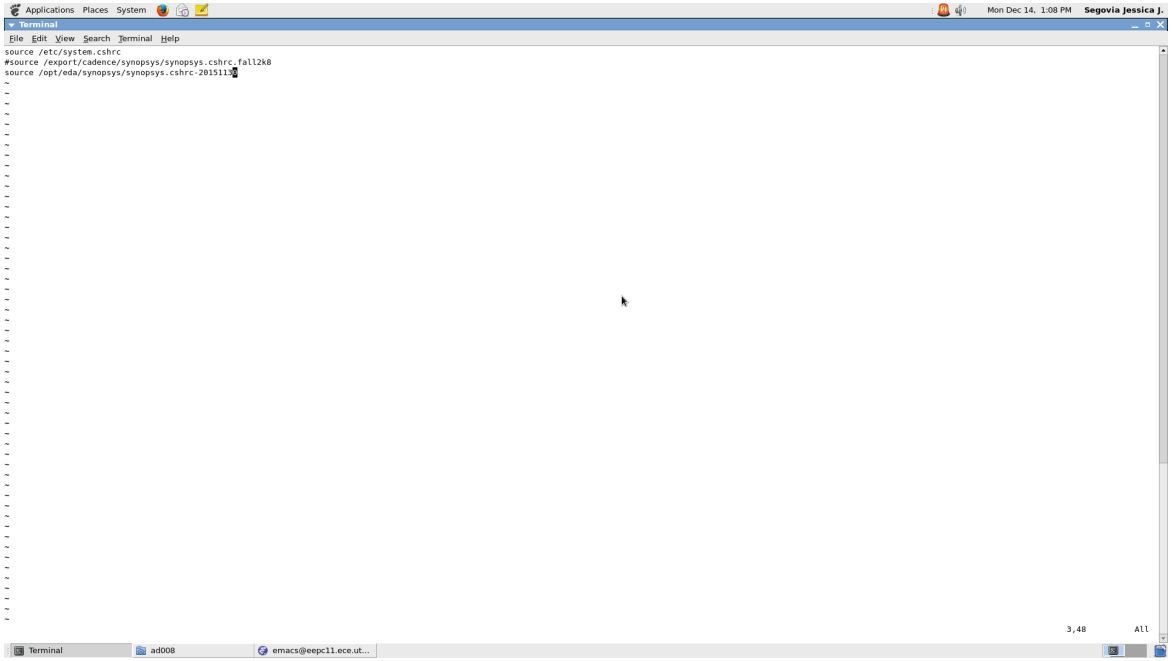
Hit the i key on your keyboard to edit the file

Type in the following at the end of the file. USM is already configured.

source /opt/eda/synopsys/synopsys.cshrc-20151130 (University specific)

If the following line is present comment out by inserting a # at the front

Source /export/cadence/synopsys/synopsys.cshrc.fall2k8



LINUX Configuration

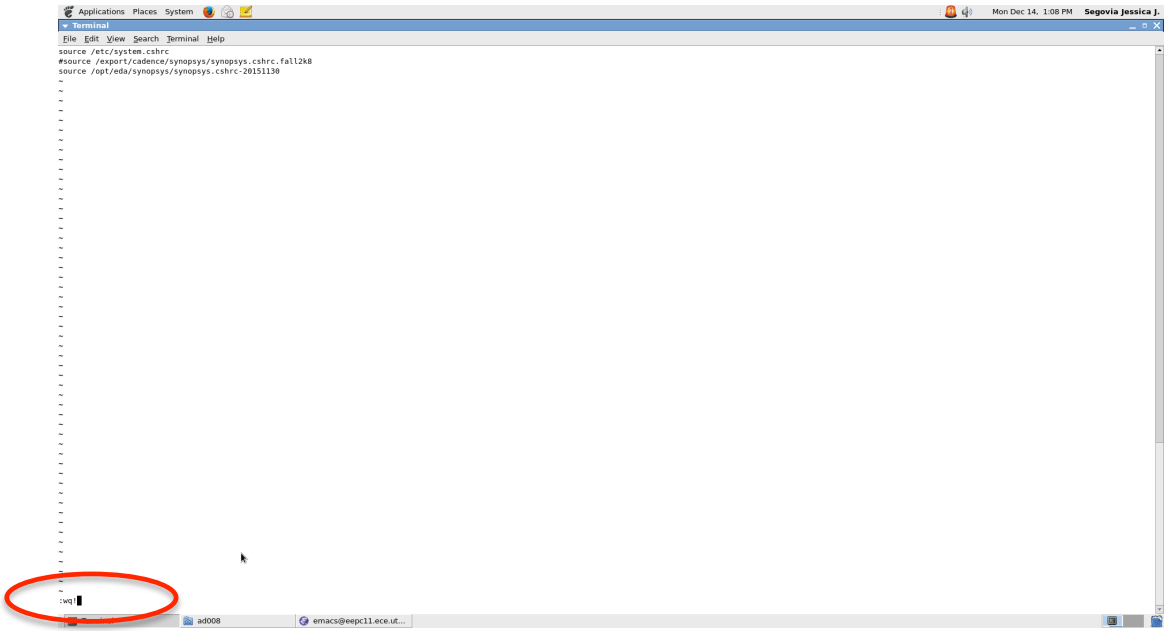
Hit the escape key on your keyboard

Type in the : key on your keyboard

Type in the following

:wq!

This will save the file.



Log in with Linux

ssh allows us to login to the system and
"-X" enables X Windows support.
UTEP can log in to eepc[1-20].ece.utep.edu.
USM can log in to 200.1.17.47.

```
pcsos1:~ ericmacdonald$ ssh -X grupo1@200.1.17.47
grupo1@200.1.17.47's password:
Last login: Wed Aug  5 10:37:54 2015 from pcsos1.elo.utfsm.cl
[grupo1@200 ~]$ pwd
/home/alm/grupo1
[grupo1@200 ~]$ cd ref
reference_design/     reference_design.tar
[grupo1@200 ~]$ cd reference_design
[grupo1@200 ~/reference_design]$ ls -arlt
total 32
drwxr-xr-x.  2 grupo1 grupo1 4096 Aug  2 17:51 golden
drwxr-xr-x.  2 grupo1 grupo1 4096 Aug  2 18:02 custom
drwxr-xr-x.  8 grupo1 grupo1 4096 Aug  2 18:06 .
drwxr-xr-x.  2 grupo1 grupo1 4096 Aug  2 18:16 sim
drwxr-xr-x.  2 grupo1 grupo1 4096 Aug  4 15:29 rtl
drwxr-xr-x.  8 grupo1 grupo1 4096 Aug  4 15:36 syn
drwxr-xr-x. 15 grupo1 grupo1 4096 Aug  4 16:21 pnr
drwxrw-rw-. 23 grupo1 grupo1 4096 Aug  5 10:38 ..
[grupo1@200 ~/reference_design]$
```

Quick Linux Tutorial

You are expected to have a working knowledge of Linux for this lab.

"pwd" – print working directory – should be your user home directory.
"cd" – change directory into reference_design.

"ls –alrt" – list directory contents.

Other useful Linux commands:

- man – help for commands
- top
- mkdir
- passwd
- cd ~
- cd ..
- rm
- grep
- more
- cat

Type xclock and you should see a clock appear if X Windows is working properly - shown on your screen but running remotely.

Copy the Reference Design

cd /opt/eda/synopsys/local/ON.PDK (UTEP location – ask administrator)

cp –r reference_design/ ~/.
This will create a directory structure.

cd ~/reference_design

RTL is where you should store your source verilog code.

SYN is where you will perform synthesis. Update the script "dc_good.tcl" in scripts and invoke "dc_shell –f dc_good.tcl | tee logs/output.syn"

PNR is where you will perform place and route. Update the script "icc_good.tcl" in scripts and invoke "ic_shell –f icc_good.tcl | tee logs/output.ic"

CUSTOM is where you store reference files. Eg: matlab files that create test vectors and expected results, simulation will use these vectors and match the results.

SIM is where you store your simulation files. Optional in this case.

GOLDEN is where you store your final files – RTL, GDS, SDC, TB.v, and scripts.

Design Compiler - Reference Design

```
module xtal_chip (clk,reset, out1, out2);
input        clk;
input        reset;
output [2:0]  out1;  // counter clocked by traditional clock - clk_in1
output [2:0]  out2;  // counter clocked by internally generated clock


reg   [19:0]  count1;
wire   [2:0]  out1 = count1[19:18];

reg   [19:0]  count2;
wire   [2:0]  out2 = count2[19:18];

wire          internal_clk;

// added to select the source of an internally generated clock
// necessary?  I had to find new source after compiling the RTL.
BUFX2 buffer(.A(count1[10]), .Y(internal_clk));

always @(posedge clk)
  if (reset) count1 <= 0;
  else       count1 <= count1 + 1;

always @(posedge internal_clk)
  if (reset) count2 <= 0;
  else       count2 <= count2 + 1;

endmodule
```

Simple reference design
Two counters – one clocked by external clock, the other by an internally generated clock. All clocks have to be identified for static timing to work correctly. Total of 40 flip-flops no where near the limit in terms of area for this chip size.

Only four outputs and two inputs, power and ground. Total is 8 pins.

Design Compiler - Prepare for synthesis

```
[grupo1@200 ~/reference_design]$ cd syn/
[grupo1@200 syn]$ ls
alib-52  command.log  default.svf  input  logs  reports  results  scripts
[grupo1@200 syn]$ dc_shell -f scripts/
add_pads.tcl          dc_good.tcl           dc_orig.tcl           placer_blockages.tcl  topo.tcl
[grupo1@200 syn]$ dc_shell -f scripts/
add_pads.tcl          dc_good.tcl           dc_orig.tcl           placer_blockages.tcl  topo.tcl
[grupo1@200 syn]$ dc_shell -f scripts/dc_good.tcl | tee logs/output.syn
```

cd to syn directory for synthesis

cd syn

Edit and update the script “dc_good.tcl” for your specific design in scripts subdirectory.

- identify file and module name (both should be identical).
- identify clocks and targeted frequency of operation.
- Run “dc_shell –f ” to invoke script as shown above in shell.

Look in reports subdirectory for quality of output by reading area and timing reports.

The tool can be opened in gui format as well – “dc_shell –gui” although synthesis is traditionally done with a script only.

Design Compiler - Checking synthesis

```
Compiling source file /home/alm/grupo1/reference_design/rtl/xtal_chip.v

Inferred memory devices in process
        in routine xtal_chip line 19 in file
                '/home/alm/grupo1/reference_design/rtl/xtal_chip.v'.
=========================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=========================================================================================
|     count1_reg      | Flip-flop |  20   |  Y  | N  | N  | N  | N  | N  | N  |
=========================================================================================

Inferred memory devices in process
        in routine xtal_chip line 23 in file
                '/home/alm/grupo1/reference_design/rtl/xtal_chip.v'.
=========================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=========================================================================================
|     count2_reg      | Flip-flop |  20   |  Y  | N  | N  | N  | N  | N  | N  |
=========================================================================================
Presto compilation completed successfully.
```

During synthesis you should see standard output.

Flip-flops will be identified. Make sure this make sense. I have two 20 bit counters in the reference design so this report looks reasonable.

Also look for latches which indicates a problem in your RTL. All if statements need an else (fully qualifying the statement).

Design Compiler - After synthesis

```
write -hier -format verilog -output results/${top_module}.postsynth.v
Writing verilog file '/home/alm/grupo1/reference_design/syn/results/xtal_chip.postsynth.v'.
1
write_sdc results/${top_module}.postsynth.sdc
1
write_sdf  results/${top_module}.postsynth.sdf
Information: Annotated 'cell' delays are assumed to include load delay. (UID-282)
Information: Writing timing information to file '/home/alm/grupo1/reference_design/syn/results/xtal_chip.postsynth.sdf'. (WT-3)
1
1
dc_shell> exit

Thank you...
[grupo1@200 syn]$ cd results/
[grupo1@200 results]$ ls
xtal_chip.postsynth.ddc  xtal_chip.postsynth.sdc  xtal_chip.postsynth.sdf  xtal_chip.postsynth.v
[grupo1@200 results]$ cd ../logs/
[grupo1@200 logs]$ ls
dc.log  output.log  output.syn  synthesis.output
[grupo1@200 logs]$ cd ../reports/
[grupo1@200 reports]$ ls
xtal_chip.area.rpt  xtal_chip.noIO.area.rpt  xtal_chip.noIO.timing.rpt  xtal_chip.power.rpt  xtal_chip.qor.rpt  xtal_chip.timing.rpt
[grupo1@200 reports]$
```

Final Report: Add timing and area reports to final report.

After script is run, look for return values of 1 for all commands. 0 is a failure.

In results subdirectory, *postsynth.v is your netlist and a sdc file with timing info.

Check the netlist to make sure it looks reasonable.

In logs, you will see the commands and output. Search for errors and “0’s”.

In reports, check all files thoroughly.

- Timing – check that critical path passes (positive slack) and is a legitimate path.
- Area – not too big. 500k square microns or less is reasonable.
- Power is interesting but not very accurate. Power is data dependent but you can improve by adding switching data from the simulator VCS.

Design Compiler - Checking synthesis

```
module xtal_chip_with_io ( clk, reset, \out1[2] , \out1[1] , \out1[0] ,
        \out2[2] , \out2[1] , \out2[0]  );
  input clk, reset;
  output \out1[2] , \out1[1] , \out1[0] , \out2[2] , \out2[1] , \out2[0] ;
  wire   n2, n4, n8, n10, n14, n16, \I_xtal_chip/N82 , \I_xtal_chip/N81 ,
         \I_xtal_chip/N80 , \I_xtal_chip/N79 , \I_xtal_chip/N78 ,
         \I_xtal_chip/N77 , \I_xtal_chip/N76 , \I_xtal_chip/N75 ,
         \I_xtal_chip/N74 , \I_xtal_chip/N73 , \I_xtal_chip/N72 ,
         \I_xtal_chip/N71 , \I_xtal_chip/N70 , \I_xtal_chip/N69 ,
         \I_xtal_chip/N68 , \I_xtal_chip/N67 , \I_xtal_chip/N66 ,
         \I_xtal_chip/N65 , \I_xtal_chip/N64 , \I_xtal_chip/N63 ,
         \I_xtal_chip/N42 , \I_xtal_chip/N41 , \I_xtal_chip/N40 ,
         \I_xtal_chip/N39 , \I_xtal_chip/N38 , \I_xtal_chip/N37 ,
         \I_xtal_chip/N36 , \I_xtal_chip/N35 , \I_xtal_chip/N34 ,
         \I_xtal_chip/N33 , \I_xtal_chip/N32 , \I_xtal_chip/N31 ,
         \I_xtal_chip/N30 , \I_xtal_chip/N29 , \I_xtal_chip/N28 ,
         \I_xtal_chip/N27 , \I_xtal_chip/N26 , \I_xtal_chip/N25 ,
         \I_xtal_chip/N24 , \I_xtal_chip/N23 , \I_xtal_chip/N13 ,
         \I_xtal_chip/count1[0] , \I_xtal_chip/count1[1] ,
         \I_xtal_chip/count1[2] , \I_xtal_chip/count1[3] ,
         \I_xtal_chip/count1[4] , \I_xtal_chip/count1[5] ,
         \I_xtal_chip/count1[6] , \I_xtal_chip/count1[7] ,
         \I_xtal_chip/count1[8] , \I_xtal_chip/count1[9] ,
         \I_xtal_chip/count1[10] , \I_xtal_chip/count1[11] ,
         \I_xtal_chip/count1[12] , \I_xtal_chip/count1[13] ,
         \I_xtal_chip/count1[14] , \I_xtal_chip/count1[15] ,
```

Top of the netlist should have your module name, IO's and nets.

You will see some starting with a back slash which is an escape character as some tools cannot handle square brackets and this character is used to escape.

Design Compiler - Checking synthesis

```
       n328, n329, n330, n331, n332;
wire   [17:0] \I_xtal_chip/count2 ;
tri   clk;
tri   reset;
tri   \out1[2] ;
tri   \out1[1] ;
tri   \out1[0] ;
tri   \out2[2] ;
tri   \out2[1] ;
tri   \out2[0] ;

DCX1 \I_xtal_chip/count1_reg[0]  ( .D(\I_xtal_chip/N23 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[1]  ( .D(\I_xtal_chip/N24 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[2]  ( .D(\I_xtal_chip/N25 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[3]  ( .D(\I_xtal_chip/N26 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[4]  ( .D(\I_xtal_chip/N27 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[5]  ( .D(\I_xtal_chip/N28 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[6]  ( .D(\I_xtal_chip/N29 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[7]  ( .D(\I_xtal_chip/N30 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[8]  ( .D(\I_xtal_chip/N31 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count1_reg[9]  ( .D(\I_xtal_chip/N32 ), .CLK(n2), .CLR(n50), .Q(\I_xtal_chip/count1[
DCX1 \I_xtal_chip/count2_reg[0]  ( .D(\I_xtal_chip/N63 ), .CLK(
      \I_xtal_chip/count1[10] ), .CLR(n50), .Q(\I_xtal_chip/count2 [0]) );
DCX1 \I_xtal_chip/count2_reg[1]  ( .D(\I_xtal_chip/N64 ), .CLK(
      \I_xtal_chip/count1[10] ), .CLR(n50), .Q(\I_xtal_chip/count2 [1]) );
```

Tri's will be inserted due to the IO's being bidirectional.

Also, look at the gates. DCX1 is a flip-flop. Scroll down to check all of the gates.

Design Compiler - Checking synthesis

```
pad_bidirhe u_clk_pad ( .DataOut(n145), .EN(n145), .pad(clk), .DataIn(n2) );
pad_bidirhe u_reset_pad ( .DataOut(n145), .EN(n145), .pad(reset), .DataIn(n4) );
pad_bidirhe \u_out1[2]_pad  ( .DataOut(n145), .EN(n50), .pad(\out1[2] ) );
pad_bidirhe \u_out1[1]_pad  ( .DataOut(n8), .EN(n50), .pad(\out1[1] ) );
pad_bidirhe \u_out1[0]_pad  ( .DataOut(n10), .EN(n50), .pad(\out1[0] ) );
pad_bidirhe \u_out2[2]_pad  ( .DataOut(n145), .EN(n50), .pad(\out2[2] ) );
pad_bidirhe \u_out2[1]_pad  ( .DataOut(n14), .EN(n50), .pad(\out2[1] ) );
pad_bidirhe \u_out2[0]_pad  ( .DataOut(n16), .EN(n50), .pad(\out2[0] ) );
NAND2X1 U91 ( .A(n143), .B(\I_xtal_chip/count1[10] ), .Y(n144) );
NOR2X1 U223 ( .A(n319), .B(n318), .Y(n143) );
NOR2X1 U224 ( .A(n328), .B(n327), .Y(n245) );
NAND2X1 U225 ( .A(n245), .B(\I_xtal_chip/count1[17] ), .Y(n331) );
NOR2X1 U226 ( .A(n325), .B(n324), .Y(n249) );
NAND2X1 U227 ( .A(n249), .B(\I_xtal_chip/count1[15] ), .Y(n328) );
NOR2X1 U228 ( .A(n322), .B(n321), .Y(n247) );
NAND2X1 U229 ( .A(n247), .B(\I_xtal_chip/count1[13] ), .Y(n325) );
```

Pad_bidirhe are the IO cells and you should see other gates like NAND2X1.

NAND2X1 is the smallest 2 input NAND gate. X1 means small and the 2 is the number of inputs. A and B are inputs and Y is the output for this library.

Design Compiler - Timing report

Worse case path for main clock.

Through the adder from bit 0 to bit 19 as expected. 20 nS to complete and finished in 8.34 nS so time to spare – which means positive slack of 11.27 nS.

Passed and made sense. Note that these timings are only for setup calculations and no holds are completed. Since the clock tree is not in place (ideal clock tree assumed) there is no clock skew which can cause hold violations. Clock skew occurs with placement.

Final Report: Add timing report to final report.

```
Startpoint: I_xtal_chip/count1_reg[0]
            (rising edge-triggered flip-flop clocked by main_clock)
Endpoint: I_xtal_chip/count1_reg[19]
          (rising edge-triggered flip-flop clocked by main_clock)
Path Group: main_clock
Path Type: max

Point                                                   Incr       Path
-----------------------------------------------------------------------------
clock main_clock (rise edge)                            0.00       0.00
clock network delay (ideal)                             0.00       0.00
I_xtal_chip/count1_reg[0]/CLK (DCX1)                    0.00       0.00 r
I_xtal_chip/count1_reg[0]/Q (DCX1)                      0.80       0.80 f
U245/Y (NAND2X1)                                        0.24       1.04 r
U246/Y (INVX2)                                          0.18       1.23 f
U237/Y (NAND2X1)                                        0.30       1.53 r
U235/Y (NOR2X1)                                         0.42       1.95 f
U236/Y (NAND2X1)                                        0.37       2.32 r
U233/Y (NOR2X1)                                         0.42       2.74 f
U234/Y (NAND2X1)                                        0.37       3.11 r
U231/Y (NOR2X1)                                         0.42       3.53 f
U232/Y (NAND2X1)                                        0.37       3.90 r
U223/Y (NOR2X1)                                         0.45       4.36 f
U91/Y (NAND2X1)                                         0.24       4.60 r
U239/Y (INVX2)                                          0.14       4.74 f
U230/Y (NAND2X1)                                        0.33       5.07 r
U228/Y (NOR2X1)                                         0.41       5.48 f
U229/Y (NAND2X1)                                        0.37       5.85 r
U226/Y (NOR2X1)                                         0.42       6.27 f
U227/Y (NAND2X1)                                        0.37       6.64 r
U224/Y (NOR2X1)                                         0.42       7.06 f
U225/Y (NAND2X1)                                        0.37       7.44 r
U240/Y (NOR2X1)                                         0.37       7.81 f
U241/Y (OAI21X1)                                        0.25       8.06 r
U242/Y (AOI21X1)                                        0.28       8.34 f
I_xtal_chip/count1_reg[19]/D (DCX1)                     0.00       8.34 f
data arrival time                                                  8.34

clock main_clock (rise edge)                           20.00      20.00
clock network delay (ideal)                             0.00      20.00
I_xtal_chip/count1_reg[19]/CLK (DCX1)                   0.00      20.00 r
library setup time                                     -0.39      19.61
data required time                                                19.61
-----------------------------------------------------------------------------
data required time                                                19.61
data arrival time                                                 -8.34
-----------------------------------------------------------------------------
slack (MET)                                                       11.27
```

Design Compiler - Area report

Check for latches (should not exist) and that the basic numbers are reasonable.

For the MOSIS chip, I expect that 3-5 times this area is possible, so 500,000 square microns should fit in the allocated space.

Final Report: Add area report to final report.

```
****************************************
Report : area
Design : xtal_chip_with_io
Version: I-2013.12-SP5-2
Date   : Wed Aug  5 10:57:32 2015
****************************************

Library(s) Used:

    c5n_utah_std_v5_t27 (File: /home/synopsys/ON.PDK/sample/logic/c5n_utah_std_v5_t27.db)
    io (File: /home/synopsys/ON.PDK/sample/logic/io.db)

Number of ports:                          8
Number of nets:                         187
Number of cells:                        185
Number of combinational cells:          137
Number of sequential cells:              40
Number of macros/black boxes:             8
Number of buf/inv:                       21
Number of references:                    11

Combinational area:              42120.000000
Buf/Inv area:                     4536.000000
Noncombinational area:           54720.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 96840.000000
Total area:                 undefined
1
```

IC Compiler

Now we will use a different tool – IC Compiler – to place and route and clock tree synthesis.

IC Compiler - Place, route, timing and checks

```
[grupo1@200 pnr]$ icc_shell -gui -f scripts/icc.tcl | tee logs/output.icc

                          IC Compiler (TM)
                        IC Compiler-PC (TM)
                        IC Compiler-XP (TM)
                        IC Compiler-DP (TM)
                        IC Compiler-AG (TM)

         Version I-2013.12-SP5-1 for RHEL64 -- Sep 24, 2014

Zroute is the default router for ICC, ICC-PC, ICC-DP and ICC-AG in IC Compiler.
         Classic router will continue to be fully supported.

               Copyright (c) 1988-2014 Synopsys, Inc.

This software and the associated documentation are confidential and
proprietary to Synopsys, Inc. Your use or disclosure of this software
is subject to the terms and conditions of a written license agreement
between you, or your company, and Synopsys, Inc.

Initializing...
```

The icc_good.tcl script will read the synthesis directory to obtain the netlist and sdc file (timing).

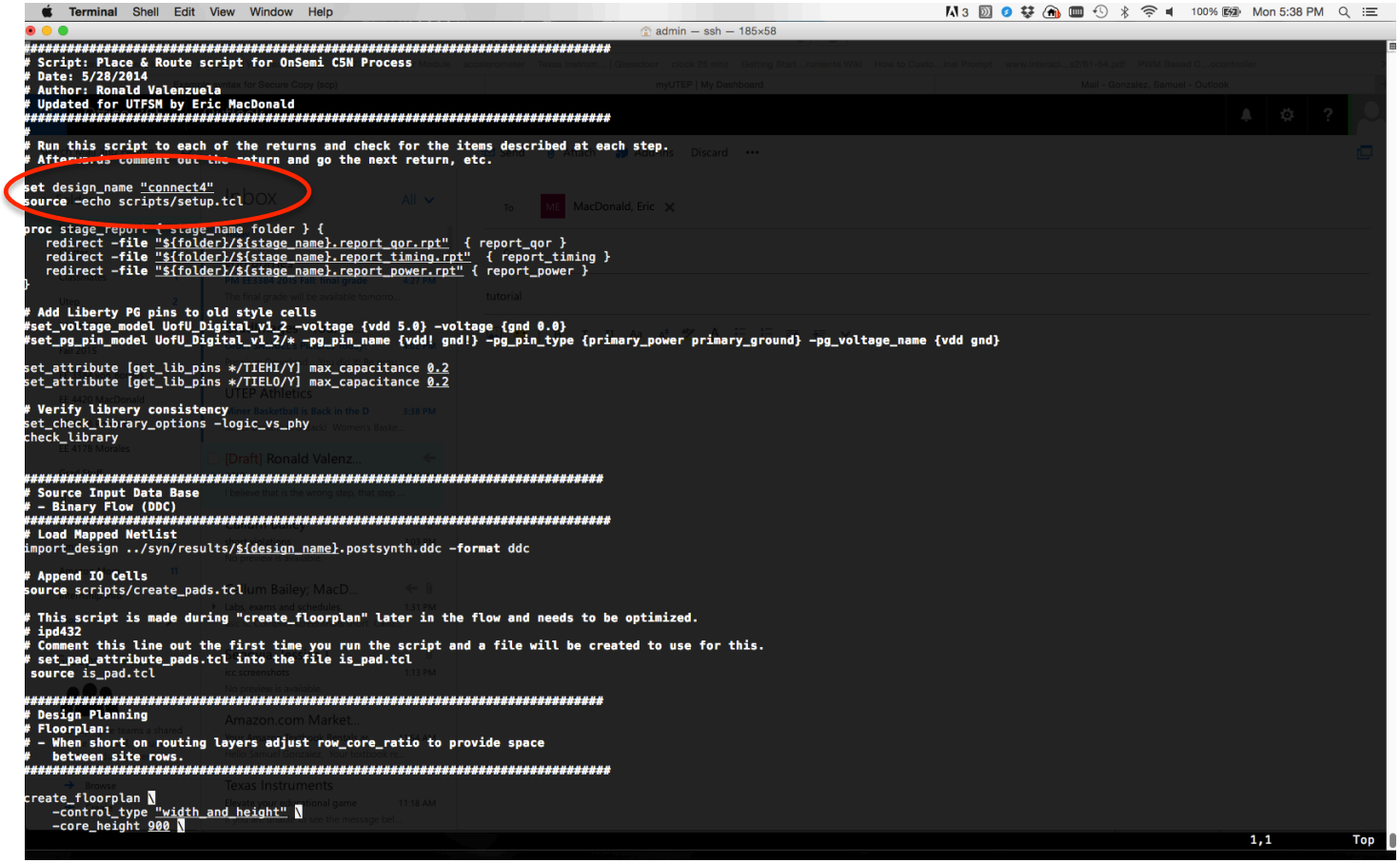
Note that the script has commented lines indicating different sections of the script that will be run piecemeal. The command “return” will stop the script and is used to run to a specific location. You can comment these out to run further in the script.

IC Compiler - Place, route, timing and checks

cd into pnr sub directory

edit icc_good.tcl script found under pnr/scripts/

-update the design_name

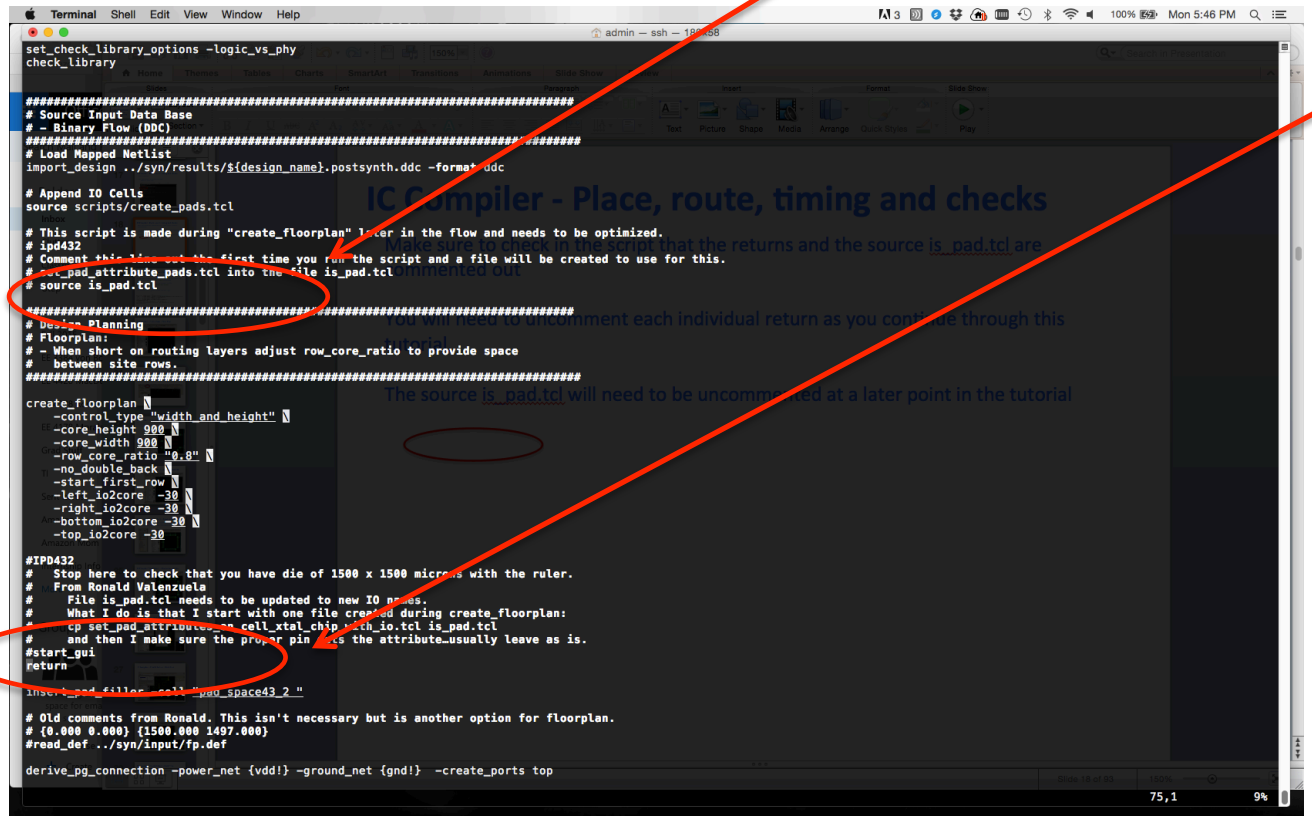


IC Compiler - Place, route, timing and checks

Make sure to check in the script that the returns and the source is_pad.tcl are commented out

You will need to uncomment each individual “return” to continue through this tutorial script.

The source is_pad.tcl will need to be uncommented at a later point in the tutorial



1st return should be uncommented

All others should be commented out “#”.

To continue to next return comment out return and uncommend the following return in icc_good.tcl script.

IC Compiler – First return – Floor Plan

Run to first "***return***" in script – start_gui to invoke viewer.

You will need to run to this point twice: once just to create an IO file and a second time using the IO file with the line "*source is_pad.tcl*" uncommented.

