The Matrix (Multiplier): Reloaded

Phat Nguyen, Edison Yang, Shem Snow, Connor Watson
ECE 5710/6710

Abstract: With as much attention as Artificial Intelligence is receiving, we wanted to explore the trend by studying matrix multipliers which are the basis of neural network inference and required for many Artificial Intelligence tasks such as image processing. We designed a 2x2 matrix multiplier capable of handling 2's complement numbers. In this report, we describe the process of using design automation tools including Verilog, ModelSim, and multiple Cadence softwares to turn our idea on a block level diagram into a full chip layout.

Overview: The process we followed to design our chip is to generate HDL code, verify that code, synthesize a gate-level netlist, perform verification again, produce a floor plan, place each cell, synthesize the clock tree, route the cells, pass all layout checks (DRC and LVS), place pads and a seal ring, and finally, pass all layout checks again.

Introduction: From the start of this project, we knew we wanted to design a scalable, adaptable chip that could be used as a major component of our group senior project. The idea being that our senior project will have a neural network used for some 'smart' task such as a Captcha test [1]. In this example, images would be processed through a neural network containing training data necessary to perform matrix multiplication and the text in the Captcha would be identified [2]. From this idea, our decision was to build a matrix multiplier.

Matrix multiplication is known to be computationally expensive and is already a well researched topic. The algorithm works in two stages. First, multiplication of two products is performed and second the resulting products are added to an accumulator variable [4]. Figure 1 shows an overview of this process. State of the art chips typically use specialized "Multiply Accumulate" (MAC) units [4].

Design: With our basis confirmed, the first step in writing RTL code was to figure out how to construct the matrix multiplier via block diagram. Before beginning on the modules, we sat down and drew out multiple rough drafts of how our circuit would work, finally settling on Figure 2 below as our top-level Verilog module. The design is conceptually separated into two pieces. The first being the design of all hardware necessary to perform each step such as multiplying the correct matrix entries together, adding the products together, and saving the final result into the correct entry of the resulting matrix. The second conceptual piece is to design a finite state machine to control all that hardware.

In addition to the two conceptual pieces, our machine is also built to operate in four stages. The finite state machine ensures the progression of each stage by sending control signals to

enable/disable the other components. To ensure each step of matrix multiplication is done in the correct order, the finite state machine communicates with a status register. In the first stage, the status register identifies when the two input matrices are loaded into the program. In the second stage, it identifies when the input matrices are finished multiplying their entries and saves each product into the register file. In the third stage, each entry-product of the input matrices already stored in the register file are moved to the adder, named as “accumulator” in the figure, which adds all products in parallel for an immediate calculation. Finally, the fourth stage takes the result from the accumulator and feeds it into the result handler, which encodes the final product and holds its value on a 24-bit wire. Another output of our chip is a “matrix_count” which is intended to act as an address specifier. The idea being that each matrix multiplier our circuit performs will be written to some external memory unit.

Verification: The next step was simulating the Matrix Multiplier in Modelsim, to validate the functionality of our design by applying several test cases in our testbenches. The waveform in Figure 3 demonstrates the functionality and matrix multiplication computation of our project. We performed formal verification by comparing the result in Modelsim to our expected answers (inputs/outputs) as shown in Table 1. Furthermore, we utilized self-checking testbenches in order to automate the verification process, ensuring correctness when given specific inputs. As a result, we were able to determine that the Matrix Multiplier Verilog code is working as intended. The summary of our results are also shown in Table 1.

Table 1 Expected/Actual Matrix Multiplication Result

Input (decimal)	Output-expected (decimal)	Output-actual (decimal)
$\begin{bmatrix}0&1\\2&3\end{bmatrix}\begin{bmatrix}3&2\\1&0\end{bmatrix}$	000001(1) 000000(0) 001001(9) 000100(4)	000001(1) 000000(0) 001001(9) 000100(4)
$\begin{bmatrix}-4&-2\\-3&-1\end{bmatrix}\begin{bmatrix}-2&-3\\0&-3\end{bmatrix}$	001000(16) 010010(34) 000110(6) 001100(12)	001000(16) 010010(34) 000110(6) 001100(12)
$\begin{bmatrix}3&2\\3&1\end{bmatrix}\begin{bmatrix}-1&-4\\-3&-4\end{bmatrix}$	110111(-9) 101100(-20) 111010(-6) 110000(-16)	110111(-9) 101100(-20) 111010(-6) 110000(-16)

$\begin{bmatrix}3 & 3\\ -2 & -1\end{bmatrix}\begin{bmatrix}-4 & -2\\ -3 & 1\end{bmatrix}$	101011(-21) 111101(-3) 111011(11) 000011(3)	101011(-21) 111101(-3) 111011(11) 000011(3)
$\begin{bmatrix}-4 & -4\\ 1 & 1\end{bmatrix}\begin{bmatrix}-4 & 3\\ -3 & 3\end{bmatrix}$	011100(28) 101000(-24) 111001(-7) 000110(6)	011100(28) 101000(-24) 111001(-7) 000110(6)

Front End: After verification, we obtained the technology-mapped Gate-Level Netlist schematic from our own RTL netlist, generated from Verilog code, using the Genus software. The resulting netlist is shown in Figure 4.

Once we were able to extract the TMSC and padding files for the logic gates in the RTL schematic, we applied the RTL file into ModelSim and ran the same testbench as before. This helped us determine that the RTL schematic generated by Genus, which is mapped to our specific technology (TSMC 180 nm process), was producing the same result as the Verilog code we wrote. Figure 5 shows the simulation result for the technology-mapped RTL schematic.

Comparing the output variables “matrix_result” and “matrix_count” shown in Figures 2 and 4, we were able to produce the same computations shown in Table 1 for both the pre-synthesis and post-synthesis RTL schematics. The equivalence of these circuits tells us we succeeded in the front-end portion of chip design. The code we wrote was now mapped to standard cells in the TSMC 180 nm process.

Timing Constraints: While generating the RTL schematic, we also had to modify our constraints file (.sdc) to account for the conditions our circuit will operate on. Our constraints file is shown below in Figure 6. It shows the clock frequency of 333k (period of 300 ns) we had come up with in order to provide a positive slack for our circuit so it operated correctly after Place & Route occurred.

