ECE 4150/6250 – Cadence Tutorial: Using Cadence Chip Assembly Router – Automatic IC Routing Tool

Objectives:

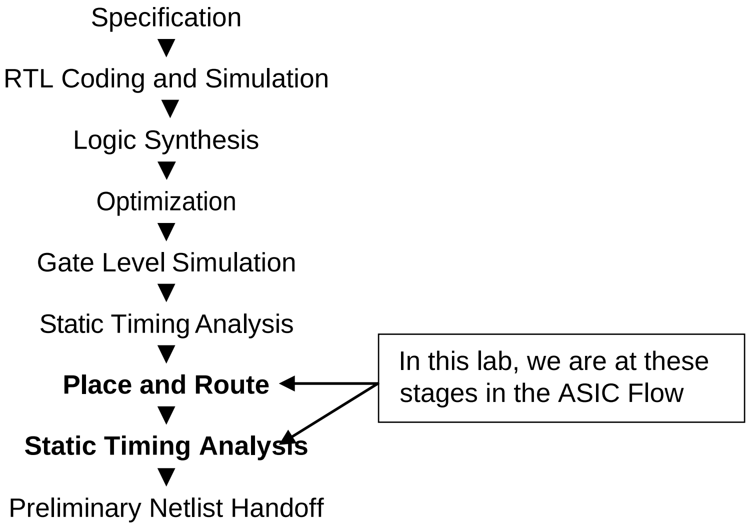
- Automatically route multiple modules together into a single core.
- Automatically route the core to a padframe.
- Verify our routed designs matches our schematic designs.

Assumptions:

- Student has a placed and routed the ripple-carry adder w/scan cells from Lab 7.
- Student is familiar with Cadence Virtuoso Layout tools.
- Student is familiar with Cadence Schematic Capture tools.

Introduction:

The ASIC design flow is as follows:



Specification
▼
RTL Coding and Simulation
▼
Logic Synthesis
▼
Optimization
▼
Gate Level Simulation
▼
Static Timing Analysis
▼
Place and Route
▼
Static Timing Analysis
▼
Preliminary Netlist Handoff

In this lab, we are at these stages in the ASIC Flow

In this lab we will use a Cadence Product Called: Chip Assembly Router (ccar). This product is capable of taking modules that already have been laid out and routing them together. This tool is able capable of routing modules to IC padframes. This is an extension of the place and route process that we have seen with Silicon Encounter. We will not be using this tool to automate the placement of cells or floorplanning of a design, but rather to automate the interconnection thereof. Many other tools exist in the industry to accomplish the same task.

After we have routed modules together, we will perform standard verification checks on them – this would be DRC, extraction and LVS. LVS is necessary, since we will be using schematics to define the interconnections between the modules we use.

We will be routing together a pair of ripple-carry adders w/scan chain modules in order to create a two-bit adder circuit. After we have created and verified this module, we will then connect that circuit into a padframe, and repeat the routing process with in the padframe. After placing and routing, we will perform verification, via LVS, against our schematic.

Note: You can change the suggested "ece4150_6250" folder name to your own one in the labs thereafter.

Part I: Setup Cadence Libraries (10 mins)

1. We will need one file, in order to run a setup script for CCAR. Copy and paste the following command in order to copy the script from the VSLI account:

```
cp /home/seas/vlsi/course_ece128/lab_files/lab8/do.do ~/cadence
```

2. Change to your cadence working directory `cd ~/cadence`

3. Start Cadence Virtuoso by typing the following commands:

```
virtuoso &
```

4. Create a new AMI 0.5 Library for later use:

- From the Library Manager window, select: File->New Library
- In the dialog box, select:
  - ◦ Name: lab8_adder
  - ◦ Path: (leave blank)
- Attach technology Library: **NCSU_TechLib_ami06**
  - ◦ or, if you do not see that in the list, look for: **AMI 0.6u C5N (3M, 2P, high-res)**
  - ◦ *Note: Be very sure to attach the 0.6u library (and not accidentally the 1.6u) as this will cause you work in this lab to fail*
- Press OK and ensure the **ripplecarry4_clk** library (from lab 7) is in the Library Manager's list of Libraries.
- We will use this library later in the tutorial.

5. Add UofU Padframe (AMI .5 technology) Library to Cadence
   - From the CIW Window (the small window at the bottom), select: Tools → Library Path Editor
   - On the bottom line, create a new entry:

Library Name:	UofU_Pads
Path:	`/home/seas/vlsi/course_all/UofU/UofU_Pads_oa3`

- **Note: The library name must be typed EXACTLY as above, double check this before continuing. An incorrect entry will appear to work, but later problems will surface**
- From the Library Path Editor menu, choose: **File → Save As** and say make sure to check "lib.defs". Save the changes, if prompted to overwrite any files, click YES to save the changes
- In the Library Manager, click on the UofU_Pads library to ensure it has padframe in it. These cells contain layout, schematic and symbol views. These are all necessary to use CCAR.

Part II: Core-Core routing: Route together the two bit adder (50 mins)

Overview of Part II:

We will use the Cadence tools to do the following:

1. Make a **schematic** that contains all of our modules wired together. This includes IO pins.
2. Using *Virtuoso-XL*, we will generate a new **layout** view, which will be based on that schematic.
3. Place our modules by hand in the new layout view. This is floorplanning.
4. Place our IO pins which we defined in our schematic.
5. Connect our power network, vdd! And gnd!, together by hand.

6. Invoke *CCAR* in order to route our design together.
7. Save our routed design and perform verification on the design.

Note: It is assumed that modules will be routed together only after they have been verified through simulation. You cannot route blocks together which you have not functionally verified.

1. Open up the **schematic** view for the ripplecarry4 adder we created in lab 7. This should be under the library **ripplecarry4_clk**. We need to generate a symbol view, in order to create schematics with this module.

2. Once you have opened up the schematic, follow the menu options **Generate → Cellview → From Cellview** in order to generate a symbol view for your ripplecarry adder. Leave all of the defaults as they are. After the **symbol** view is generated, save the view and close it.

3. In the Cadence Library Manager, create a new cell in your lab8_adder library. This needs to be a **schematic** view. You can name this "**adder2**". After you have instanced them together, you will need to wire them up as a two cascaded adders and assign pins. See the following figures for examples of what your schematic should look like. When you are finished generating your schematic, check and save your schematic but do NOT close your schematic.

   Note: We are using WIDE wires for this example, large designs you will want to properly use WIDE wires for buses.

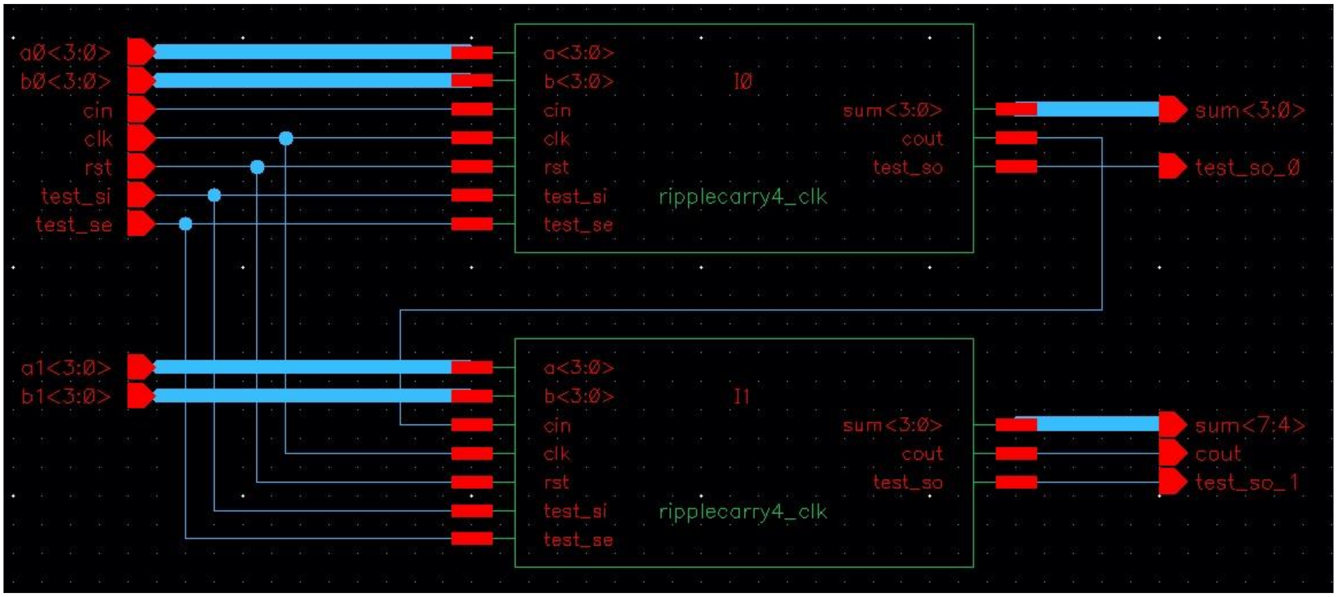


Figure 1 Adder2 schematic

4. From your schematic view, go to **Launch → Layout XL.** This will launch Virtuoso XL, which is a more advanced layout engine than what was used in ECE126. Accept the default startup options. This will automatically create a **Layout** view of your adder2 cell.

   If prompted to accept a license file, click yes to continue.

5. Your layout view is initially blank, so we need to tell Layout XL to automatically generate your layout. Go to **Connectivity → Generate → All from Source**, and accept the defaults for the Generate Layout dialog. This will create two instances of your ripplecarry adder, a blue **prboundary** box and some pin nets. These pin nets are located near the bottom left of the prboundary. You'll need to zoom in closely in order to see them.