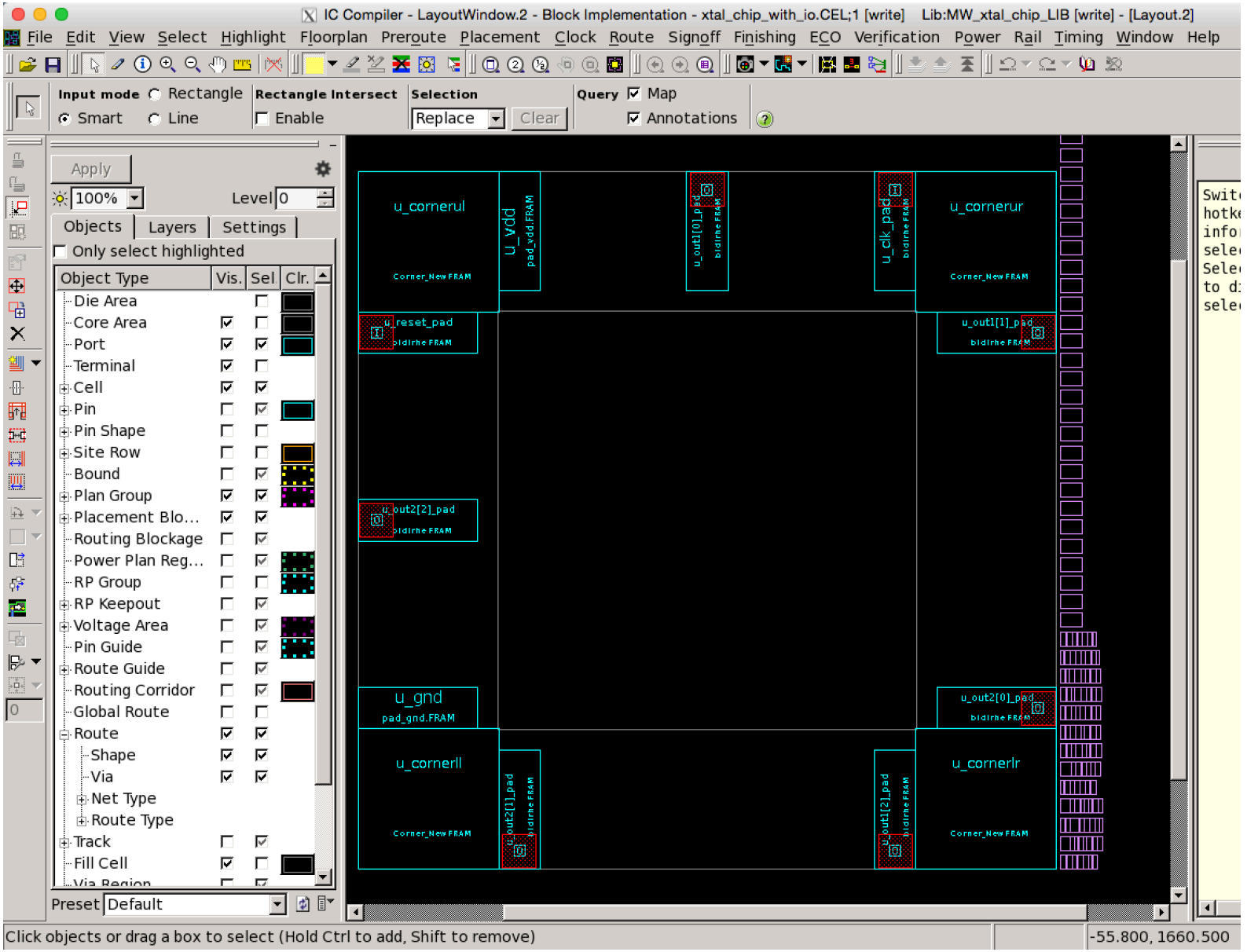
Run the first time and change the IO file name of a newly created file:
cp set_pad_attributes_on_cell_[your_design_name]_io.tcl is_pad.tcl

Check that the file has all IO declared.

Uncomment the "source is_pad.tcl" line in the script and run second time.

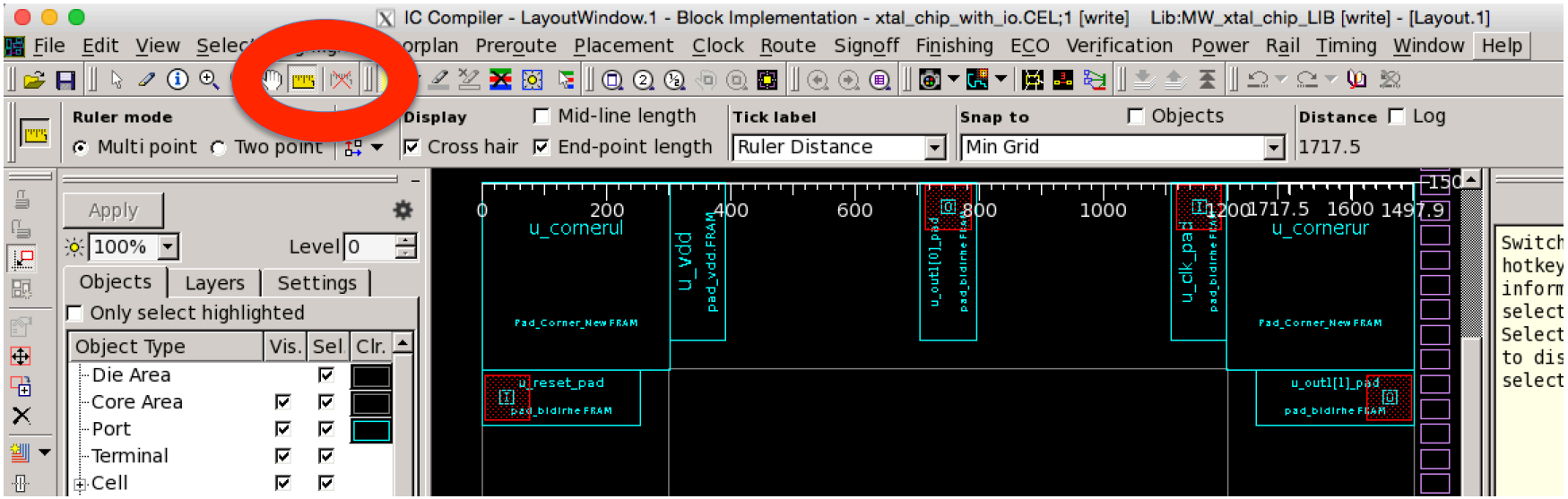
```
drwxr-xr-x.  3 grupo1 grupo1    4096 Aug  9 12:55 MW_xtal_chip_LIB
-rw-rw-r--.  1 grupo1 grupo1     546 Aug  9 12:55 set_pad_attributes_on_cell_xtal_chip_with_io.tcl
-rw-rw-r--.  1 grupo1 grupo1  142846 Aug  9 13:00 command.log
drwxr-xr-x. 15 grupo1 grupo1    4096 Aug  9 13:00 .
-rw-rw-r--.  1 grupo1 grupo1   41310 Aug  9 13:00 icc_output.txt
[grupo1@200 pnr]$ cp set
set_pad_attributes_on_cell_xtal_chip_with_io.tcl  set_pads.tcl
[grupo1@200 pnr]$ cp set_pad_attributes_on_cell_xtal_chip_with_io.tcl is_pad.tcl
[grupo1@200 pnr]$
```

IC Compiler – First return – Floor Plan



IC Compiler – First return – Floor Plan

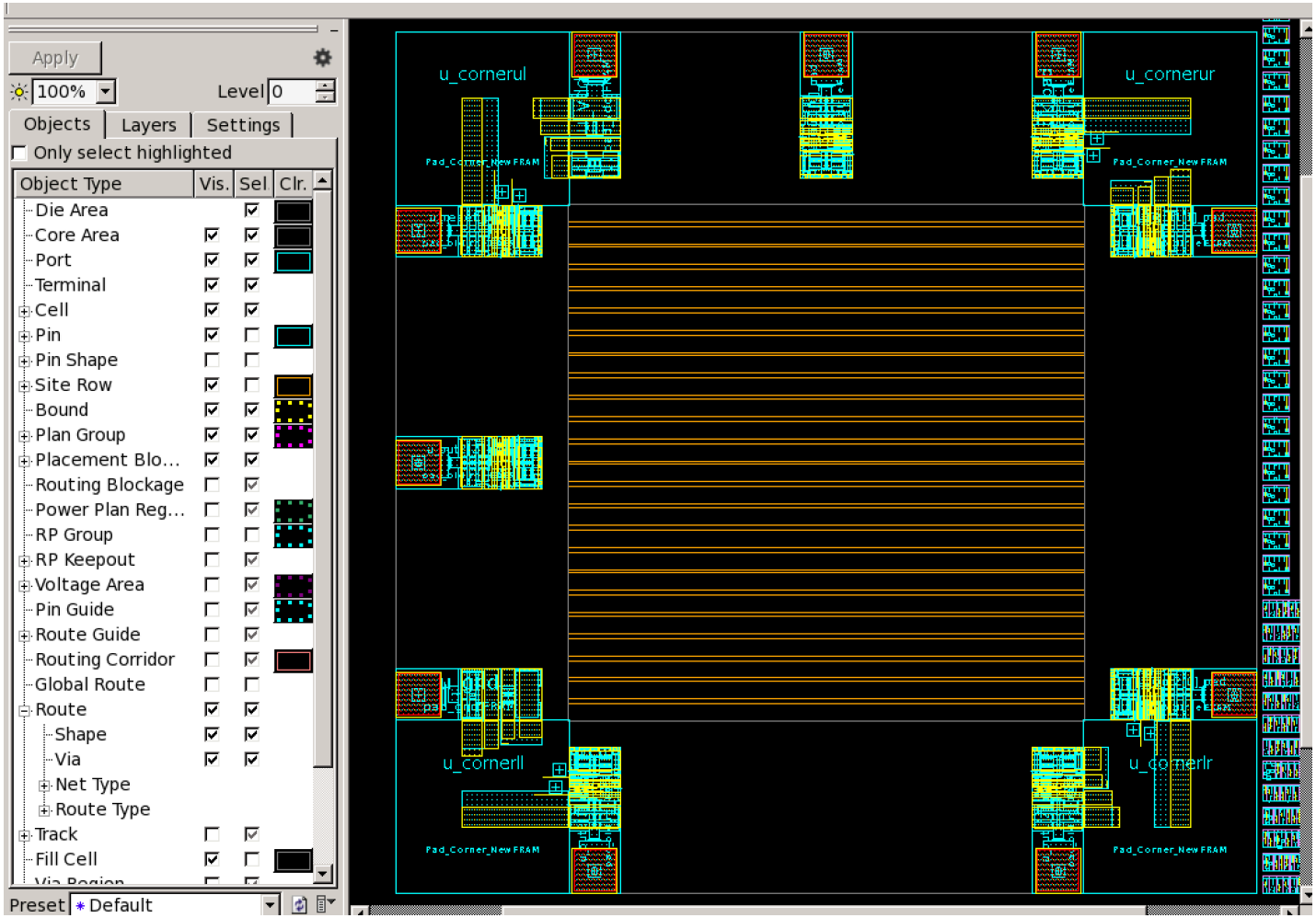
Use the ruler to check sizes of the die. 1500 microns is the target size for each side.



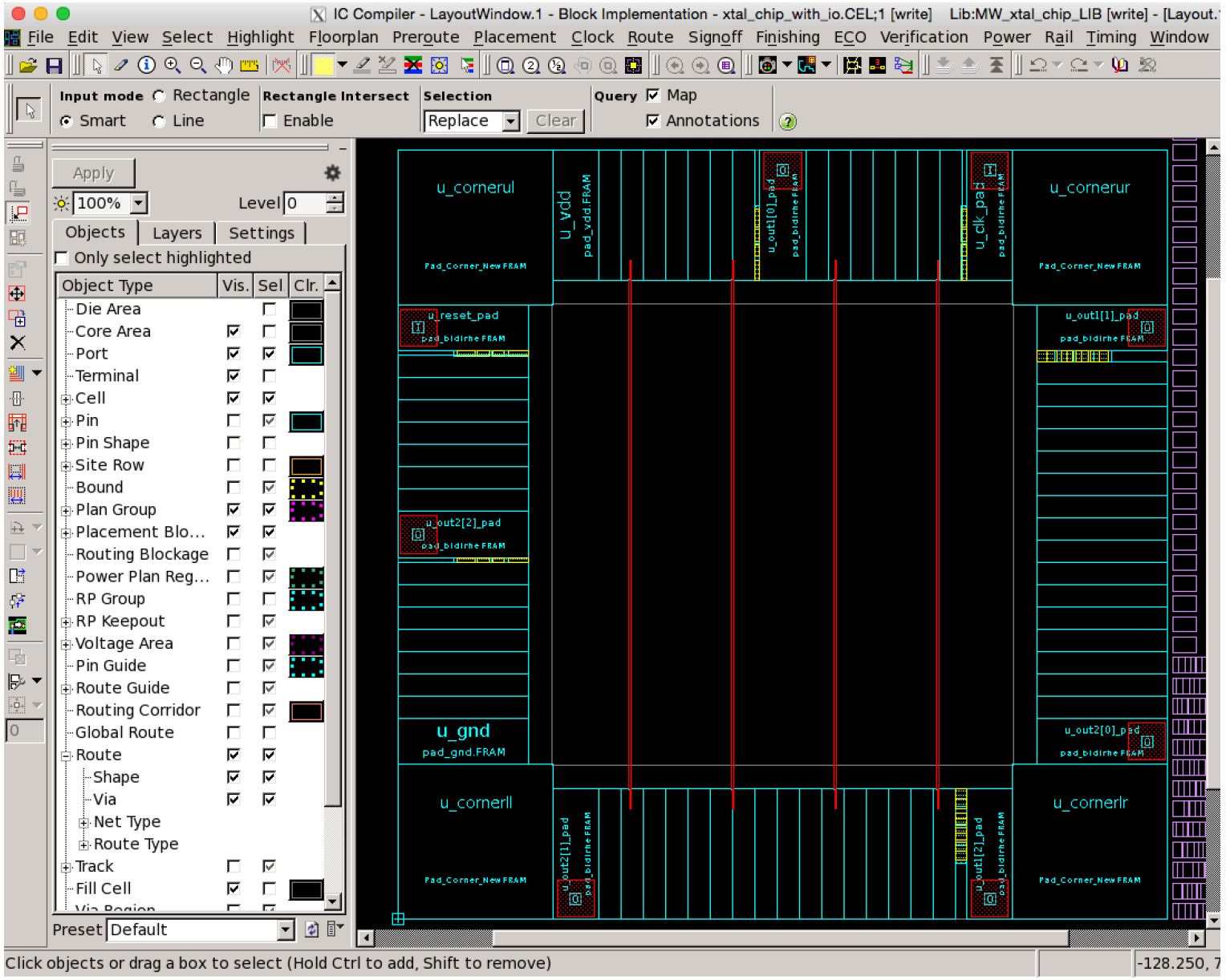
IC Compiler – First return – Floor Plan

Change the visibility selections to see different layers. Pins and Site rows to see metals and where the standard cells will be placed later.
Note that all unplaced cells are to the right – out of bounds for now.

Final Report: Add screen shot to final report.

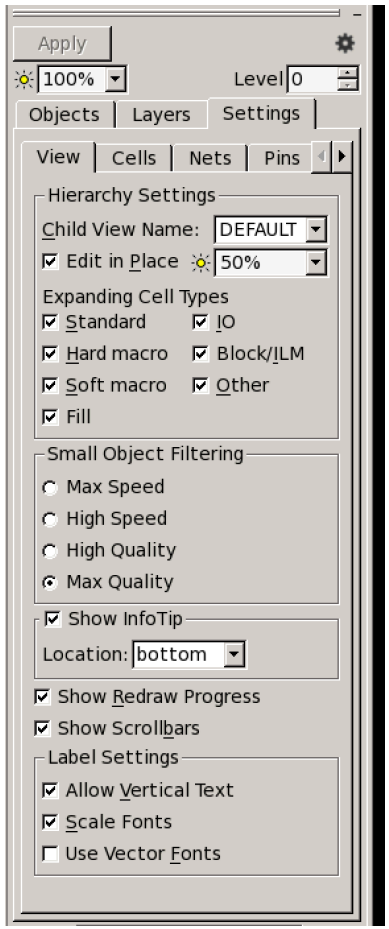


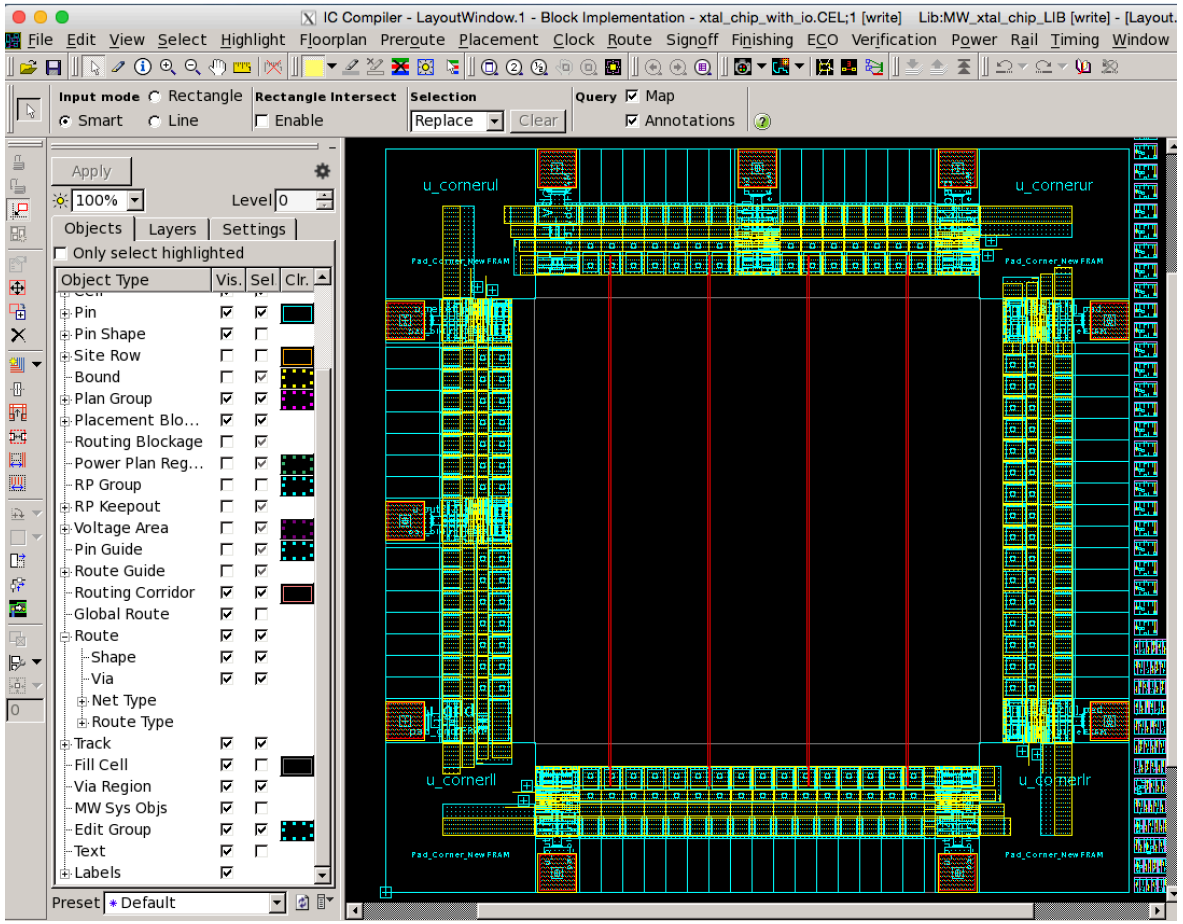
IC Compiler - Second return – Power and IO Fill



IC Compiler - Second return – Power and IO Fill

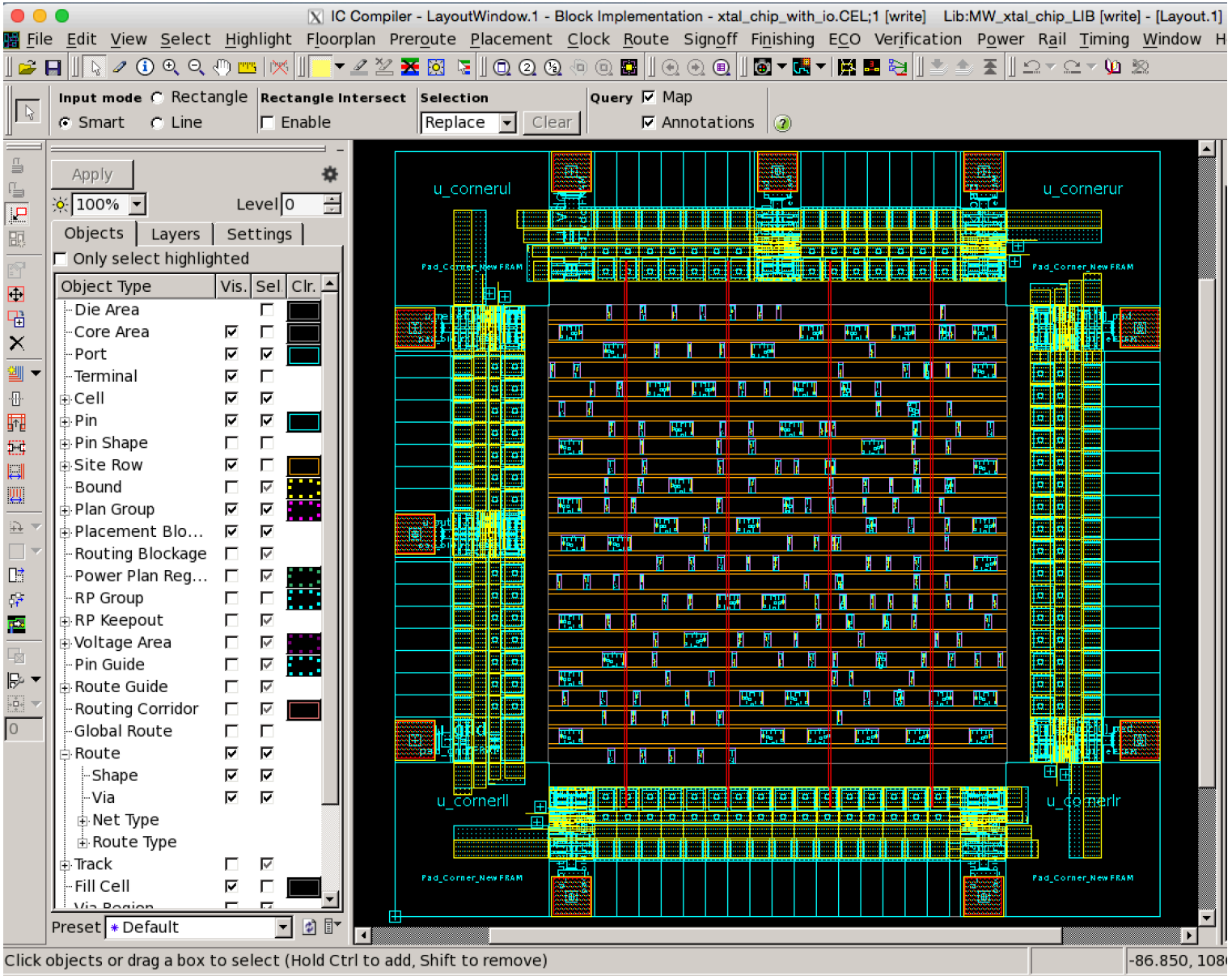
Should see power lines at all three levels and ring with vertical straps. IO Ring is complete but you must set the visual settings to “Max Quality”. Change levels to 1 or 2 to see into lower hierarchical levels. May need to change color of poly to see transistors.





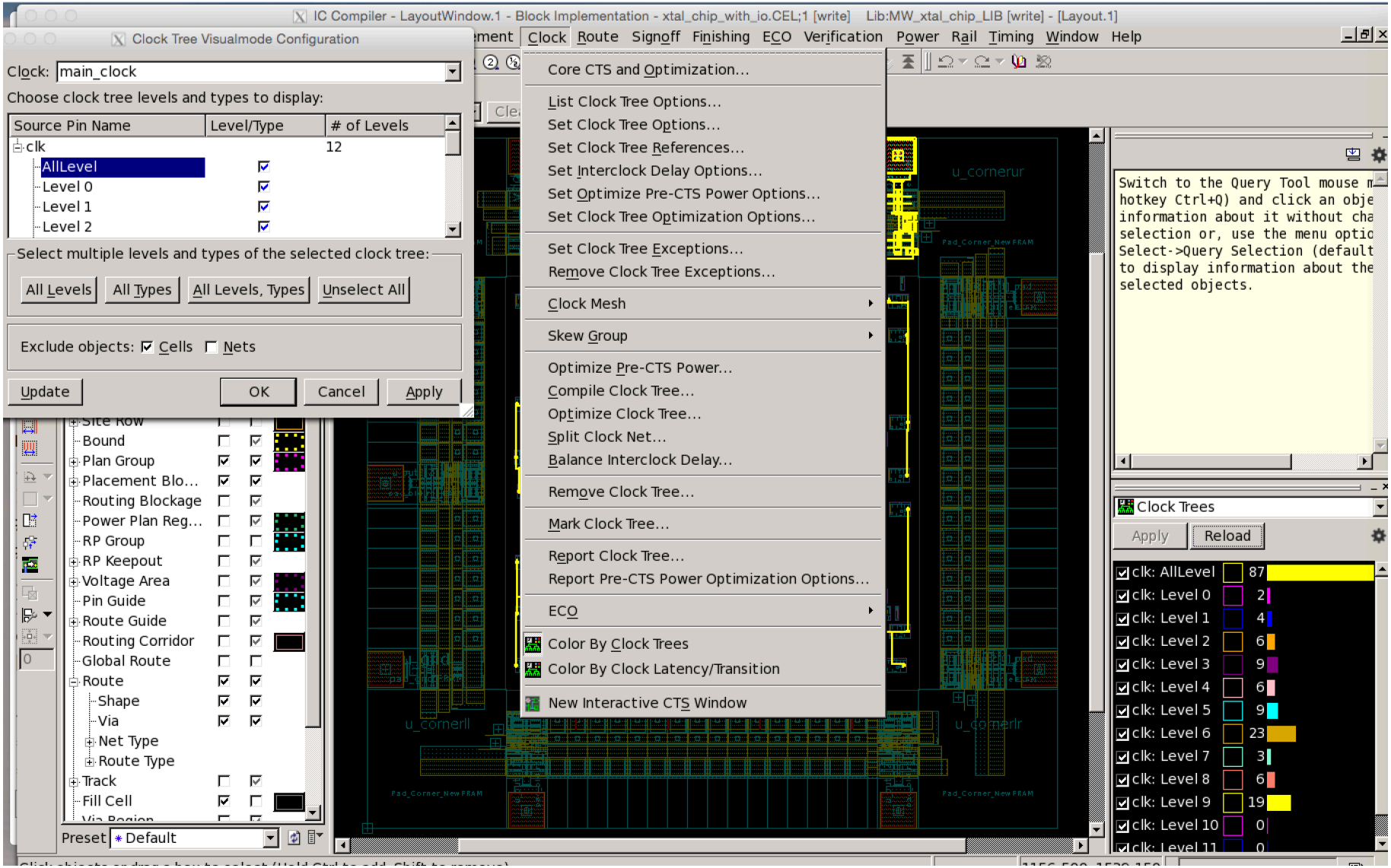
IC Compiler - Third Return – Placement of Cells

Final Report: Add screen shot to final report.

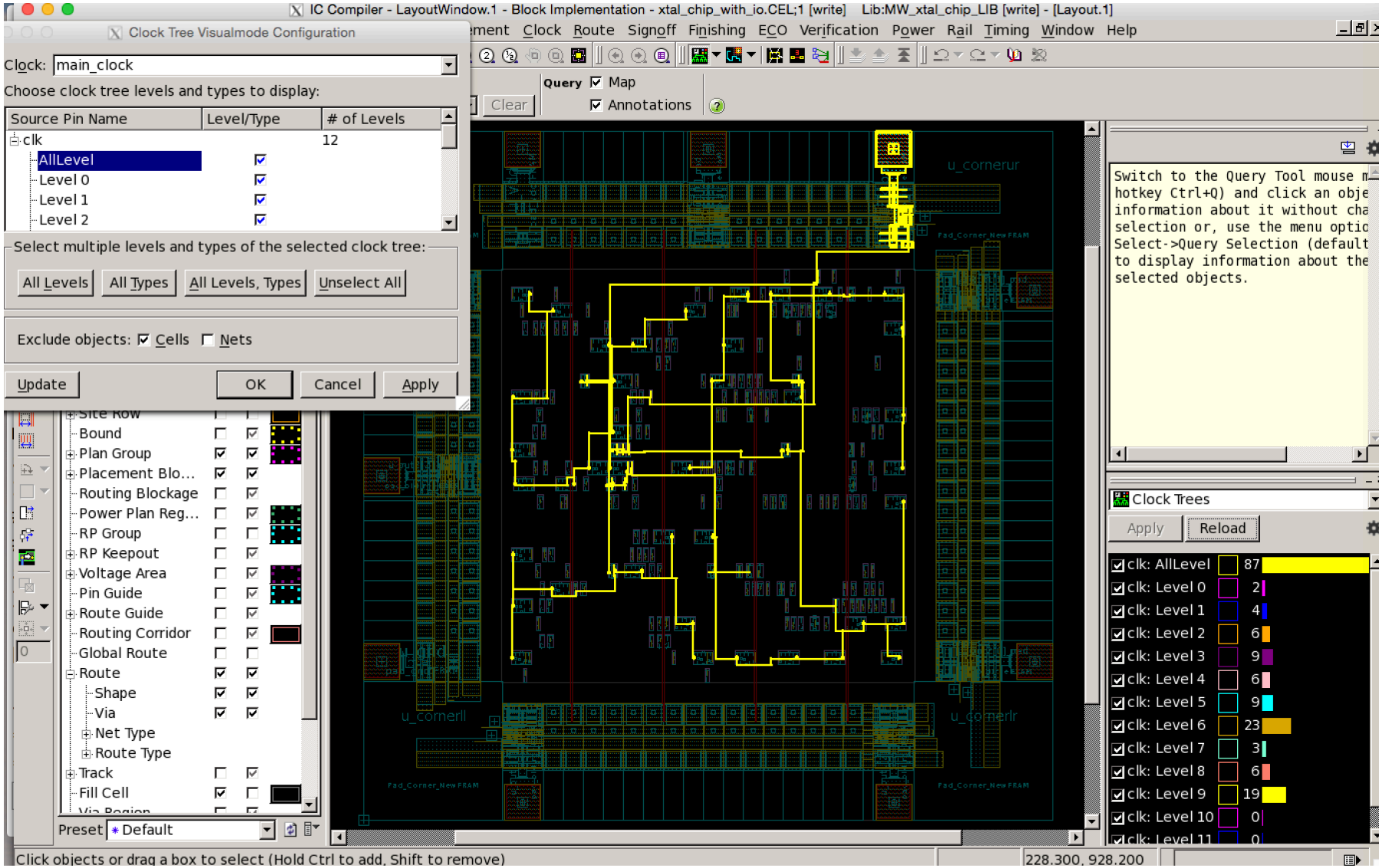


IC Compiler – Fourth Return - Clock Tree

After running placement, invoke “Report Clock Tree”
Use “Color By Clock Trees” to see clock path



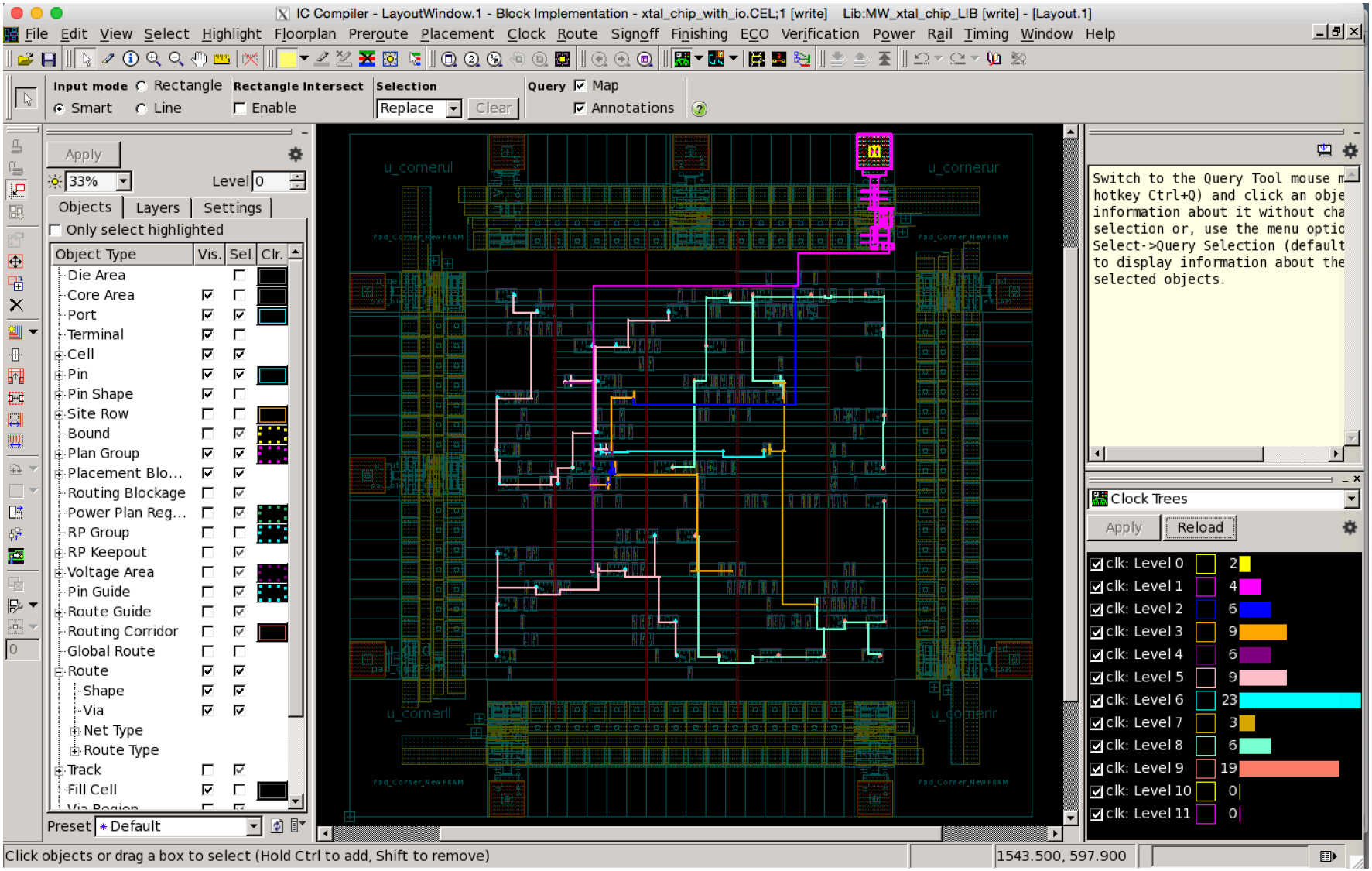
IC Compiler – Fourth Return - Clock Tree



IC Compiler – Fourth Return - Clock Tree

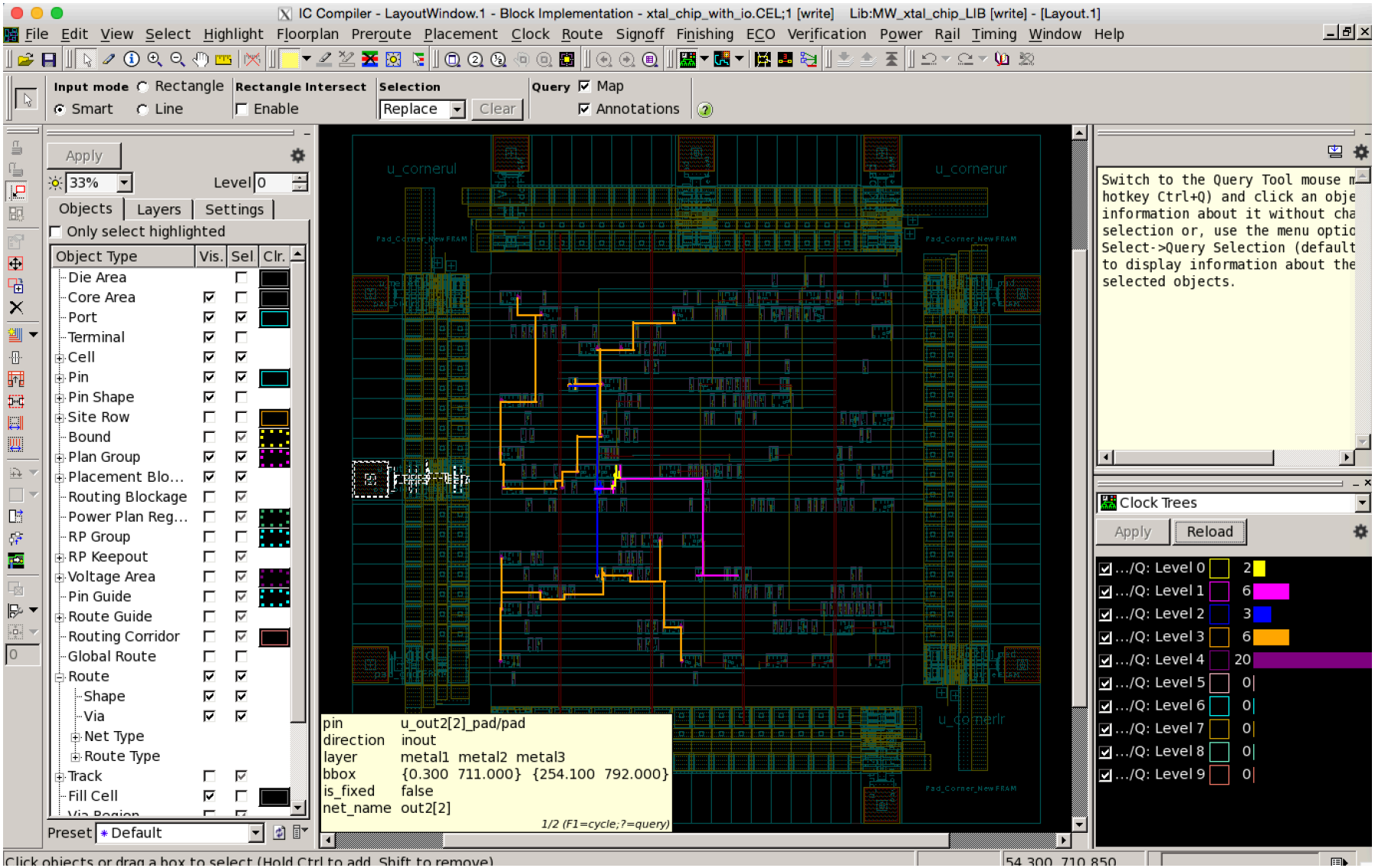
Selecting main_clock

Final Report: Add screen shot to final report.