Performances: We were also able to extract the area, performance, and power information from the synthesis report generated with the gate level netlist. The area is shown in Figure 7 and 8.

The maximum area we are allowed is 2x2 mm or 4000000 square micrometers. We see that our area only covers less than 13000 micrometers. We are nowhere near the limit on area and so we decided to make the trade-off of expanding area in order to improve performance. We did this by performing all additions in the process during one clock cycle which requires components to be in parallel rather than what state of the art chips do and placing components in series. In theory,

this means our cell area should be higher than that of state of the art chips and our performance should also be higher.

Figures 9 and 10 showcases our performance of the circuit given the constraints in our .sdc file. We were advised to have a good amount of positive slack as it showcases there is good latency for our circuit to operate. With that, we have obtained a positive slack of 146,150 ps which is a good latency for our design. Finally, Figure 11 showcases total power of 13,330 nW with leakage power of 280 nW and dynamic power of 13,020 nW.

After some research, we found various performance statistics for state of the art matrix multiplier chips using the 180 nm process [3]. Area seems to be consistently around 3.1-3.2 square millimeters for every chip we found so we believe area is not a constraint we can lower as it likely depends on the fabrication process. We relied on cell count to make inference about area. Comparing our results to existing chips, we see that our area of 1.63 square millimeters containing 384 cells uses almost double the number of cells compared to state of the art non-pipelined matrix multipliers with 193 gates in a 3.1 square millimeters space [4]. However, compared to pipelined chips with roughly 3200 cells and an area of 3.15 square millimeters , our design uses only 12% of the number of cells [5].

Regarding power and performance, the results varied greatly, likely due to different manufacturing capabilities. In general, our power and performance were ten times better. Most likely, the chips we compared ours to had more functionality which we did not implement, namely, the capability of multiplying matrices larger than 2x2. It is difficult to find a fair comparison as it does not make sense for a company to spend millions of dollars fabricating a circuit as trivial as a 2x2 matrix multiplier that can only handle 3-bit numbers.

Another chip, a 16x16 MAC chip [6], reinforces our conclusions. It displayed statistics of ~140 mW, 8.5 ns delay, and with our previously established area estimations, it is clear that our chip is better in every measure. However, our chip has scaled down functionality and if sized up to an industry-usable standard would be worse in terms of power, size, and delay. One additional benefit that our chip has over most state of the art comparisons is that it uses 2’s compliment form. This gives our chip niche uses in projects that need to compute matrices made of positives and negatives in small magnitudes.

Back End: Once the synthesis was complete, we then used the generated netlist code as well as the constraint netlist to physically place our circuit using Innovus. What was modified, as compared to what we have previously done in labs, was changing the overall density to 0.30, width of floorplan to 280 nm, height of floor plan to 200 nm, and a global maximum density of 0.45 when placing our RTL schematic. Units of density are a ratio of area occupied by logic

elements and the total area of the chip. Another thing our group did was to remove the SRAM placement when doing the floorplan, this is due to our multiplier not needing any memory accesses as we are only utilizing registers.

Integrating the floorplan, placement, clock tree synthesis, and routing, results in Figure 12.

Once completed, we then finally place our layout into Virtuoso to verify that our placement and routing comply with TSMC's 180nm library. This will help reduce overall errors towards area design and help ensure our product is safe for production, which we are not going to cover for this project. Figure 13 shows the layout with VDD and VSS labeled for validation:

With that, we ran our layout with the DRC: virtuoso/calibre/tsmc180nm_drc_runset. As you can see in Figure 14, we have no conflicts with the given design rules, so our place and route is all well placed.

With the DRC passed, we then have to check our LVS to make sure it still runs the same way as our RTL schematic using the given file, virtuoso/calibre/tsmc180nm_lvs_runset. As you can see in Figure 15, we maintain having zero errors, but there were some extraction warnings with a few labels that were not necessary with our design. These extraction issues were discussed with the TAs and it was concluded that these were insignificant.

Once we knew that our actual layout and schematic were working properly, it was time to add the sealring which is used to protect our circuit from contaminants and moisture. We also ran the associated DRC files: virtuoso/calibre/tsmc180nm_drc_runset_polyfill and virtuoso/calibre/tsmc180nm_drc_runset_metalfill to help mold our polyfill and metalfill designs to fit our circuit. This resulted in Figure 16.

To account for the seal ring, we rerun the previous checks. This step is performed before our chip goes to production and it covers the full chip parameters. Figures 17 and 18 show that our DRC and LVS checks still pass as before:

With these results matching, we extract our results into a .gds file format named Phat_Shem_Connor_Edison.gds. With this our circuit is completed and is now able to be produced; however, due to some regulations on how the production is with the University's agreement with TSMC, it will not be covered in this project.

Conclusion and reflection: Even though our design is not as large-scale as we initially wanted because of the limit on the number of pins we can use, we are very happy with our results. We got to experience the cyclical development of circuits where verification revealed the need to re-design certain aspects of the circuit.

As was the case in the labs, seeing each step of the process be successfully passed is extremely satisfying, and just like with the RISC layout, it was impressive seeing our design laid out, logic gate by logic gate. Replicating the steps of each lab was a fairly simple process, but the small differences, occasional new steps, and slow clearing through of the DRC and LVS errors we had in each iteration was tedious and time consuming. Much time was spent communicating with the TAs to understand where our errors were coming from. For our final issue, we eventually happened into a conversation with another student who had encountered a similar problem, and got a working suggestion from them.

We found that it was not so easy to compare performances to the state of the art since no existing chips do the same operation. We were only able to make estimates which led to conclusions about performance significantly improving as the complexity of a circuit decreases. We also observed the common area-performance trade-off where if there is space on the die, it makes sense to use it all for the sake of improving performance. As the scope of our chip expands, especially to the level that a neural network would require, the area and power consumption would grow exponentially, hence the industry's emphasis on keeping the size and power consumption minimal even at the expense of performance. For our purposes, choosing performance is preferable for a future expansion into a senior project so long as the scale is still small enough to only need 2x2 or 3x3 matrices.