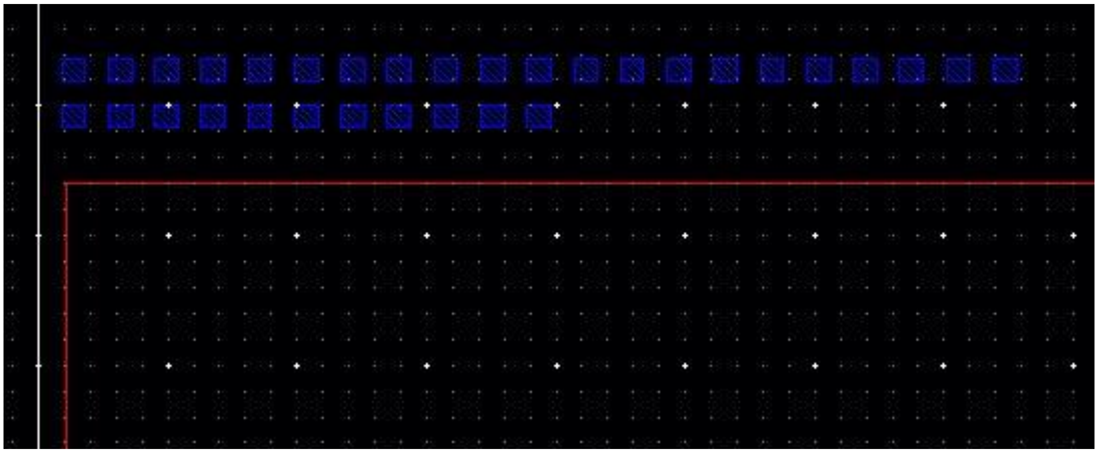


Figure 2 Automatically generated pins

6. Your job is to move your ripplecarry4 blocks into the prboundary block, along with your IO pins. You'll notice when you move your ripplecarry adder modules, lines appear which show connections between the modules and pins. Move and rotate modules as necessary, and move pins as needed.

7. For simplicity, place the input pins to the left of the modules and the output to the right of the modules. The property viewer, brought up by the Q key, will be helpful here. Your block should look something like the following when you are done.

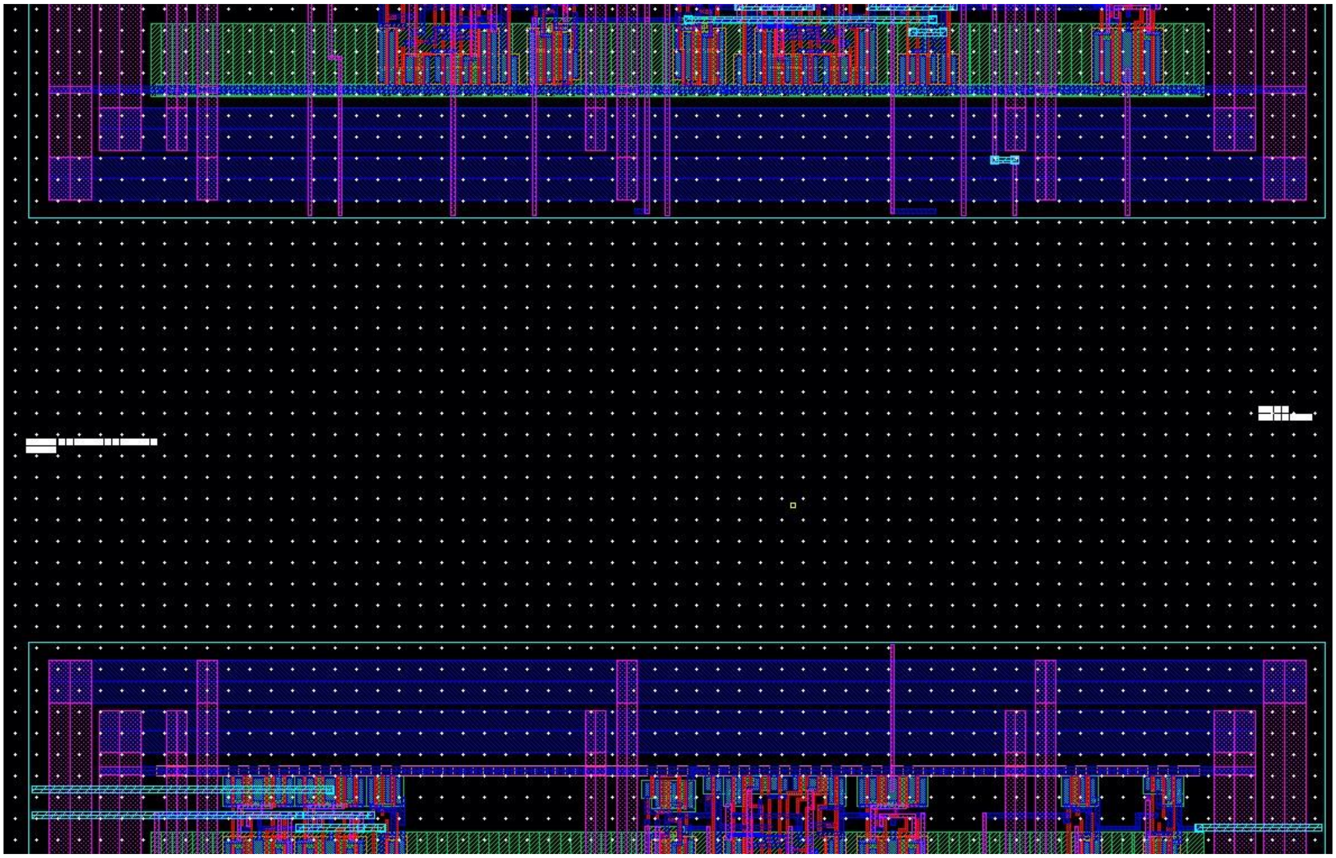


Figure 3 Floorplanned Modules, showing pins on the left and right.

8. If you wish, you may shrink the prboundary box. This will reduce the overall footprint of your design. Keep in mind that this is the boundary that CCAR will use when routing though, so if you shrink it too small then your design may not be able to be routed.

9. Now you'll have to connect power and ground nets together by hand. Using metal 1 and metal 2 as appropriate, route your power and ground rings together.