IC Compiler – Fourth Return - Clock Tree

Selecting “secondary_clock” – Note no IO included in this internal clock. Note: this design had a second clock but you may not have one.



IC Compiler – Fourth Return - Clock Tree

Around 0.2 ns of skew is considered acceptable. However, skew isn't a problem if you pass hold timings later.

========== Clock Tree Summary ==========

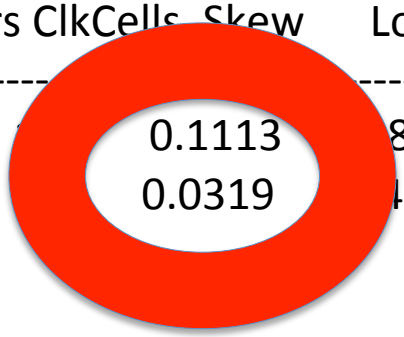
Clock	Sinks	CTBuffers	ClkCells	Skew	LongestPath	TotalDRC	BufferArea
main_clock	39	12	14	0.0493	2.0112	0	4536.0000
second_clock	20	4	4	0.0311	0.9414	0	2016.0000

Post Routing (skew increased :/):

========== Clock Tree Summary ==========

Clock	Sinks	CTBuffers	ClkCells	Skew	LongestPath	TotalDRC	BufferArea
main_clock	39	12		0.1113	813	0	4536.0000
second_clock	20	4		0.0319	401	0	2016.0000

1



Final Report: Add this output to your final report.

IC Compiler – Fourth Return - Clock Tree

Check timing report in log. Need “Met” next to slack.

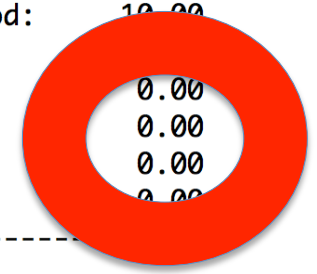
```
  U340/Y (NOR2X1)                                          0.47 *     9.33 f
  U341/Y (NAND2X1)                                         0.41 *     9.75 r
  U347/Y (NOR2X1)                                          0.48 *    10.23 f
  U348/Y (NAND2X1)                                         0.42 *    10.65 r
  U354/Y (NOR2X1)                                          0.58 *    11.22 f
  U355/Y (NAND2X1)                                         0.46 *    11.68 r
  U361/Y (NOR2X1)                                          0.44 *    12.12 f
  U362/Y (OAI21X1)                                         0.26 *    12.38 r
  U363/Y (AOI21X1)                                         0.32 *    12.70 f
  I_xtal_chip/count2_reg[19]/D (DCBX1)                     0.00 *    12.70 f
  data arrival time                                                  12.70

  clock second_clock (rise edge)                        5120.00    5120.00
  clock network delay (propagated)                         3.00    5123.00
  I_xtal_chip/count2_reg[19]/CLK (DCBX1)                   0.00    5123.00 r
  library setup time                                      -0.40    5122.60
  data required time                                               5122.60
  --------------------------------------------------------------------------
  data required time                                               5122.60
  data arrival time                                                 -12.70
  --------------------------------------------------------------------------
  slack (MET)                                                      5109.90


1
# IPD432 check for your clock tree and use the clock tree tools to visualize the skew in your design
start_gui
```

IC Compiler – Fourth Return - Clock Tree

Check QoR report for hold violations after the tree.

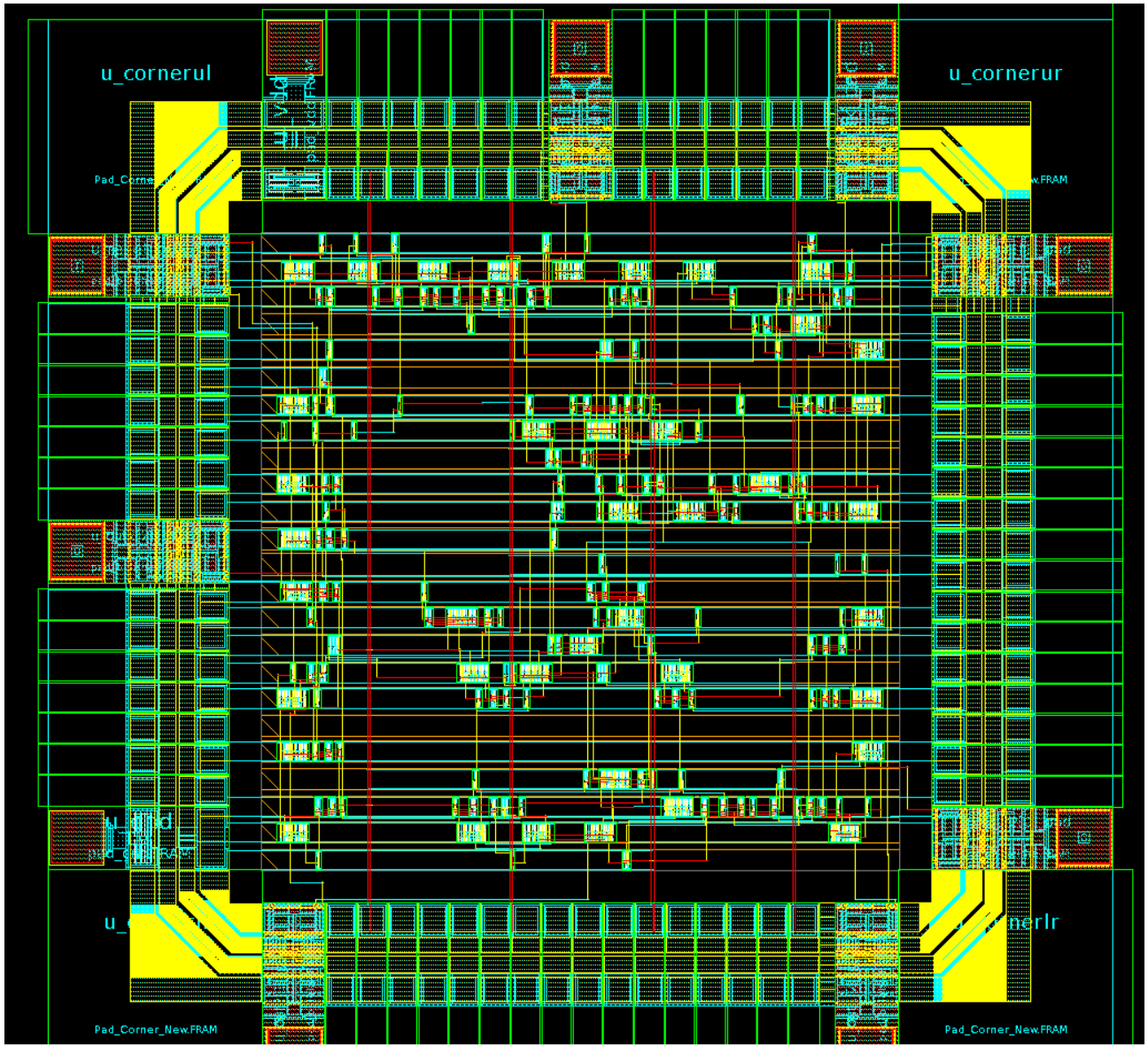
Note that hold violations are only possible after a clock tree has been synthesized as clock skew causes hold violations. Previous synthesis timing reports only performed setup violations checks.

```
****************************************
Report : qor
Design : xtal_chip_with_io
Version: I-2013.12-SP5-2
Date   : Wed Aug 12 10:08:17 2015
****************************************

  Timing Path Group 'clk'
  -----------------------------------
  Levels of Logic:                22.00
  Critical Path Length:            8.42
  Critical Path Slack:             1.19
  Critical Path Clk Period:       10.00
  Total Negative Slack:
  No. of Violating Paths:          0.00
  Worst Hold Violation:            0.00
  Total Hold Violation:            0.00
  No. of Hold Violations:          0.00
  -----------------------------------

  Timing Path Group 'generated_clk'
  -----------------------------------
  Levels of Logic:                20.00
  Critical Path Length:            8.50
  Critical Path Slack:         10231.11
  Critical Path Clk Period:    10240.00
  Total Negative Slack:            0.00
  No. of Violating Paths:          0.00
  Worst Hold Violation:            0.00
  Total Hold Violation:            0.00
  No. of Hold Violations:          0.00
  -----------------------------------
```

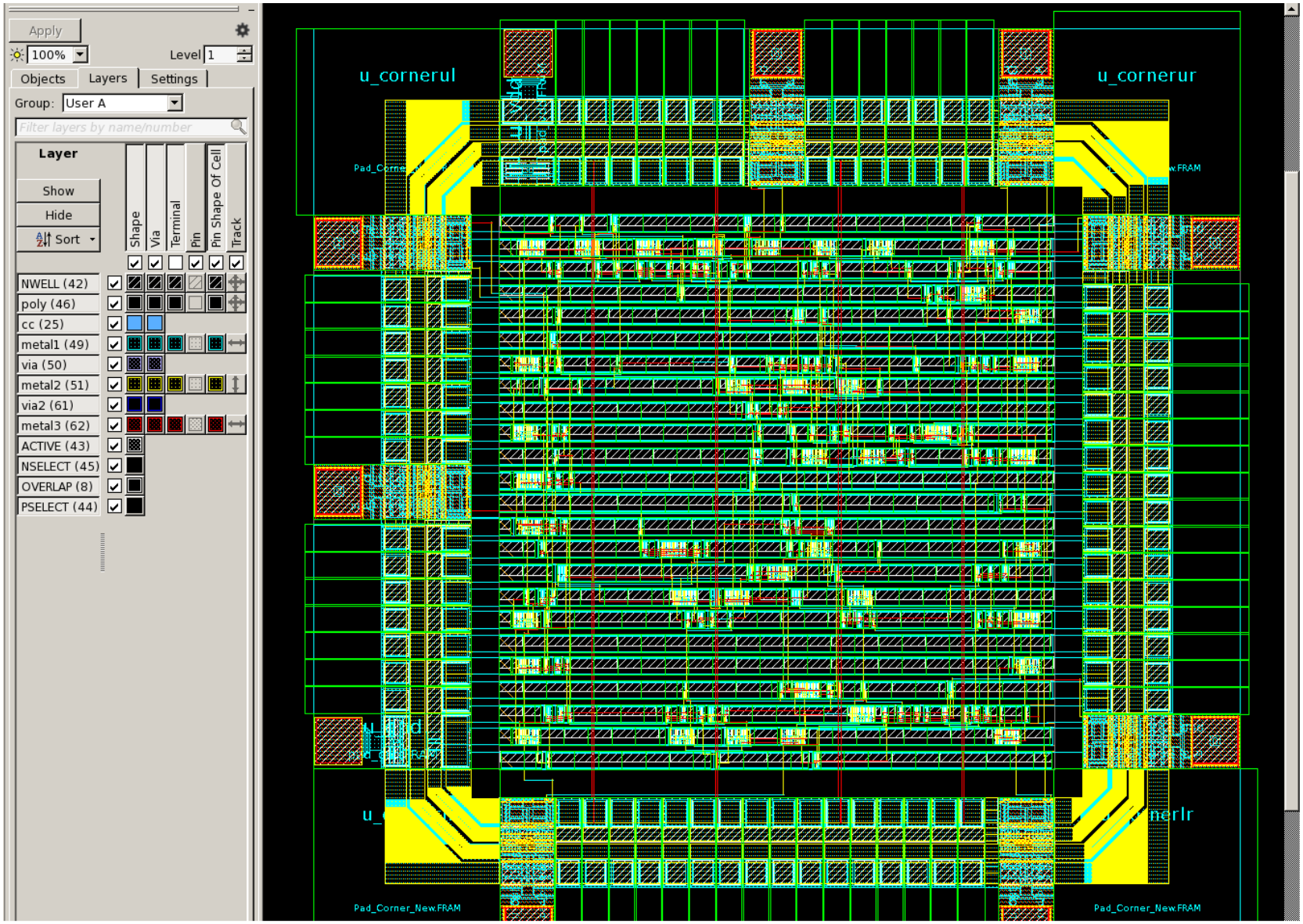


IC Compiler – Fifth Return - Routing

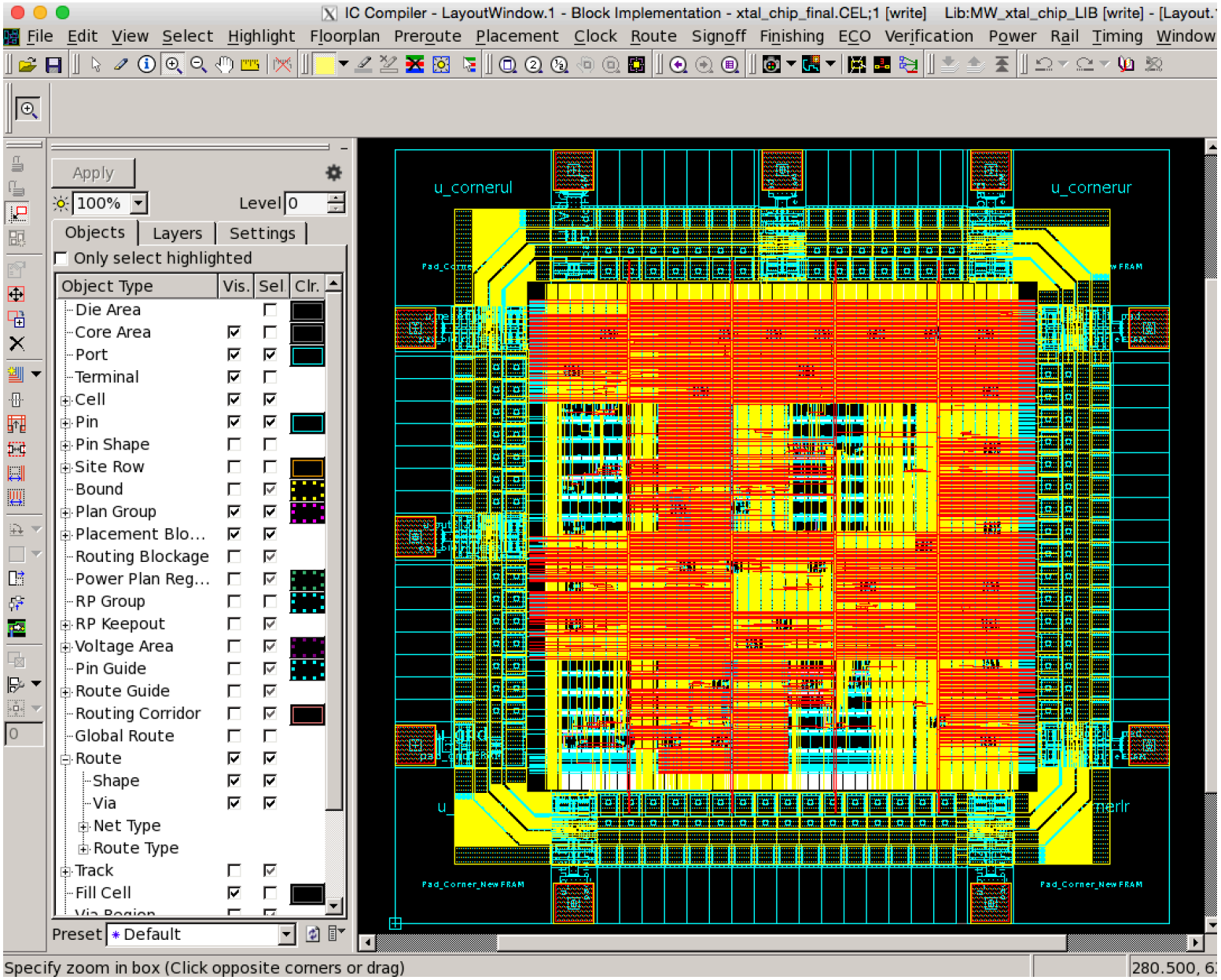
Final Report: Add screen shot to final report.



IC Compiler – Well filler



IC Compiler – Metal Density Fill



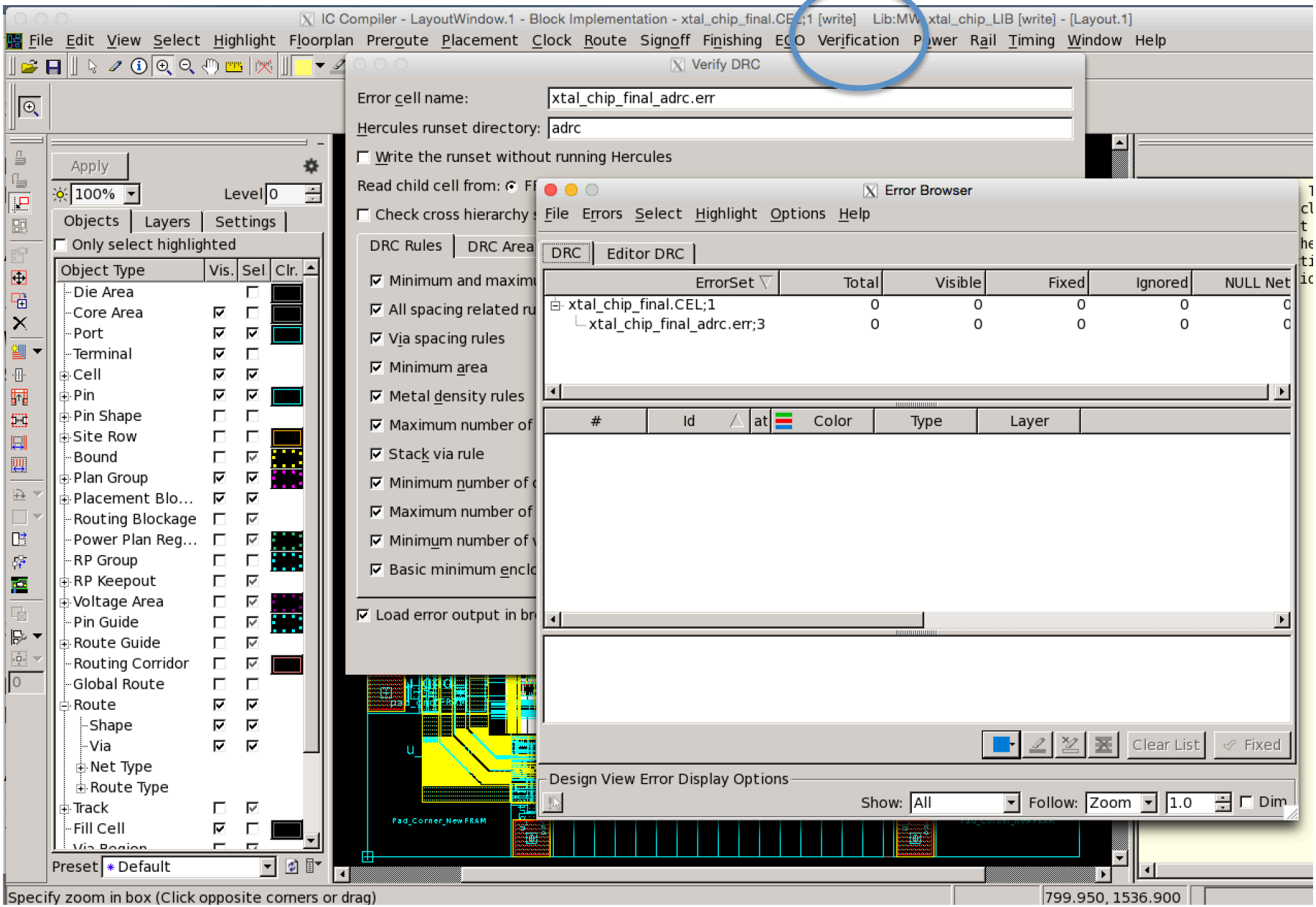
IC Compiler – Final Check Off

```
verify_pg_nets

Checking [gnd!]:
        There are no floating shapes
        All the pins are connected.
        No errors are found.
Checking [vdd!]:
        There are no floating shapes
        All the pins are connected.
        No errors are found.
Checked 2 nets, 0 have Errors
Update error cell ...
1

verify_lvs

** Total Floating ports are 14.
** Total Floating Nets are 0.
** Total OPEN Nets are 0.
** Total Electrical Equivalent Error are 0.
** Total Must Joint Error are 0.
```

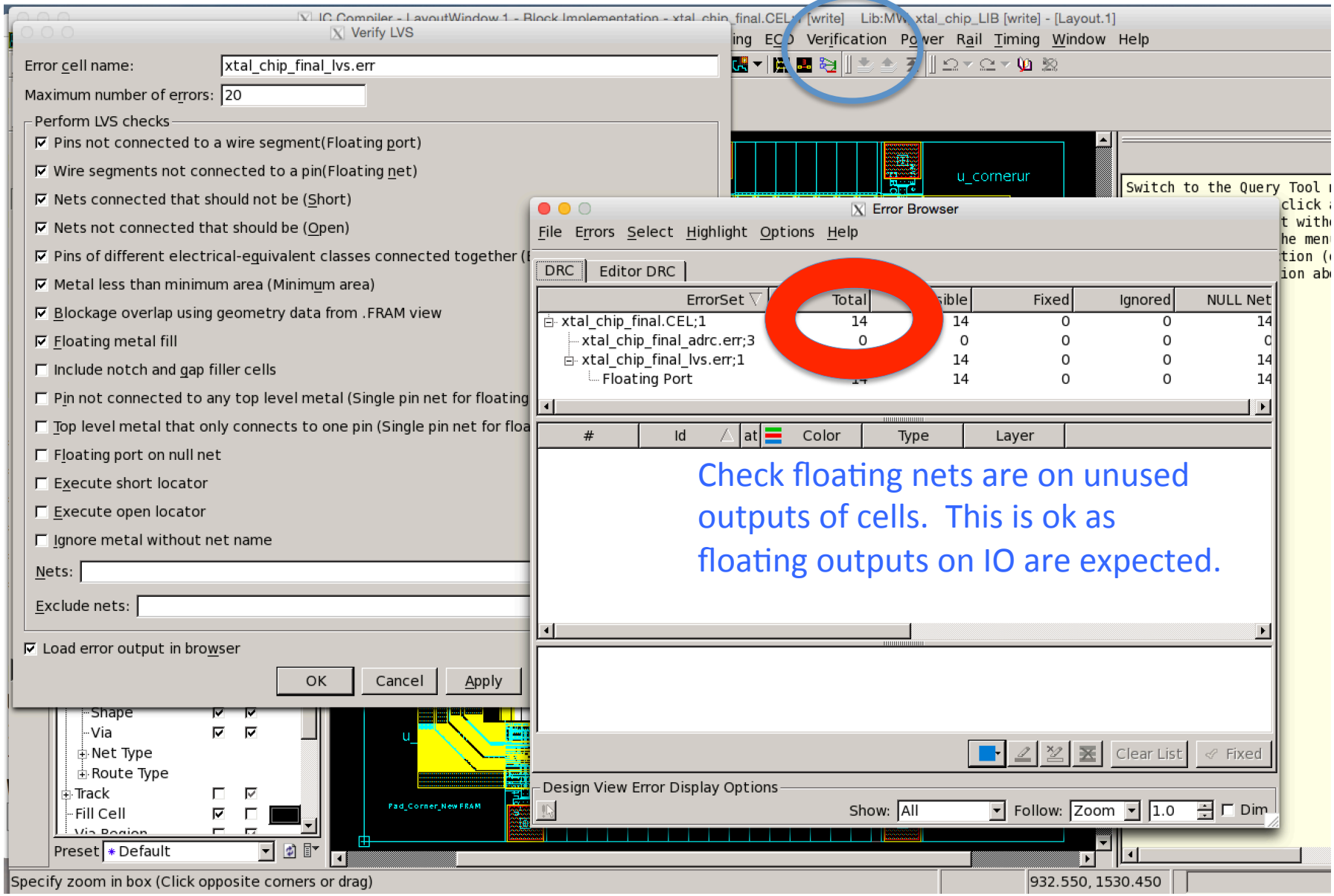
Check floating nets are on unused outputs of cells. This is ok. IOs will almost always have floating outputs.

Check the reports directory for timing and QoR reports for passing all timing checks including holds.

IC Compiler – Final Check Off

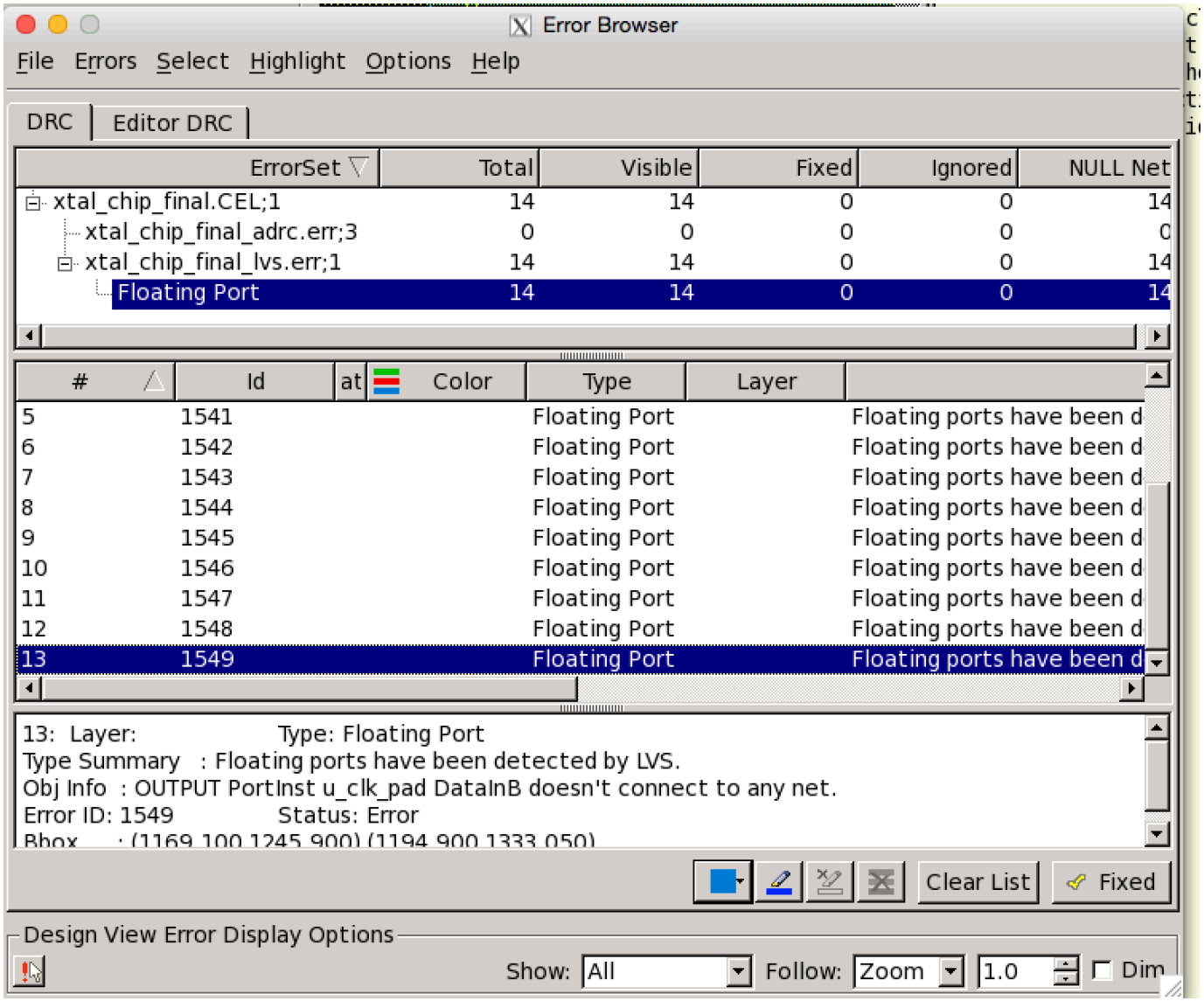


IC Compiler – Final Check Off



IC Compiler – Final Check Off

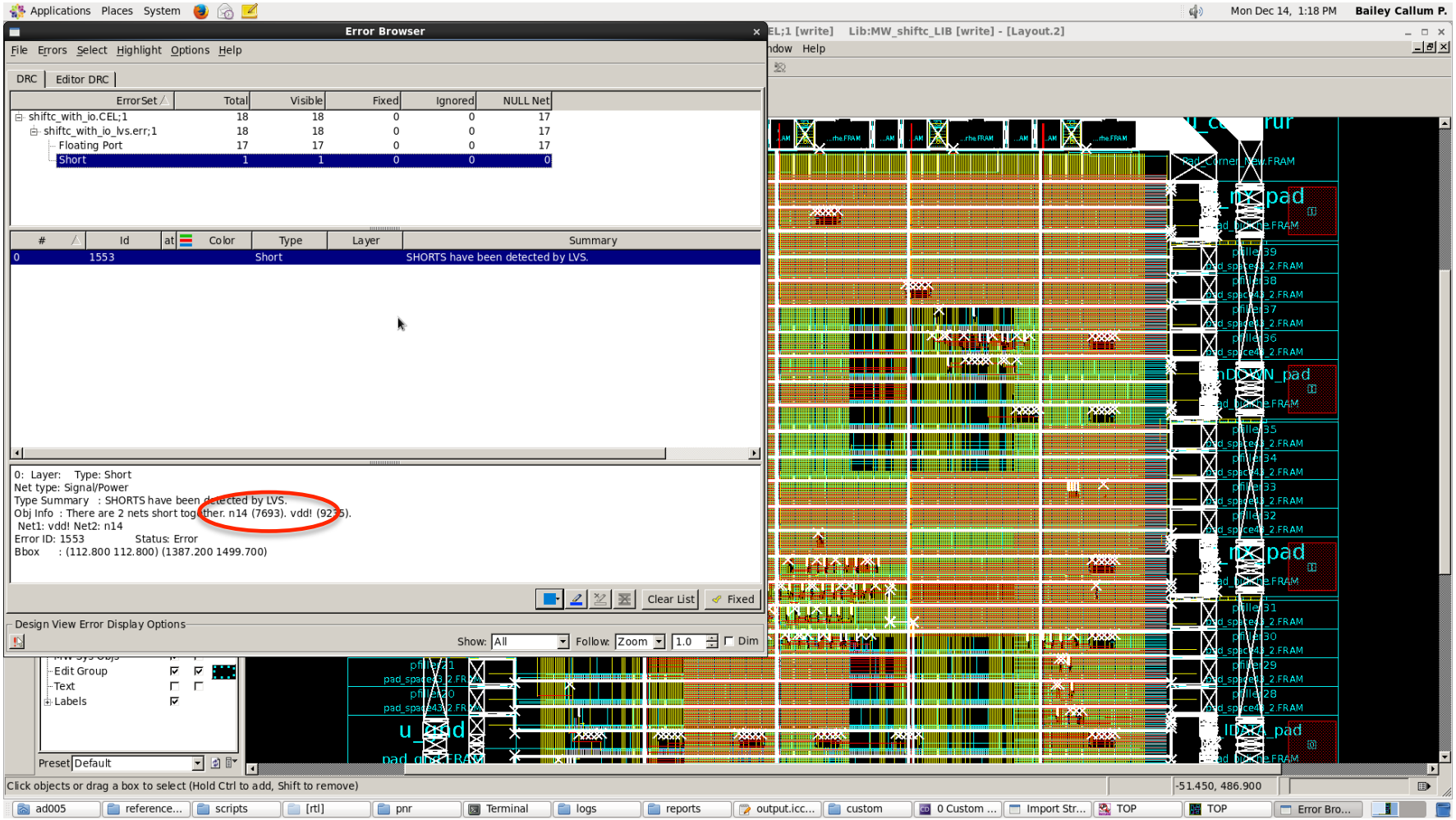
Data_in and Data_inB from IO must float if they are output pins. Data_inB will likely not be used even for inputs.