Our chip may never be fabricated, but understanding of the principles around its design, and gaining the experience with Cadence tools will be a great help to each of us in the future.

References:

[1] M. Kopp, M. Nikl, and M. Holeňa, “Breaking CAPTCHAs with Convolutional Neural Networks.” Available: https://ceur-ws.org/Vol-1885/93.pdf (accessed Nov. 27, 2024)

[2] Z. Al-Qadi and A. Musbah, “Performance analysis of parallel matrix multiplication algorithms used in image processing.,” CiteSeer, 2009. https://www.researchgate.net/profile/Musbah-Aqel/publication/267261129_Performance_Analysis_of_Parallel_Matrix_Multiplication_Algorithms_Used_in_Image_Processing/links/5744025108ae9ace841b446c/Performance-Analysis-of-Parallel-Matrix-Multiplication-Algorithms-Used-in-Image-Processing.pdf (accessed Nov. 27, 2024).

[3] J. Lee, M. Kim, and S. Park, "Time-Domain Multiply–Accumulate Unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 3, pp. 1234–1245, Mar. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10107445

[4] J. Chen, J. Li, Y. Li, and X. Miao, "Multiply accumulate operations in memristor crossbar arrays for analog computing," *Journal of Semiconductors*, vol. 42, no. 1, p. 013104, Jan. 2021. [Online]. Available: https://www.jos.ac.cn/en/article/id/7e0268f4-0de0-4254-ab02-6eb999f684d2

[5] A. A. Deshmukh and S. S. Dorle, "Design and VLSI Implementation of Pipelined Multiply Accumulate Unit," in *Proceedings of the 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Trivandrum, India, Dec. 2009, pp. 859–861. doi: 10.1109/ACT.2009.5395507.

[6] M. Masadeh, O. Hasan, and S. Tahar, “Input-Conscious Approximate Multiply-Accumulate (MAC) Unit for Energy-Efficiency,” IEEE Access, vol. 7, pp. 147129–147142, 2019, doi: https://doi.org/10.1109/access.2019.2946513.

Figures:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Figure 1: Visual Representation of Matrix Multiplication Algorithm

