Cache Coherence

McKay Mower
Electrical and Computer Engineering
University of Utah
Salt Lake City, USA
u1182423@utah.edu

Braxton Chappell
Electrical and Computer Engineering
University of Utah
Salt Lake City, USA
u1097560@utah.edu

Tyler Liddell
Electrical and Computer Engineering
University of Utah
Salt Lake City, USA
u1175601@utah.edu

***Abstract*—Modern processors are becoming more complex and will continue to become more complex in the future. A massive problem of multi-core processors is the idea of keeping data consistent through all memory and cache memory. This idea is called cache coherence. In this paper, we describe a cache coherent system and review our design before and after place and route.**

***Index Terms*—Cache coherence, place and route**

I. Introduction

Modern day processor caches are continuously becoming more complex. For instance, current processors can have up to 5 levels of cache before a request is sent to main memory. However, an interesting problem is introduced when we have this many levels of cache. This problem is called 'cache coherence' and is established when multiple caches in a multi-processor system are sharing data known as 'global memory'.

As an example, imagine we have 2 processors, each with only a level 1 cache (for simplicity). If both processors are using data X from memory which has an initial value of 10, and one processor tries to change X to 20, then the caches become inconsistent. Thus, we need a way of solving this problem [1].

We introduce the goal of this project: to establish a cache coherent interface between 2 simulated processors and 1 simulated memory (RAM). We implement a 256-byte direct-mapped cache that incorporates a write-through cache policy [2]. We also implement a cache controller that uses a snooping-based coherence mechanism with a write-update coherence protocol.

II. Description of Steps

Creating a coherent cache system requires a few elements. First, a cache system was built out. This involved deciding on a cache policy, mechanism and overall design.

A. Cache Design

The design used a 256-byte direct mapped cache incorporating a write-through cache policy, using half-addressable 16-bit words. This allowed single byte access and writing. These cache modules receive 25-bit requests where 1 bit is for r/w, 8 bits are for data, and the last 16 bits represent the address. The 16-bit address is interpreted by the cache as 8 bits for the tag, 7 bits for the index, and 1 bit for the offset. Doing so allows very efficient and simple indexing and checking of the cache entries.

The control flow for a cache is outlined as follows. When a request is made, it will branch based on if the request was a read or a write request. For a read request, it will check if the requested index in the cache matches the requested tag, and is also a current valid entry. If it is, it will return the data on the 8-bit data out line. If it does not match or is not valid, the cache will make a memory read request and wait for a response. On receiving a response, it will write the data into the cache, and also populate the data out bus. A simpler flow is seen for a write request. The cache writes the data into the corresponding entry in its cache, and makes a memory write request.

The cache design was the most complex of the entire project. However, because of the design choices made when creating the caches, implementing a coherence mechanism was made very simple. This is done using the cache coherenter, which is explained in detail below.

B. Coherenter Design

When one CPU makes a write to cache and then memory, that data needs to be consistent in other caches if a different CPU is to execute instructions that would rely upon that data. The "coherenter" (YES THIS IS TOTALLY A WORD) is the module that keeps the coherency of the caches. The coherenter takes in inputs from two caches: 25 bits from one and 25 bits from the other. The coherenter checks if the incoming memory request is a write. If the request is a write, the coherenter will signal the other cache to change the validity bit for that cache entry to 0. Now if the other CPU tries to read information from the cache at that index, the validity bit being 0 causes the cache to go to the memory for that information.

C. Serialization of inputs and outputs

With a design involving two caches, a cache coherenter, and a memory output bus, too many IO pads would be required. A staggering 25+1 bits are required for a cpu request+ready signal, another 25+1 for a memory request+ready signal, and a final 8+1 for data output+ready signal. In our system, this would lead to a required 26*4+9*2=122 IO pads. This required serializing inputs and outputs using two new modules, namely the input collector and output emitter. The input collector reads in a serialized data stream and populates an internal bus. Conversely, the output emitter takes an internal bus and serializes the data onto a single line. Since these modules are extremely small, they can be driven at a much faster rate than the rest of the system, reducing the performance hit taken from serializing data transfer. By placing one of these modules at each input or output bus described above, the required IO pads were drastically reduced.