IC Compiler – Fixing a short in ICC

If LVS returns a short error, you will have to fix the problem.

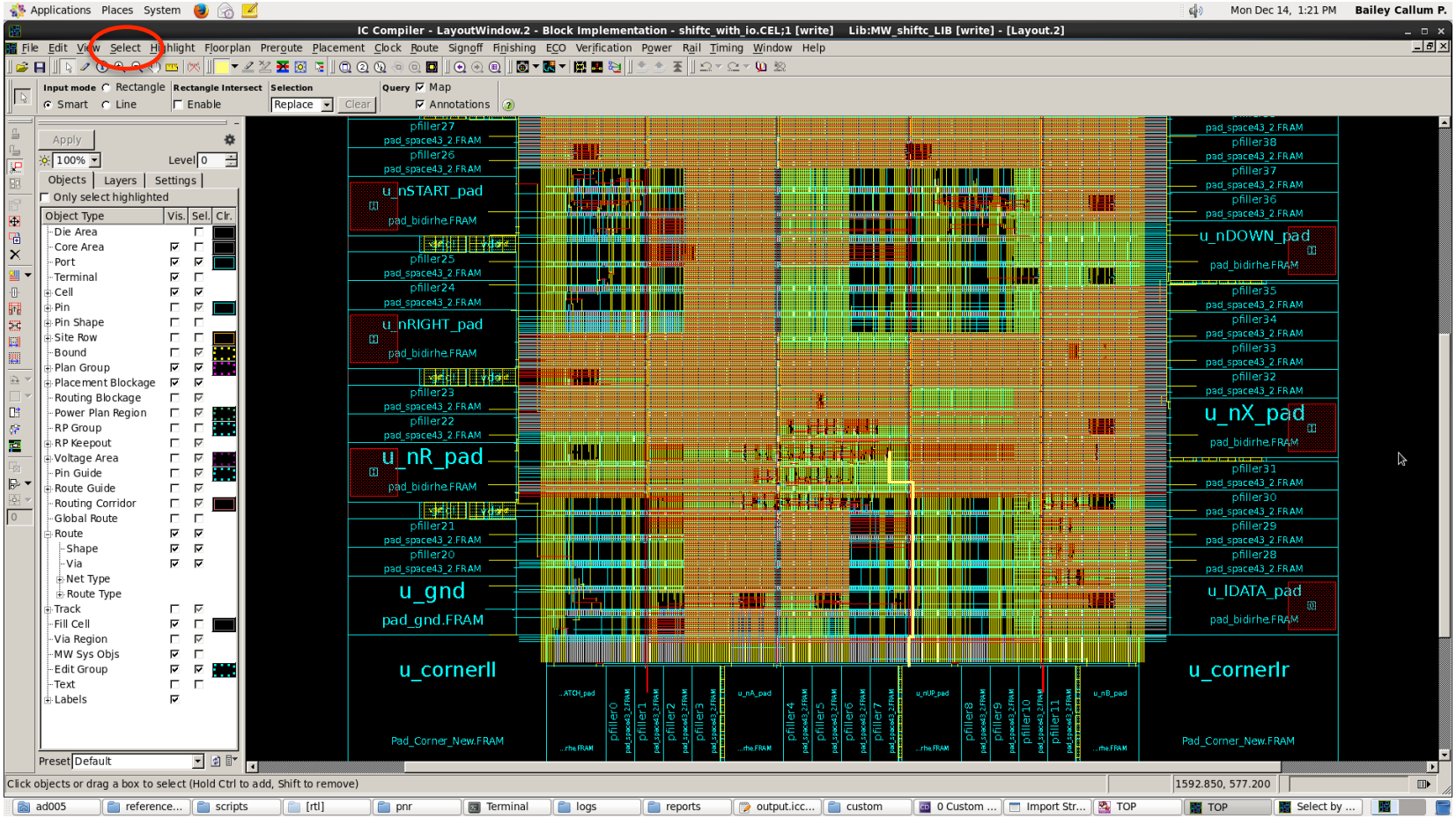
Here Net n14 is causing a short with vdd



IC Compiler – Fixing a short in ICC

Under select tab you should find a “select by name” field.

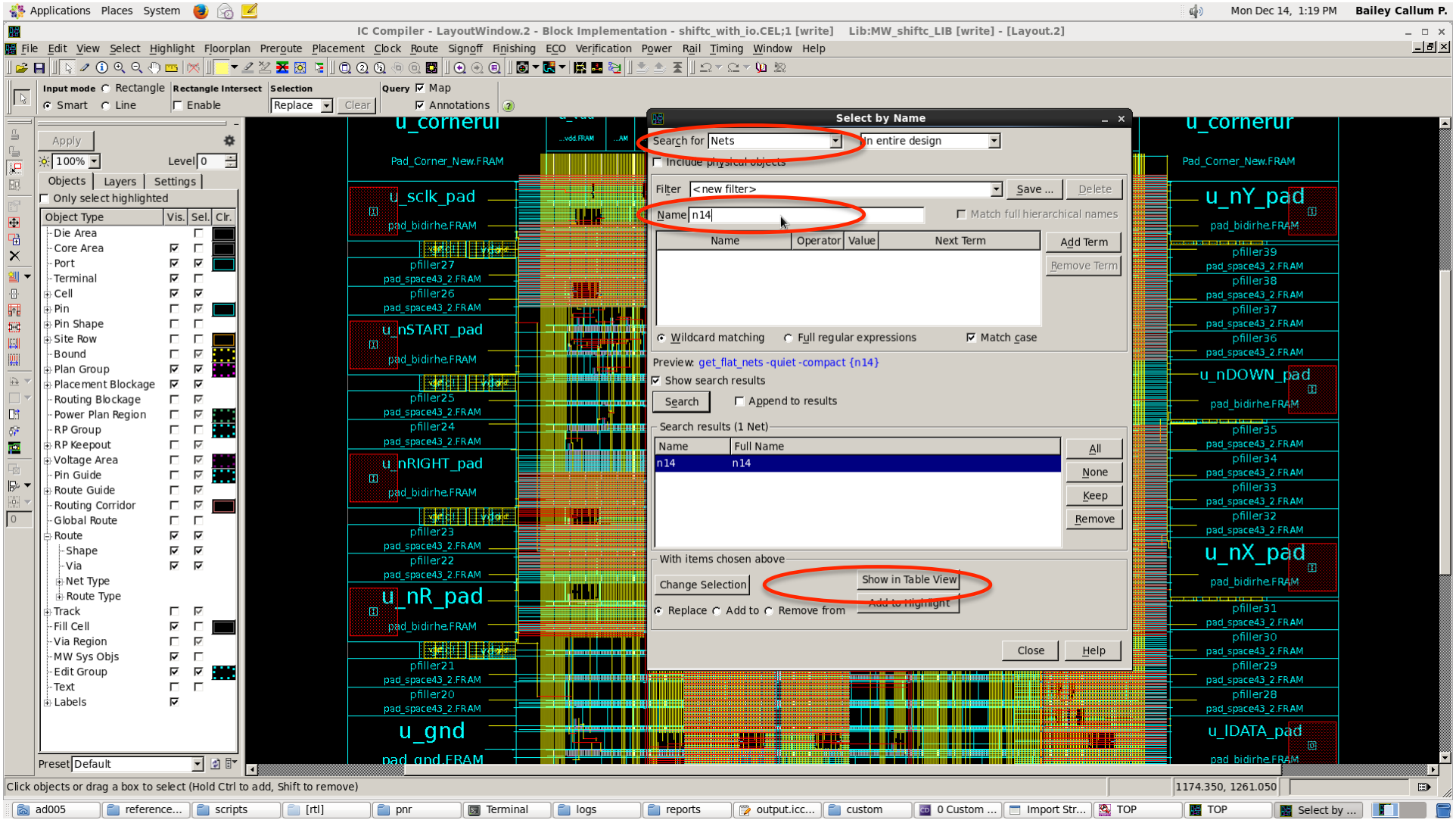
Click to search for net causing the short.



IC Compiler – Fixing a short in ICC

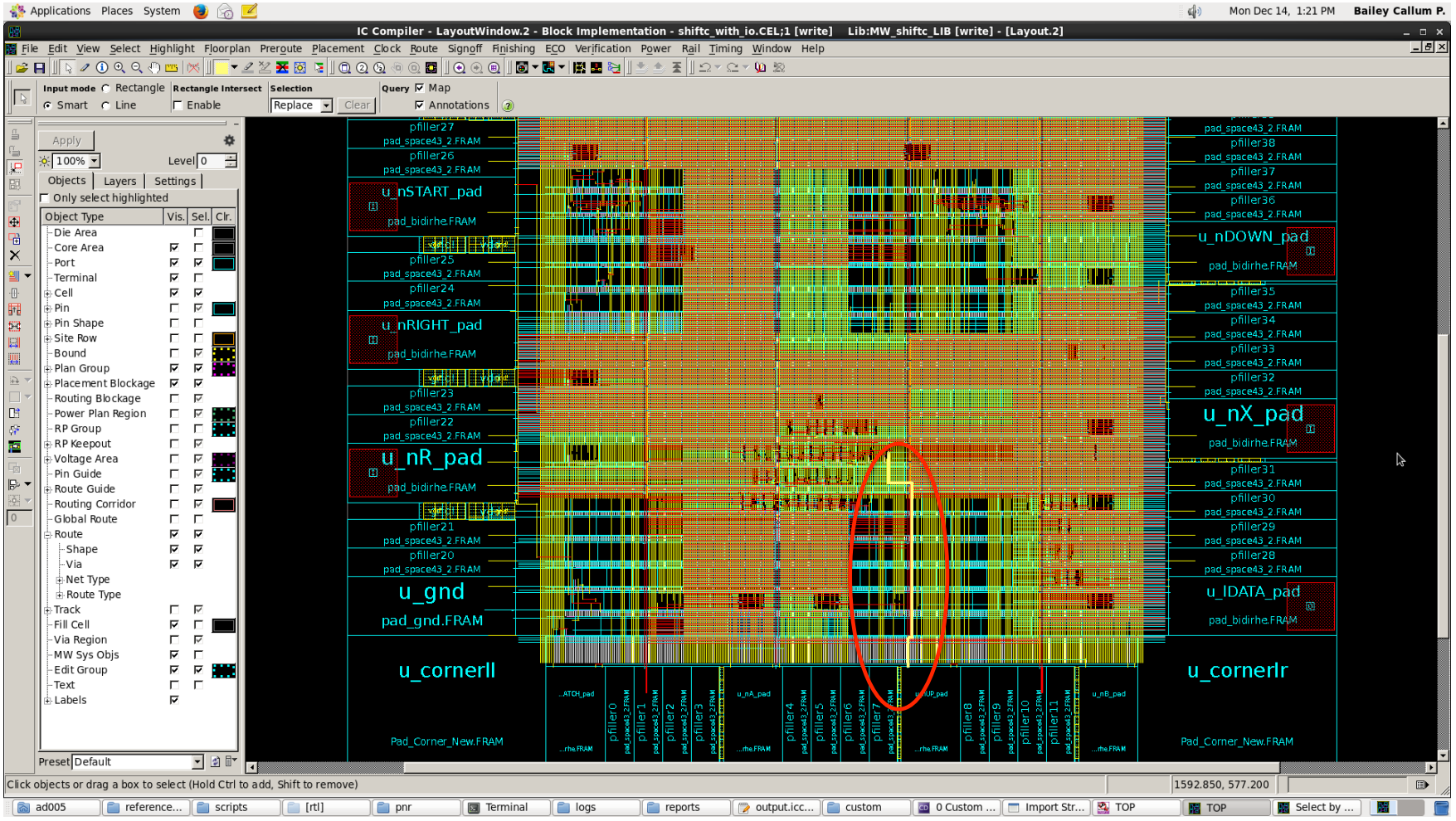
Fill in the name – here it is n14.

Click Add to Highlight.



IC Compiler – Fixing a short in ICC

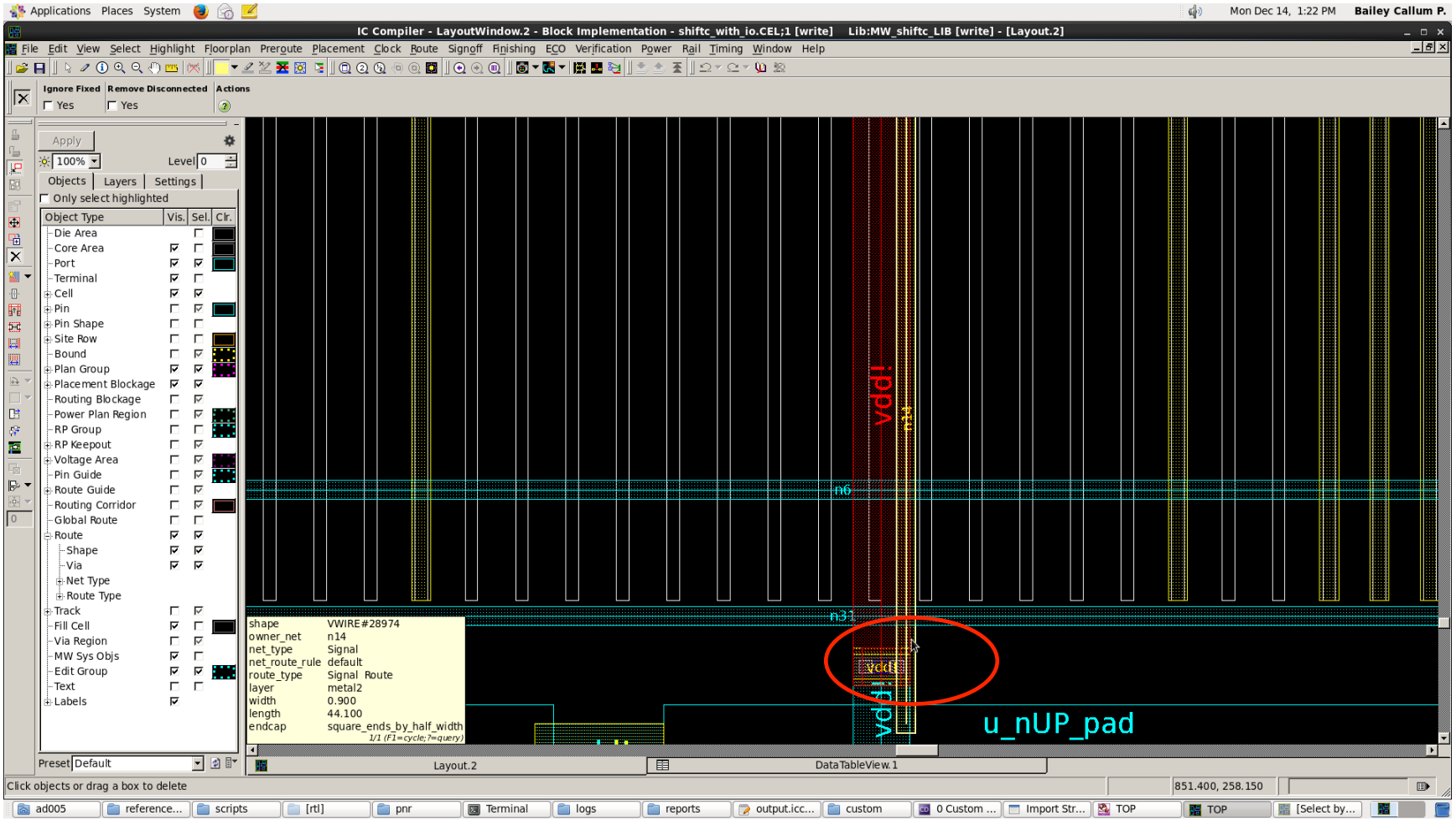
The net should appear highlighted



IC Compiler – Fixing a short in ICC

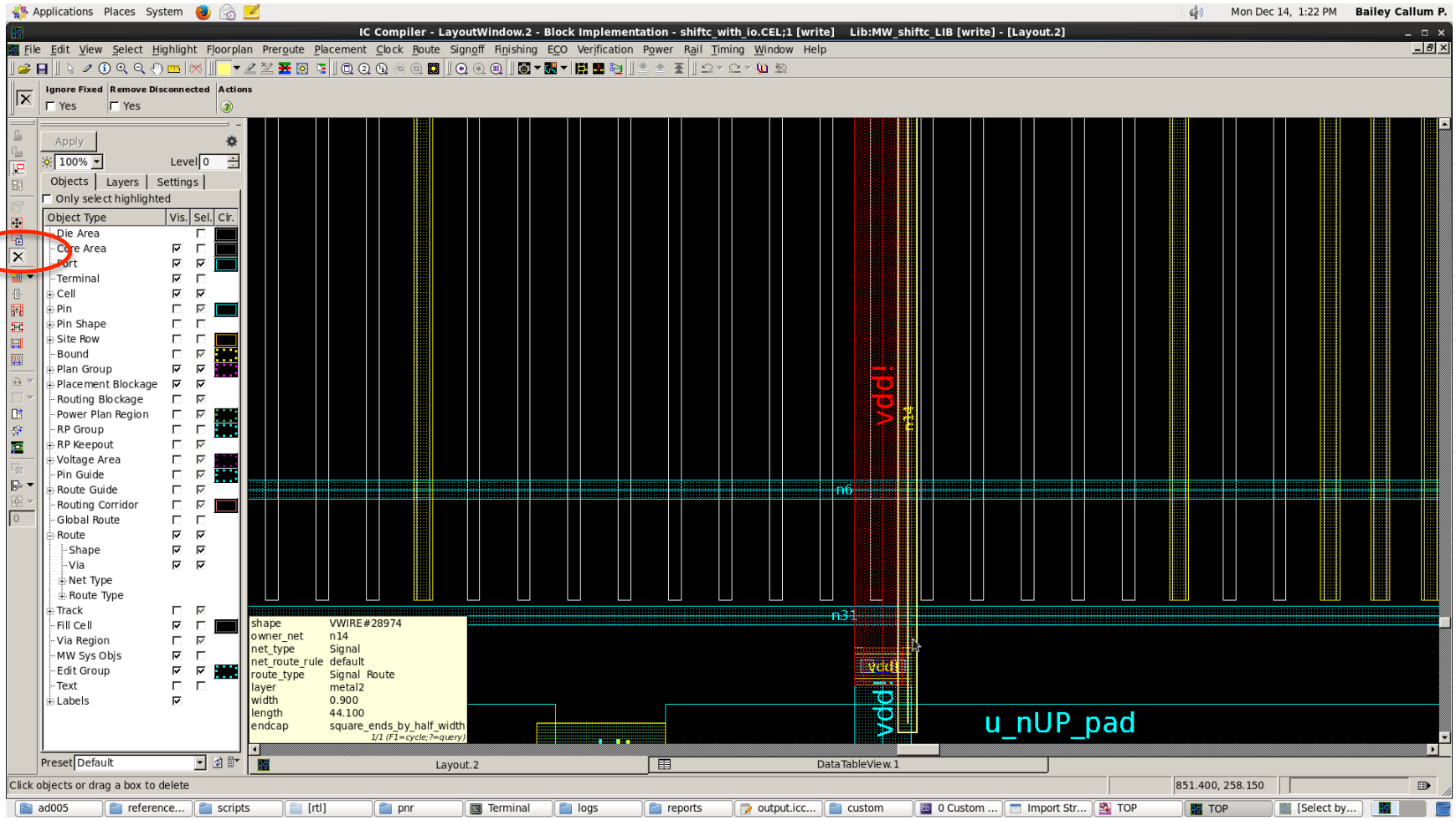
Zoom in to see the net and search for short.

Net n14 is causing a short with vdd on Metal 3 layer (yellow).



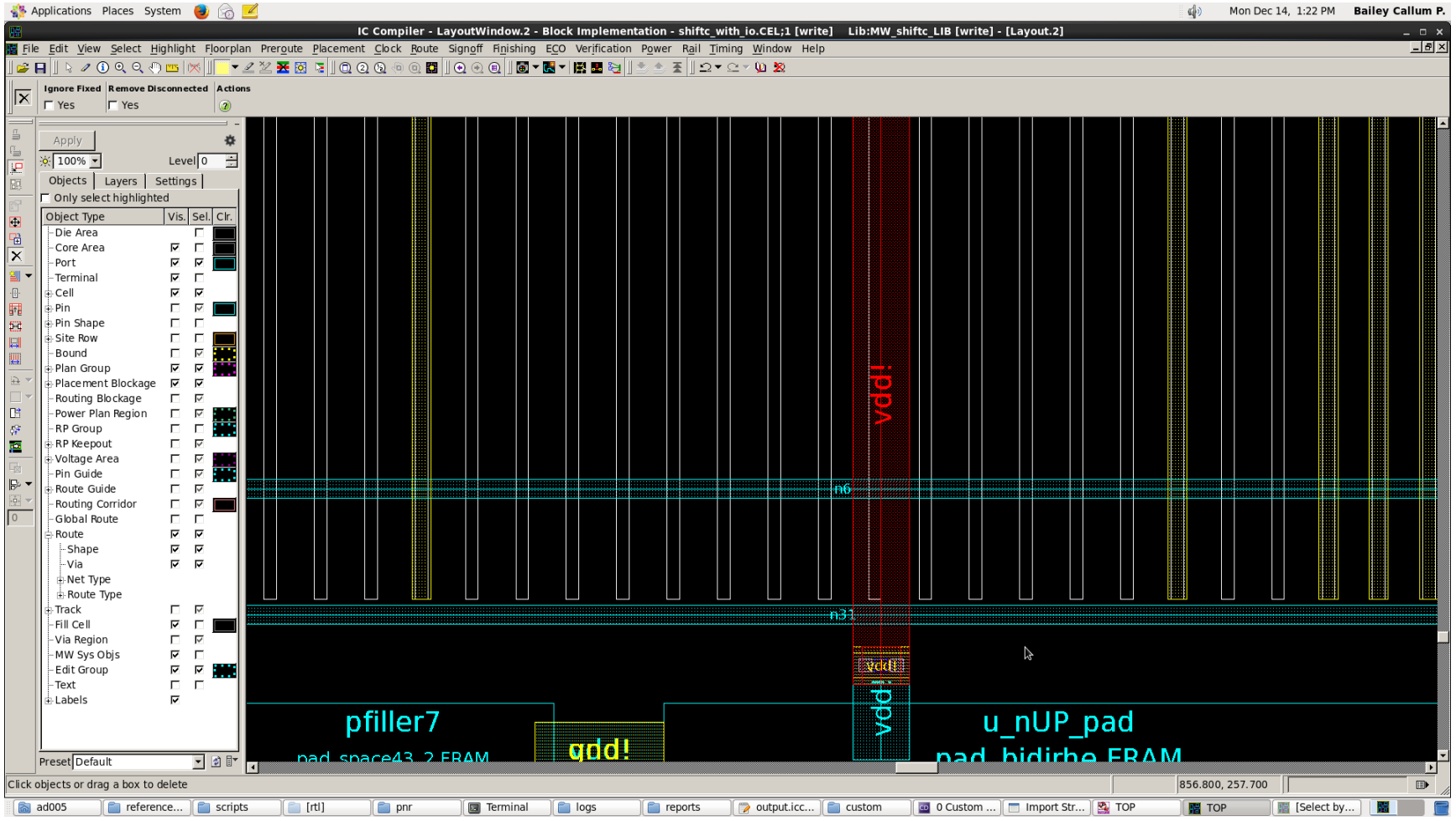
IC Compiler – Fixing a short in ICC

Select delete tool and delete the Net n14 by selecting.



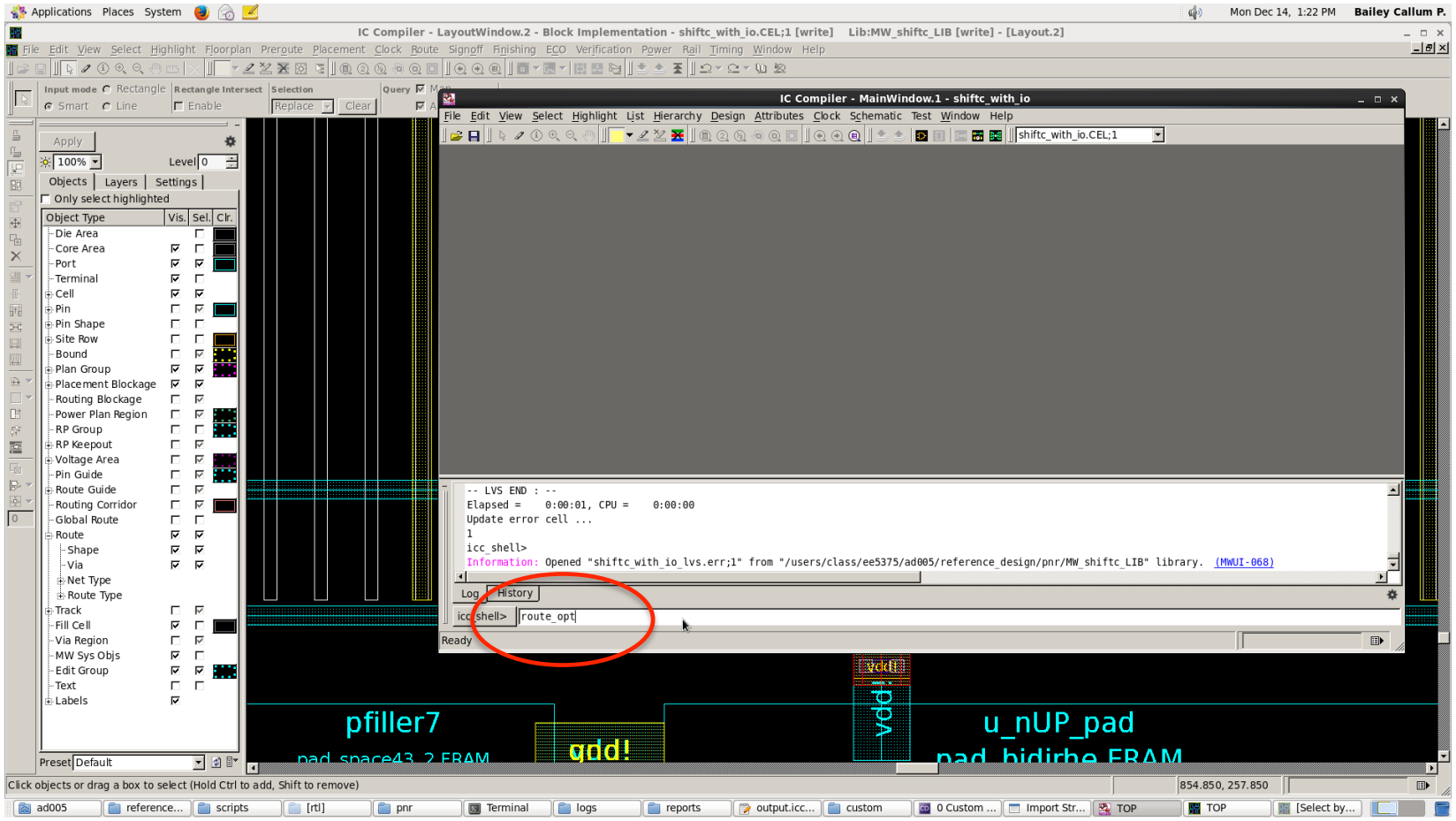
IC Compiler – Fixing a short in ICC

Net has been deleted.



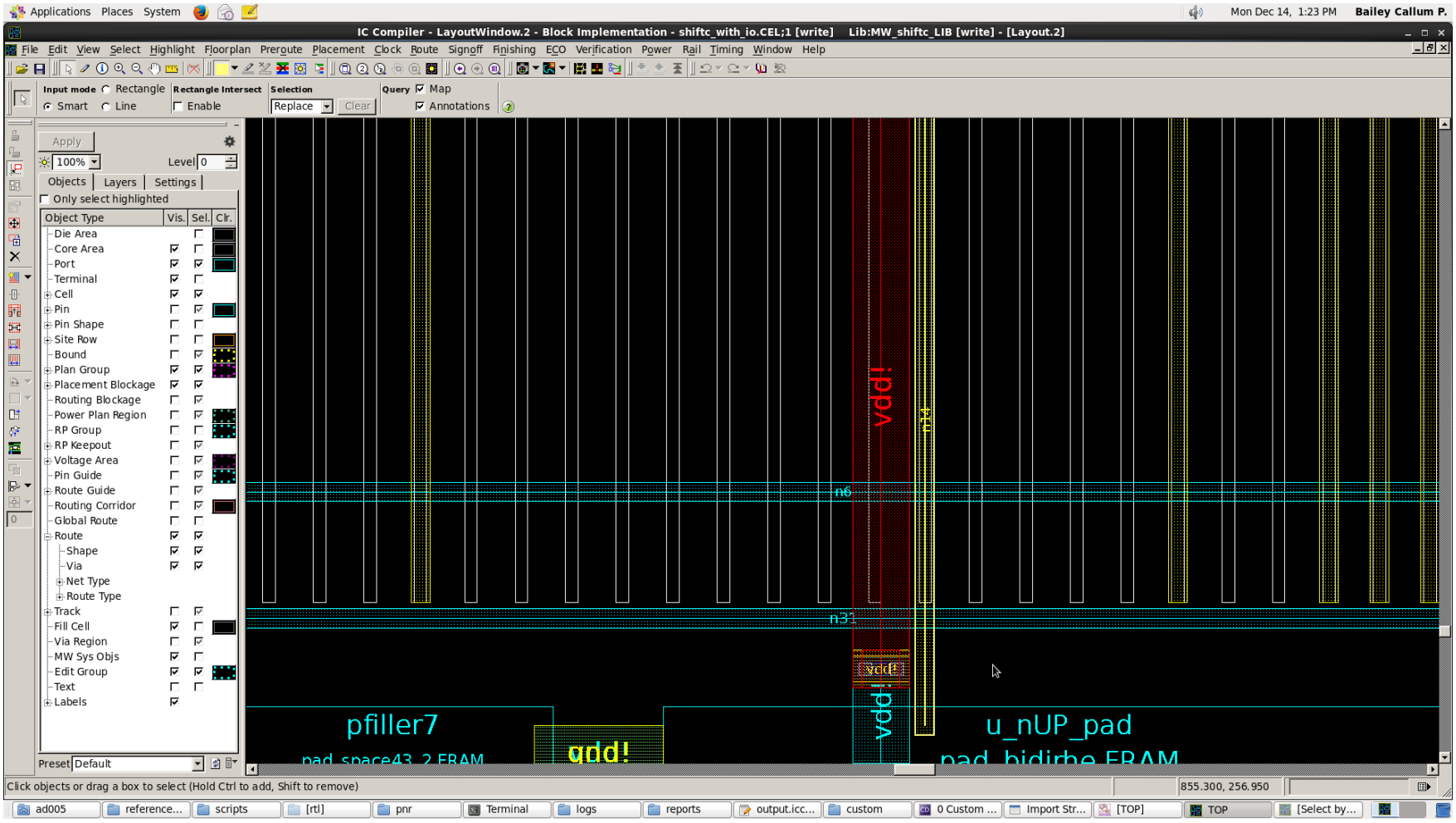
IC Compiler – Fixing a short in ICC

Reroute the net with opt_route command in ICC command line.



IC Compiler – Fixing a short in ICC

Short should now be fixed.

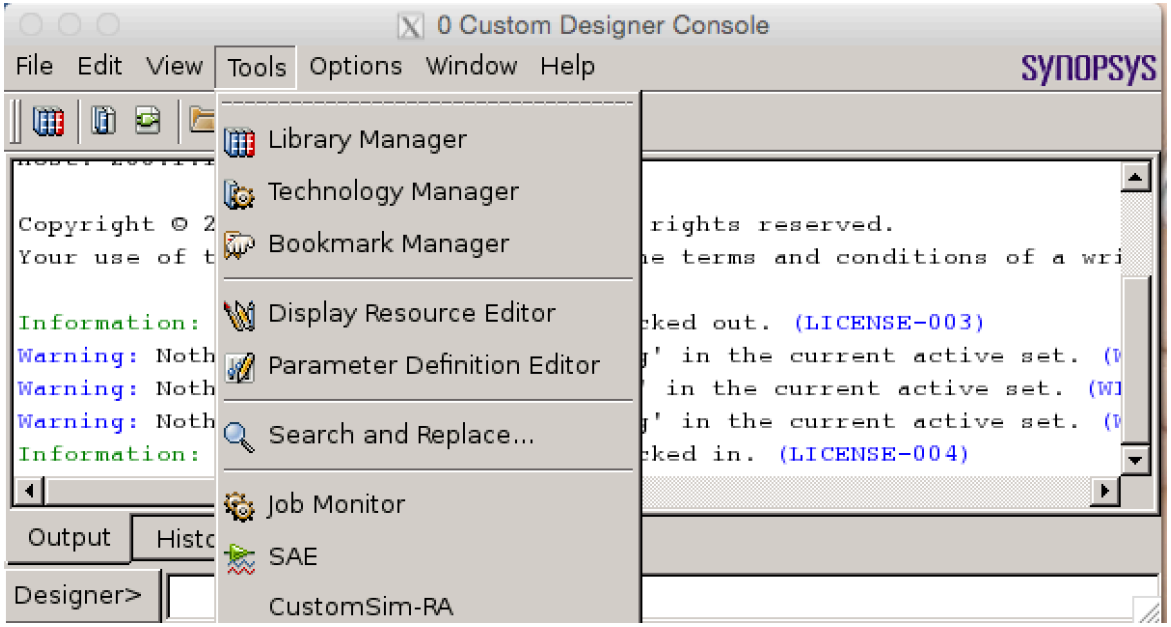


Custom Designer – Import GDS

After running through the entire IC Compiler script and verifying, go to “custom” directory and invoke “cdesigner &”, create a library, read in the GDS, add metal 3 art (optional) and run final DRCs. Steps shown in following slides

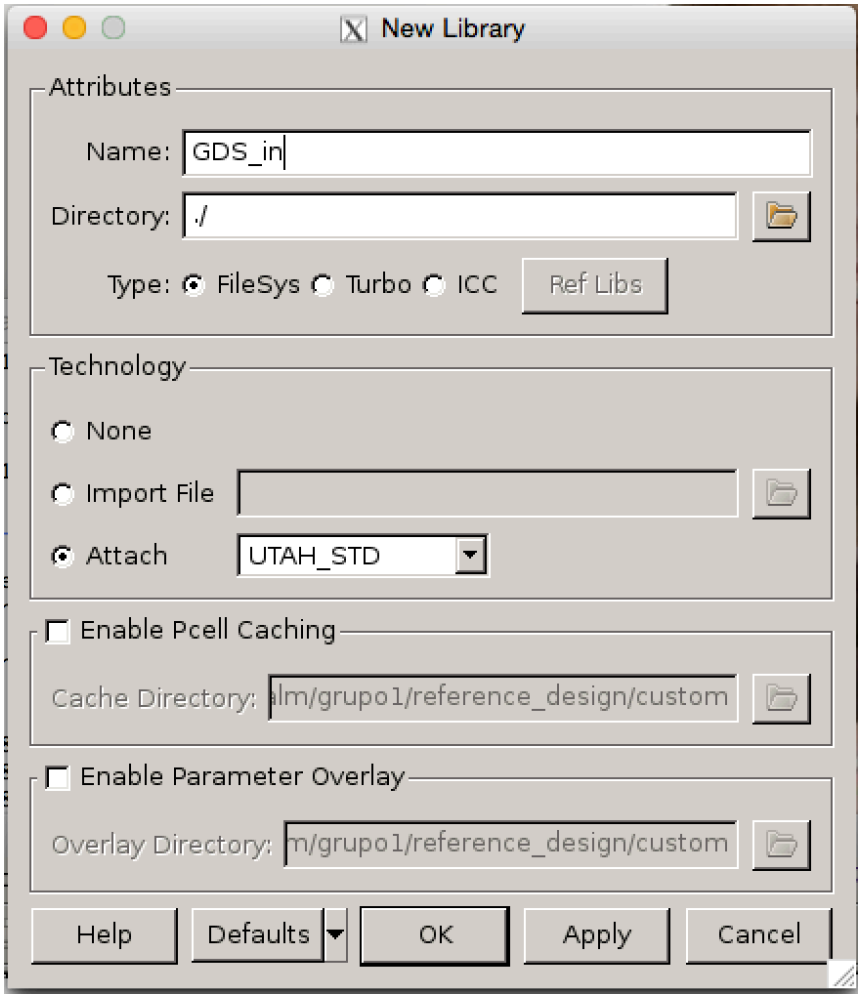
We will be performing DRCs on the core level and will not check the IO’s as these are known to be good and have false problems with DRC rules.