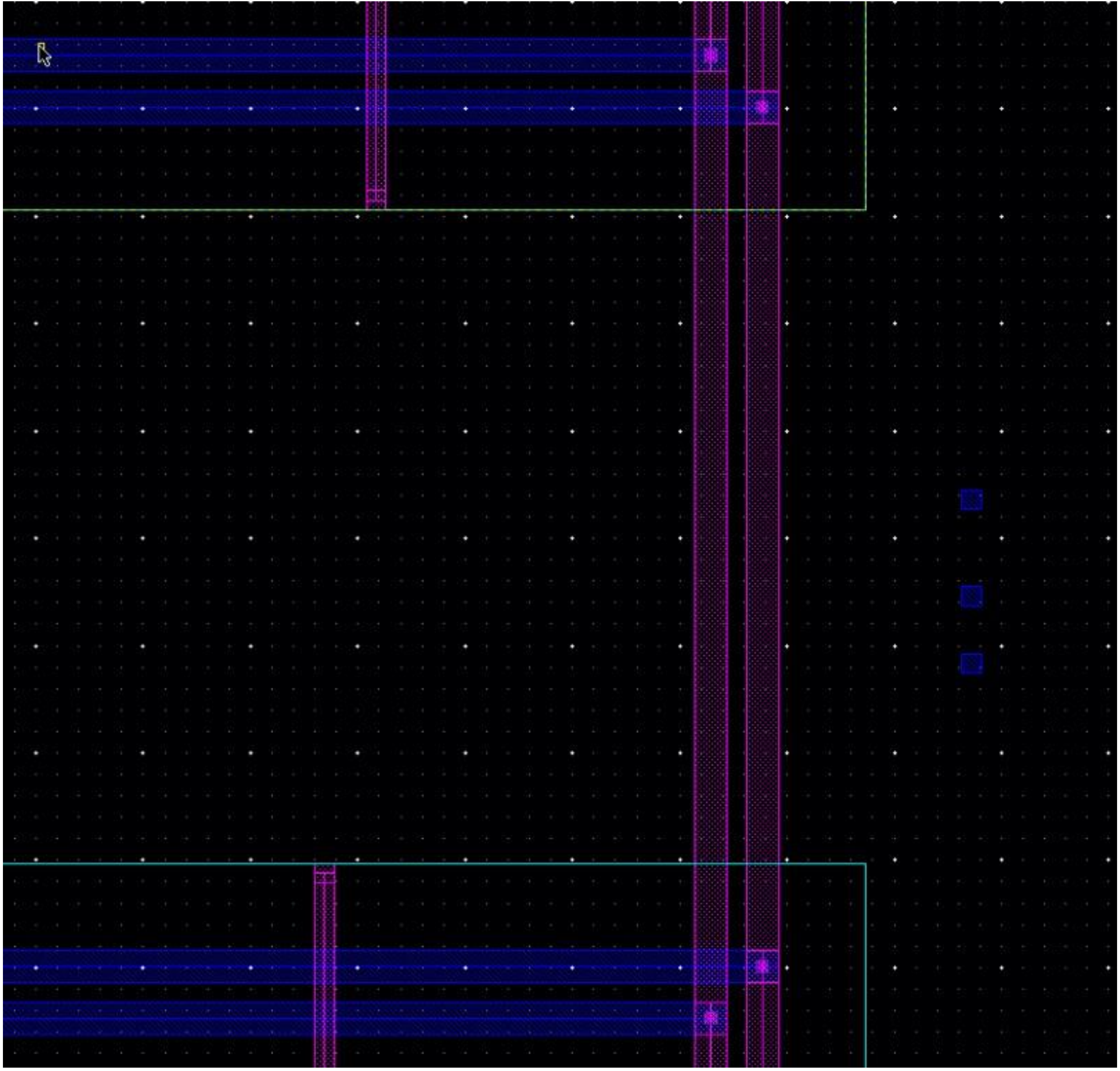


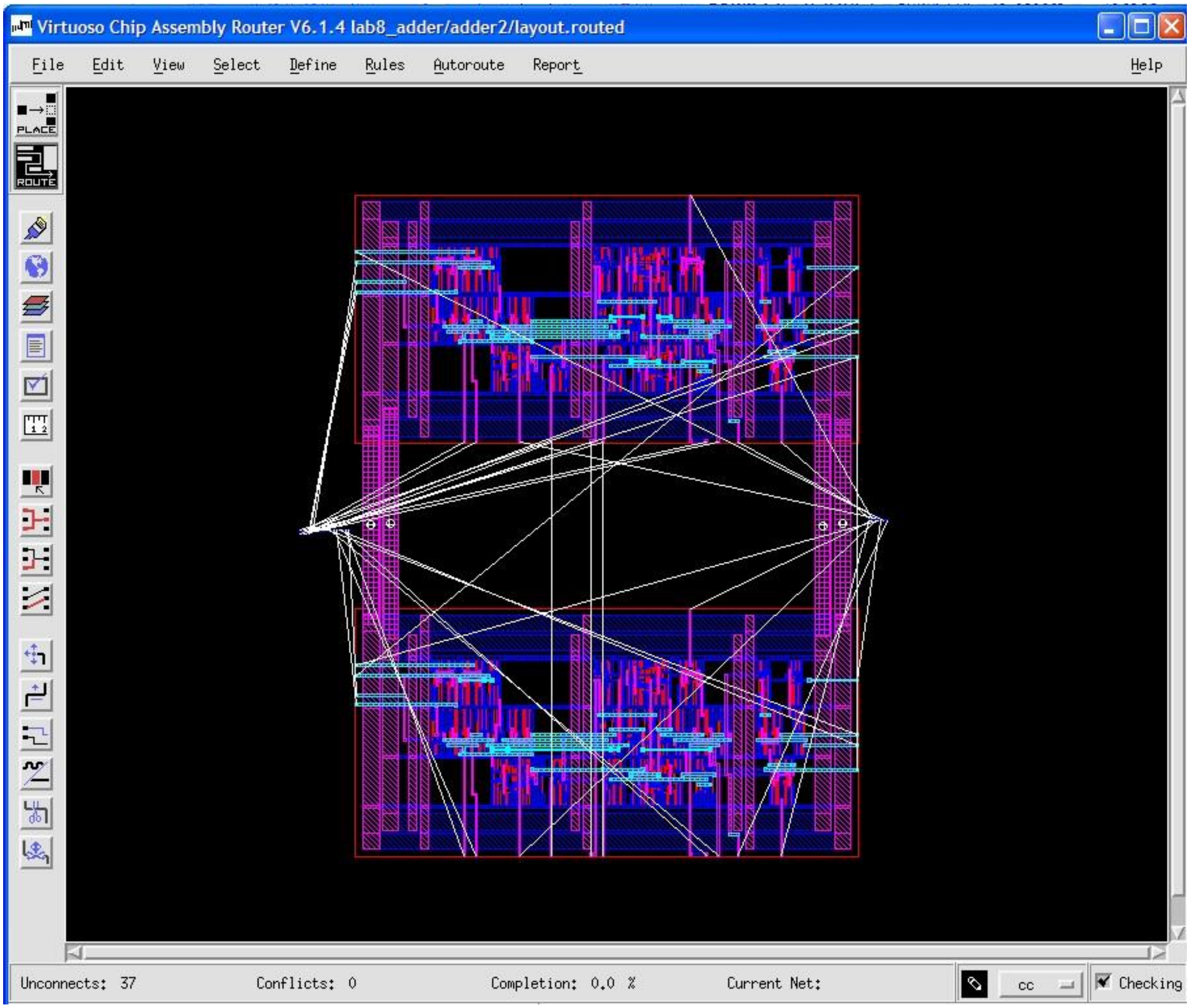
Figure 4 Power and Ground Routing

10. After adding power and ground routing, double check your power and ground nets to make sure that they are not bridged. Bring up the Mark Nets tool by going to **Connectivity → Nets → Mark**. Click on one of the power rails, and you should see that entire net become highlighted. If you have a bridge, you will see both power nets highlighted at the same time. If that has occurred, you will need to remedy that situation before continuing further.

11. Before continuing, make sure your modules, pins and PRBoundary are where you want them to be. You can certainly change quite a bit here, but if you restrict yourself too much then CCAR may not be able to route your design and you will have to floorplan your design again. Be certain to save your layout at this point.

12. At this point, we are ready to invoke CCAR. In order to do this, we must first enable CCAR. First, go to **Launch → Layout XL.** Then, go to **Window → Toolbars → Chip Assembly Router** and a new pane will come up in bottom of your Layout window.

13. Go to the bottom of your layout window and choose **Start Router** in order to start CCAR. You can accept the default settings. When CCAR starts up, you should receive an OA Read Messages warning dialog box. You should have a warning for your module, and 5 warnings about viaDefs. You can ignore these warnings.



Unconnects: 37 Conflicts: 0 Completion: 0.0 % Current Net:

Figure 5 CCAR Window

14. We'll need to initialize a few items in CCAR. Go to **File → Execute Do File** and browse for the do.do file we imported to our ~/cadence directory in **Part I.**

15. Next, we'll need to tell router how to use the metal layers to route our design. Click the layers button on the left side of the screen, to bring up the layers window.

16. Set Metal3 as Horizontal routing, Metal 2 as Vertical routing, and Metal 1 as Horizontal routing. Leave other layers alone, since we do not wish to route our design with Vias or Poly.

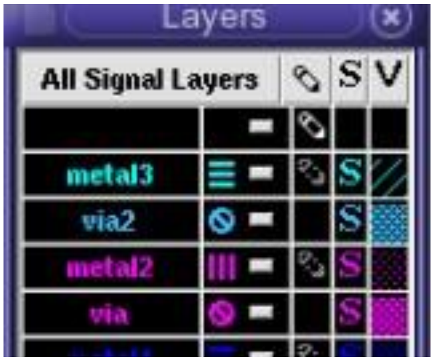


Figure 6 Layers setup for routing

17. We'll now need to add routing costs. Go to **Rules → Costs/Taxes** and set "Wrong Way Routing" to 2. This will give the router a cost penalty for misusing a metal layer.

18. Normally you want CCAR to route all of your signals. If you have nets which you do not wish to have auto routed, you can go to **Edit → [Un] Fix Nets** to change the routing status of your nets. We will not be editing anything for this tutorial under that menu.

19. Now we can have CCAR route our nets for us.

- Go to **Autoroute → Global Route → Local Layer Direction** and tell it to get its layer directions from the **Layer Panel.**
- Go to **Autoroute → Global Route → Global Route** and tell the router how many passes you'd like it to try before giving up. The default, 3, is fine for our design.
- Go to **Autoroute → Detail Route → Detail Router** in order to do your wire detailing. You can leave the number of passes at the default, 25, for our design. This part is fun, since you can see CCAR trying out all the different routes. If this doesn't work, you may need change your number of passes, routing costs or to go back to Layout-XL and redo your floorplan.
- Go to **Autoroute → Clean** in order to clean up messy parts of your routing.
- Go to **Autoroute → Postroute → Remove Notches** In order to remove little gaps or other tiny features which are unnecessary.

See the figure below to see a routed core-core design.

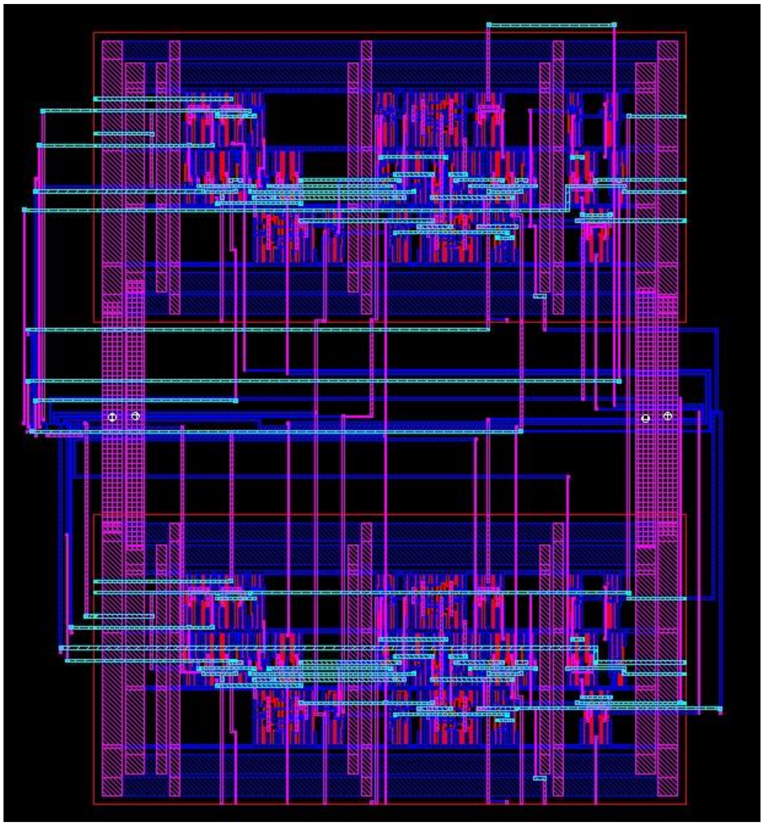


Figure 7 Routed Core-Core design.

20. Go to **File → Save** to save your design. You can save it with the defaults and then exit out of CCAR.

21. Your new layout.routed cellview should be automatically opened in Layour-XL. Perform DRC and Extraction on this cellview. Open up your extracted cell, and then perform LVS between your adder2 schematic and extracted view.

    If your netlists fail to match, you'll need to identify and correct the issue before continuing on.