D. *Top Level Design*

After all of the pieces were built out, the final step was hooking them all together in a top level module. Two caches, one coherenter, two input collectors for the cache requests, two input collectors for the memory responses, two output emitters for the memory requests, and a final two output emitters for the data out back to the CPUs. Creating the top level in this way reduces the overall IO requirement to only 19 pins.To increase the speed of the serialized data transfer, the top level takes both a regular clock used to drive the caches and coherenter, and a fast clock used to drive the input collectors and output emitters.

III. ModelSim

A. *Pre-synthesis*

Testing was done for each file that was created for a component. Cache was the first device to be written in Verilog. Because the project was focused only on the cache, its coherency, and not on CPUs/memory that might use the device, mock CPU requests and memory needed to be implemented. Other components tested were the Coherenter, the collector, and the emitter. With all the components of the overall device functioning properly on their own, the top level is tested.

Testing of the cache required mocked components. The CPU requests could be generated manually, or randomly, with 16 or 24 bits, depending on if it is a read or write. The mock memory was an array of 65536 bytes. This number is generated from having 16 bit addresses for the data.

With the auxiliary components set, testing of the caches write and read ability could be done. To do so, Verilog tasks were written to automate the process. These tasks are split for the read and write test process, as well as a reset for the cache. The write task takes an input of a 16-bit address for 8-bits of data, totalling 24 bits. This task writes the data to the cache and the memory, then checks it to see if the write was successful. The read task only takes a 16-bit address. Both tasks use another task called "check_data". This will compare the data in the cache to the data in the memory to see if they are the same. Another task that is utilized to test the cache is the "invalidate" task. This task is written to test the interaction of the coherenter and the cache and to ultimately see if our system will function as intended.

For the smaller files, the coherenter, the emitter, and collector, tests were written to catch any small bugs and to see if operation is as intended. These files had operations much more direct and simpler, therefore, testing was utilized as a check.

With the confirmation of the smaller components operating as intended, top level testing can begin. For this, the same auxiliary components the cache testbench used were generated. The tasks in the cache testbench were modified slightly for two CPUs instead of just one. Although, the "invalidate" task was not included because the coherenter would be doing that naturally, not through a test.

For the test of the top level, general function and edge cases were first tested. The main trigger to test was if a CPU writes to one cache, the other cache will have an invalid bit in the index. This means that the CPU would go to the main memory to get the data and the cache will then be written with the new data at that index and turning the validity bit to 1. After this was confirmed, then general use was tested using random addresses, random data, and random CPUs. With the use of the tasks to check the data, the data and addresses didn't matter as they could be checked in the main memory, allowing the use of random numbers. With this test bench passing, confirmation to move to synthesis is given.

B. *Post-synthesis*

Before the test benches could be run on the post-synthesis file, some changes to the pre-synthesis code needed to be made. These changes are made to fix an inferred latch problem that was only exposed after synthesis. Having the state machine to be triggered by the clock, and not on the change of a state (even though that was run on the clock) solved the inferred latch issue.

Testing the design after synthesis was very straight forward. By replacing the top level Verilog file with the newly synthesized file, we were able to run the exact same testbenches and observe the results.

IV. Place and Route

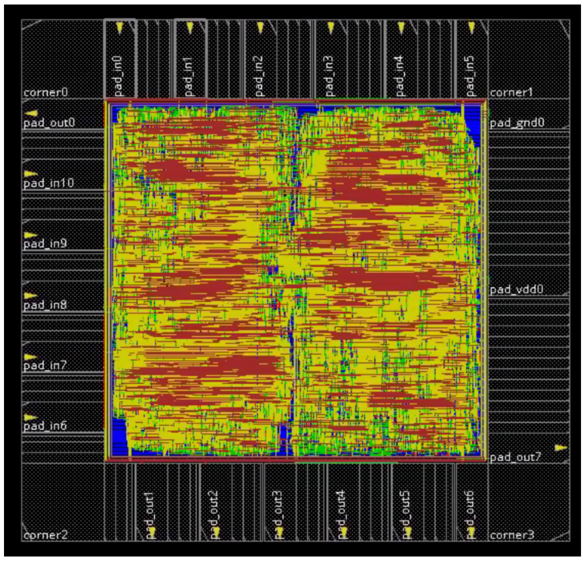


Fig. 1: Innovus layout of project before pads have been filled in.