Before you run the DRCs - you can use “add net” after selecting “metal3” and add metal art work (e.g. your name, a diagram, etc.). Example to follow.



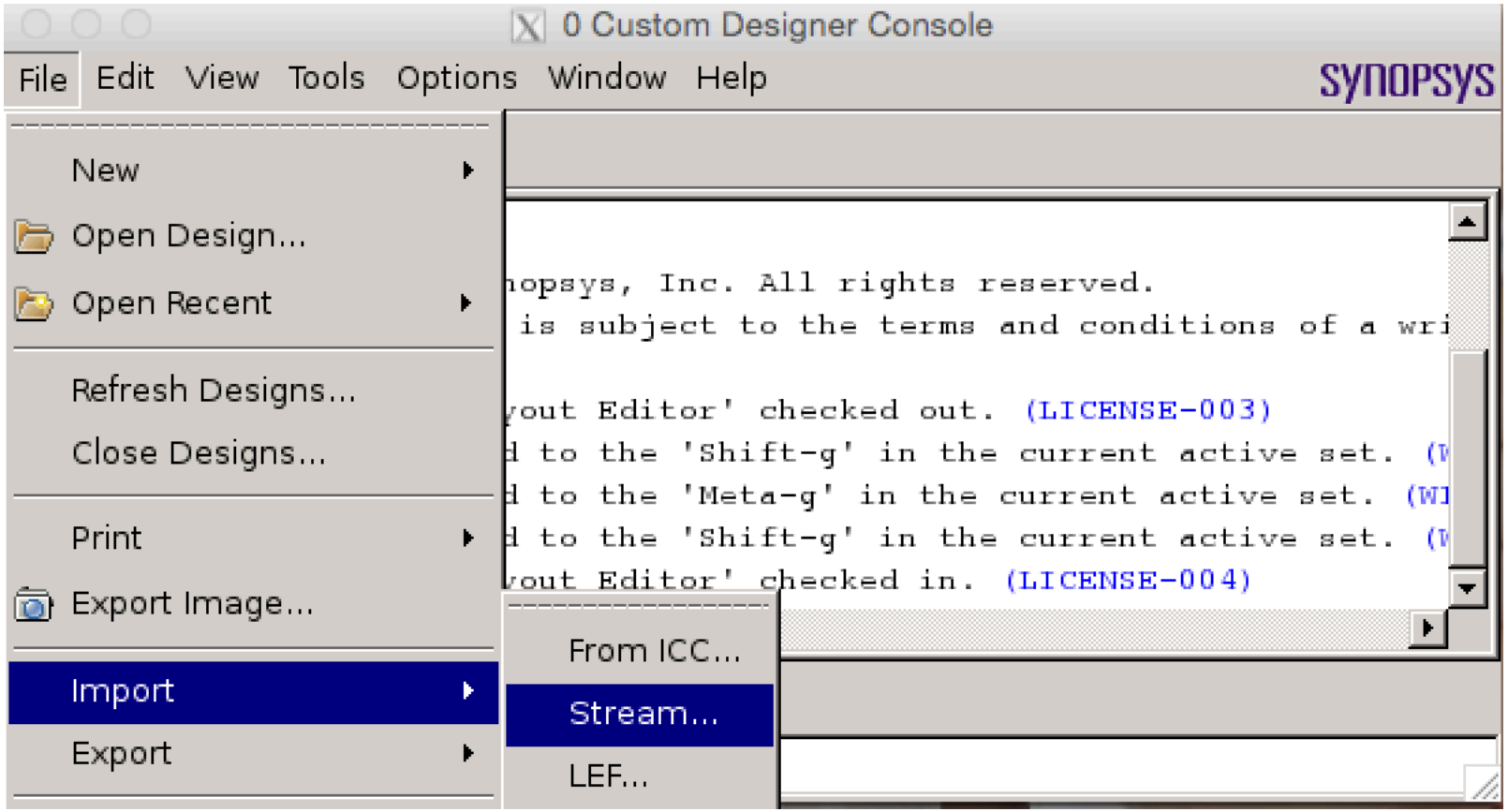
Custom Designer – Create new library

Create “new library” to “import stream” the GDS into with the correct technology.



Custom Designer – Import GDS

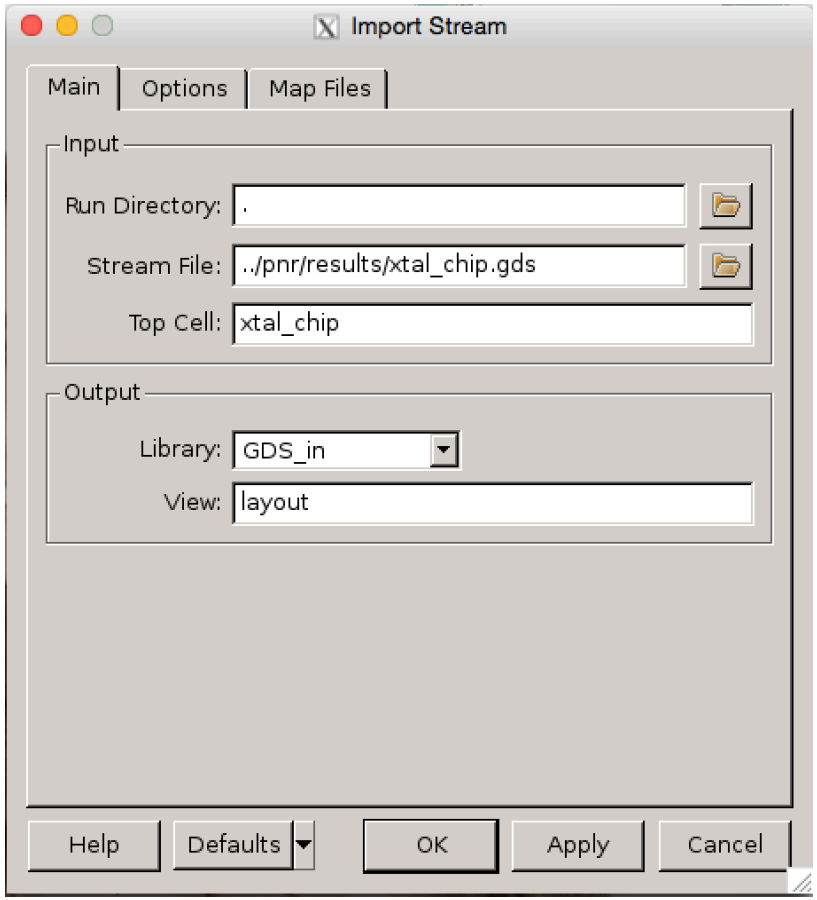
Import “stream” allows us to read in the GDS file created by IC Compiler and later to be sent to MOSIS for fabrication. The GDS shows all geometries of the fabricated device.

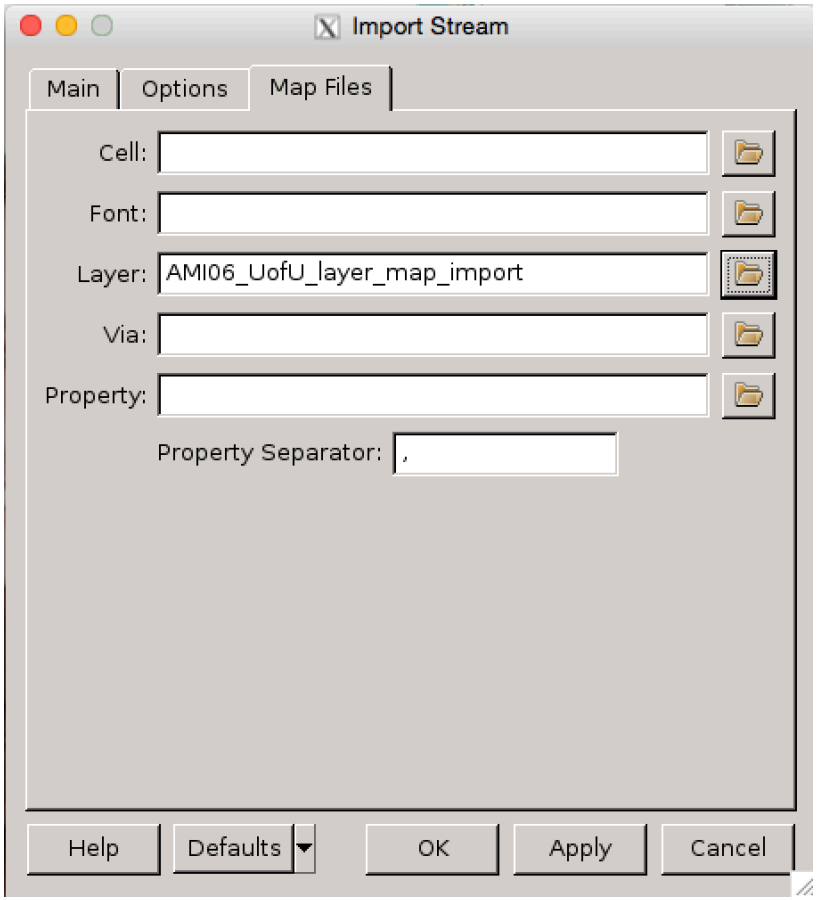


Custom Designer – File -> Import Stream

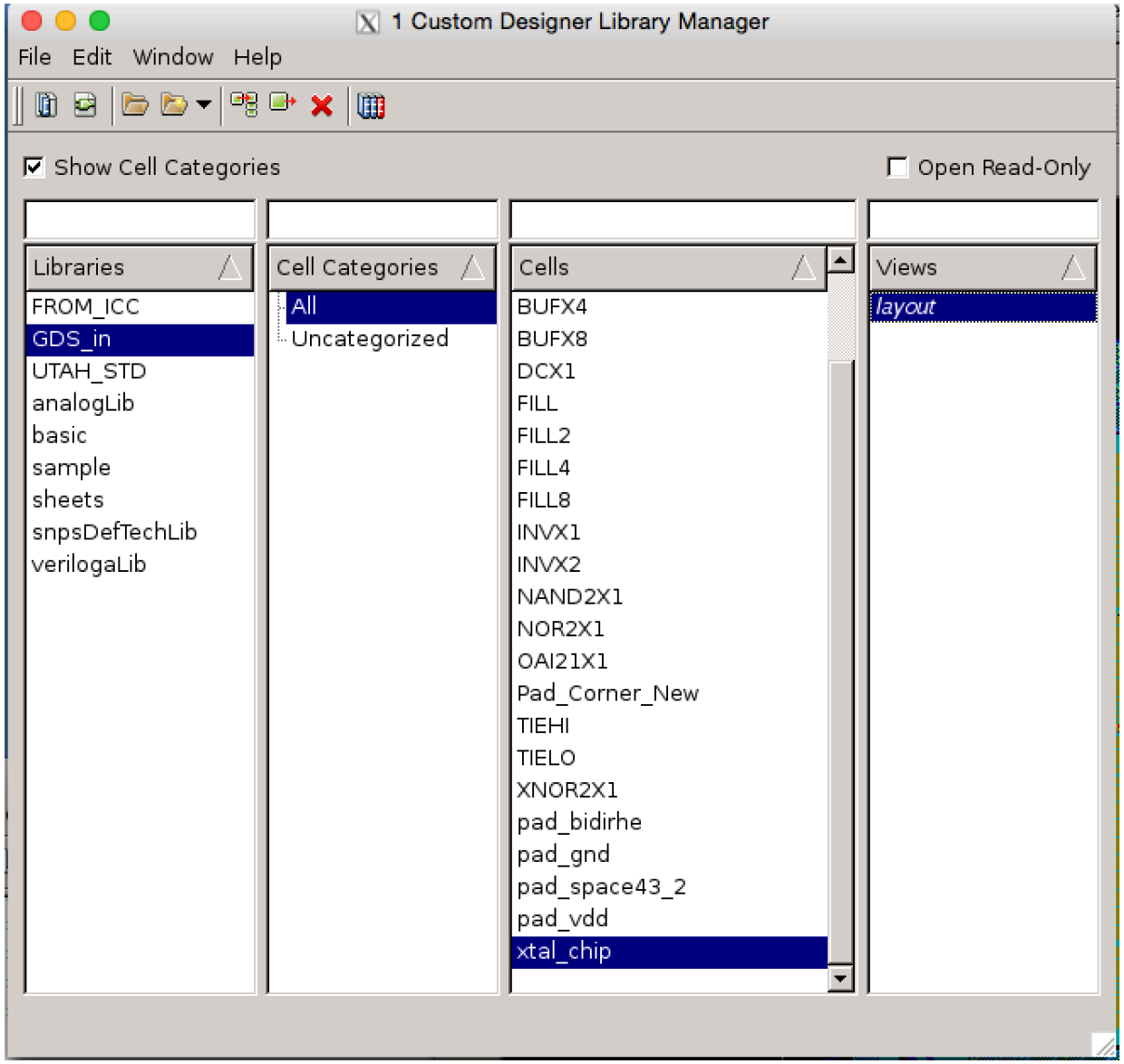
IC Compiler script command:
“save_mw_cel -as ${design_name}” determines the Top Cell name.

IC Compiler script command:
“write_stream -lib_name MW_${design_name}_LIB -format gds results/${design_name}.gds
determines the file name of the GDS filename.



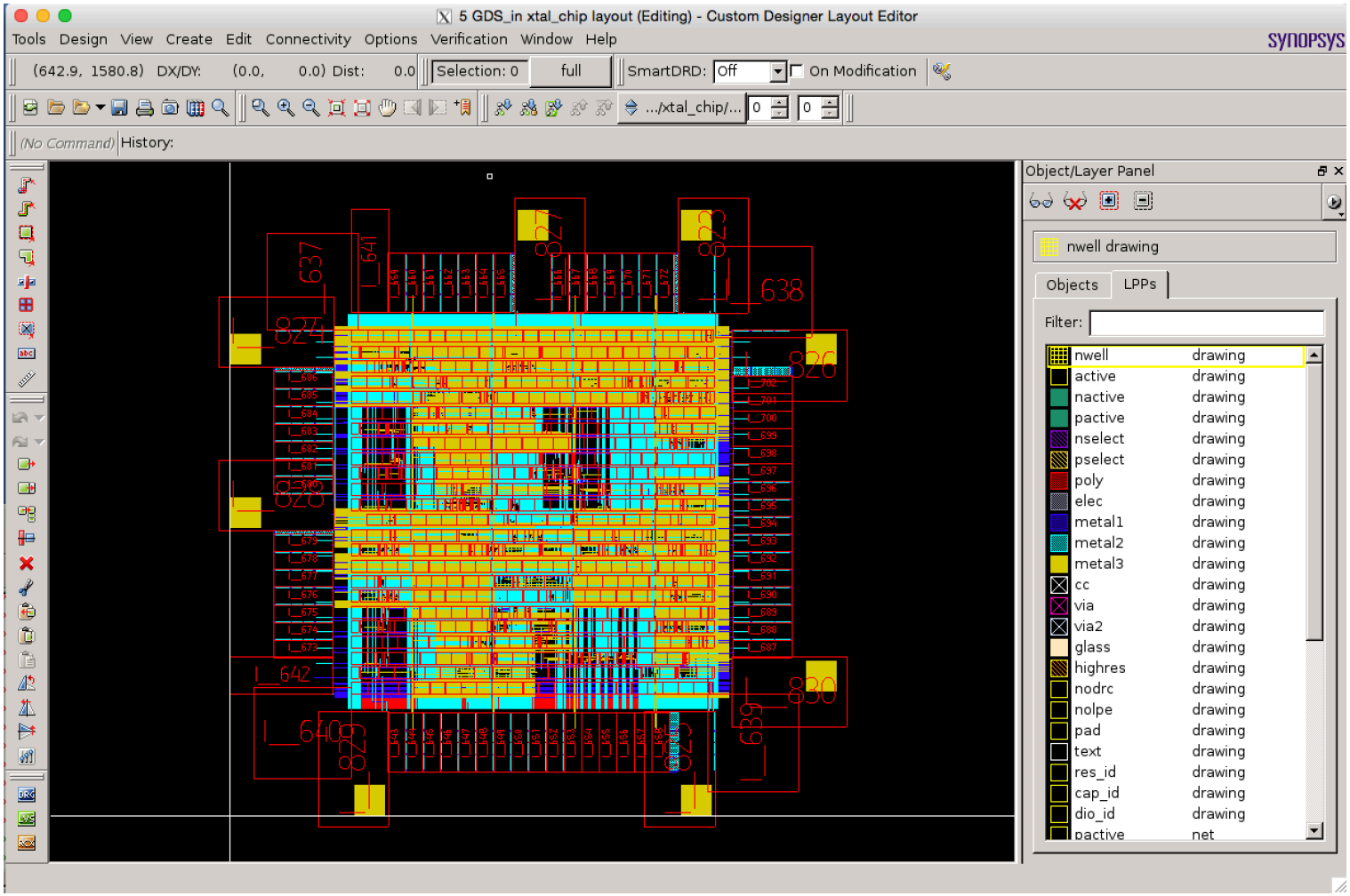


Custom Designer – Open Layout



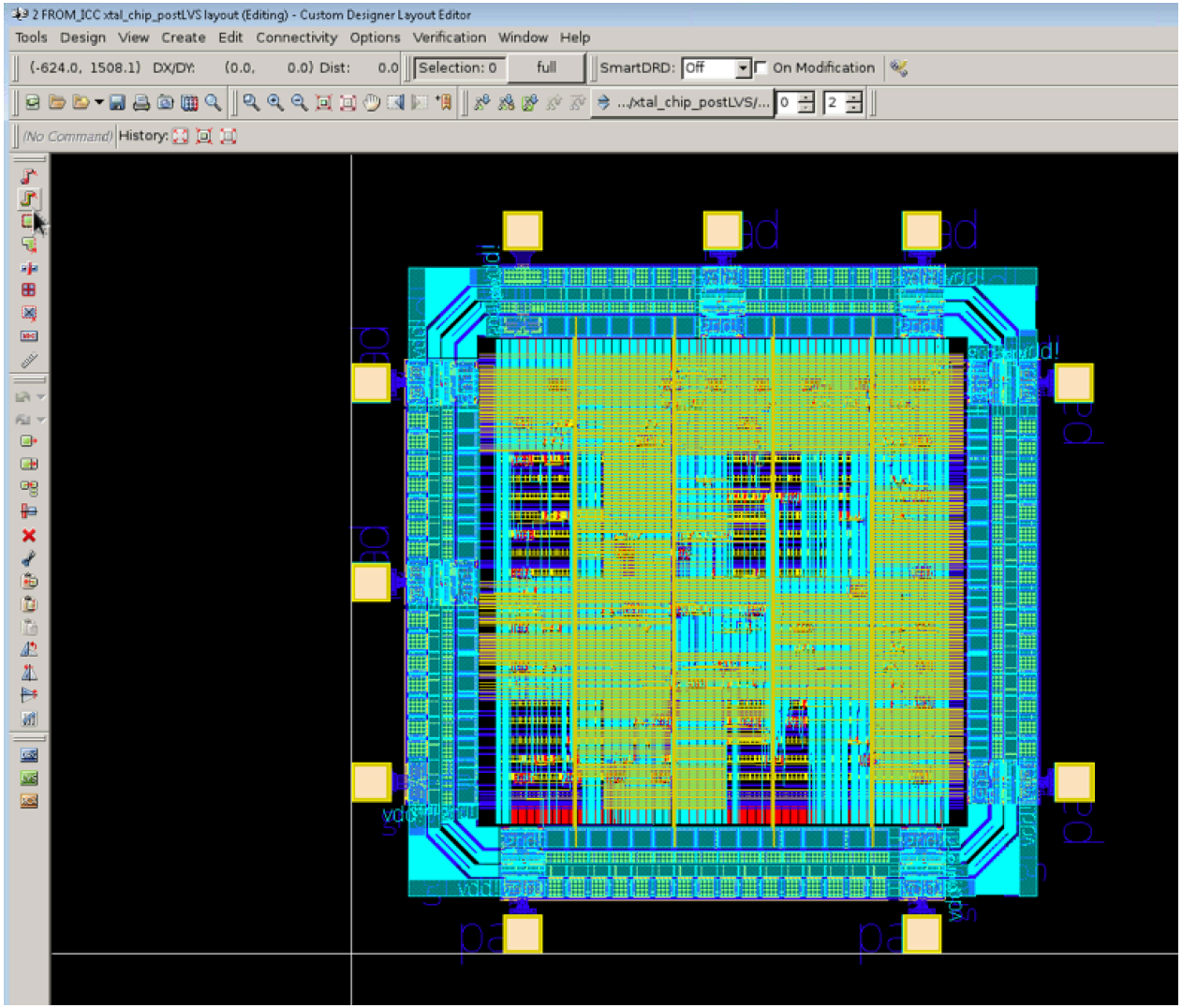
Custom Designer – Draw Images in Metal 3

f – fits the design to the window.



Custom Designer – Draw Images in Metal 3

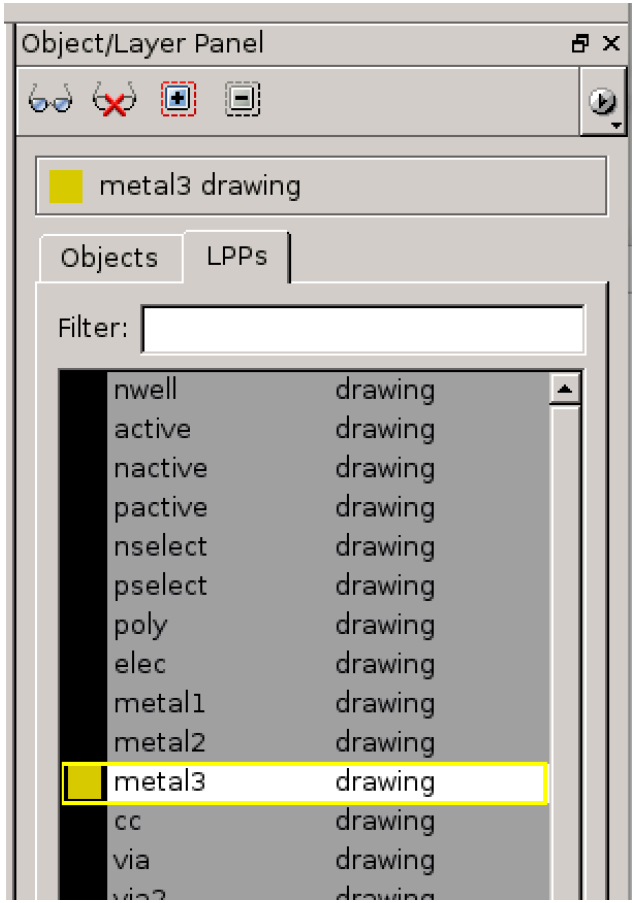
Shift F – shows all lower hierarchical layers. This shows chip after shift F.

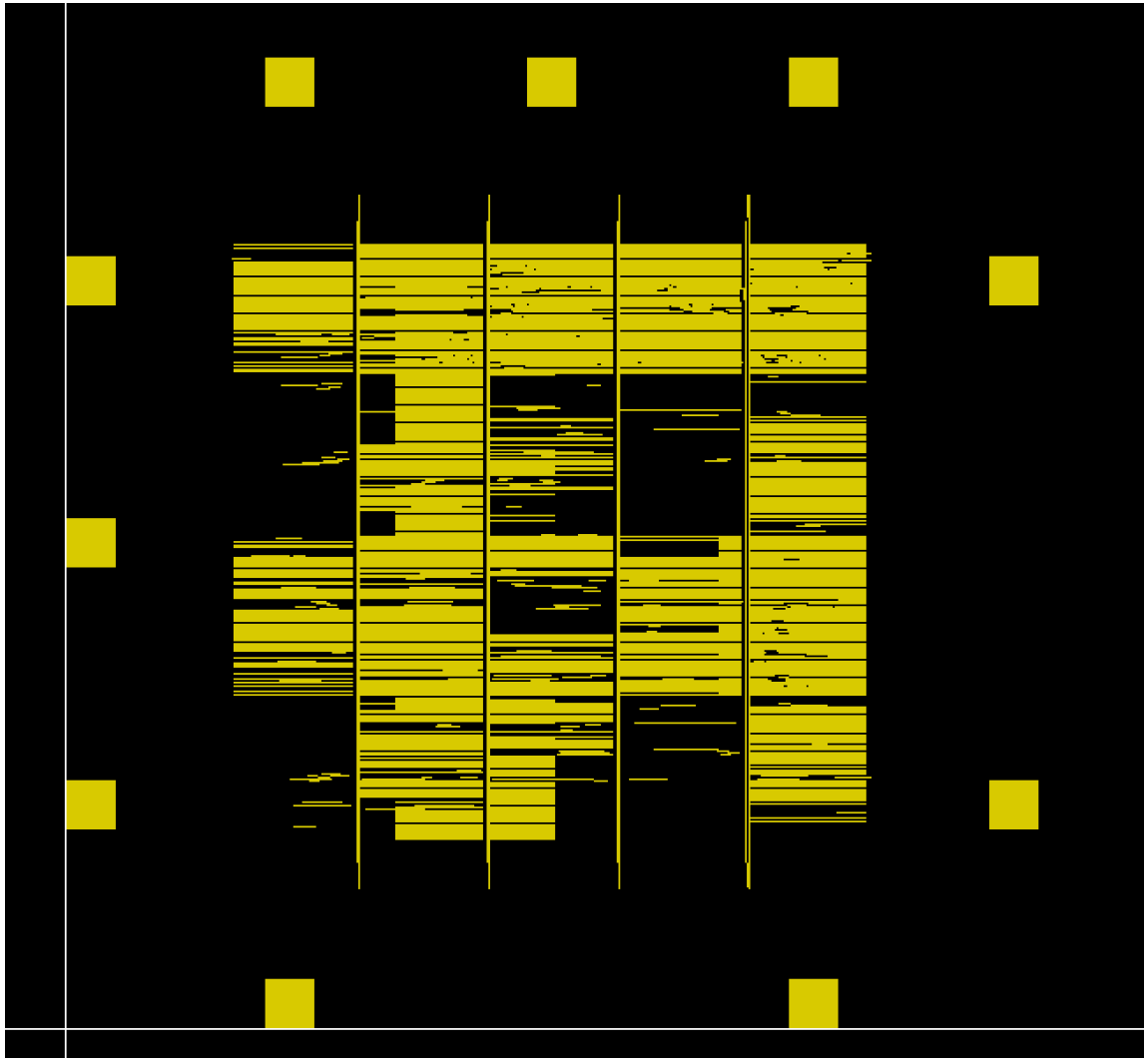


Final Report: Add screen shot to final report.

Custom Designer – Draw Images in Metal 3

On the right you can select metal 3 and the glasses icon to only see Metal 3.
Control R – redraws after a change.

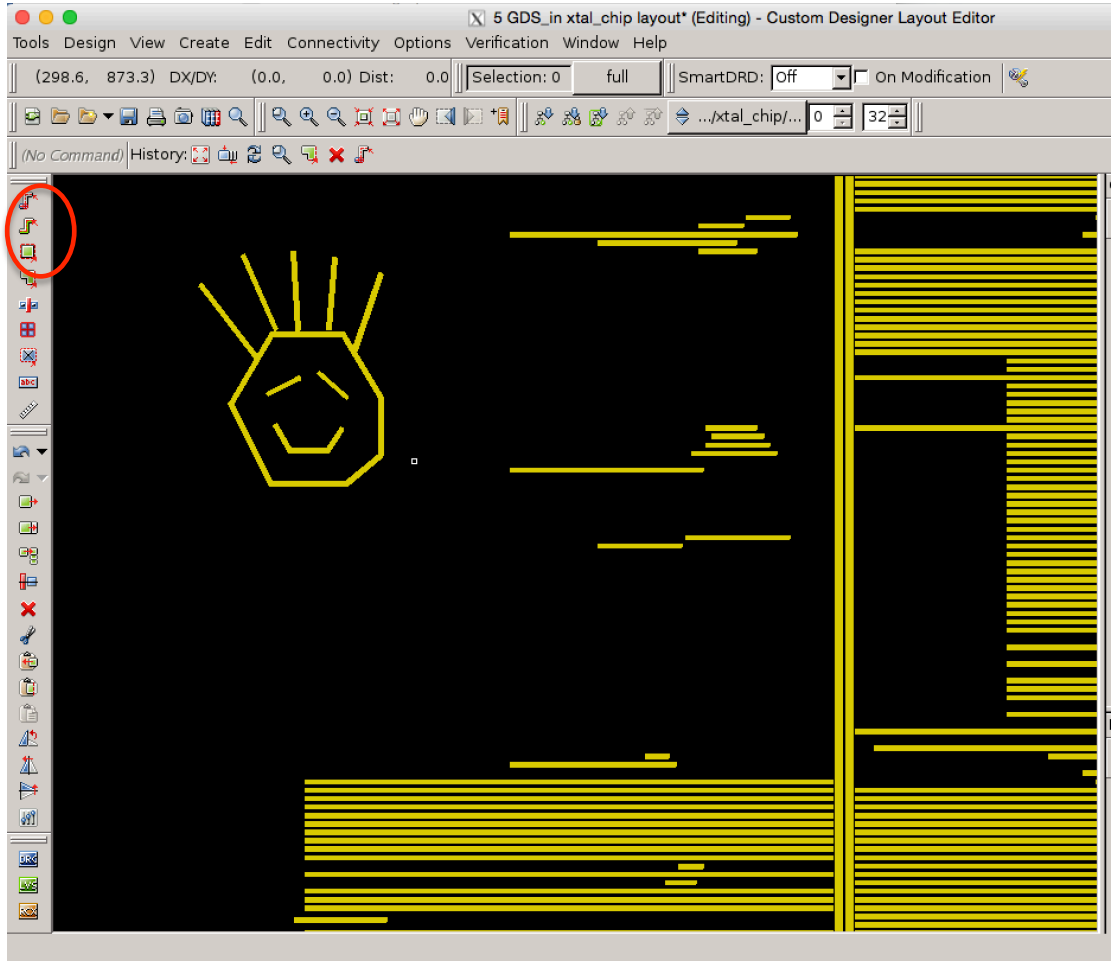




Custom Designer – Draw Images in Metal 3

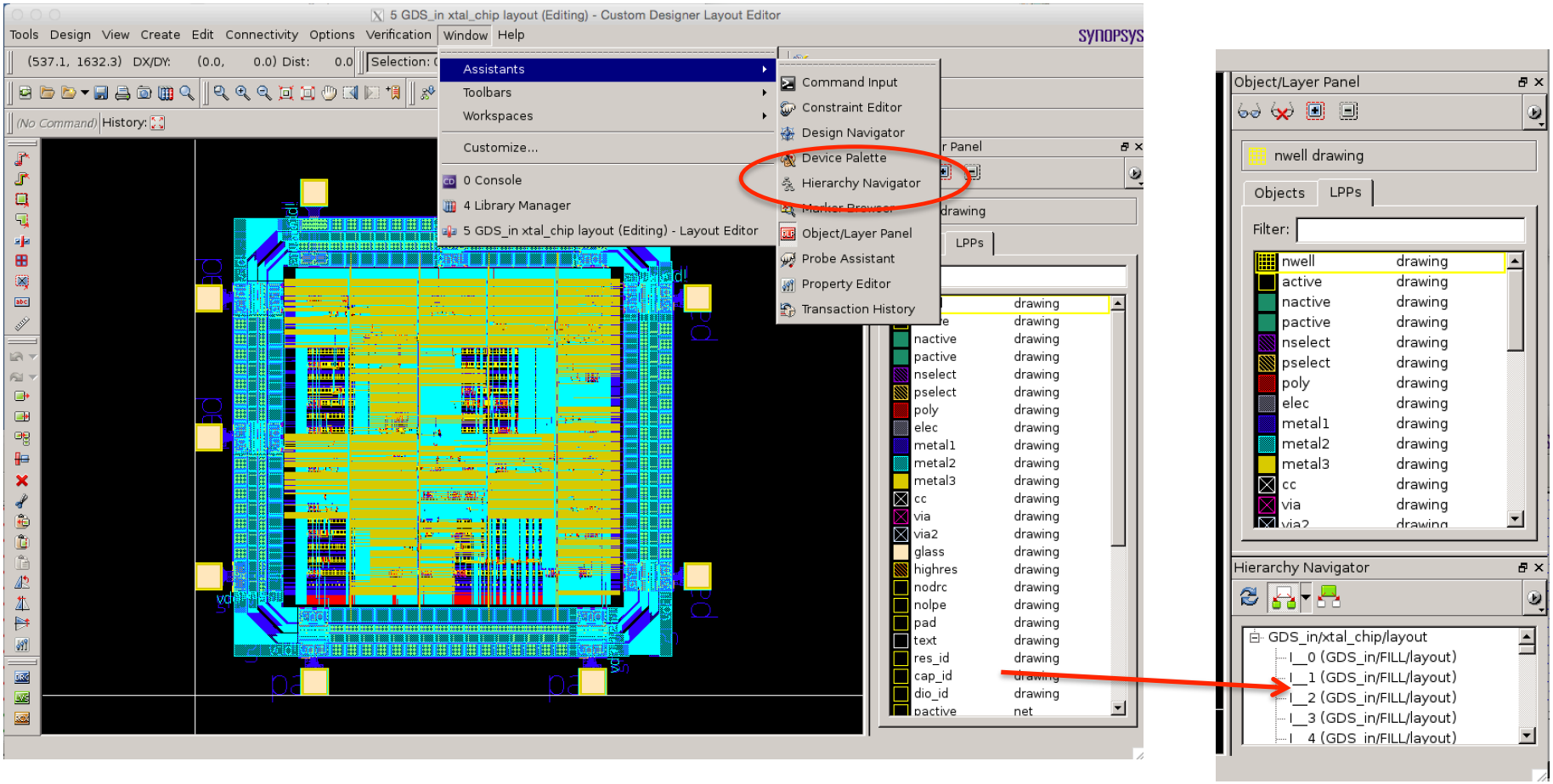
On the right you can select create interconnect and draw lines. There is a submenu that allows you to change thickness and permitted angles.