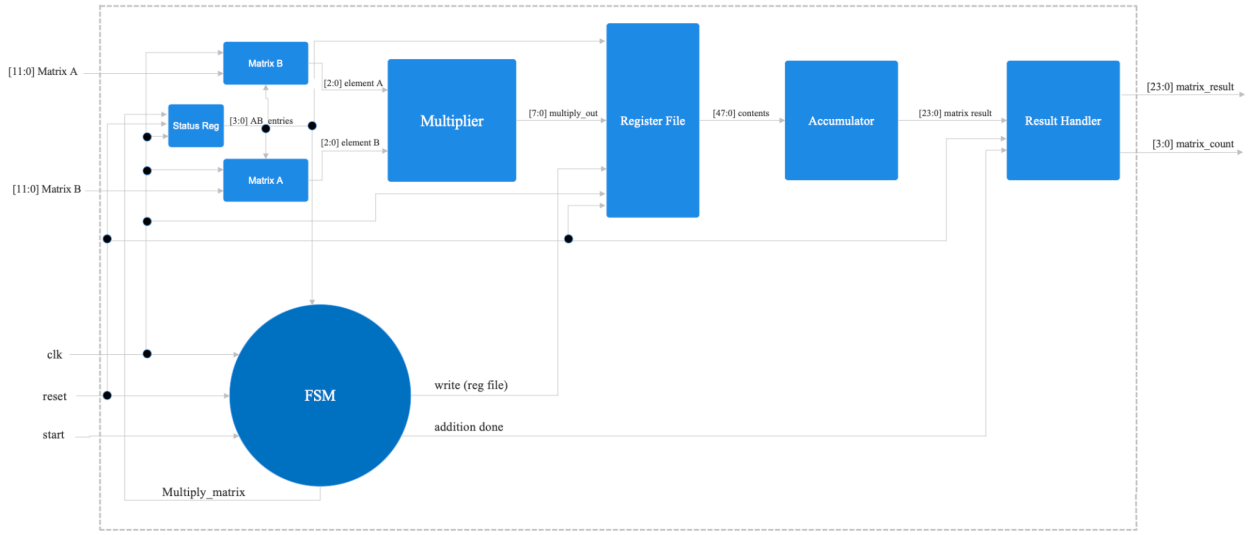


Figure 2: Block Diagram of Circuit

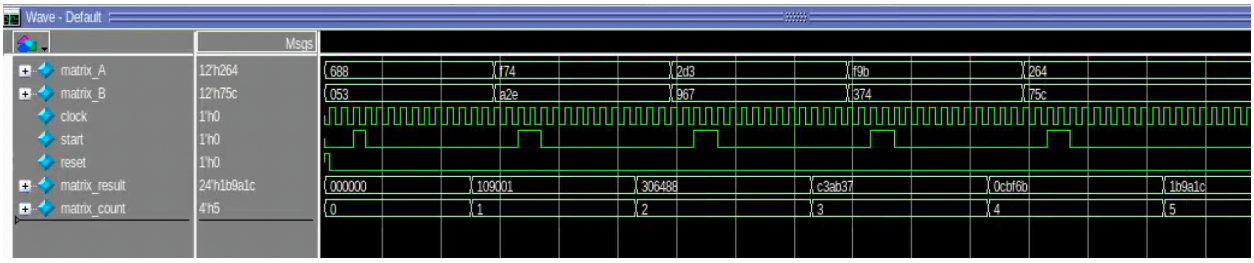


Figure 3: Pre-synthesis Bit Matrix Multiplication Waveform

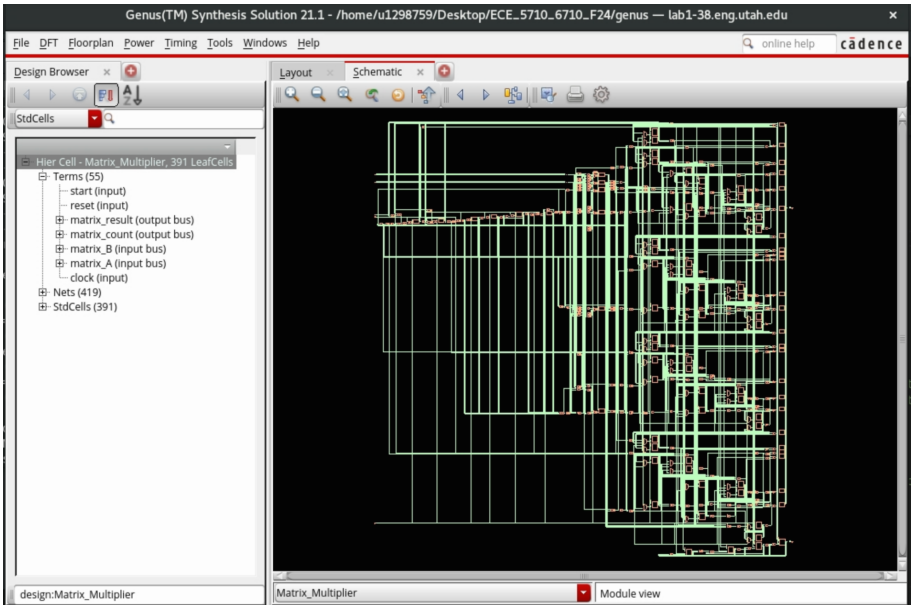


Figure 4: RTL schematic of Matrix Multiplier

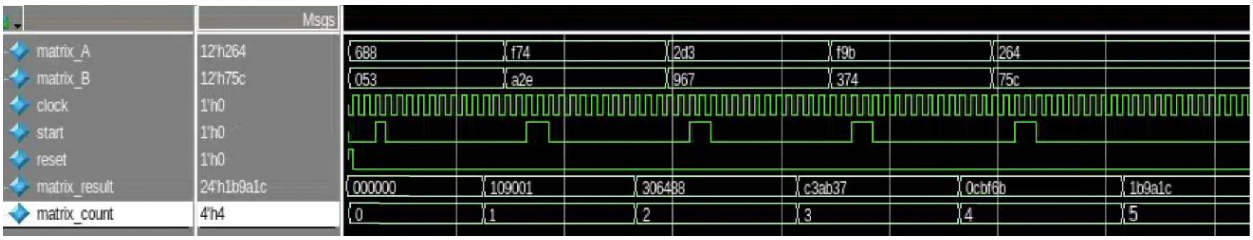


Figure 5: Post-synthesis Bit Matrix Multiplication Waveform with a RTL schematic

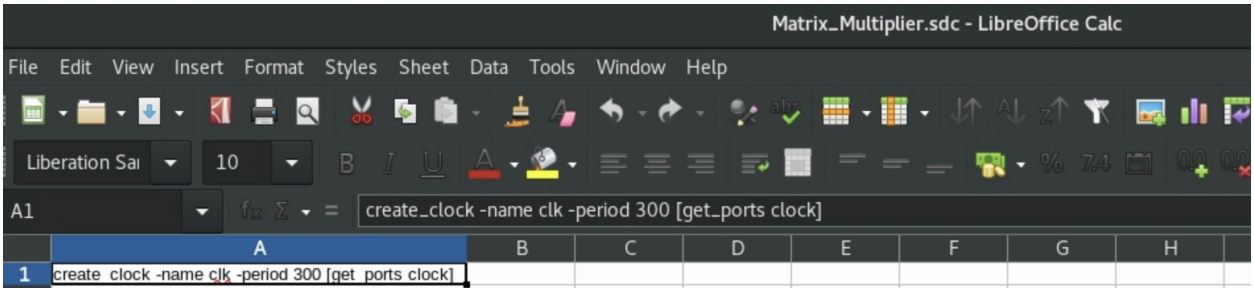
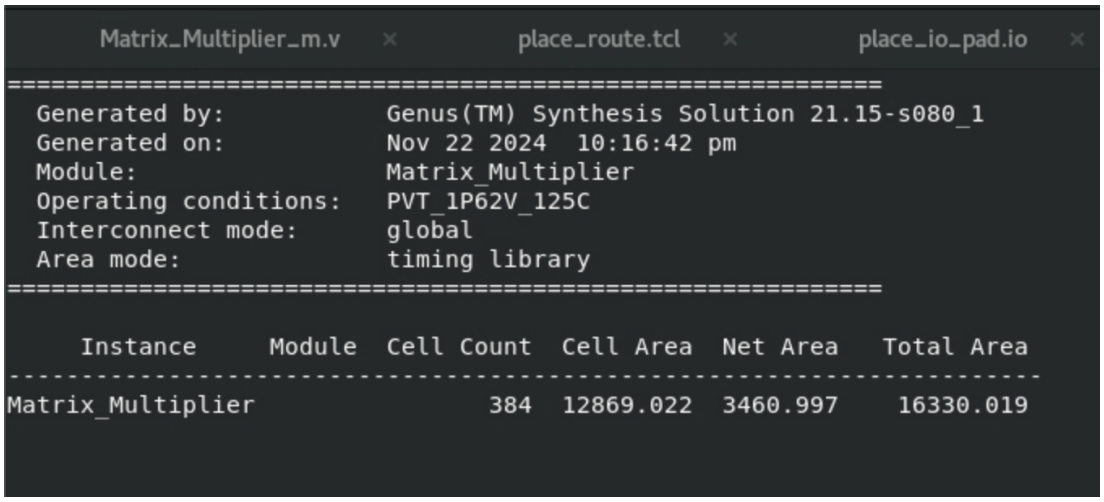
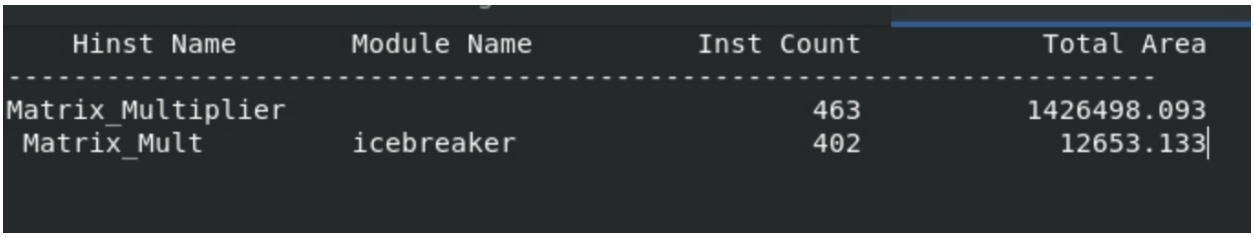