In Figure 1, the layout is shown. From the Figure, we are able to make out what seems to be cache 0, and cache 1 dominating the left and right sides of the layout. The coherenter is a fairly small and simple chunk of hardware, so it is difficult to make out exactly where it is. DRC and LVS was checked before the room for pads was added. When green lights were given for both things, that is when the pads were added.

Figure 2 adds the pads and sealring in and shows the rest of the chip in a different view. The chip shown is supposed to be the ”tape out ready” chip design, and it almost is. The only thing that wasn’t figured out on this chip design were some density errors. Listening to other projects, density errors are not specific to this project using the tsmc technology.

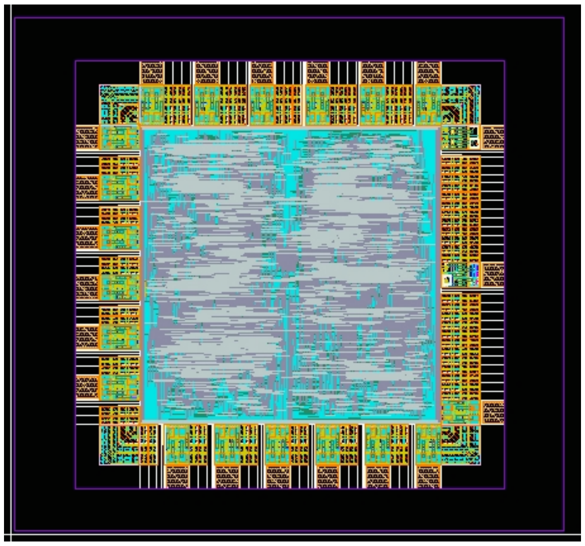
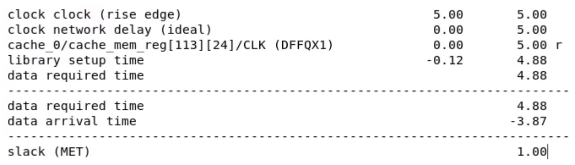


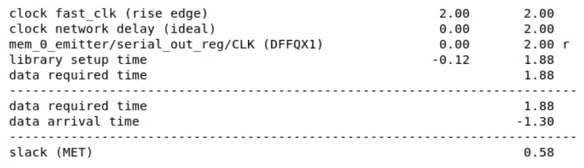
Fig. 2: Innovus layout of project close to tape-out ready. This layout has density errors shown in the DRC.

When it comes to timing, slack was kept higher than zero to try and account for the timing differences of synthesis and layout. Unfortunately, the added slack time was not enough for the layout to pass timing. An attempt to change the timing was made, but the amount of things that changed to get the first synthesis file to work caused the attempt to fail. The amount of things that were changed made the process not so streamlined. Trying to recreate the process took more time than it was worth with the deadline approaching quickly. Figure 3 and 4 shows the timing of the clock and fast clock, respectively.



```
clock clock (rise edge)                            5.00      5.00
clock network delay (ideal)                        0.00      5.00
cache_0/cache_mem_reg[113][24]/CLK (DFFQX1)        0.00      5.00 r
library setup time                                -0.12      4.88
data required time                                           4.88
-----------------------------------------------------------------------
data required time                                           4.88
data arrival time                                           -3.87
-----------------------------------------------------------------------
slack (MET)                                                  1.00
```

Fig. 3: Timing of the normal clock after synthesis. Not after place and route. Shown with a clock cycle of 5ns used, the optimized circuit will take 1ns less.



```
clock fast_clk (rise edge)                         2.00      2.00
clock network delay (ideal)                        0.00      2.00
mem_0_emitter/serial_out_reg/CLK (DFFQX1)          0.00      2.00 r
library setup time                                -0.12      1.88
data required time                                           1.88
-----------------------------------------------------------------------
data required time                                           1.88
data arrival time                                           -1.30
-----------------------------------------------------------------------
slack (MET)                                                  0.58
```

Fig. 4: Timing of the fast clock after synthesis. Not after place and route. Shown with a clock cycle of 2ns for the fast clock.

With the use of 2ns for the fast clock, 5ns for our normal clock is pushing the use of our circuit anyways. Because the fast clock is for the SPI lines for our device, and the worst case will have a 24 bit instruction, this means that our devices could operate at 48ns to use power more effectively. This would be slow. Therefore, to speed it up, SPI lines running in parallel could be a better approach to speed up the design and maintain IO pin constraints.