Warning: Sharp acute angles and insufficient space between lines can cause DRC errors later.



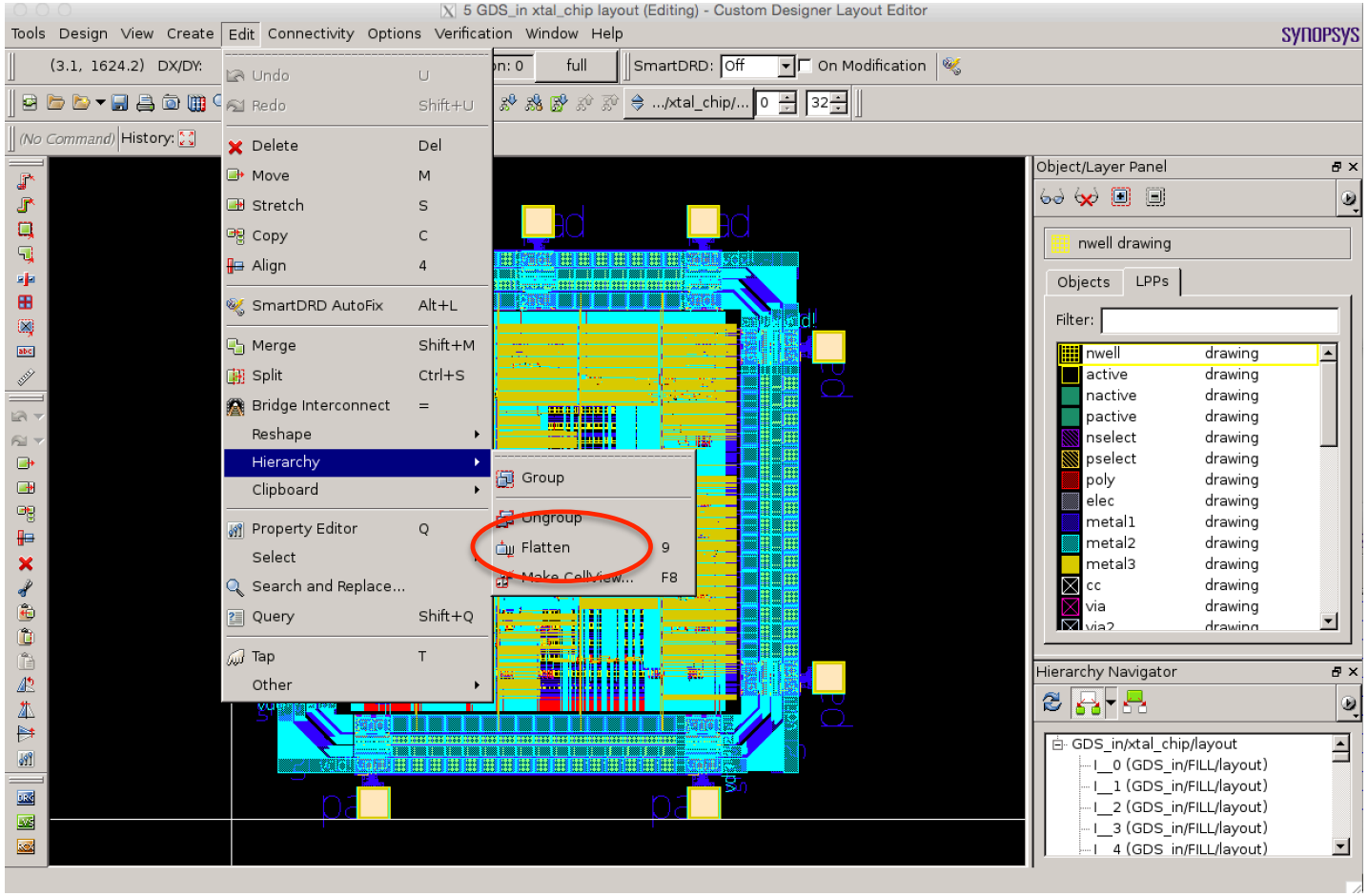
Custom Designer – Run DRCs

Before we can run checks, we must flatten the design. Open Hierarchical Navigator.



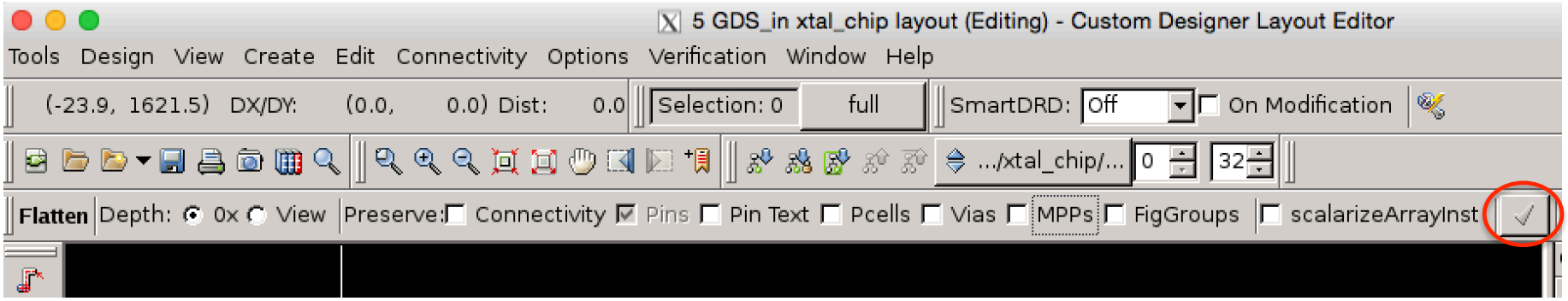
Custom Designer – Run DRCs

Select flatten from edit menu.



Custom Designer – Run DRCs

There will be “flatten” submenu in the tool bar and set the switches according to the picture below.



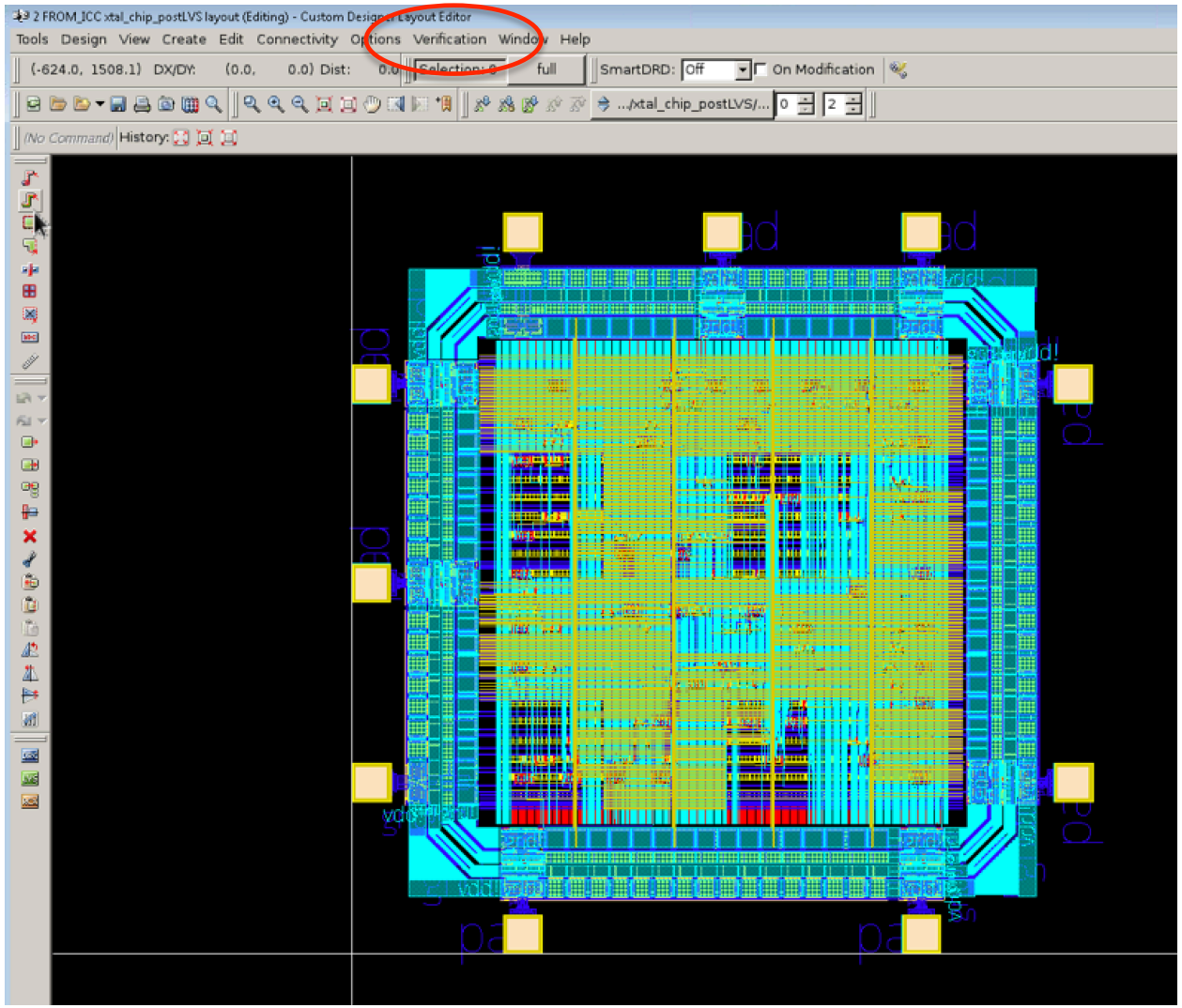
With the mouse, select the entire design by dragging a box around everything including IO’s.

Then click the check mark on the far right of the menu.

You should see only one instance with sub-instances in the Hierarchical Navigator after you click on the refresh.

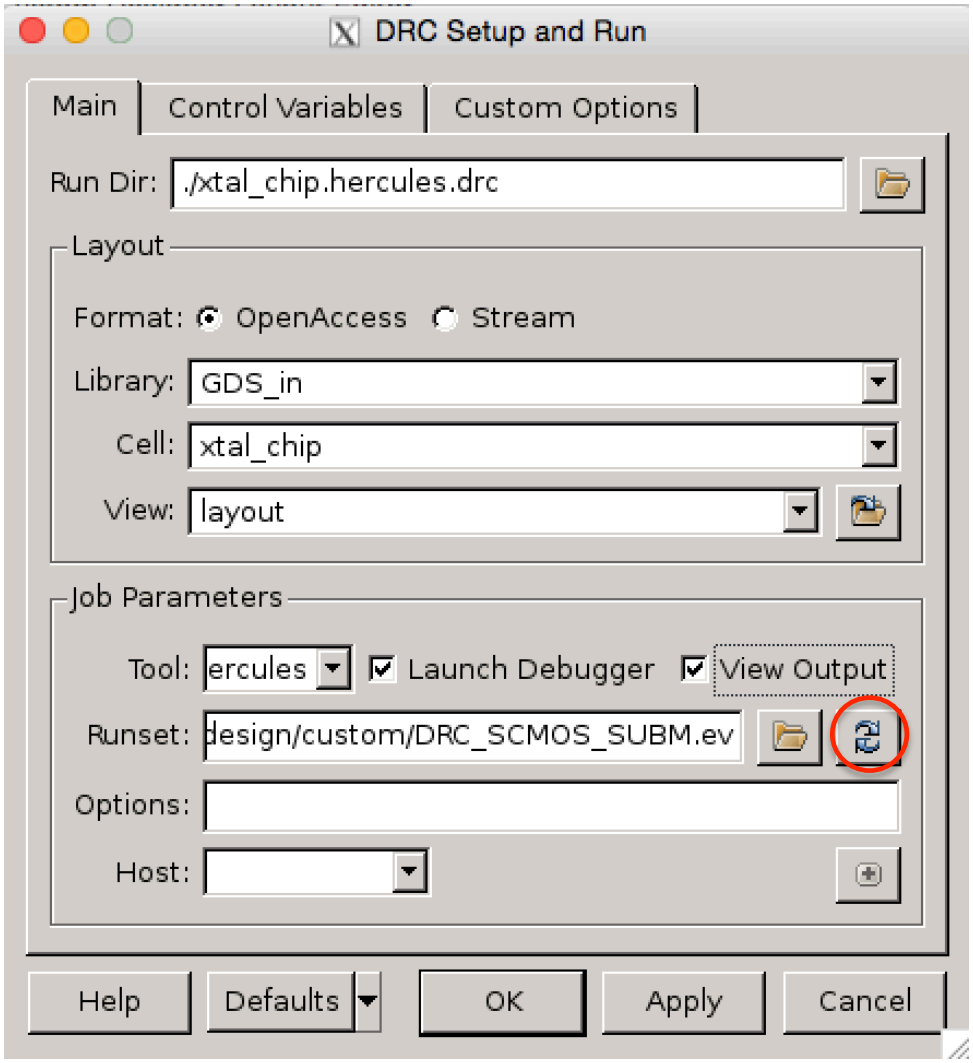
Custom Designer – Run DRCs

Use the verification menu to run final DRCs regardless of whether you added any metal patterns. Only select the internal core and avoid the IO's as there is an issue with the rules for the cells which is not real and can be ignored.



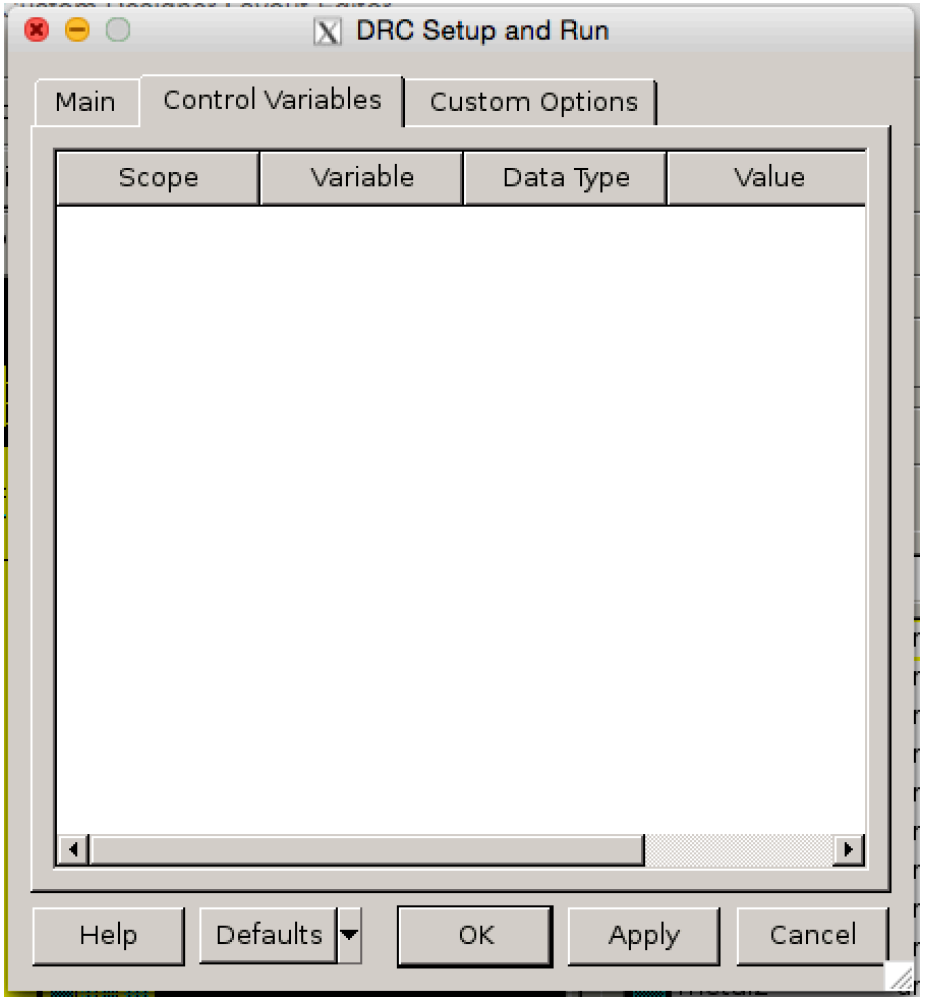
Custom Designer – Run DRC

Fill in the following fields of the DRC setup pane – “Main”. The runset file should be in your custom directory. Press the refresh button next to runset field.



Custom Designer – Run DRC

If you press the refresh button in the Main panel next to the runsets, you should see fields in this “control variable” section. There are none in this picture as Hercules (DRC checker) was not yet installed.

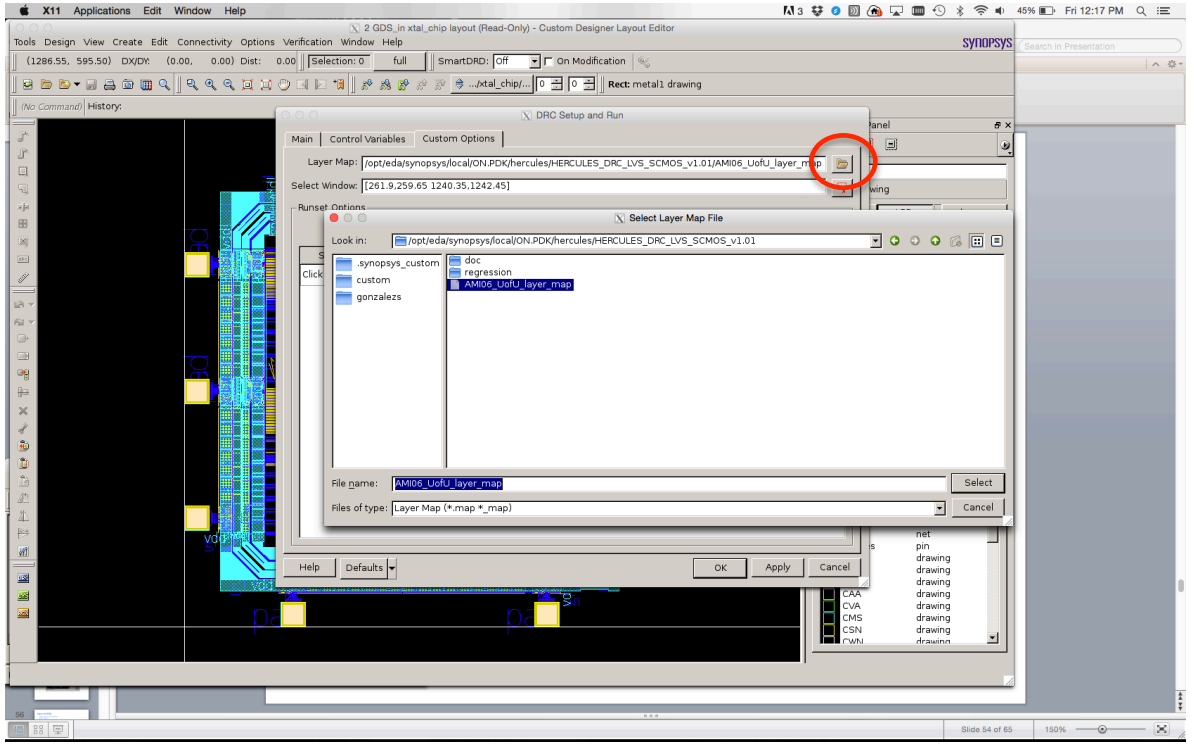


Custom Designer – Run DRC

In Custom Options, click the “select window” button and you will be allowed to select from the main window what will be DRC checked. Since there are problems in the IO that are not real DRCs and the IO’s are proven, we are only going to check the interior section of the chip. Select the interior by drawing a box around the chip but avoiding the IO rings. Coordinates will show in the field and should be similar to what is shown here – approximately 250, 1250, 1250, 1250.

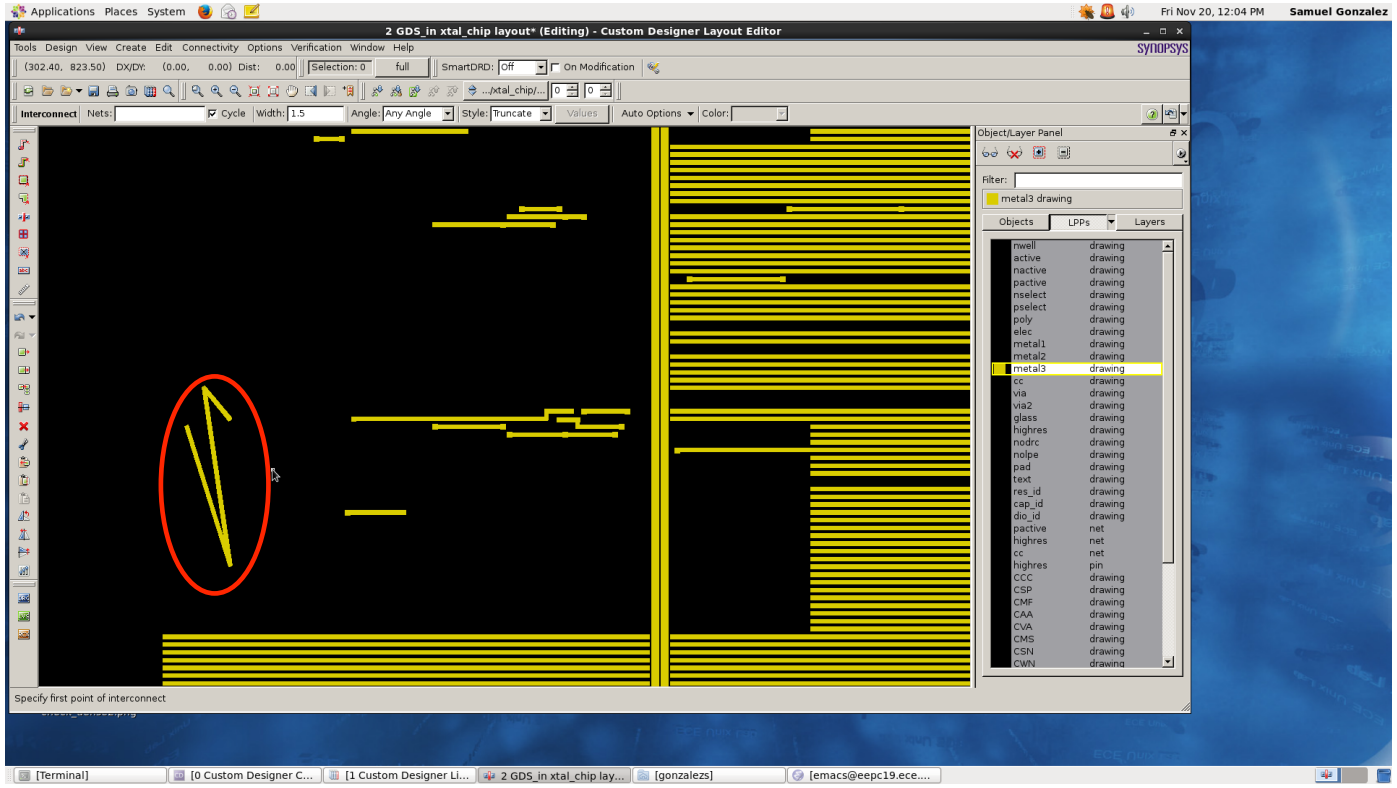
Select Layer Map located in custom directory

Now click OK to run DRCs.