Figure 6: Constraints File of our Matrix Multiplier



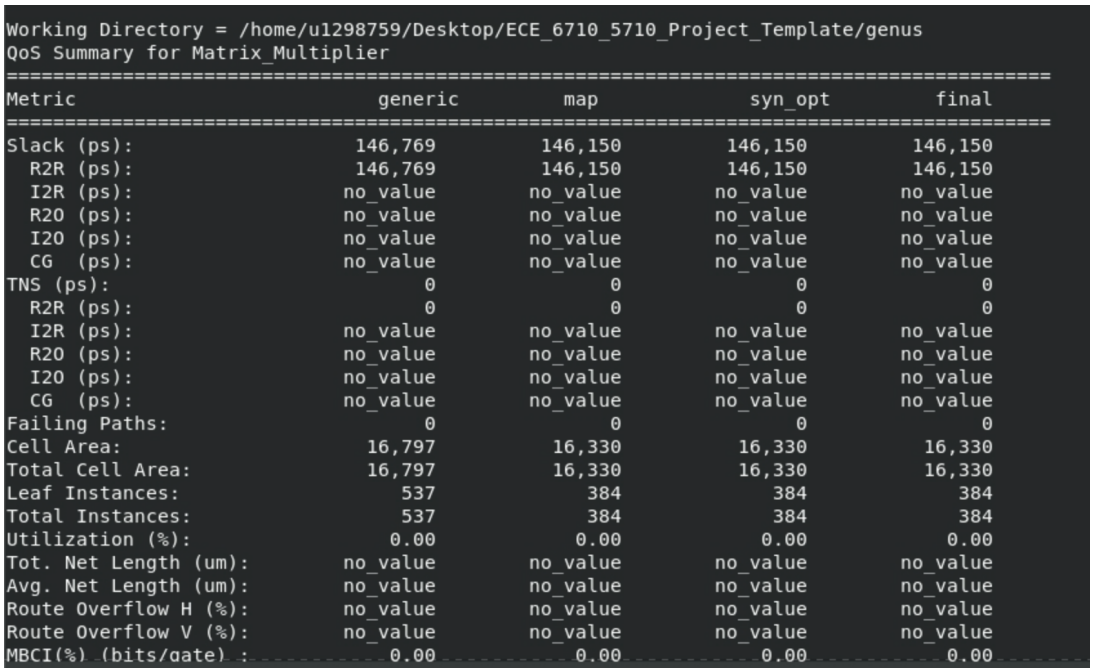
```
Matrix_Multiplier_m.v   ×   place_route.tcl   ×   place_io_pad.io   ×
============================================================
  Generated by:           Genus(TM) Synthesis Solution 21.15-s080_1
  Generated on:           Nov 22 2024  10:16:42 pm
  Module:                 Matrix_Multiplier
  Operating conditions:   PVT_1P62V_125C
  Interconnect mode:      global
  Area mode:              timing library
============================================================

     Instance     Module  Cell Count  Cell Area  Net Area   Total Area
-------------------------------------------------------------------------
Matrix_Multiplier               384  12869.022  3460.997    16330.019
```

Figure 7: Total Area of our Matrix Multiplier from Genus



```
    Hinst Name       Module Name          Inst Count          Total Area
-------------------------------------------------------------------------
Matrix_Multiplier                              463          1426498.093
 Matrix_Mult         icebreaker                402            12653.133
```

Figure 8: Total Area from Innovus



```
Working Directory = /home/u1298759/Desktop/ECE_6710_5710_Project_Template/genus
QoS Summary for Matrix_Multiplier
================================================================================================
Metric                          generic         map            syn_opt         final
================================================================================================
Slack (ps):                   146,769         146,150         146,150         146,150
  R2R (ps):                   146,769         146,150         146,150         146,150
  I2R (ps):                  no_value        no_value        no_value        no_value
  R2O (ps):                  no_value        no_value        no_value        no_value
  I2O (ps):                  no_value        no_value        no_value        no_value
  CG  (ps):                  no_value        no_value        no_value        no_value
TNS (ps):                           0               0               0               0
  R2R (ps):                         0               0               0               0
  I2R (ps):                  no_value        no_value        no_value        no_value
  R2O (ps):                  no_value        no_value        no_value        no_value
  I2O (ps):                  no_value        no_value        no_value        no_value
  CG  (ps):                  no_value        no_value        no_value        no_value
Failing Paths:                      0               0               0               0
Cell Area:                     16,797          16,330          16,330          16,330
Total Cell Area:               16,797          16,330          16,330          16,330
Leaf Instances:                   537             384             384             384
Total Instances:                  537             384             384             384
Utilization (%):                 0.00            0.00            0.00            0.00
Tot. Net Length (um):      no_value        no_value        no_value        no_value
Avg. Net Length (um):      no_value        no_value        no_value        no_value
Route Overflow H (%):      no_value        no_value        no_value        no_value
Route Overflow V (%):      no_value        no_value        no_value        no_value
MBCI(%) (bits/gate) :            0.00            0.00            0.00            0.00
```