In the appendix, there are more photos. One of them shows LVS passing our ”top_level_padded” file. This is the file that is almost ”tape out ready”. Two others are the area and power. Total power was given at 16.01mW, and total area was 848941. They are in the appendix so you can see them better. Repeat photos are in the appendix as well for a better look.

When it comes to meeting our requirements, we tried to set the generic ones given in the project handout. We were afraid of the area because of the size that cache can get. The pins were also a constraint that we had and changed our design to make work. When it comes to power, we weren’t sure were to set our goal as. Ultimately, trying to reach these goals, besides the pins, went out the window. Trying to figure out the tools became priority as the pretty picture of a tape-out ready device was more desired.

Noticing other projects in the course have leakage currents of in the pW range, the reported leakage current for this project needs to be addressed. The leakage current reported for this project is at 915nW. The best possible explanation of this is probably due to the amount of gates used in the cache of this project. Although, this is a guess.

V. Conclusion

Cache coherency is a major problem in modern cache and memory design. We have discussed what cache coherency is and how it is important to VLSI. The way to keep memory consistent throughout the system is to adhere to a cache coherent design. We have discussed the design of our cache coherent system. This system has been tested and synthesized using ModelSim, Design Compiler and Innovus. It passes LVS but there are 2 density errors regarding the POLY layer when trying to run DRC. However, this seemed to be a common issue with other projects.

References

[1] “Cache coherence,” Mar. 2022, page Version ID: 1078177075. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cache_coherence&oldid=1078177075

[2] “Bus snooping,” Dec. 2021, page Version ID: 1058114803. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bus_snooping&oldid=1058114803