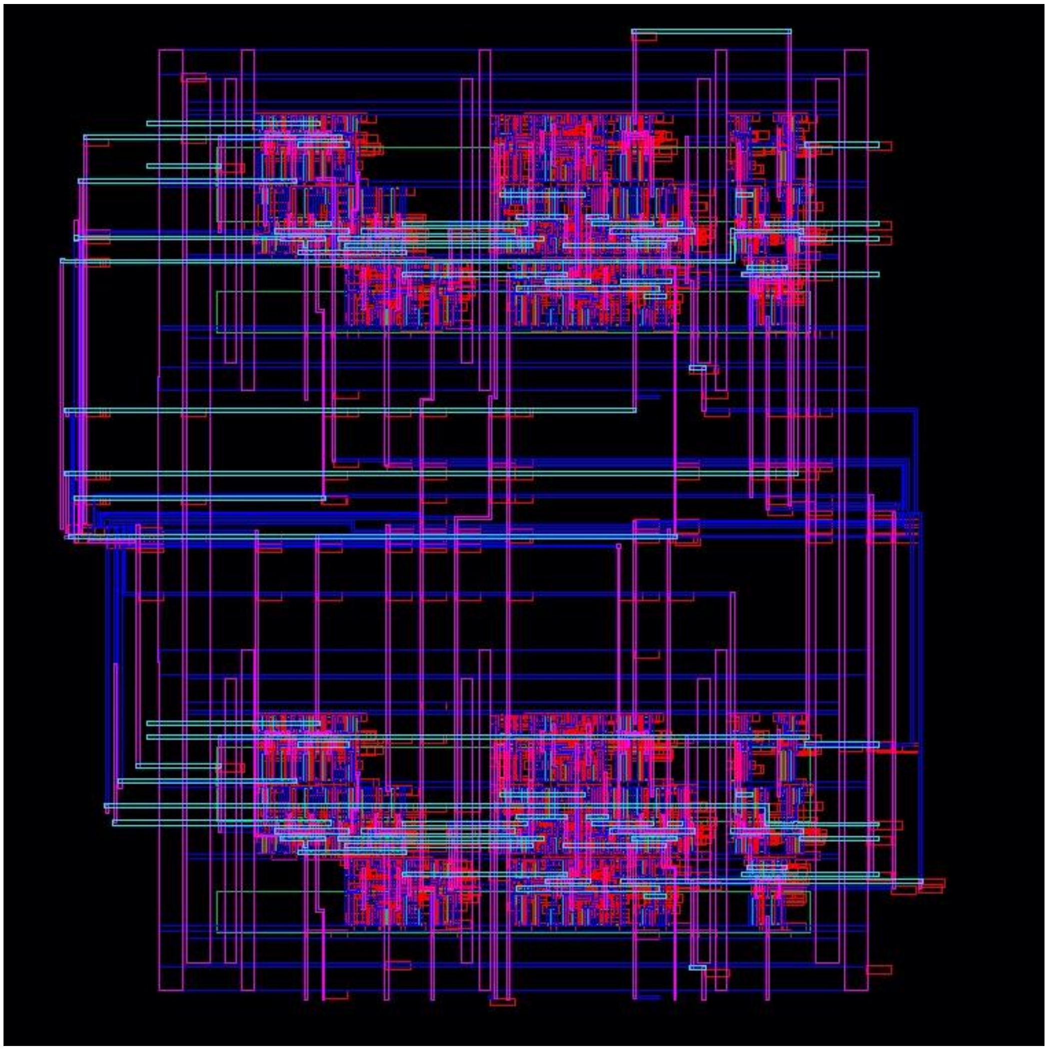


Figure 8 Extracted adder2

Part III: Core-Pad routing: Route the adder2 module to padframe (50 mins)

Overview of Part III:

We will use the Cadence tools to do the following:

- Make a **schematic** that contains all of our padframe with IO pins.
- We will generate a **symbol** of that schematic.
- We will then wire up the padframe and adder2 modules together in a new **schematic**.
- Using *Virtuoso-XL*, we will generate a new **layout** view, which will be based on that schematic.
- Place our modules by hand in the new layout view. This is whole-chip floorplanning.
- Connect our power network, vdd! And gnd!, together by hand.
- Invoke *CCAR* in order to route our design together.
- Save our routed design and perform verification on the design.

1. Close and reopen virtuoso from the same terminal window. This will resolve an LVS licensing error we will otherwise encounter at the end of **Part III**.

2. Open your schematic for your adder2 modules in your lab8_adder library, and generate a symbol view from that cell. We'll need that later on in **Part III**.

3. You'll want to copy a padframe from the UofU_Pads library to your **lab8_adder** library. We will be using the Frame1_38 module in our design. There are a few different padframes available to you though.

   - Frame1_38: 1 Tiny Chip unit frame, 900x900 microns interior area.
   - Frame2h_70: 2 Tiny Chip unit frame, 2300x900 microns interior area.
   - Frame2v_70: 2 Tiny Chip unit frame, 900x2300 microns interior area.
   - Frame4_78: 4 Tiny Chip unit frame, 2300x2300 microns interior area.

   Select the Frame1_38 cell, and right click on it and select **Copy**. The copy dialog will open up, and make sure that you select your local library, lab8_adder, as your destination.

4. You'll now need to modify your local copy of Frame1_38 to have the needed IO pins available. By default, these padframes only have VDD and GND connections already made, all other pads are Noconnect. The different types of pads available are as follows:

   - **pad_bidirhe**: Bidirectional pad with a high-enable signal.
   - **Pad_in**: An input pad, connecting signals from the pads to your core. A pair of outputs are available, DataIN and DatainB. **Pad_out**: An output pad, connecting signals from your core to the pads. The port you'll be using with this pad is DataOut.
   - **Pad_nc**: This pad does not connect to your core. All pads which are not used should be left as pad_nc types.

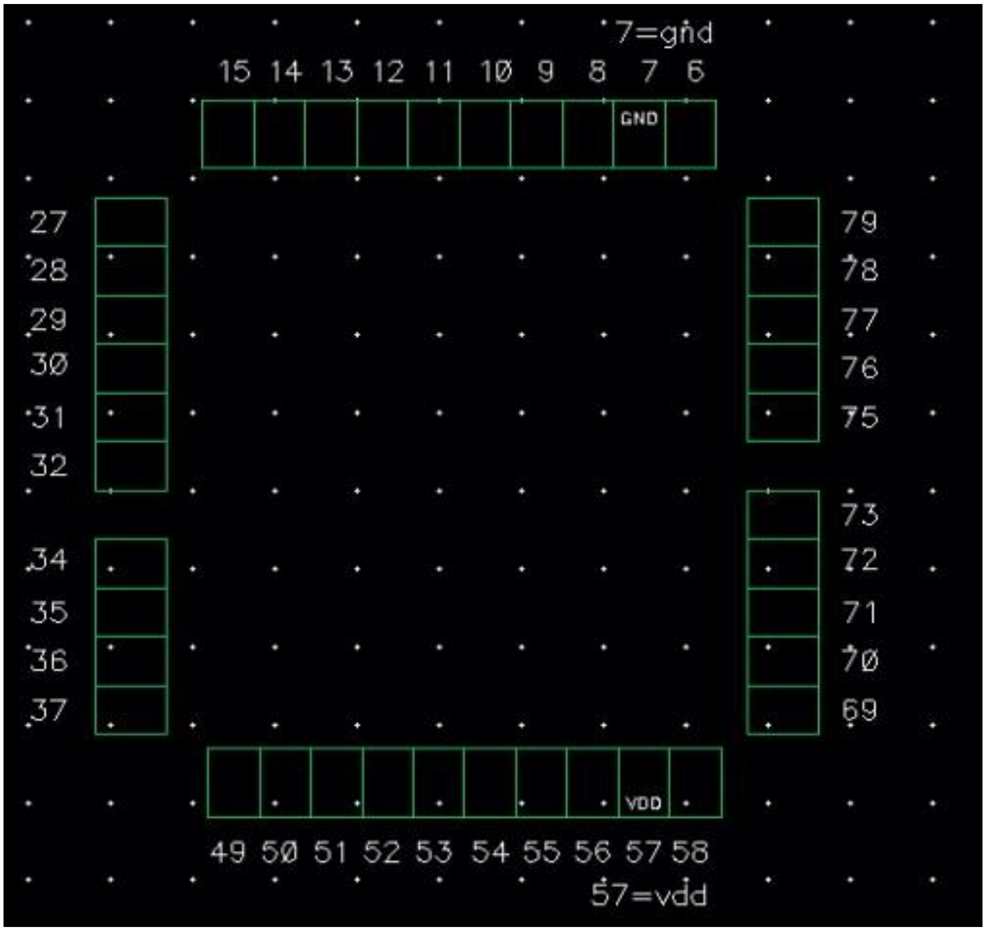


Figure 9 Padframe Prior to Editing

5. You will want to open your copy of Frame1_38 and modify the numbered pads to be the following types. To save time for this lab, we only route part of the inputs and outputs. Use the property editor (type Q to use the tool) in order to change the pad types.