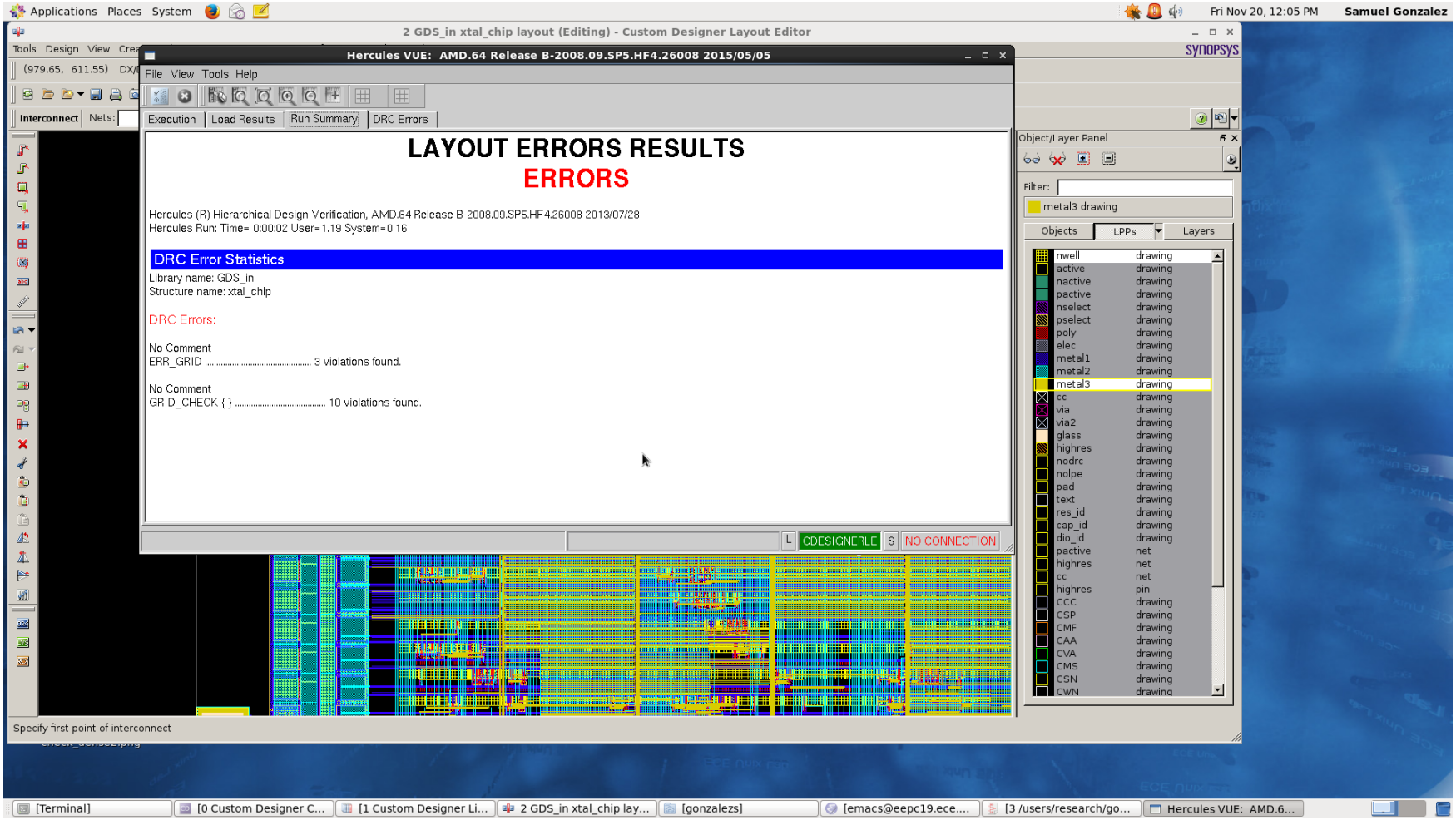
Custom Designer – Example DRC failure

Metal 3 with DRC errors



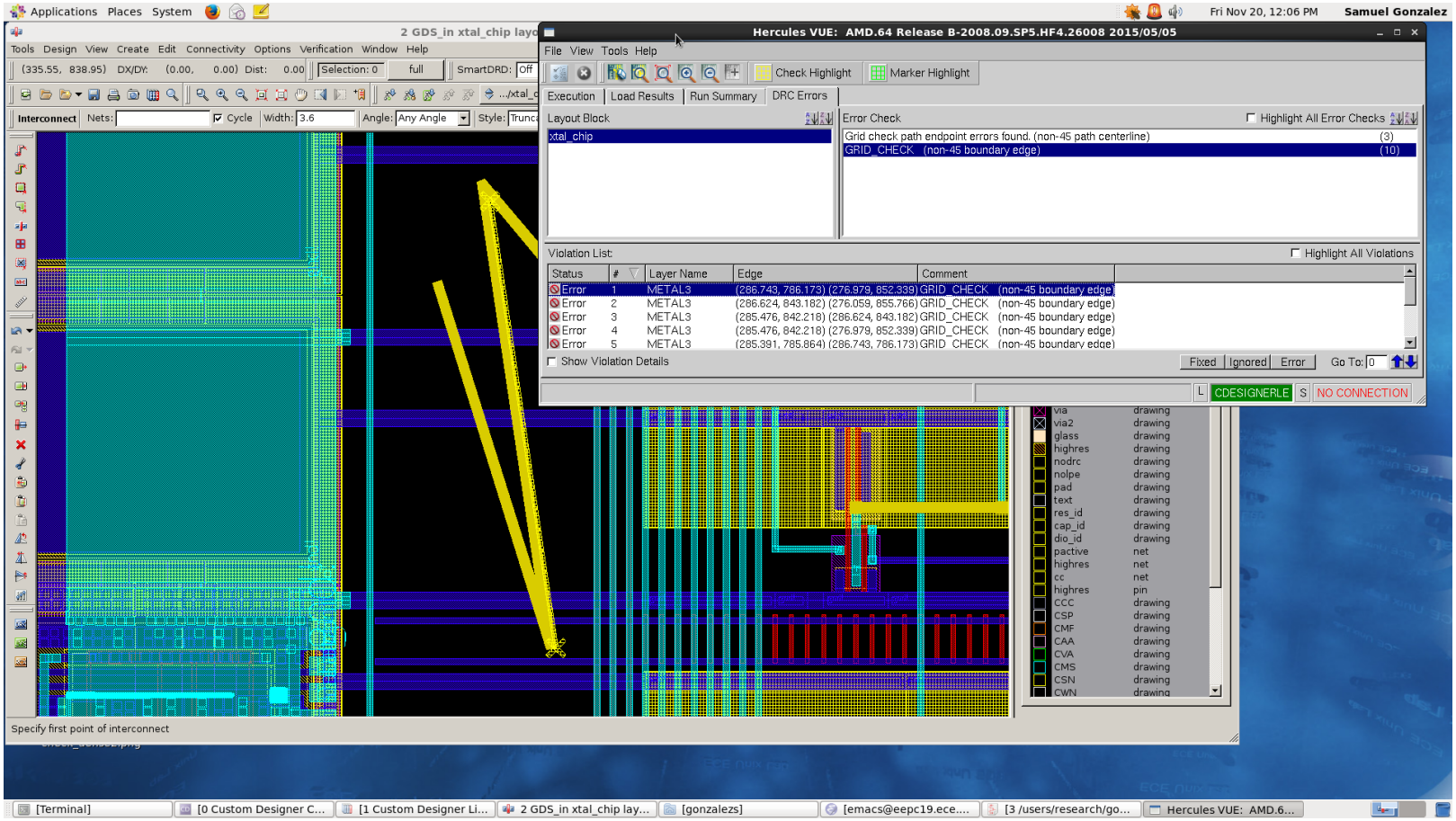
Custom Designer – Example DRC failure

Metal 3 DRC errors found



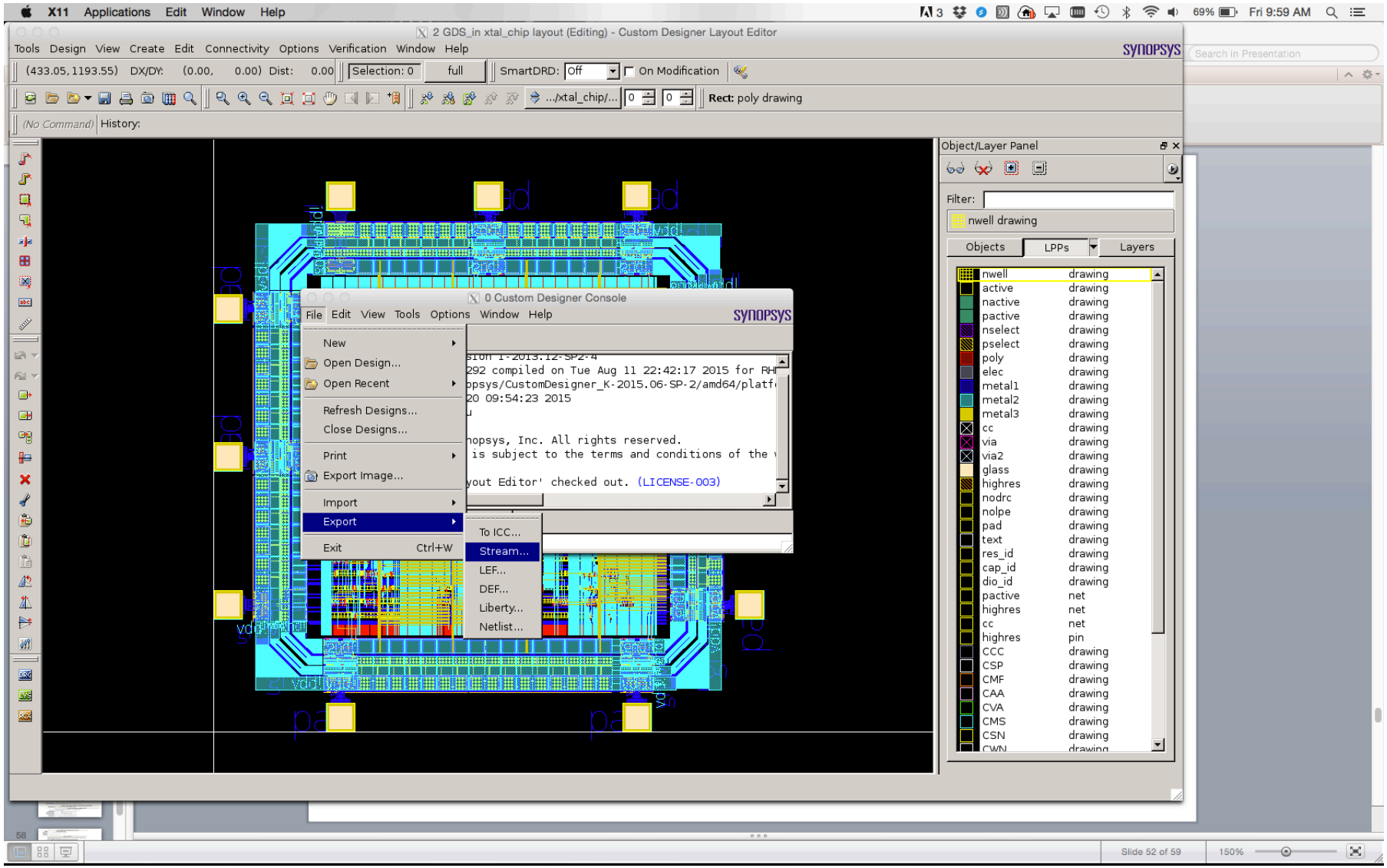
Custom Designer – Example DRC failure

Select errors to show location of error
Fix error and run DRC checks again



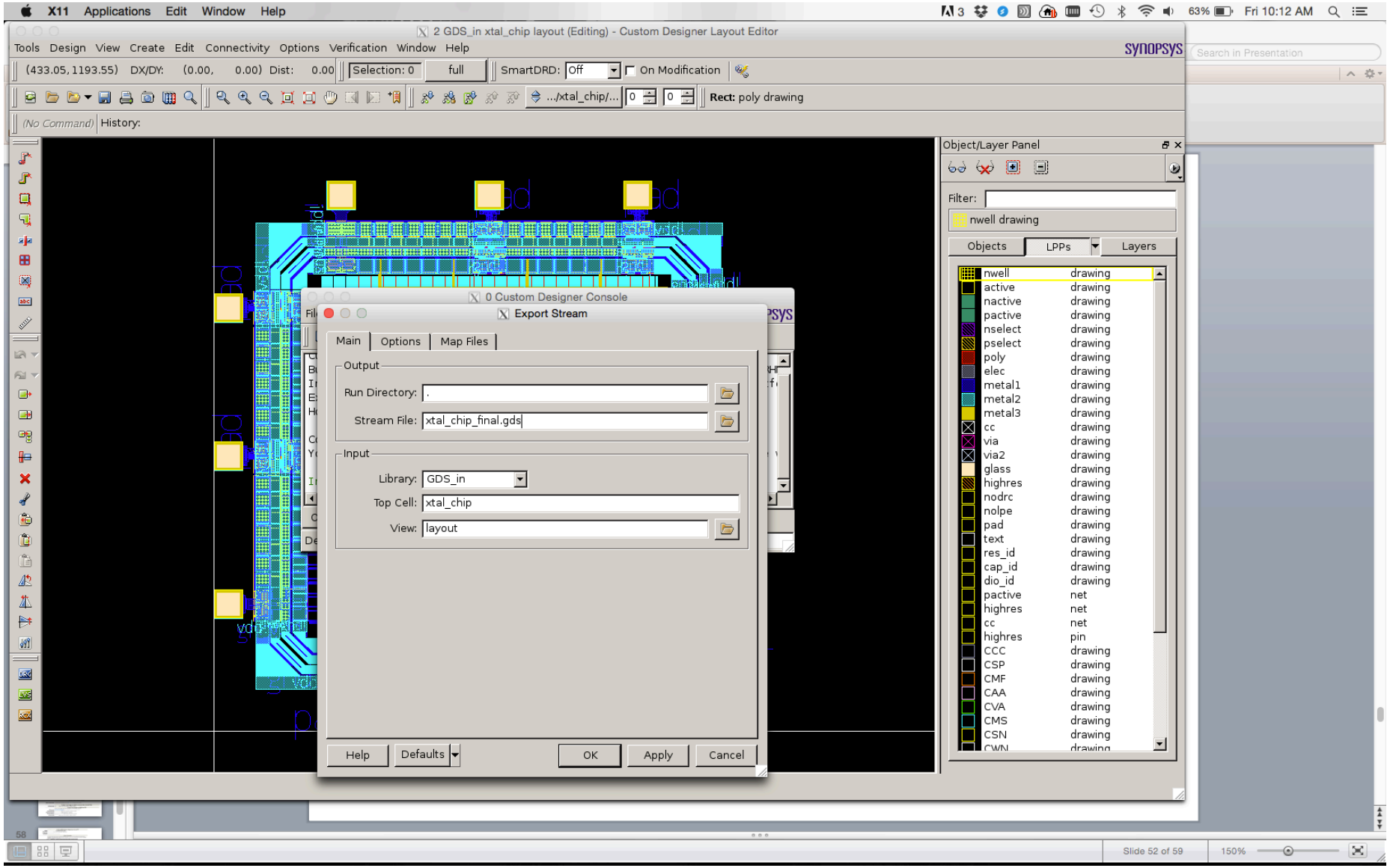
Export GDS file

File -> Export -> Stream



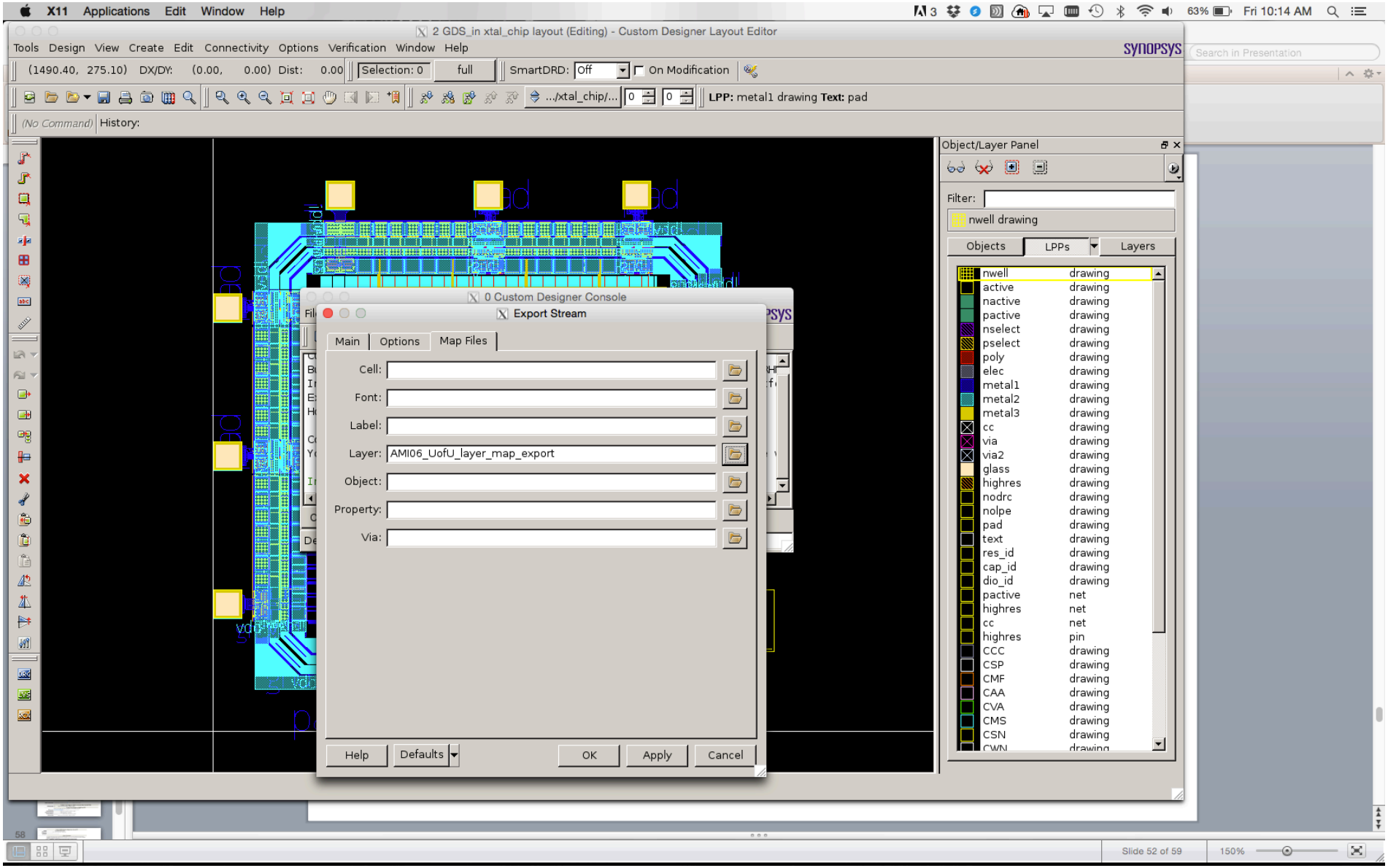
Export GDS file

Stream File – name of final gds file, Library – GDS_in, Top_cell – Name of your top cell.



Export GDS file

Layer Map – AMI06_UofU_layer_map_export



Metal Density Check

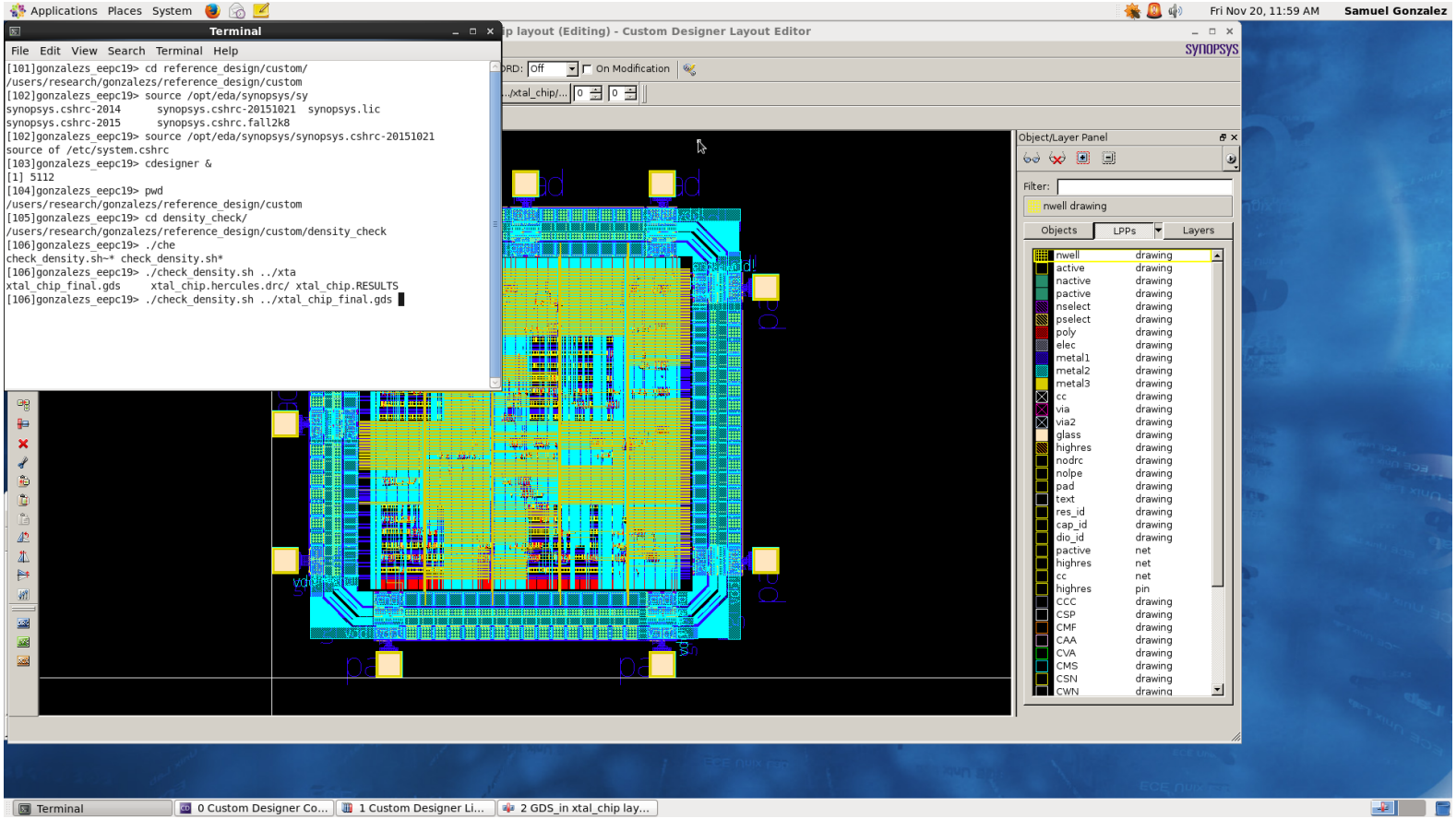
For fabrication there are minimum metal densities. The script below checks densities for each metal layer.

cd ~/reference_design/custom/density_check

Update cell in reportDensity.rs to your top cell.

Run check_density script on the GDS file as shown below in shell.

Look at [Design_Name].LAYOUT_ERRORS file for errors.



Metal Density Check

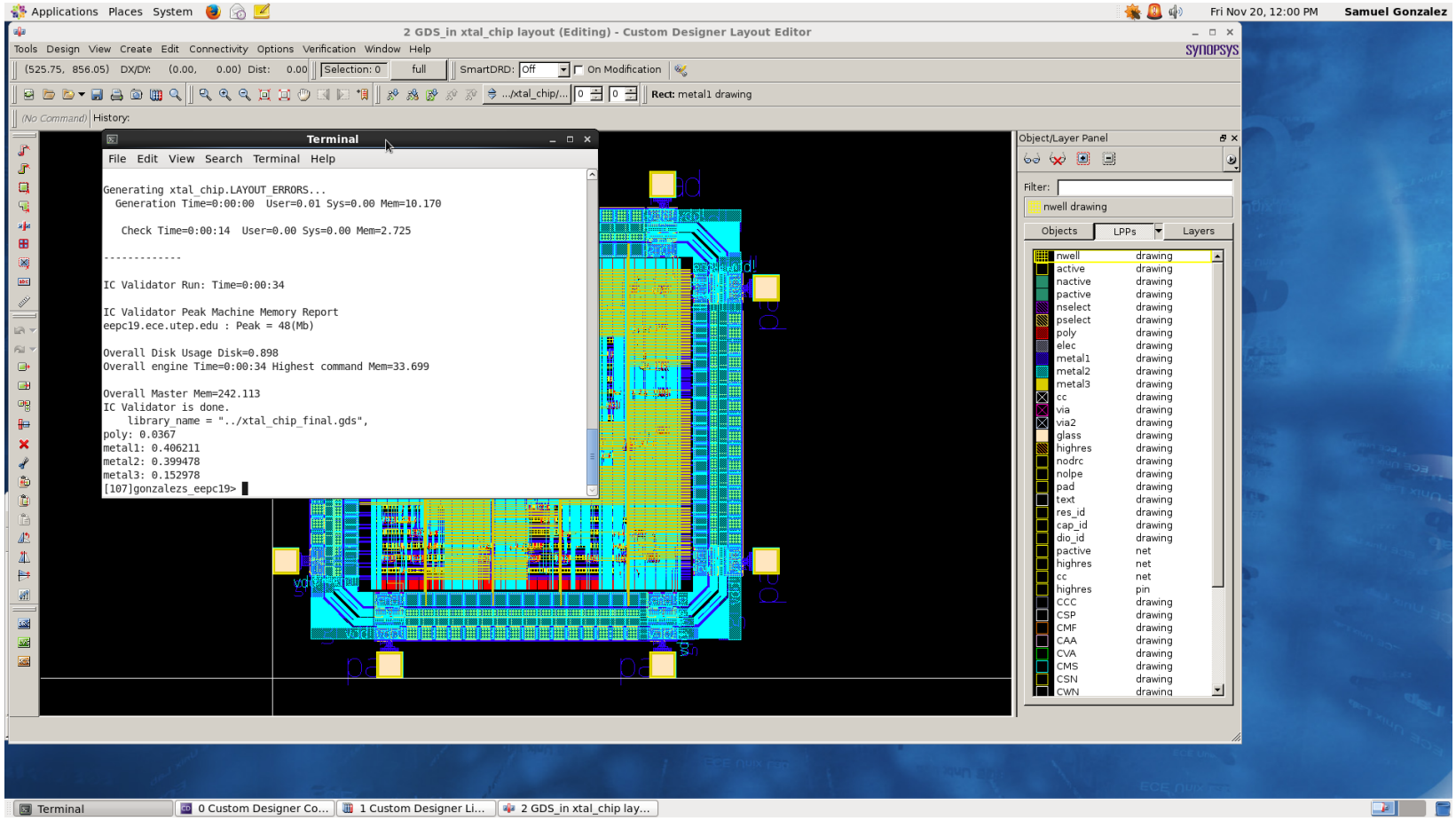
The required density for each layer is shown below.

Poly between 12–30%

Metal 1 between 30-80%

Metal 2 between 30-80%

Metal 3 between 30-80%



Adding more Layers (If densities not met)

If you failed to meet the minimum densities for each layer complete the following steps.

Be sure to run DRC again and fix any issues that may arise before exporting gds.

Check densities again and repeat adding layers if densities are still not met.

If densities are met continue with the Mosis submission portion of the tutorial.

Great care must be taken in the following steps. If you short a connection or similar layer you will cause problems with your chip.

If the changes you make cause errors and you need to revert to a design without those changes reimport gds file from ICC and start over.

Adding more metal to layers (If Densities not met)

Useful commands are

Copy
- hit c on keyboard
- click on what you would like to copy
- click location where you would like to place copy

Move
- hit m on keyboard
- click on what you would like to move
- move to location you want

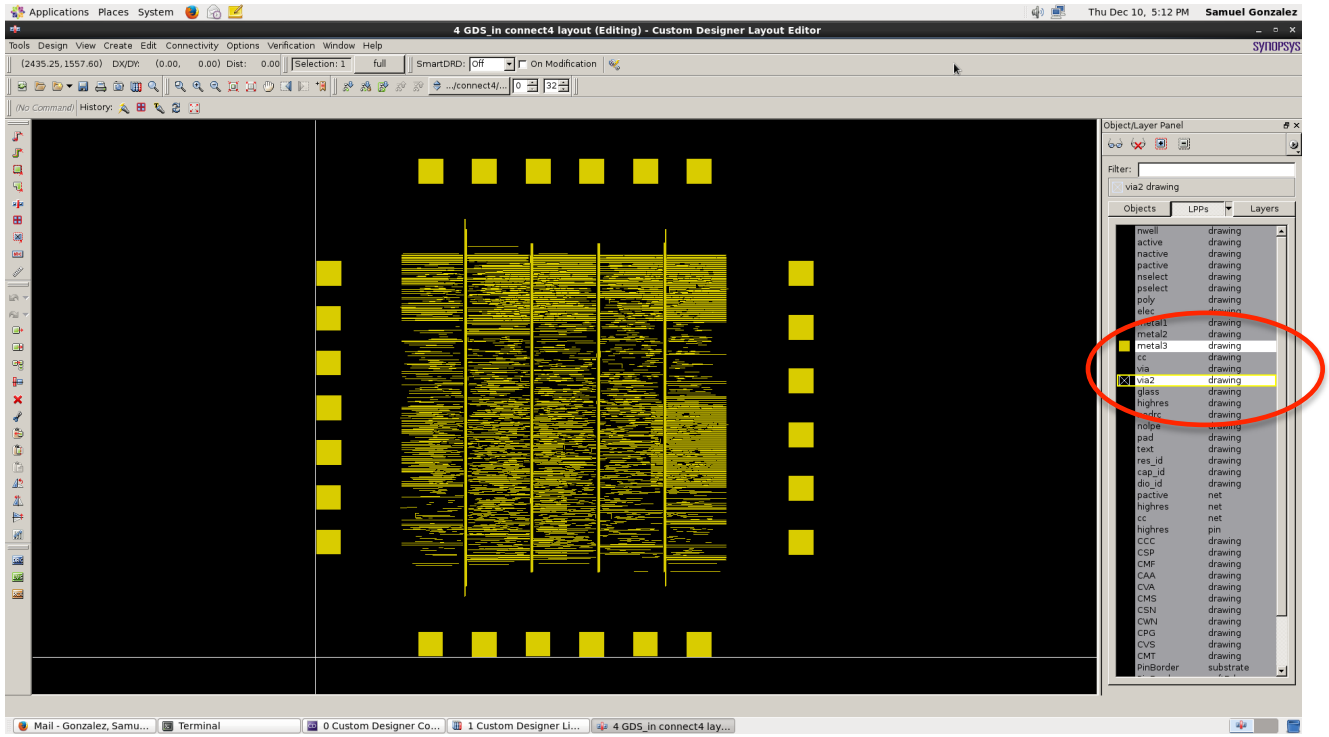
Stretch
- hit s on keyboard
- click on what you would like to stretch / resize
- move mouse to resize

Adding more metal to layers (If Densities not met)

The two common layers that may fail to meet densities are Poly and Metal 3

To fix Metal 3 densities make visible only Metal 3 and via2 layer

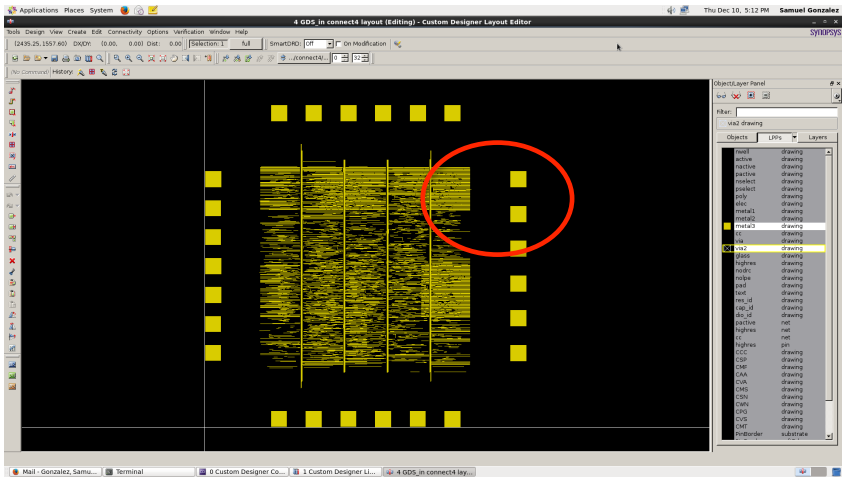
The via2s will not be immediately visible

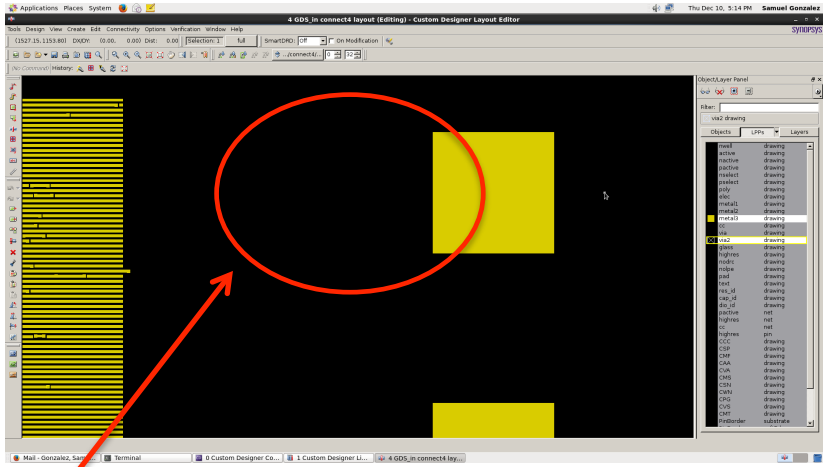


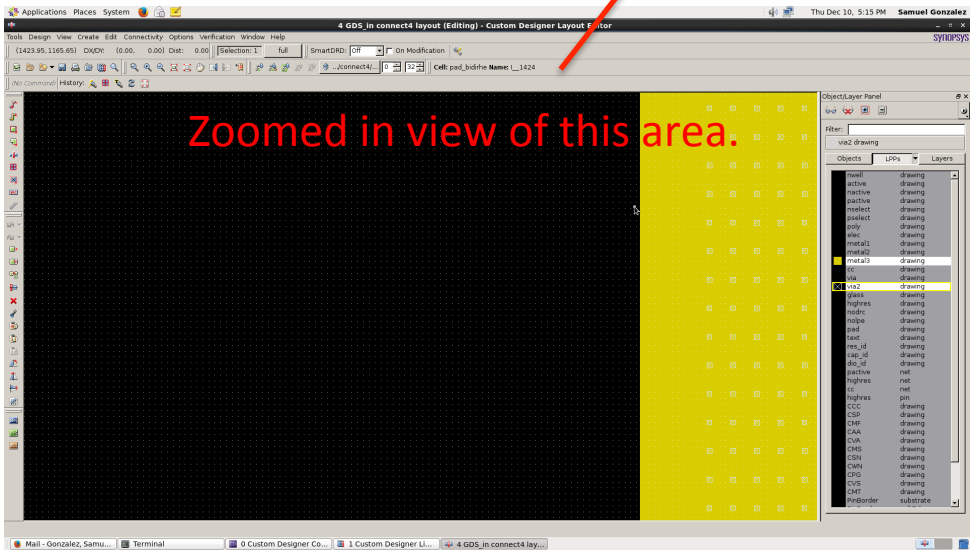
Adding more metal to layers (If Densities not met)

Zoom in to an area where there is space to add Metal 3.

If via2s are still not visible zoom in further

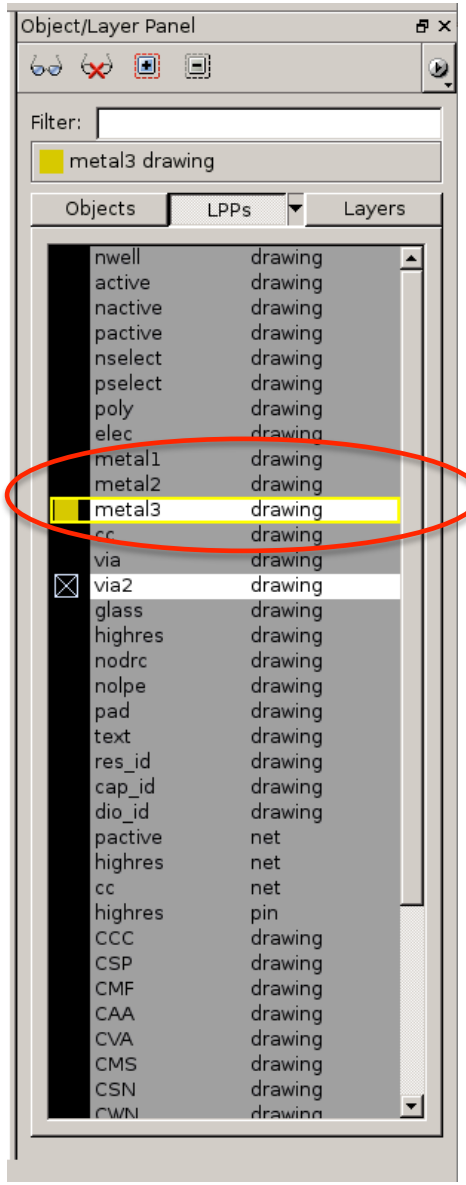






Adding more metal to layers (If Densities not met)

Select Metal 3

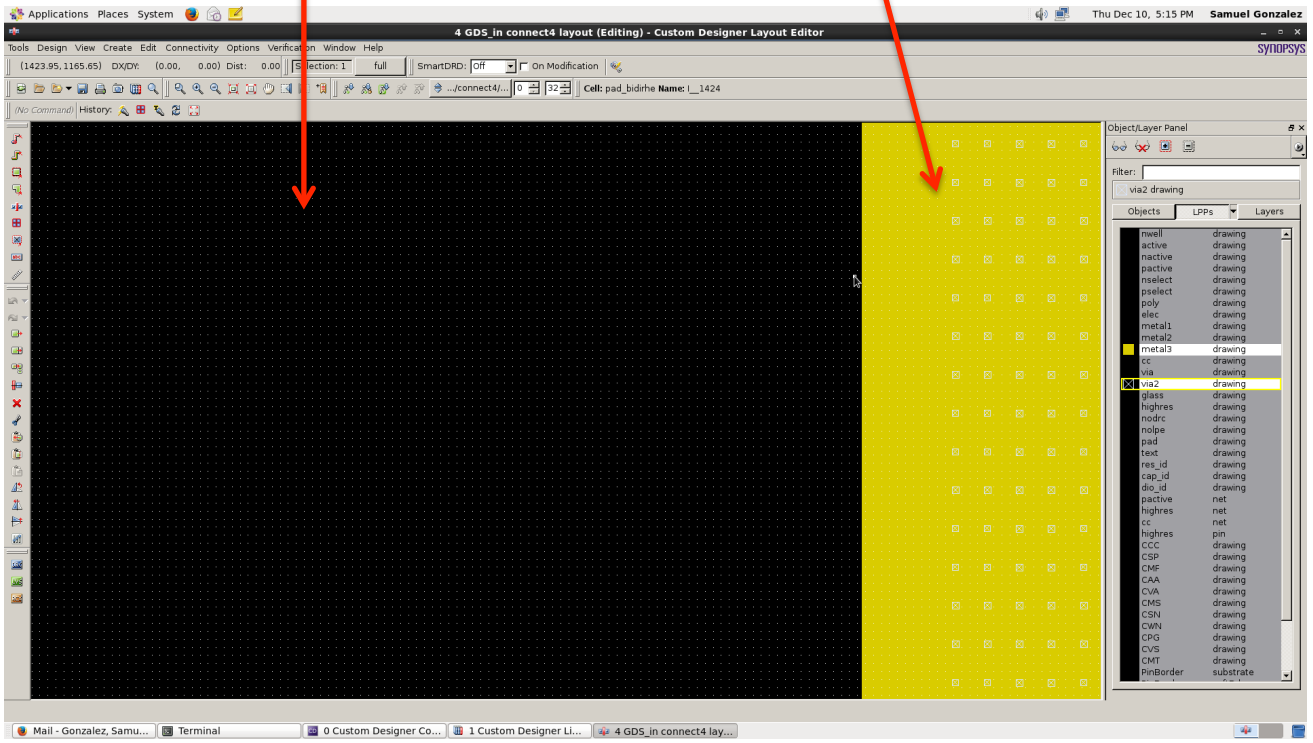


Adding more metal to layers (If Densities not met)

Scroll through the area you intend to add Metal 3 layer to and verify there are no via2 connections.