VI. Appendix

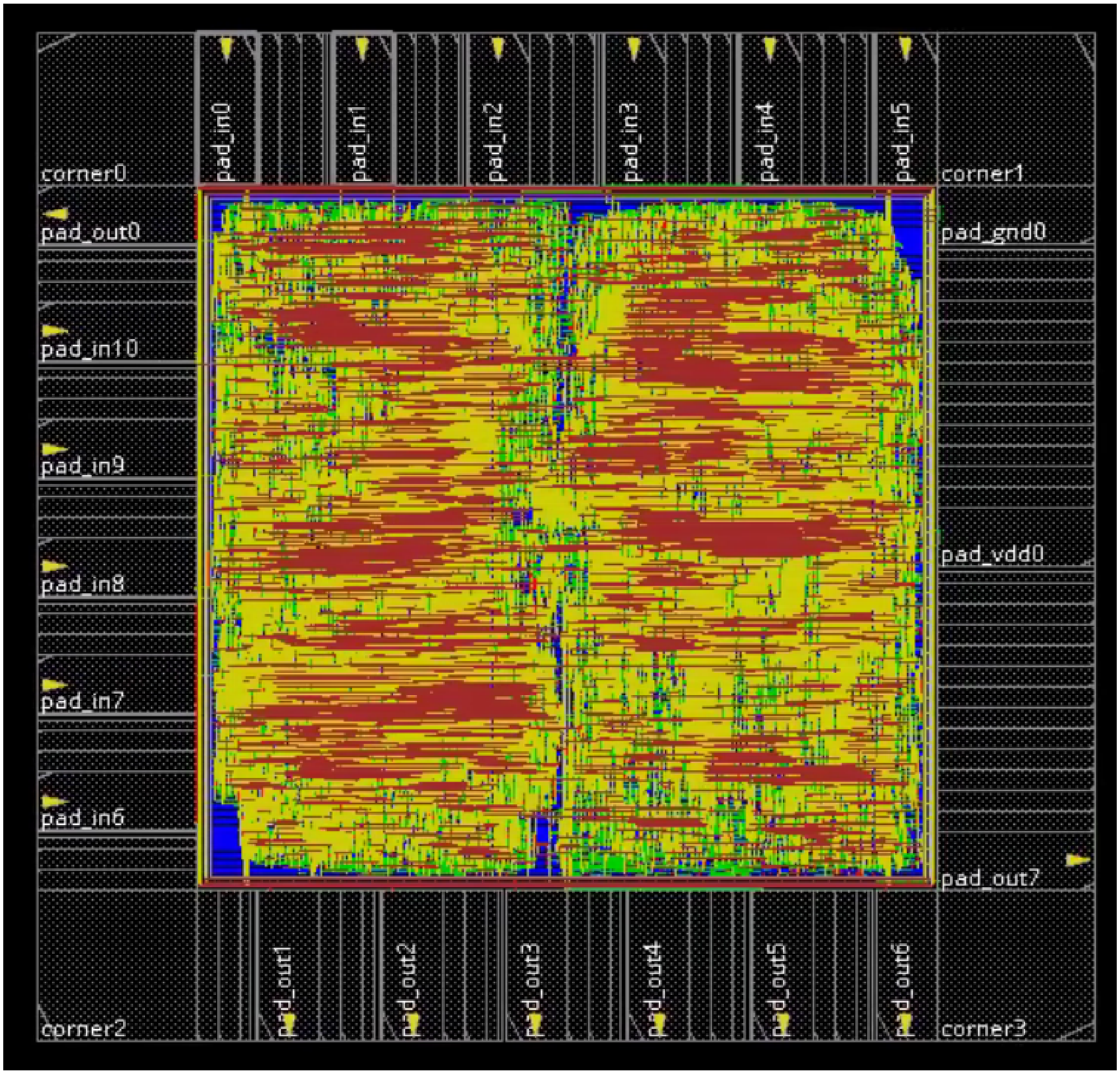


Fig. 5: Innovus layout of project before pads have been filled in. Expanded to show more detail.

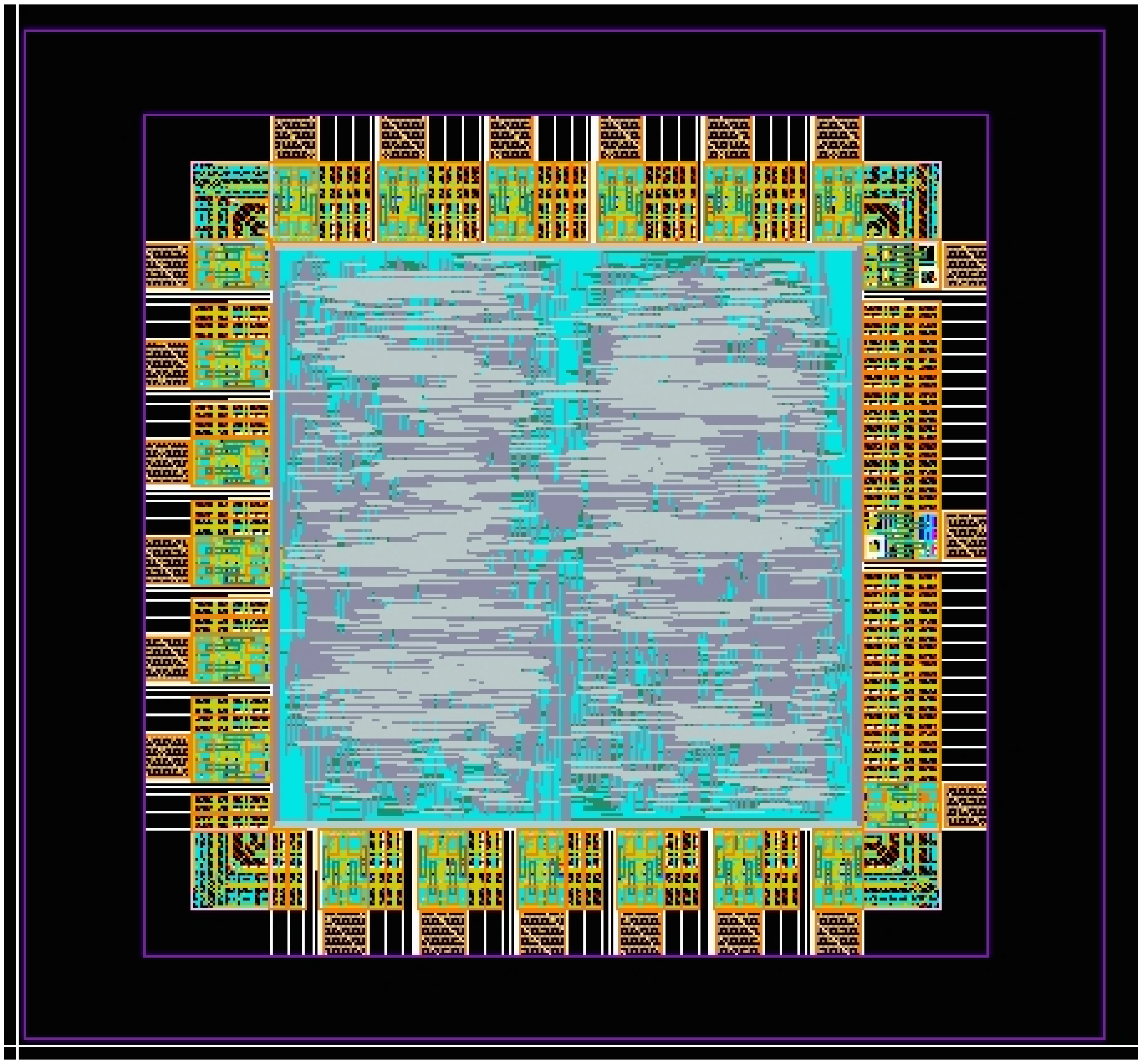
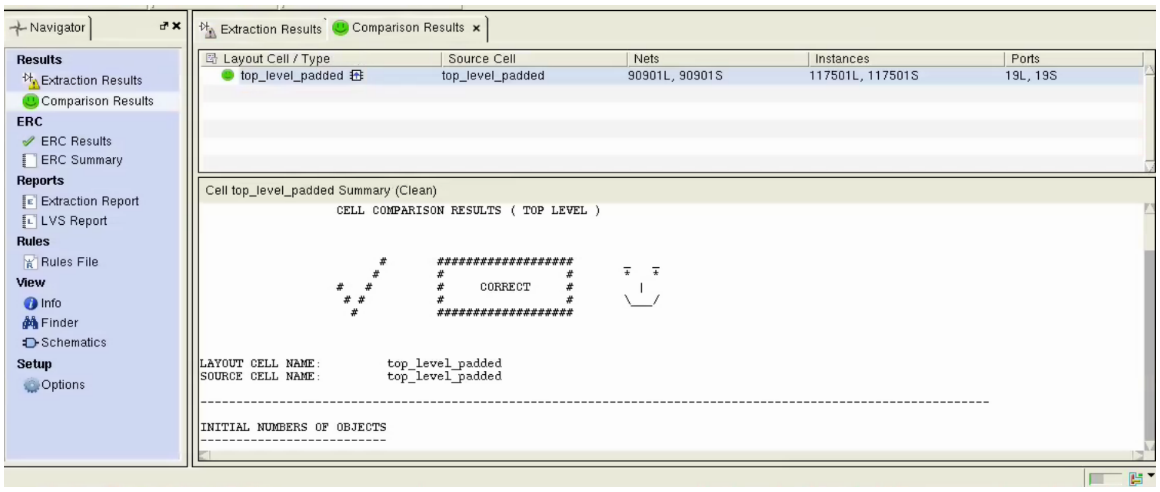