Pin Name	Pad Number	Pad Type
cin	27	pad_in
clk	28	pad_in
rst	29	pad_in
test_se	30	pad_in
test_si	31	pad_in
cout	79	pad_out
test_so_0	78	pad_out
test_so_1	77	pad_out

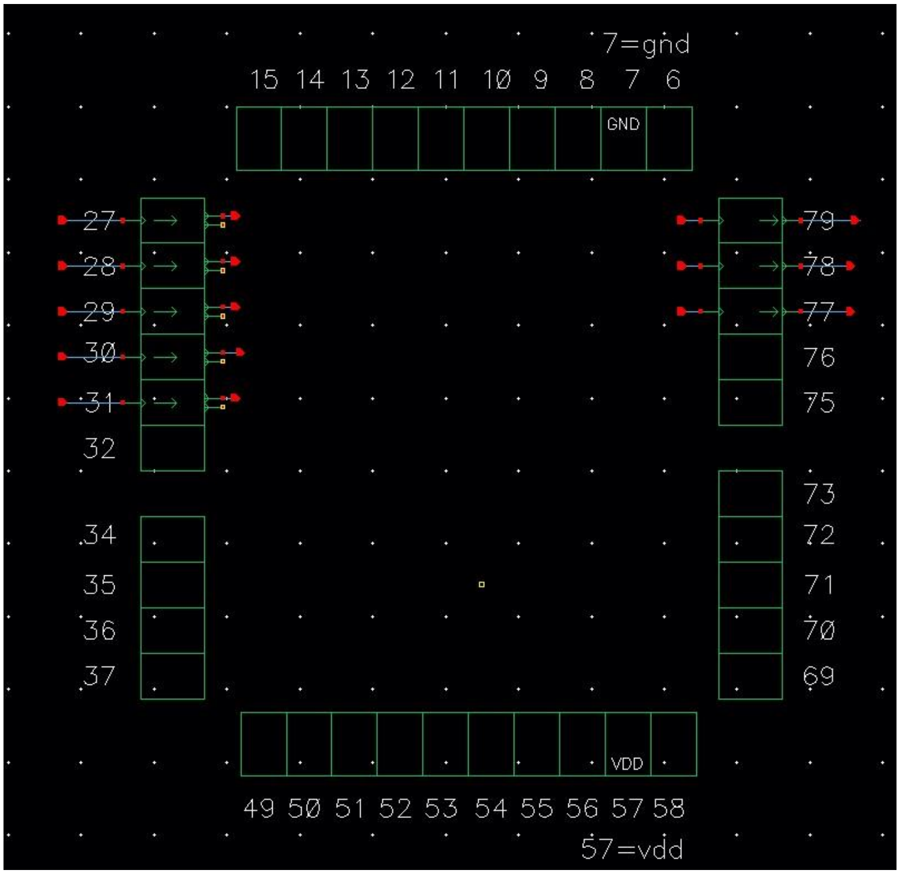


Figure 10 Padframe After Editing

6. After you've modified the pad types, we'll need to connect pins to your pad frame, in order to generate a pad frame symbol. It is best that you use the suffix "_i" for any signals that connect between your padframe and core, in order to denote that they are internal signals. After you've added input and output pins for your schematic, save your schematic.

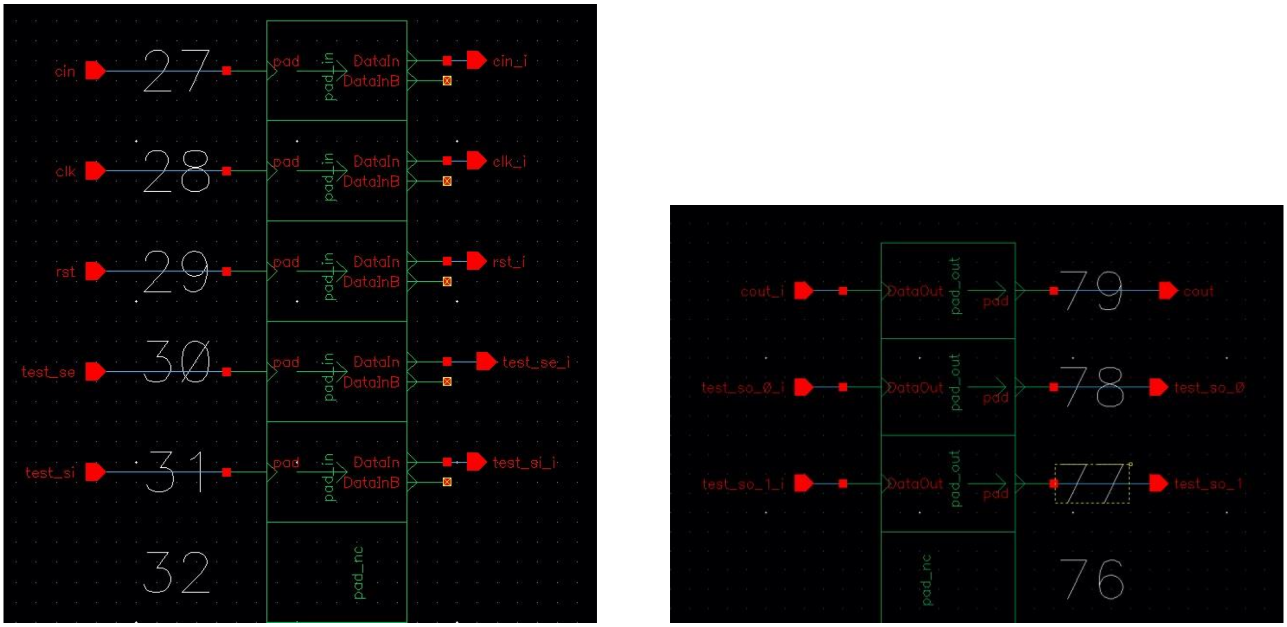


Figure 11 Example of adding pins to the padframe

7. Go to **Create → Cellview → From Cellview** in order to create a schematic symbol for your Padframe. You can close your schematic views for your padframe.

8. Create a new schematic cell in your lab8_adder library, titled **wholechip**. Instance your adder2 and padframe symbols in this schematic. Wire up your cores together. The inputs and outputs to adder2 symbol should connect to the "_i" signals on your padframe symbol, and pins should be placed on your padframe cell.

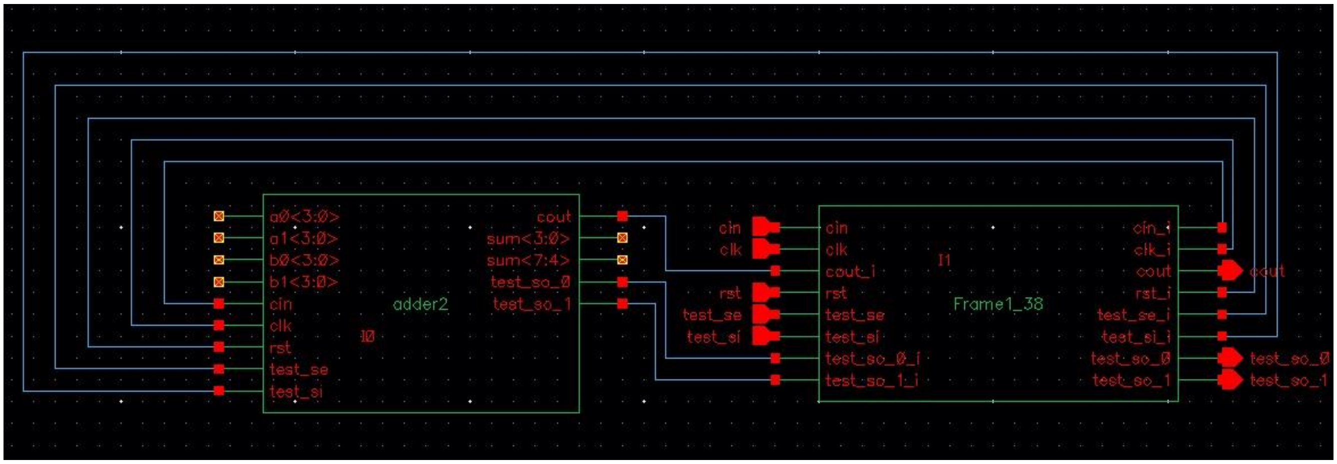


Figure 12 Wholechip Schematic

9. Next, we'll need to modify the padframe layout in order to have it match your padframe schematic. Open your padframe layout view, and change the corresponding numbered pads to their appropriate types (pad_in and pad_out).

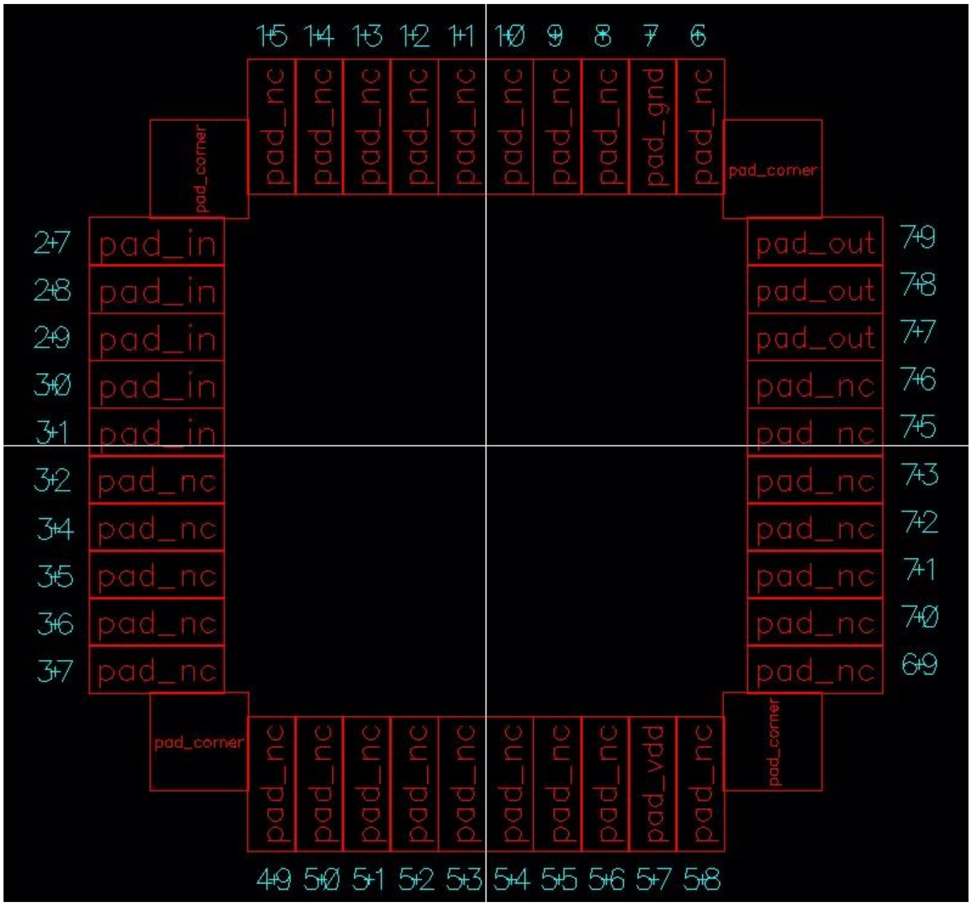
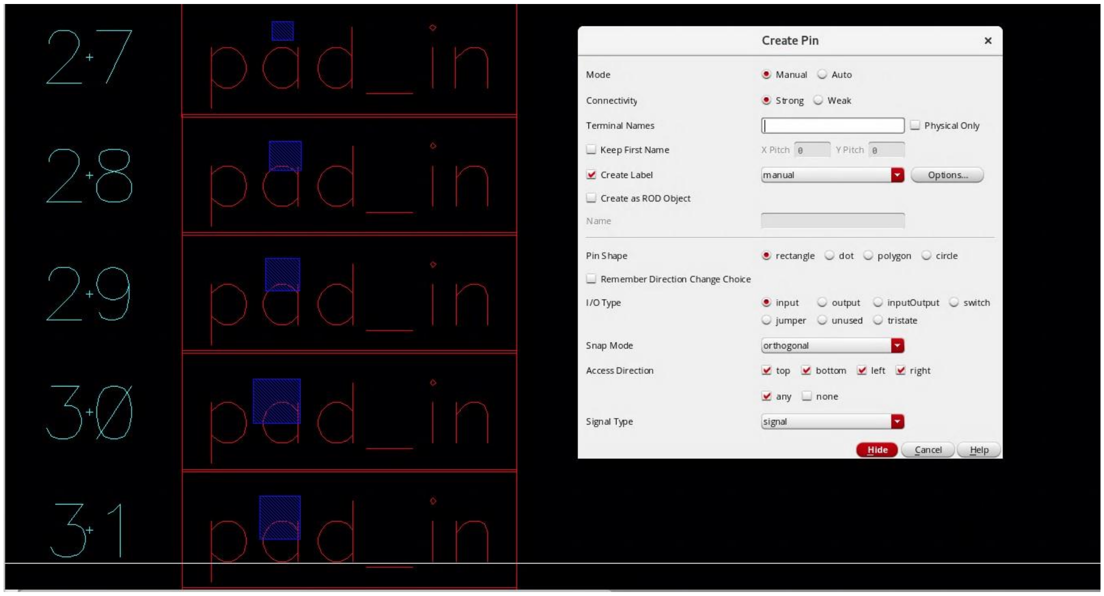