Figure 9: Total Performance of our Matrix Multiplier from Genus

```
#############################################################
#  Generated by:      Cadence Innovus 21.15-s110_1
#  OS:                Linux x86_64(Host ID lab1-38.eng.utah.edu)
#  Generated on:      Wed Nov 27 16:09:48 2024
#  Design:            Matrix_Multiplier
#  Command:           report_timing > ${rpt_dir}/timing_final.rpt
#############################################################
Path 1: MET Setup Check with Pin Matrix_Mult/regfile_contents_reg[16]/CLK
Endpoint:   Matrix_Mult/regfile_contents_reg[16]/D (^) checked with  leading
edge of 'clk'
Beginpoint: Matrix_Mult/mata_element_reg[0]/Q      (v) triggered by trailing
edge of 'clk'
Path Groups: {clk}
Analysis View: wc
Other End Arrival Time          -0.045
- Setup                          0.267
+ Phase Shift                  300.000
= Required Time                 299.689
- Arrival Time                  156.204
= Slack Time                    143.485
     Clock Fall Edge                 150.000
     + Clock Network Latency (Prop)  0.028
     = Beginpoint Arrival Time       150.028
     +-------------------------------------------------------------------------------------------+
     |              Instance              |     Arc      |   Cell   | Delay | Arrival | Required |
     |                                    |              |          |       |  Time   |   Time   |
     |------------------------------------+--------------+----------+-------+---------+----------|
     | Matrix_Mult/mata_element_reg[0]    | CLK ^        |          |       | 150.028 |  293.513 |
     | Matrix_Mult/mata_element_reg[0]    | CLK ^ -> Q v | DFFQX1   | 0.418 | 150.446 |  293.931 |
     | Matrix_Mult/g6174__1705            | A v -> Z v   | AND2X1   | 1.256 | 151.702 |  295.187 |
     | Matrix_Mult/g6166__9945            | B v -> Z v   | AND2X1   | 0.587 | 152.289 |  295.774 |
     | Matrix_Mult/g6110__2883            | A v -> Z ^   | NOR2X1   | 0.247 | 152.536 |  296.021 |
     | Matrix_Mult/g6097__2802            | B ^ -> Z v   | NOR2X1   | 0.128 | 152.664 |  296.149 |
```

Figure 10: Total Performance from Innovus

```
Instance: /Matrix_Multiplier
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ------------------------------------------------------------------------
       memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
     register     1.55788e-07  6.14811e-06  7.44577e-07  7.04847e-06  53.07%
        latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        logic     1.04947e-07  1.51854e-06  2.93226e-06  4.55575e-06  34.30%
         bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        clock     1.19826e-10  1.39228e-08  1.66387e-06  1.67791e-06  12.63%
          pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
           pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ------------------------------------------------------------------------
    Subtotal     2.60855e-07  7.68057e-06  5.34071e-06  1.32821e-05 100.00%
  Percentage           1.96%       57.83%       40.21%      100.00% 100.00%
  ------------------------------------------------------------------------
```