Fig. 6: Innovus layout of project close to tape-out ready. This layout has density errors shown in the DRC. Expanded to show more detail.

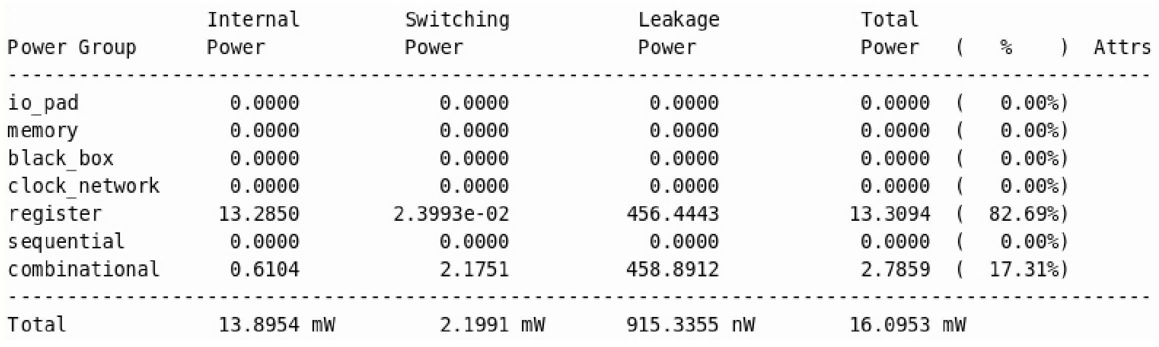


Layout Cell / Type	Source Cell	Nets	Instances	Ports
top_level_padded	top_level_padded	90901L, 90901S	117501L, 117501S	19L, 19S