Figure 13 Padframe Layout after editing the pads

10. After modifying the pad types, we need to add shape pins to correspond to the pins we added to your padframe schematic. Be certain to make sure the following conventions are kept.

- Pad_in: Pad shape pin is an input pin, made with metal1.
- Pad_in: DataIn shape pin is an output pin, made with metal2.
- Pad_out: Pad shape pin is an output pin, made with metal1.
- Pad_out: DataOut shape pin is an input pin, made with metal2.



Create Pin

Mode ● Manual ○ Auto

Connectivity ● Strong ○ Weak

Terminal Names [ ] □ Physical Only

□ Keep First Name X Pitch 0 Y Pitch 0

☑ Create Label manual Options...

□ Create as ROD Object

Name

Pin Shape ● rectangle ○ dot ○ polygon ○ circle

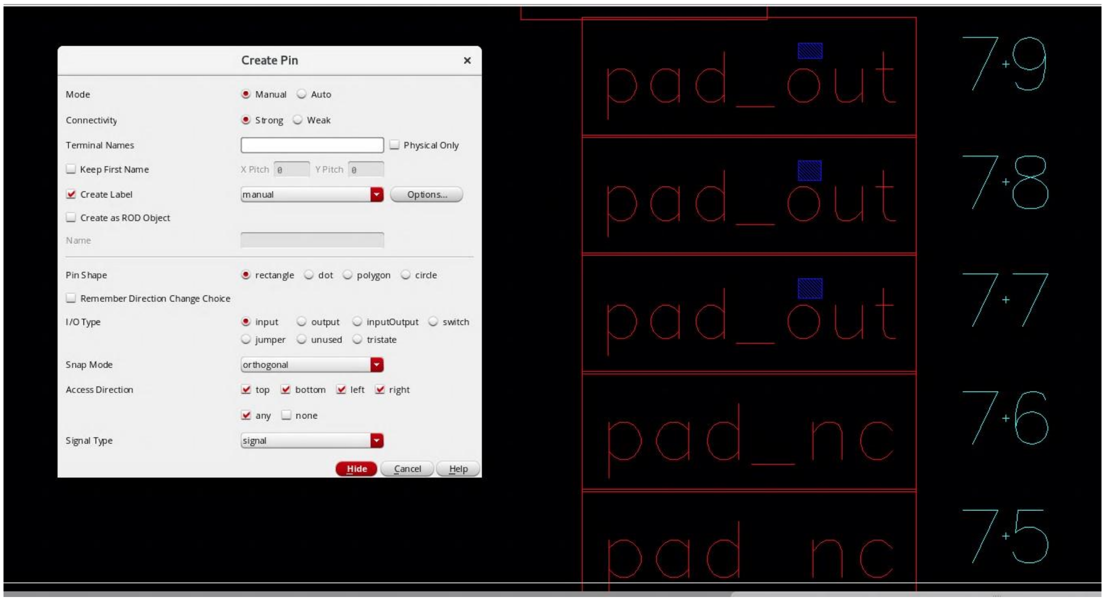
□ Remember Direction Change Choice

I/O Type ● input ○ output ○ inputOutput ○ switch ○ jumper ○ unused ○ tristate

Snap Mode orthogonal

Access Direction ☑ top ☑ bottom ☑ left ☑ right ☑ any □ none

Signal Type signal



Create Pin

Mode ● Manual ○ Auto

Connectivity ● Strong ○ Weak

Terminal Names □ Physical Only

□ Keep First Name X Pitch 0 Y Pitch 0

☑ Create Label manual Options...

□ Create as ROD Object

Name

Pin Shape ● rectangle ○ dot ○ polygon ○ circle

□ Remember Direction Change Choice

I/O Type ● input ○ output ○ inputOutput ○ switch ○ jumper ○ unused ○ tristate

Snap Mode orthogonal

Access Direction ☑ top ☑ bottom ☑ left ☑ right ☑ any □ none

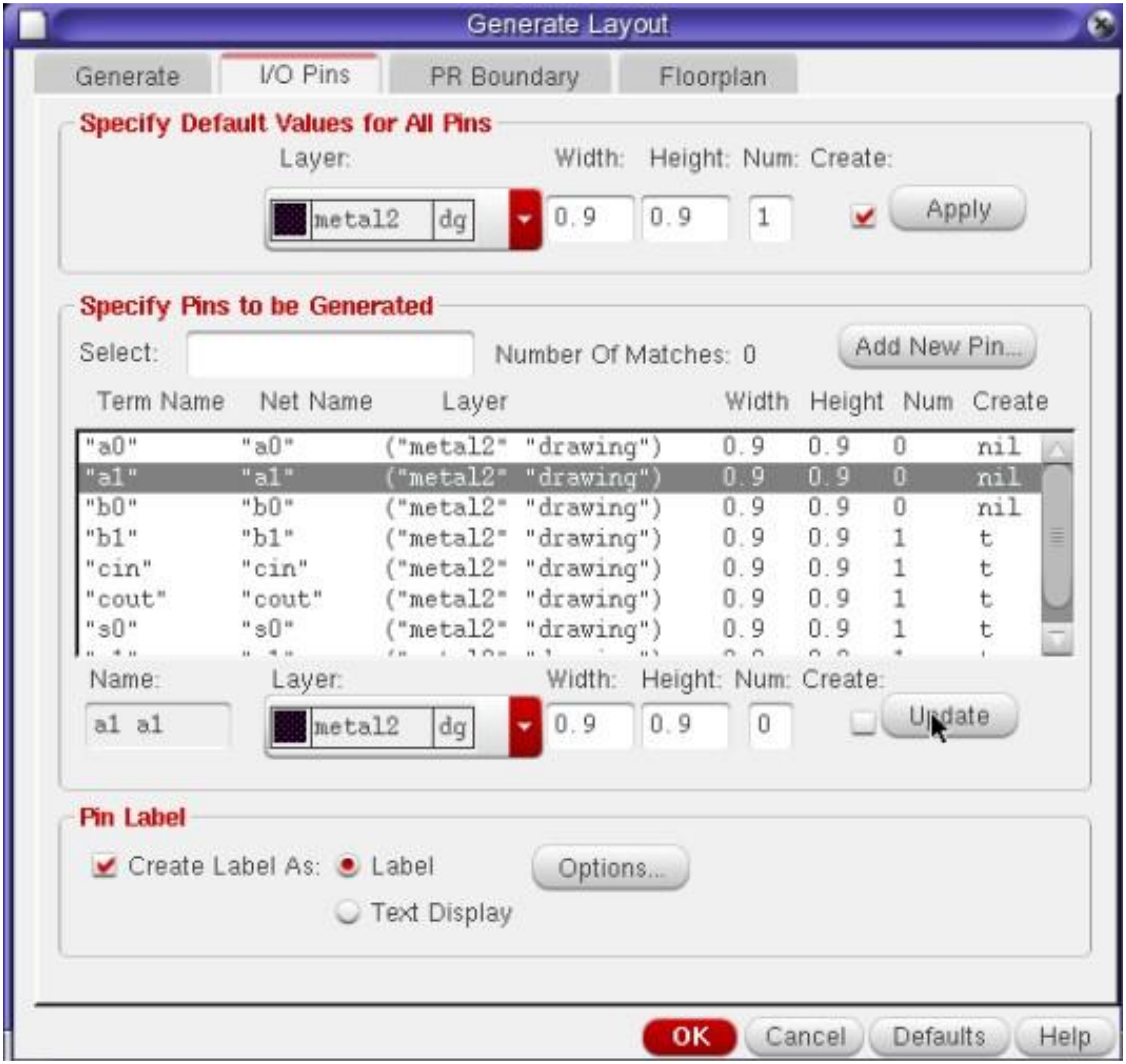
Signal Type signal

After you’ve added the pins to the padframe, you can save the padframe and close it.