Figure 11: Total Power of our Matrix Multiplier

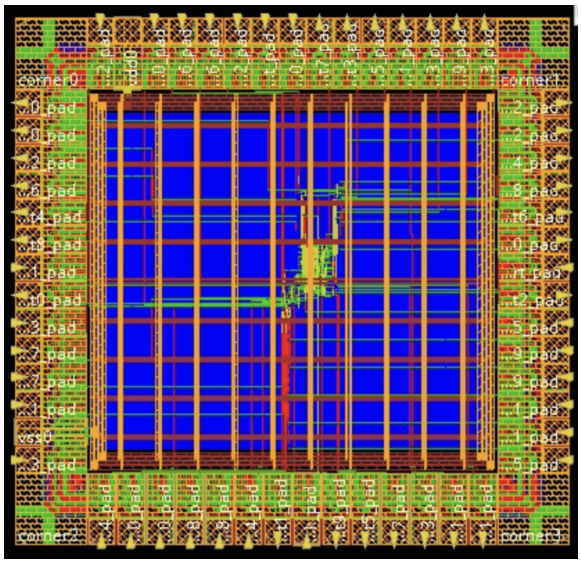


Figure 12: Matrix Multiplier Layout

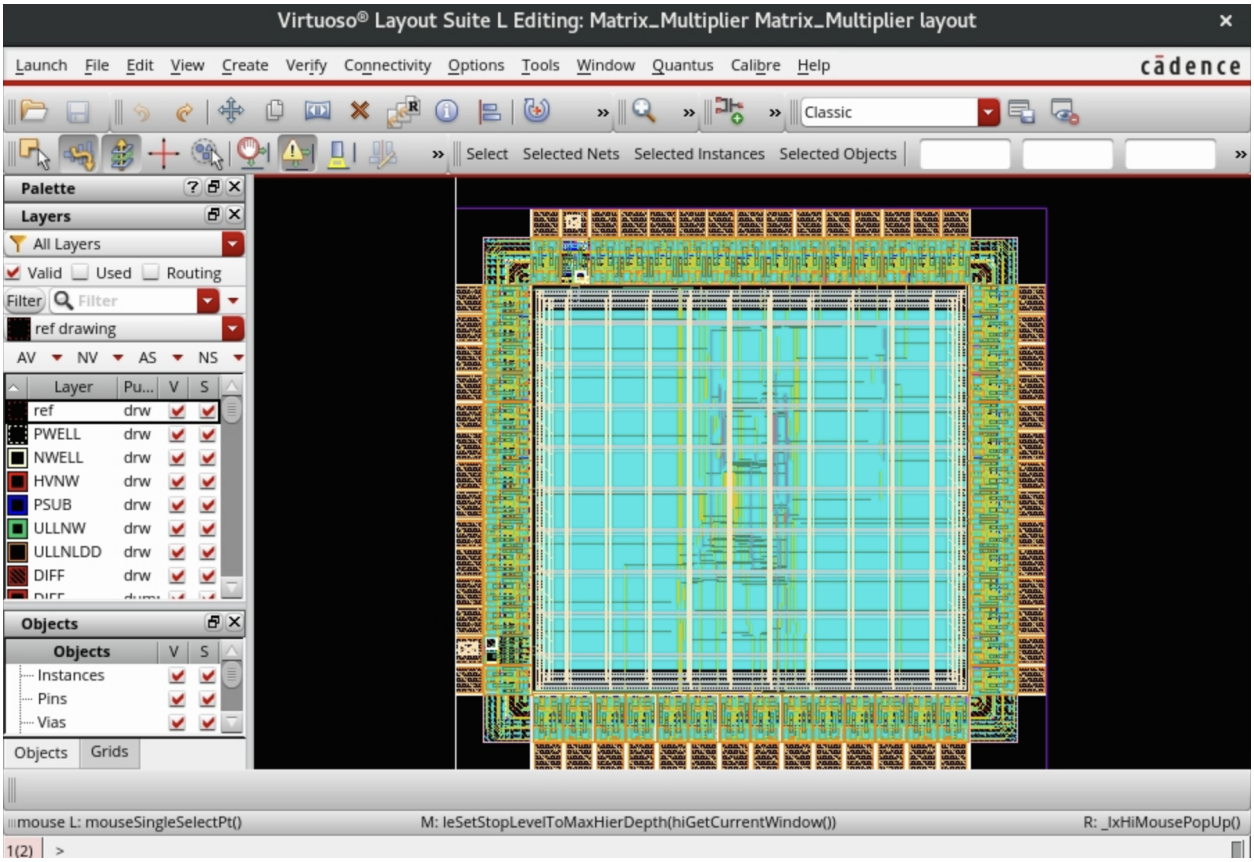


Figure 13: Matrix Multiplier Layout with Voltage and Ground Labeled

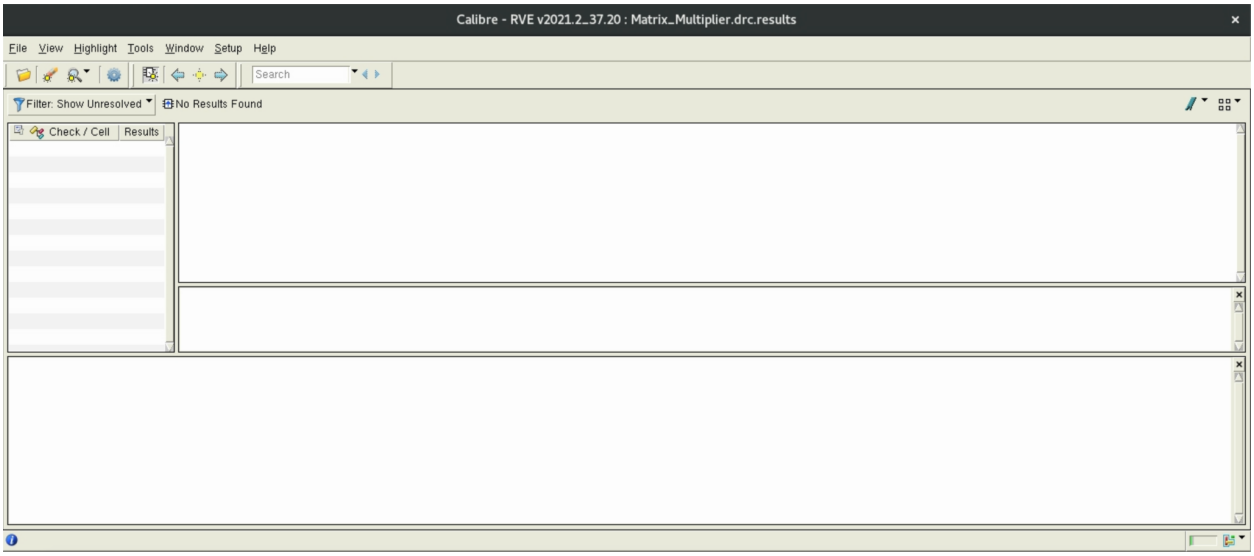


Figure 14: Matrix Multiplier DRC Check