Cell top_level_padded Summary (Clean)

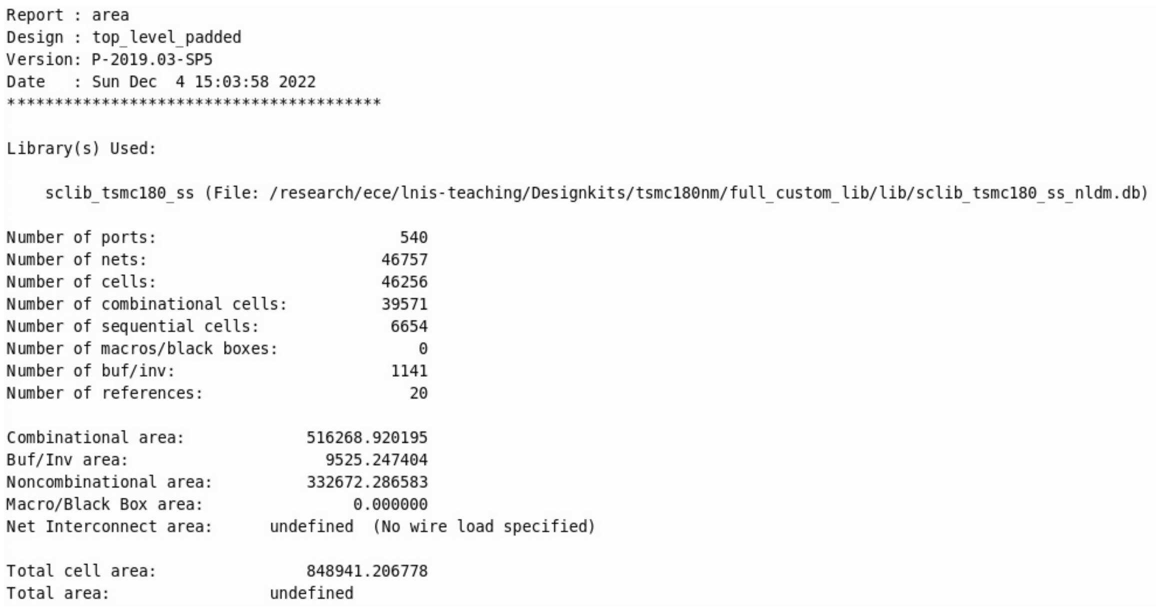
```
                    CELL COMPARISON RESULTS ( TOP LEVEL )


                       #       ###################
                      #        #                 #
                  #   #        #     CORRECT     #
                   # #         #                 #
                    #          ###################


LAYOUT CELL NAME:         top_level_padded
SOURCE CELL NAME:         top_level_padded

--------------------------------------------------------------------------------------------------------------

INITIAL NUMBERS OF OBJECTS
--------------------------
```

Fig. 7: LVS showing passing results of our top_level_mapped file.



```
                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000              0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000              0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000              0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000              0.0000            0.0000  (   0.00%)
register          13.2850         2.3993e-02           456.4443           13.3094  (  82.69%)
sequential         0.0000            0.0000              0.0000            0.0000  (   0.00%)
combinational      0.6104            2.1751            458.8912            2.7859  (  17.31%)
--------------------------------------------------------------------------------------------------
Total             13.8954 mW        2.1991 mW        915.3355 nW        16.0953 mW
```

Fig. 8: RPT output of the power.



```
Report : area
Design : top_level_padded
Version: P-2019.03-SP5
Date   : Sun Dec  4 15:03:58 2022
****************************************

Library(s) Used:

    sclib_tsmc180_ss (File: /research/ece/lnis-teaching/Designkits/tsmc180nm/full_custom_lib/lib/sclib_tsmc180_ss_nldm.db)

Number of ports:                          540
Number of nets:                         46757
Number of cells:                        46256
Number of combinational cells:          39571
Number of sequential cells:              6654
Number of macros/black boxes:               0
Number of buf/inv:                       1141
Number of references:                      20

Combinational area:            516268.920195
Buf/Inv area:                    9525.247404
Noncombinational area:         332672.286583
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:               848941.206778
Total area:                 undefined
```

Fig. 9: RPT output of the area.