Note: If you made any mistakes in connecting your padframe, CCAR will fail later on.

11. We now have the necessary components in order to route our core-frame together. In your schematic view for wholechip, go to **Launch → Layout XL.** Accept defaults to allow it to create the layout view for wholechip.

12. Your layout view is initially blank, so we need to tell Layout XL to automatically generate your layout. Go to **Connectivity → Generate → All from Source**. We do not need IO pins created in this pad-core routing, so go to the IO pins tab, and select each pin and uncheck the “Create” checkbox and hit update. After doing this for all pins, Under the generate tab, uncheck “Generate IO Pins”. Hit OK to generate your padframe view.



Specify Default Values for All Pins

Layer: metal2 dg Width: 0.9 Height: 0.9 Num: 1 Create: ☑

Specify Pins to be Generated

Select: Number Of Matches: 0

Term Name	Net Name	Layer	Width	Height	Num	Create
"a0"	"a0"	("metal2" "drawing")	0.9	0.9	0	nil
"a1"	"a1"	("metal2" "drawing")	0.9	0.9	0	nil
"b0"	"b0"	("metal2" "drawing")	0.9	0.9	0	nil
"b1"	"b1"	("metal2" "drawing")	0.9	0.9	1	t
"cin"	"cin"	("metal2" "drawing")	0.9	0.9	1	t
"cout"	"cout"	("metal2" "drawing")	0.9	0.9	1	t
"s0"	"s0"	("metal2" "drawing")	0.9	0.9	1	t

Name: a1 a1 Layer: metal2 dg Width: 0.9 Height: 0.9 Num: 0 Create: □

Pin Label

☑ Create Label As: ● Label
○ Text Display

Figure 14 Removing IO pins from the design

You'll notice that the generated view contains your adder2.layout view, not your adder2.layout.routed view. You'll need to manually change that instance to contain the appropriate cell view.

13. Similar with the core-core routing, we need to connect power to the core from the pads. Use the dedicated Pads in order to do this. You should be using 28.8 um wide strips of metal1 off of padframe, and connect that to the power routing of your core.

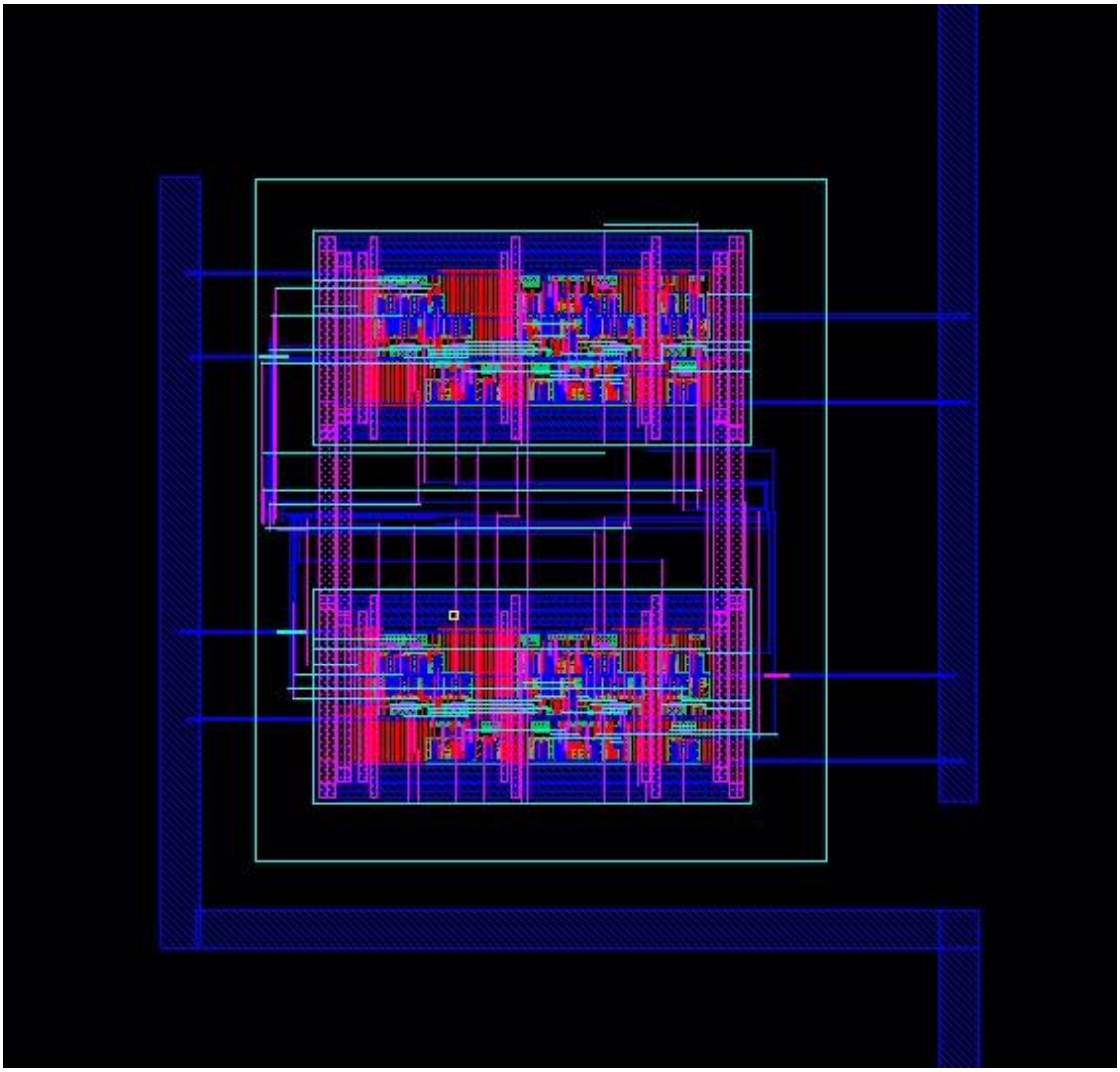


Figure 15 Power routed core 1

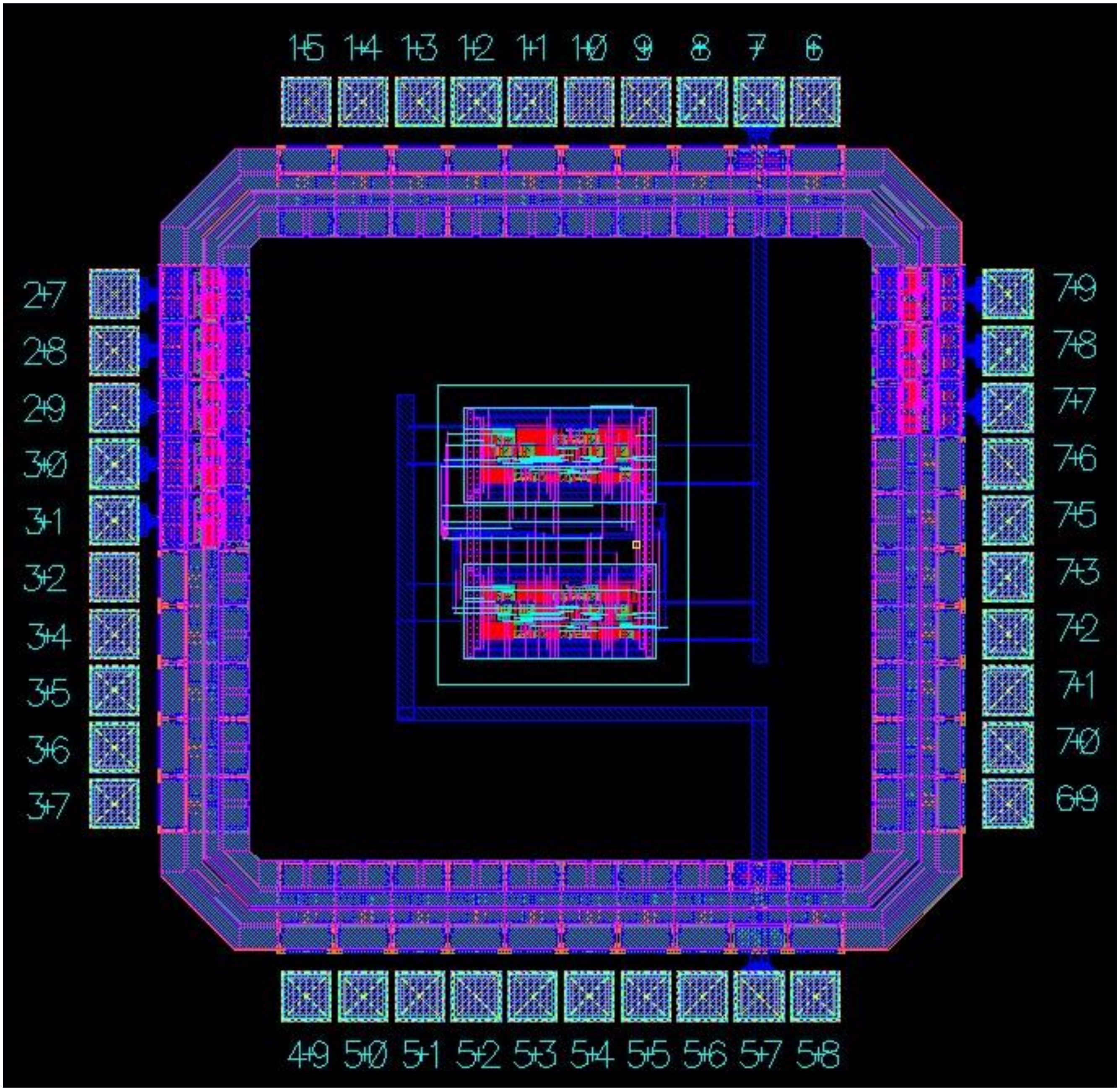


Figure 16 Power Routed Core 2

14. Once you've completed the power routing, you can now repeat the CCAR process in order to you're your core-pad module. You will perform the same steps as **Part II, Steps 11-20.** The verbose steps will be eliminated from **Part III**, since there is no additional information or steps to be given. Screen shots of the routed core are provided.