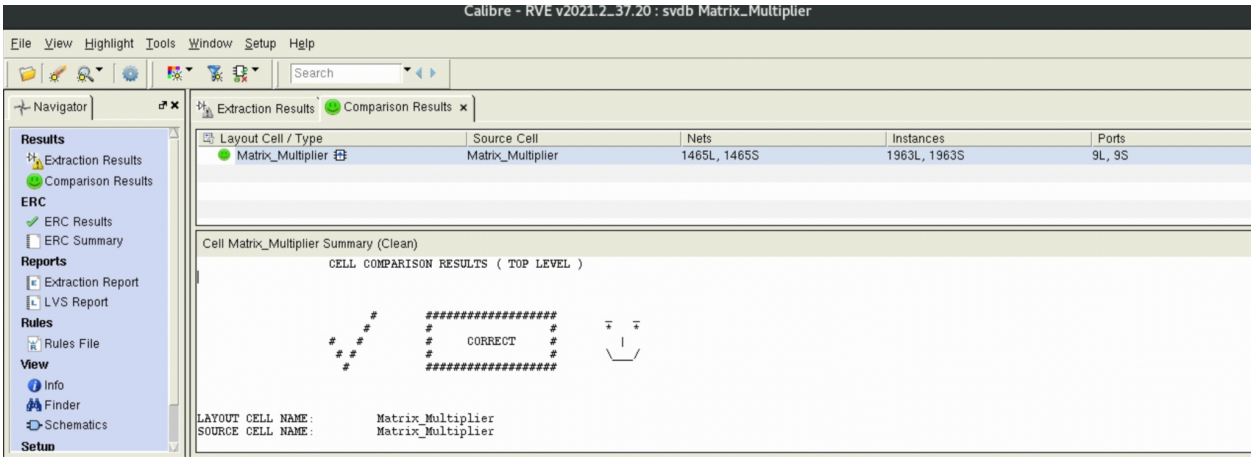


Figure 15: Matrix Multiplier LVS Check

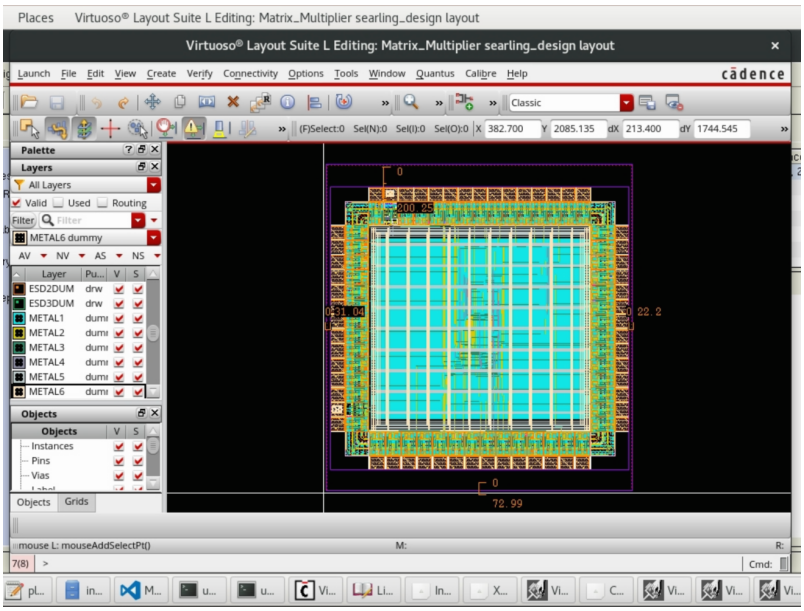


Figure 16: Matrix Multiplier Layout with Sealring

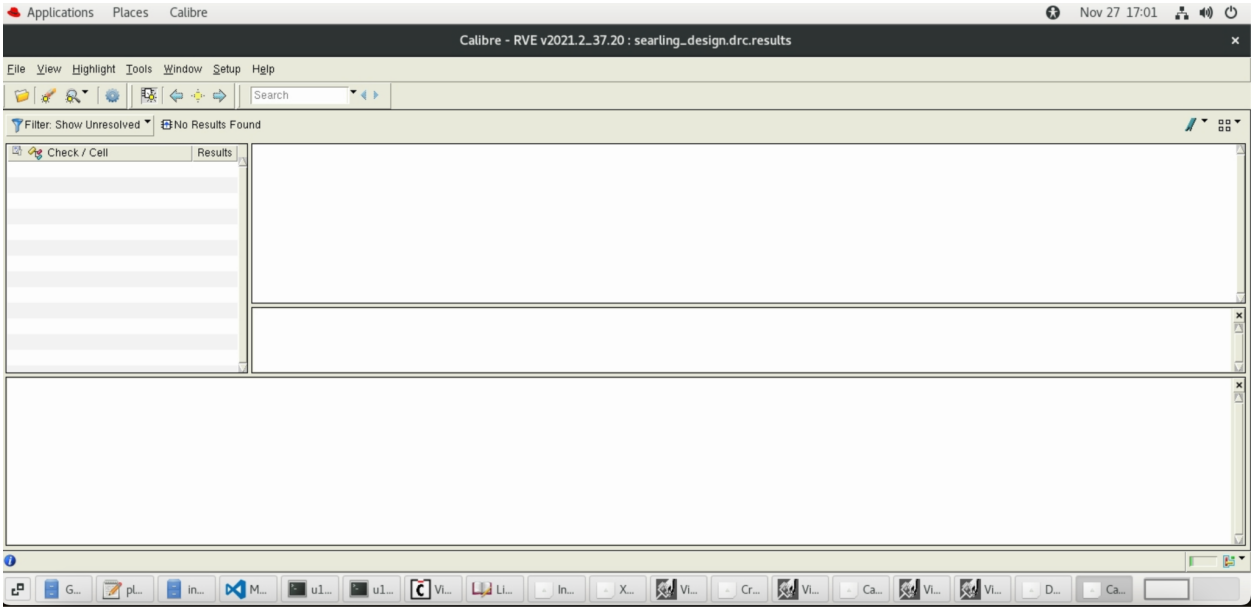


Figure 17: Matrix Multiplier DRC Fullchip Check

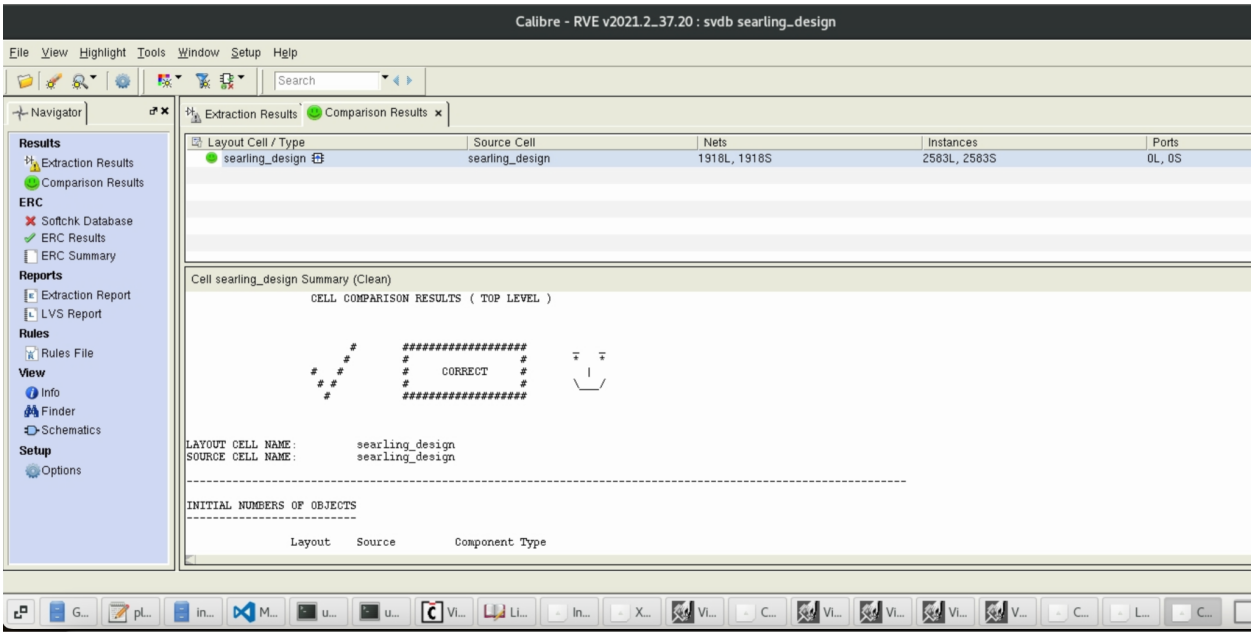


Figure 18: Matrix Multiplier LVS Fullchip Check