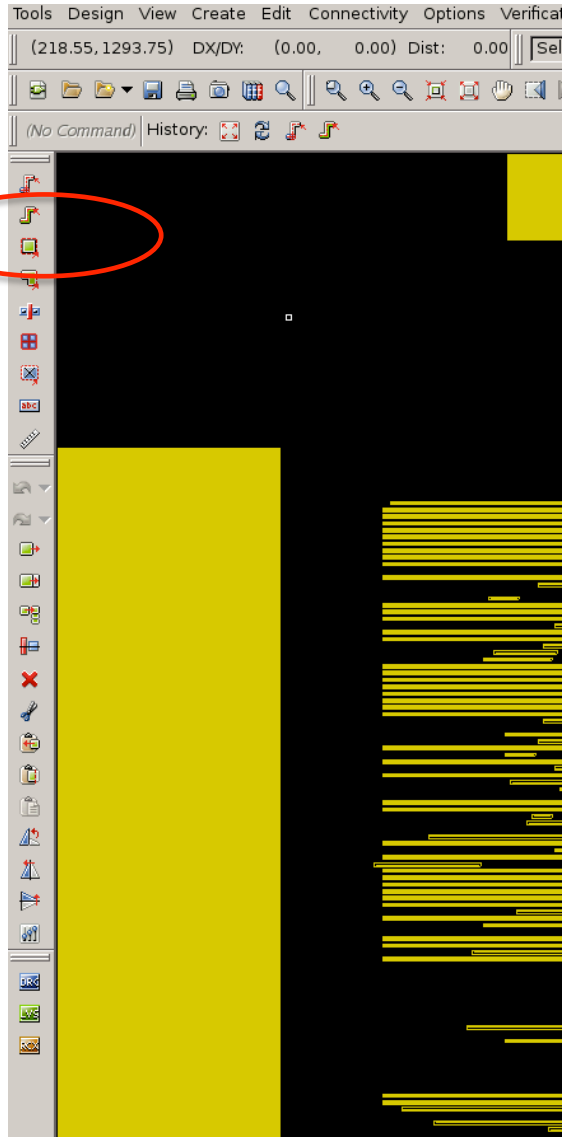
No via2 here

Via2 present here



Adding more metal to layers (If Densities not met)

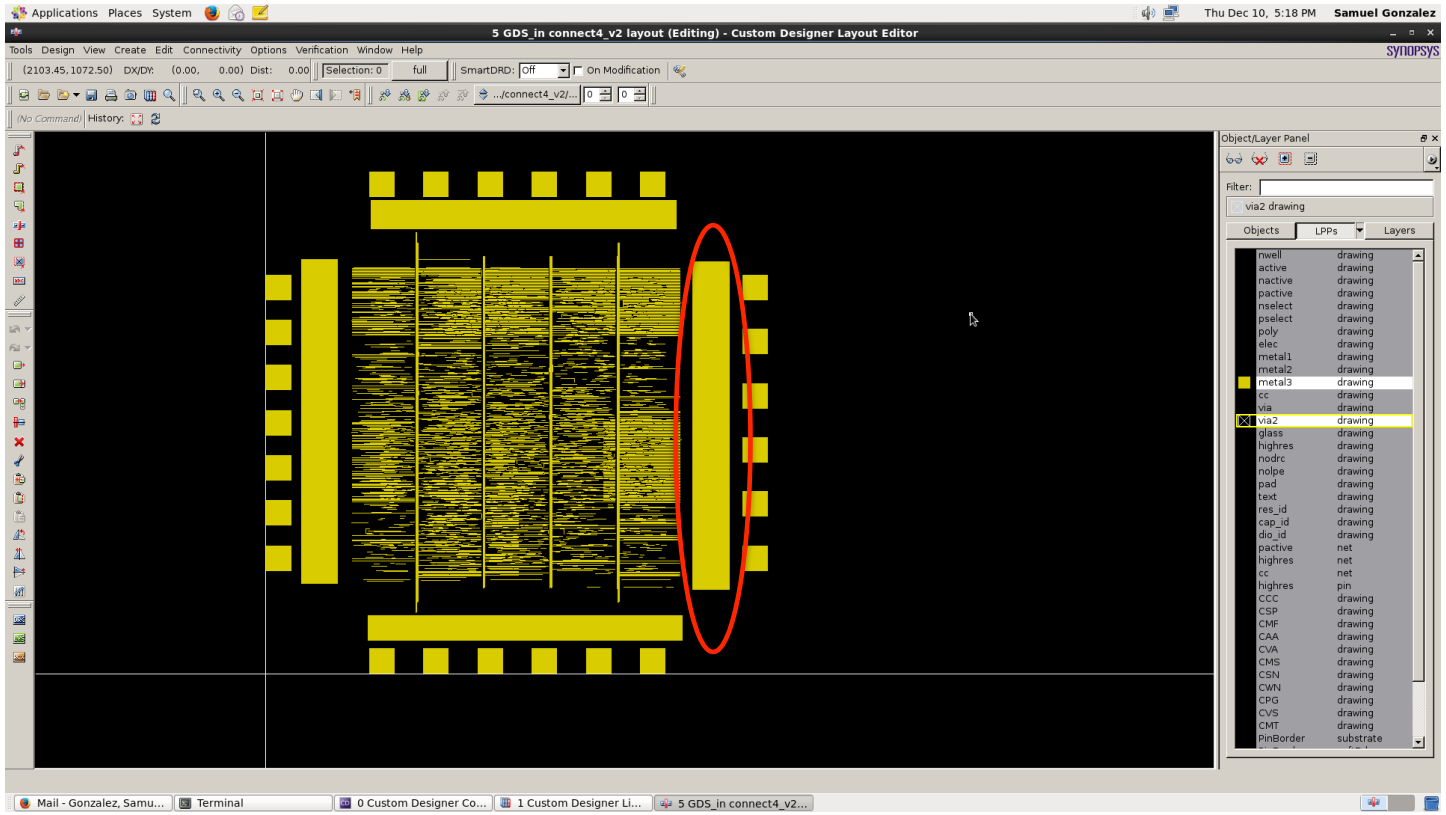
Select path tool or rectangle tool.



Adding more metal to layers (If Densities not met)

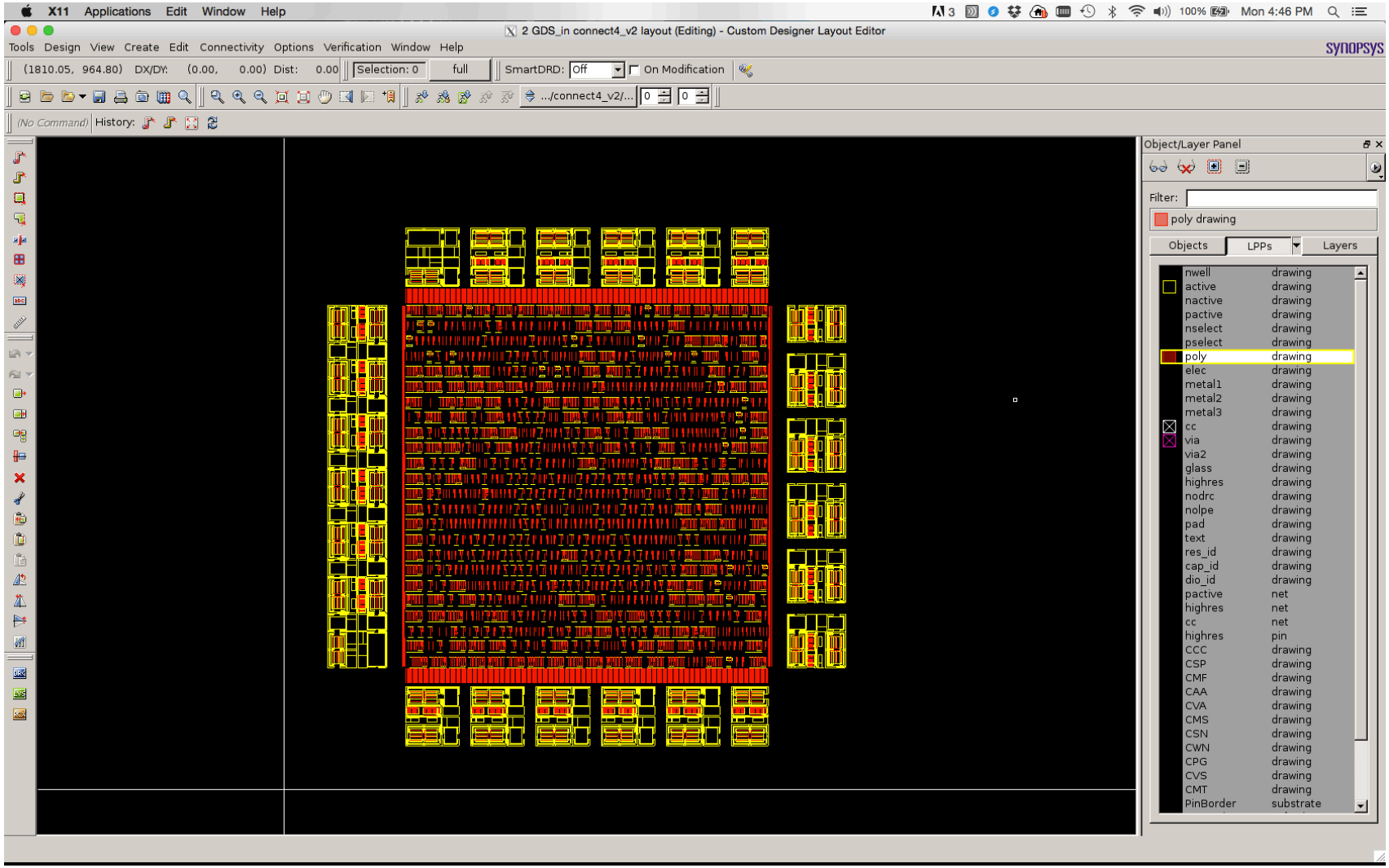
Fill in Metal 3 in area where there is no existing Metal 3 or via2s

This design only had areas available above the pad. Your design may have open spaces within the core where you can add Metal 3 layers. Every chip will be different.



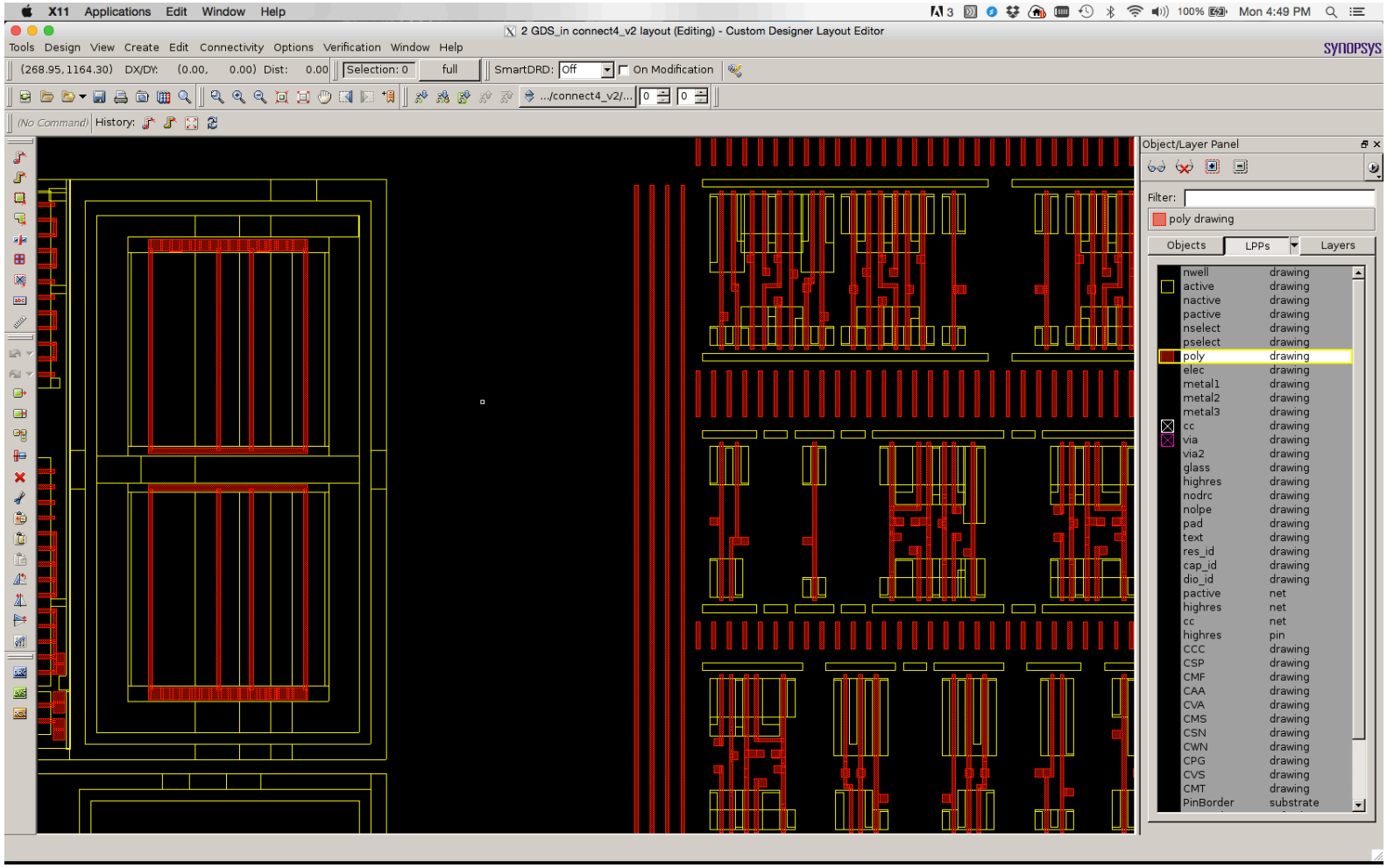
Adding more metal to layers (If Densities not met)

To fix Poly make visible Poly, active, cc, via layers



Adding more metal to layers (If Densities not met)

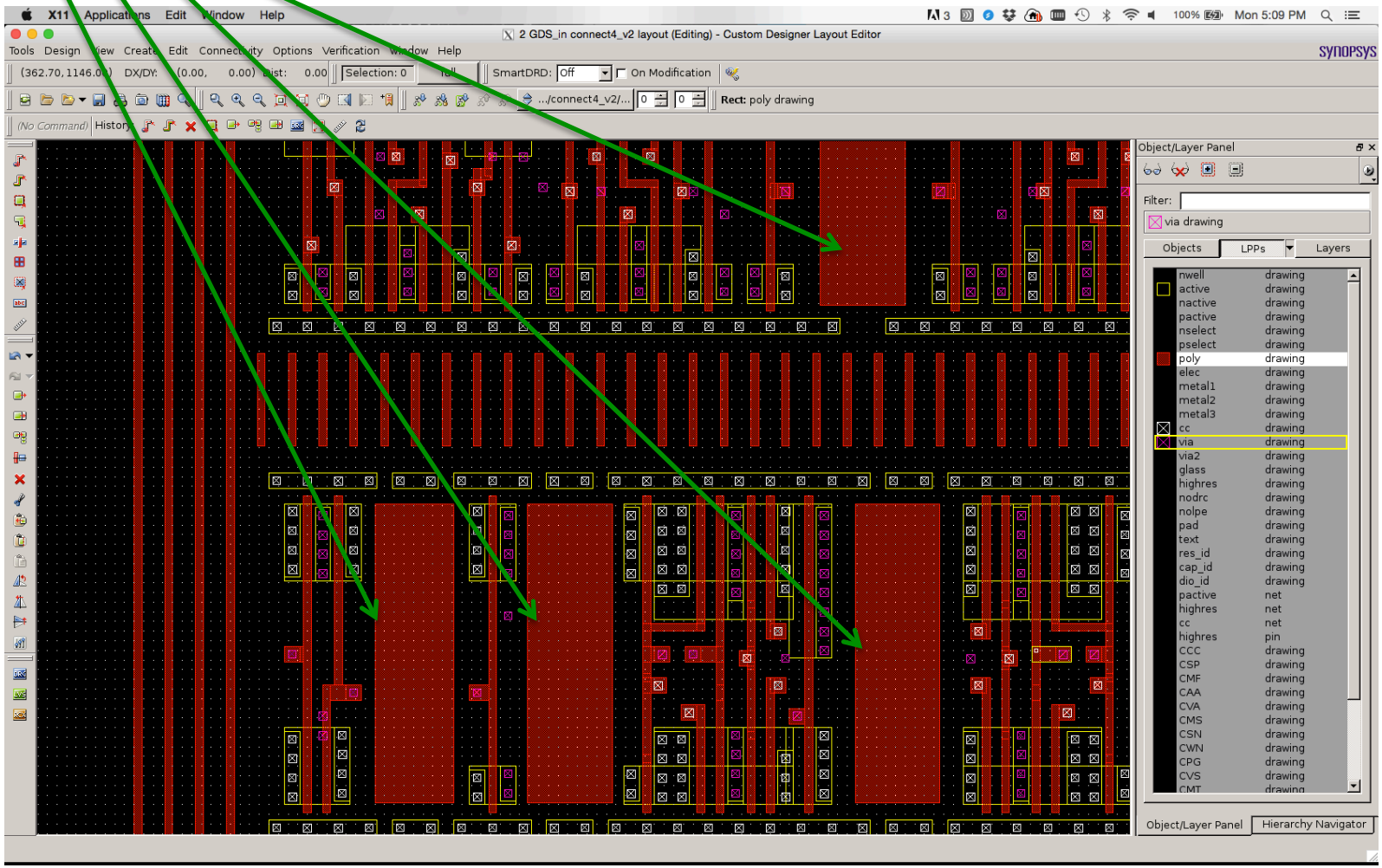
Zoom in to areas where you can add Poly layer with no poly, cc, vias or active areas present.



Adding more metal to layers (If Densities not met)

Zoom in to areas where you can add Poly layer

Added Poly layers



Adding more metal to layers (If Densities not met)

Don’t forget to run DRC again and fix issues if any are present.

Export gds file and run check_density script on new gds file.

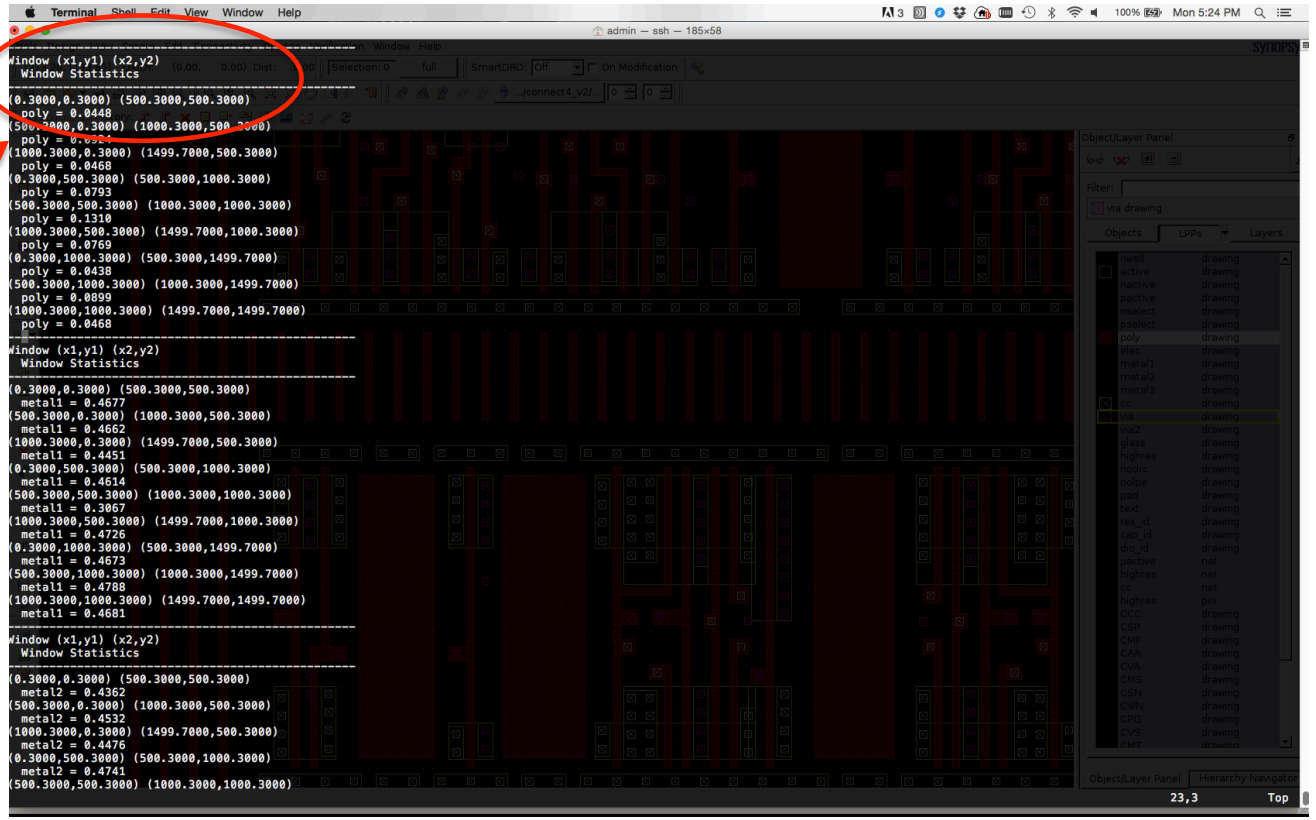
Add additional layers if density is still not being met.

Look at density.txt file to find approximate location where density is not being met.

Location

x-axis
from 0 – 500 micron

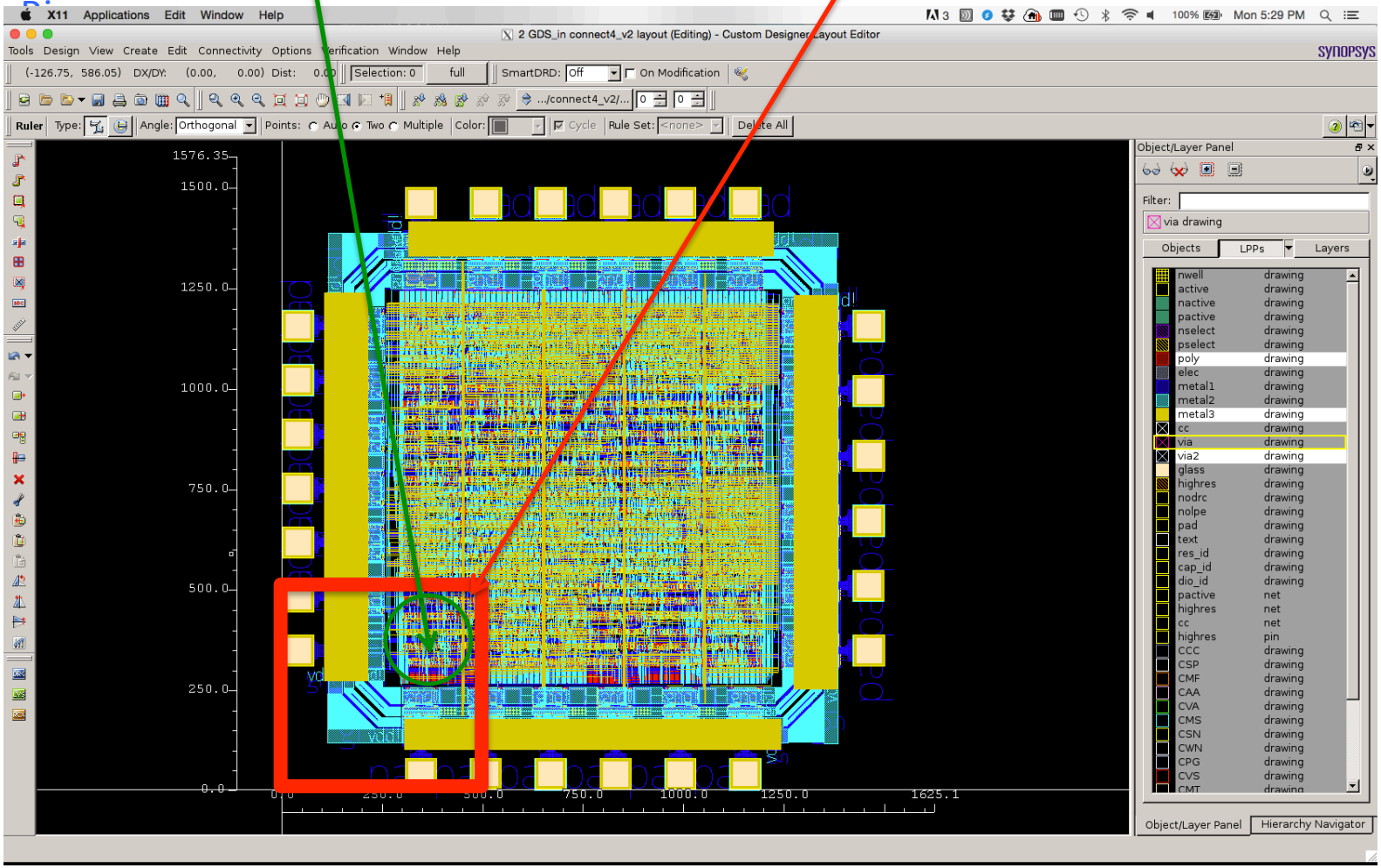
Y-axis
From 0 -500 micron



Adding more metal to layers (If Densities not met)

Location on chip for x-axis 0 – 500 and y –axis 0 – 500

Try to stay inside the core for the Poly fill and avoid going outside the Pad



MOSIS Submission Tutorial for ON Semiconductor 0.5u C5F CMOS

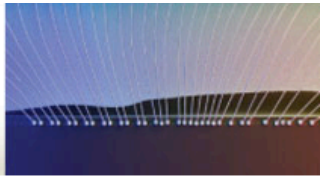
Example
design
and
project details

You will be
provided with
a five digit
design number
and password.

MOSIS Project Management

Logged in as Project: 86820 - xtal_chip

To view a project status report, update or cancel an existing project, queue an IC design for fabrication, queue a previously fabricated design for refabrication or submit a test results report.



Project Information

Design Number	86820
Date Submitted	13-JAN-12 09:43:59 am
Project Status	Shipped
Design Layout	Final layout
Fab ID	V21GBX
Project Document(s)	Download
Fabrication Options	EPI
Run Date Requested	23-JAN-2012
Area	2.223 sq millimeters
Checksum	Binary CRC checksum: 3317540580, Binary CRC byte count: 8839168

Administrative Information

Account Name	5695-MEP-INS/UTFSM-E
Account Contact Name	Agustín González
Account Contact E-mail	agustin.gonzalez@usm.cl
Design Contact E-mail	emac@utep.edu
Design Contact Phone	915-490-3488
Cost	V21G: 1 MEP unit(s) is used.
Quantity Ordered	5
ECCN	EAR99 -- student project for MEP Technology ECCN: EAR99
BIS 711 received	Yes
Approved for export	Yes

Design Details

Size in X	1482
Size in Y	1499.7
Wafer Technology	AMI_C5F
Fabrication Restricted to	AMI
Layout Format	GDS
Top Cell Name	final_chip
Fill	MOSIS
Bonding Pad Count (Customer)	28
Bonding Pad Count (MOSIS)	28
Maximum Die Size	7620.0 X 7620.0
Layers (Density)	ACTIVE, CONTACT, GLASS, METAL1(36.7%), METAL2(35.5%), METAL3, N_PLUS_SELECT, N_WELL, PADS, POLY(19.3%), P_PLUS_SELECT, VIA, VIA2
Needs Library Instantiation	N

Packaging Information

Quantity	Packaged	Package Id	Bonding Diagram Received	Bonding Diagram From	Die Thickness (mils)	Production ID Date Shipped
5	Y	DIP28	08-MAR-12	Customer	10	V21G 02-MAY-12

Special Handlings

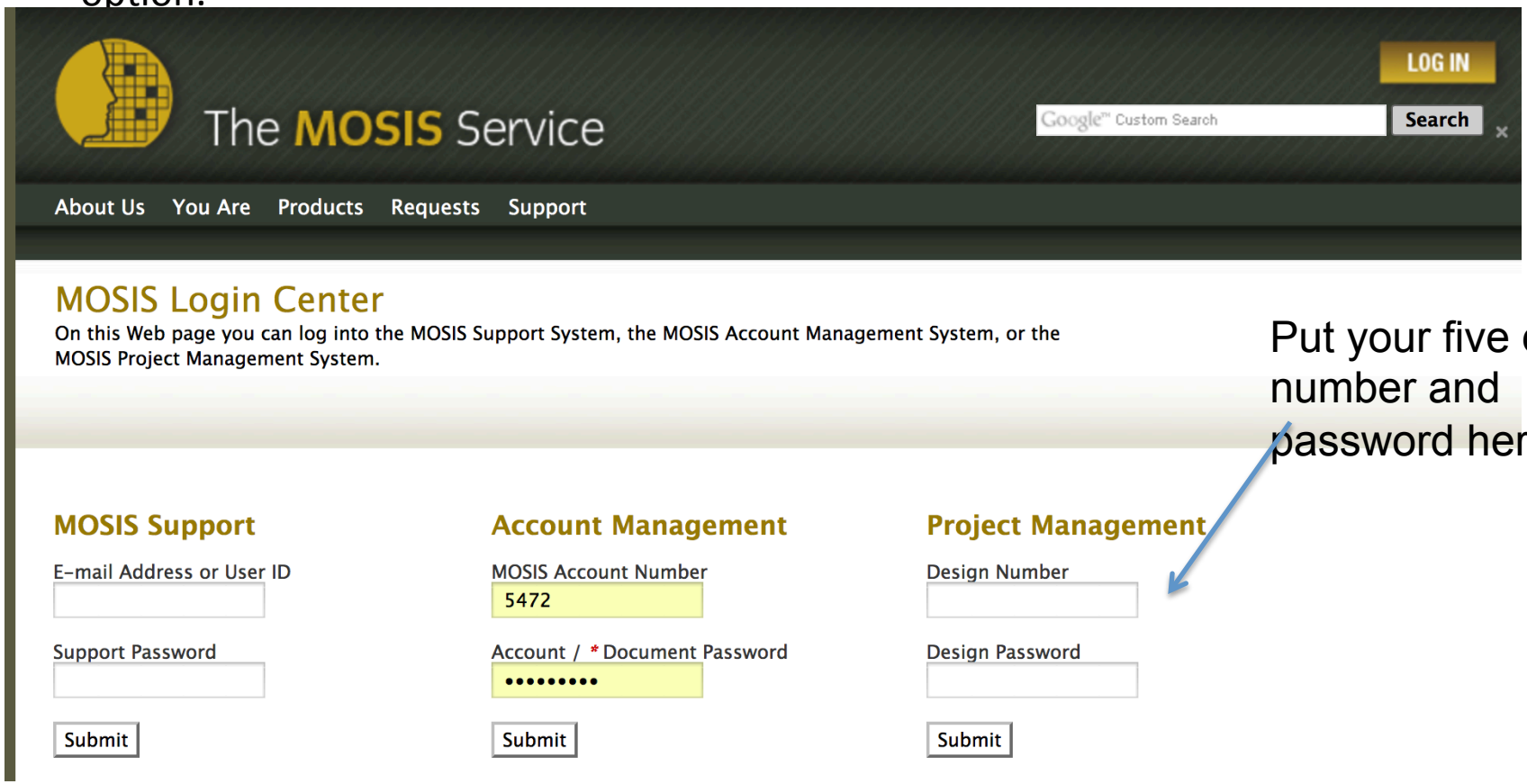
Special Handling	13-JAN-12 09:43 AM	please package all 5 die in 28-pin ceramic DIP packages with taped lids

Overview

After being assigned a design name, number and password go to:

www.mosis.com

Select “fabricate request” and fill in the form for the secure https upload option.



Secure Web Forms

Fabricate Form

Send WebForm comments and suggestions to: webmaster@mosis.com

Use this form to request fabrication of an IC project. You must know the project's Design Number and Design Password. When you have successfully submitted this form, MOSIS will either wait for you to send your layout to us via secure web form (HTTPS) or via FTP or will attempt to retrieve your layout file via FTP, depending on which Layout Transfer Method you selected. Once your layout file is received and the computed checksum matches the one you declared in this form, your design will be queued for manufacturability review. At the conclusion of the check, you will receive a separate e-mail message. If your design passes manufacturability review, it will be queued for fabrication on the next available run that is compatible with your technology code and meets any foundry restriction you have specified. See the MOSIS Fabrication Schedule for the dates of scheduled runs.

If your layout file has been successfully transferred to MOSIS and your project did not get into the fab queue due to missing or incorrect information (pad count, design size estimate, technology code, lambda, etc), you should use the Update Form to add missing information or correct wrong values.

If your project is already in the fab queue and you want to submit a revised design, you must first cancel fabrication with the Cancel Fabrication Form.

For more detailed information, see the Fabricate Documentation.

Required Parameters

Design Number: 86791 *(5-digit or 6-digit design number assigned to project by MOSIS)*

Design Password: ••••• *(Password you chose at project creation)*

Use your design number

Layout Parameters

Layout Status: (•) Final (ready for fabrication) ( ) Preliminary (not to be fabricated)

Layout Format: ( ) CIF (•) GDS (Select one - see MOSIS Layout Conventions)

Compression/Encryption: (•) Uncompressed ( ) Compressed ( ) PGP-encrypted

Checksum Type: (•) CRC ( ) Traditional "CIF"

Checksum: 2011636139

Count: 741376

Run the crc program on your gds for these two numbers

Top Structure: xtal_chip *(Required only for GDS format)*

Select Layout Transfer Method

Layout Transfer Method: I will upload layout via secure web form (HTTPS) *(Fill out corresponding section below)*

To Send Your Design to MOSIS via Secure Web Form (HTTPS)

HTTP Send Hours: 168 *(Optional hours until expiration - default is 8 hours)*

To Send Your Design to MOSIS via FTP

For help, see the Design File FTP Server FAQ. Server is at ftp.design.mosis.com.

FTP Send Host: *(Name or IP address of host you will send from)*

FTP Send Password: *(Password you will give to ftp server at login)*

Retype Send Password: *(Retype password for verification)*

FTP Send Filename: *(Name of file you will put on ftp server)*

FTP Send Hours: *(Optional hours until expiration - default is 8 hours)*

To Have MOSIS Retrieve Design from You via FTP

FTP Host: (See Submitting Designs Via FTP)

FTP Username: (User name for login, such as "anonymous")

FTP Password:

FTP Filename:

Project Packaging

Fill out this section only if you have not specified packaging parameters yet or if you want to change the previously specified parameters. A new bonding diagram will be generated (if applicable) if there are any non-blank fields in this section. See Project Packaging for detailed help for this section.

	Quantity	Package Name	Rotation in Package	Bonding Diagram Supplier	Downbond Locations
Quantity Packaged:	5	DIP28	Unchanged	Customer	Unchanged (or N/A)
Quantity Unpackaged:		N/A	N/A		

Required if Not Previously Provided

Run Type: Unchanged (Select one)

Run Date Requested: (For non-dedicated runs only. Must match a scheduled date. Example: 24-apr-2011)

Run Name: (Name of previously created dedicated run. See the Create Dedicated Run form.)

IP Included: (Vendor of licensed intellectual property in design (e.g., ARM). Select all that apply.)

☑ Unchanged ☐ None ☐ Aragio ☐ ARM ☐ Virage
☐ Other (Specify in Special Handling)

Fill Authorized: Unchanged (Only applicable to TSMC and ON Semi)

Technology Code: (See Technology Codes and Layer Maps for a list)

Lambda: Unchanged (Select one)

Foundry: Unchanged (Select one)

Substrate Options: Unchanged or N/A (Only for those processes that offer a substrate choice)

Top Metal Options: Unchanged or N/A (Only for TSMC 0.18 and 0.25 processes)

Bonding Options: Unchanged or N/A (Only for IBM processes)

Bumping Options: Unchanged or N/A (Only for IBM C4 (leaded or lead-free) processes)

Intended Disposition: Unchanged (Required by export control regulations)

Design Kit Version: (For IBM design technologies only)

Design Size X: 1500 (In microns)

Design Size Y: 1500 (In microns)

Pad Count: 28

PO Number: (For commercial accounts and MEP packaging)

Quote Id: (For assigned quote or MEP proposal ID only)

Add or Update Information (Optional)

E-mail Address: emac@utep.edu (For confirmations from MOSIS)

Phone Number: 915-490-3488

Design Name: xtal_chip (Name you choose for this project)

Quantity Ordered: 5

Description: (Paragraph describing project)

student project in the MOSIS educational program

Use your email address, phone number and design.

Add or Update Information (Optional)

E-mail Address: emac@utep.edu *(For confirmations from MOSIS)*

Phone Number: 915-490-3488

Design Name: xtal_chip *(Name you choose for this project)*

Quantity Ordered: 5

Description: *(Paragraph describing project)*
student project in the MOSIS educational program

Routing Label: *("Deliver-to" address WITHIN your company)*
Envia a Agustín Gonzalez en la departamento ingeniería electónica

Special Handling: *(Special packaging, shipping or die size requirements)*
package all 5 die in 28 pin ceramic Kyrocera DIP packages with taped lids

Put any information that will make receiving the chips easier

Submit Form Reset Form

Last Updated: September 19, 2005