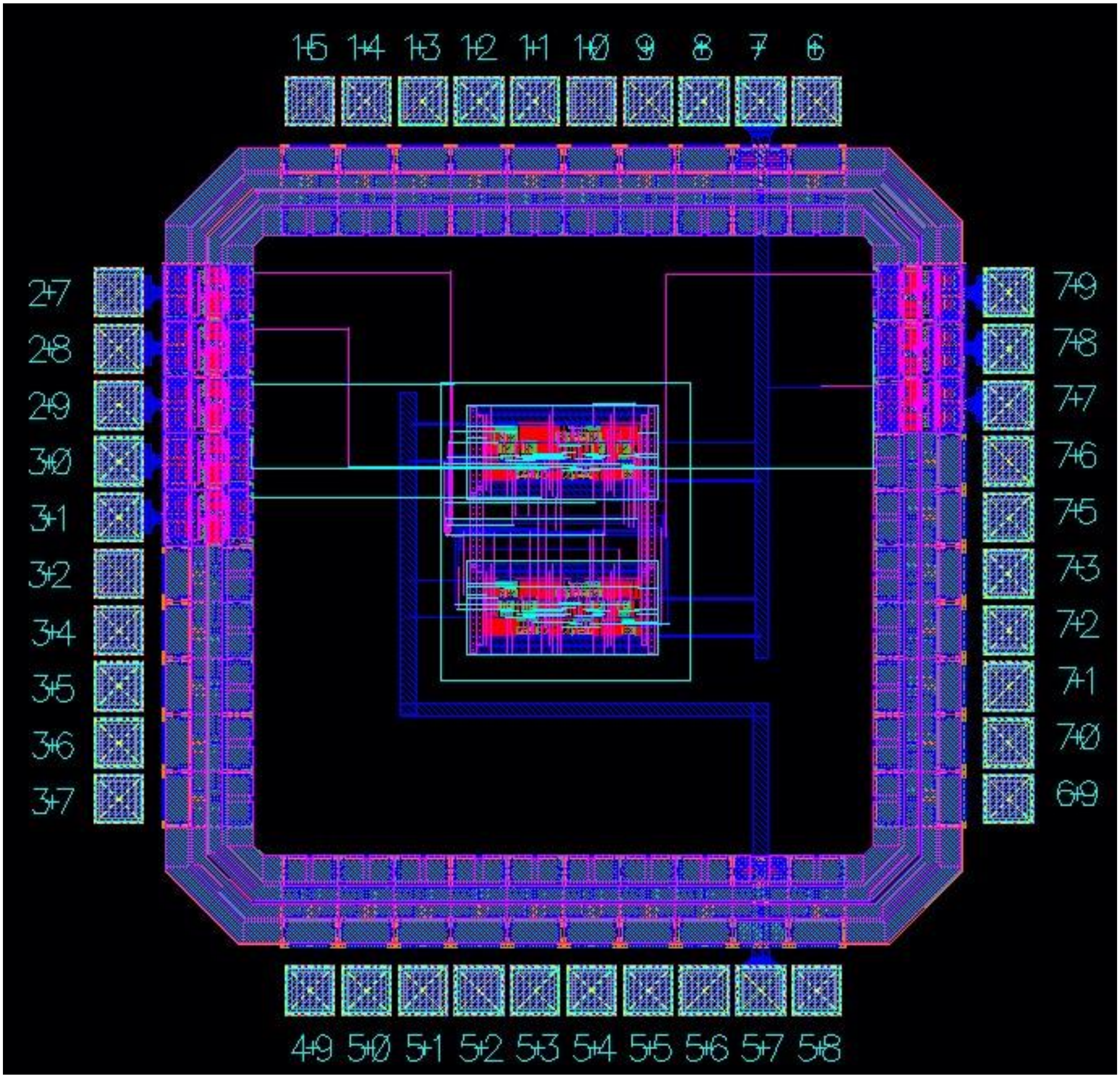


Figure 17 Fully routed wholechip

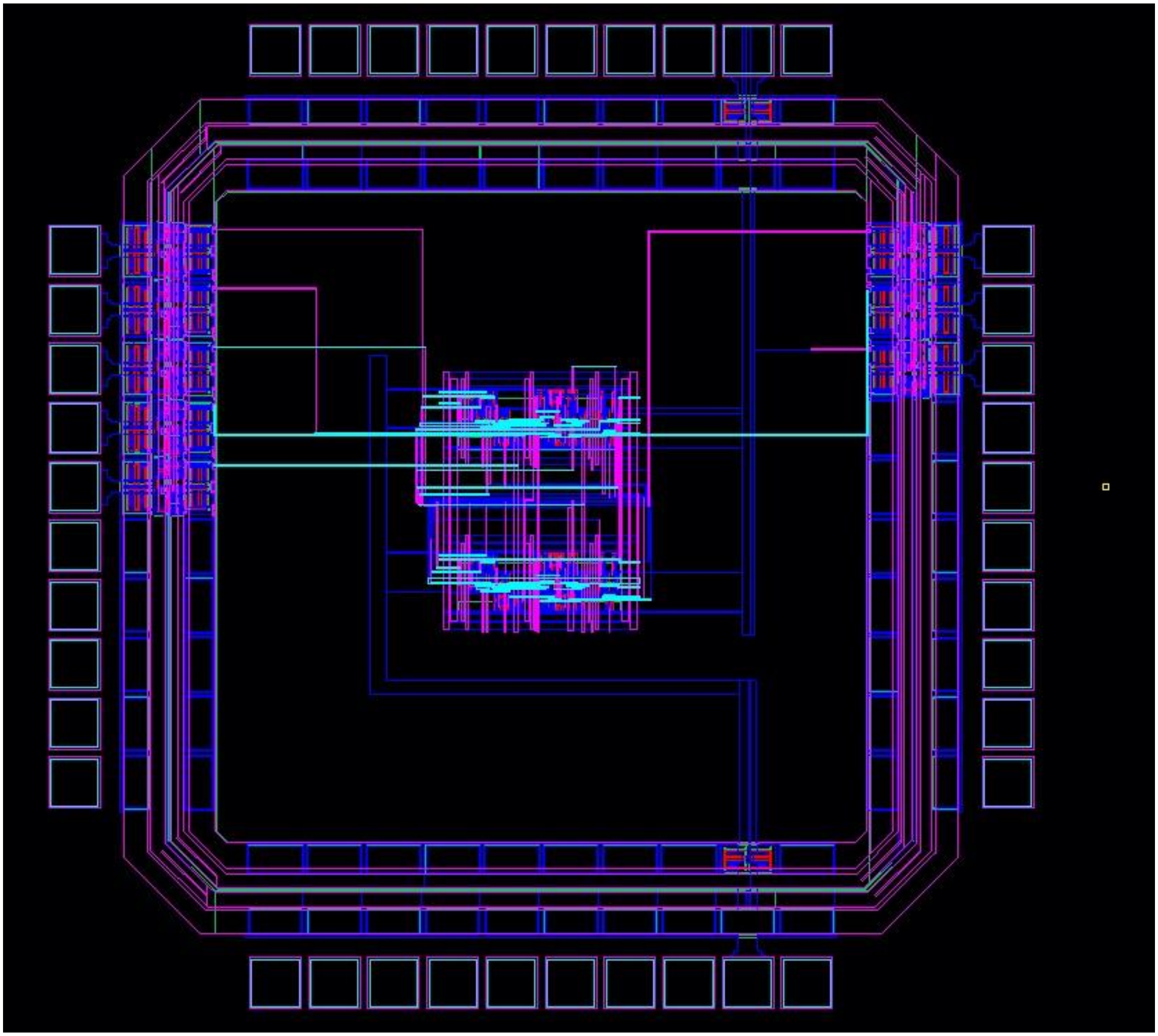


Figure 18 Extracted Whole Chip

Lab Report (post to our BB under “Lab8” in Lecture section; Our grader will grade it along with assignments)

Show the following items in a lab report:

- The following are worth 1 point apiece
  - Adder2 – Schematic
  - Adder2 – Symbol
  - Adder2 – Layout
  - Adder2 – Layout.Extracted
  - Adder2 – Extracted
  - Adder2 – LVS Report
  - Padframe – Layout after modifying it
  - Padframe – Schematic after modifying it
  - Wholechip – Schematic
  - Wholechip – Symbol
  - Wholechip – Layout
  - Wholechip – Layout.Extracted
  - Wholechip – Extracted
  - Wholechip – LVS Report
- The following is worth 5 points
- 1 Paragraph, summarizing what you did in the lab and your results and or conclusions. Note that a paragraph is at least 5 sentences long.

References

Digital VLSI Chip Design with Cadence and Synopsys CAD Tools, 2010, Erik Brunvand. Person Education Inc.

Appendix

For students viewing this tutorial who do not have access to our server, this is a listing of files referenced in this tutorial.

- **Do.do** - Setup instructions for Cadence Chip Assembly Router tool.
- **UofU Padframe** – Available from the companion site for Professor Brunvand’s VLSI textbook http://www.cs.utah.edu/~elb/cadbook/

Do.do:

```
rule IC (inter_layer_clearance -1 (layer_pair cc via)) rule
IC (inter_layer_clearance -1 (layer_pair poly via)) rule
IC (inter_layer_clearance -1 (layer_pair active via)) rule
IC (inter_layer_clearance -1 (layer_pair nactive via))
rule IC (inter_layer_clearance -1 (layer_pair pactive
via))
setup_check (antenna_rule off) (conflict on) (corner_corner_check off) \
  (xtalk on) (end_cap off) (length on) (limit_way off) \
  (min_width_wire  on) (min_mask_edge_length off) (max_vias off) \
  (miter off) (order off) (polygon_wire on) (protected off) \
  (reentrant_path on) (same_net_check  on) (segment off) \
(stub off) (use_layer off) (use_via off)
```