BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

VLSI Test & Testability

(Semester 1, Academic year 2025-26)

PROJECT-3

RTL to Layout implementation of systolic array matrix multiplier using the Cadence design suite tools

Submitted to

Dr. Abhijit Asati

By

R V Arun (2024H1230173P)

Sai Sreenivasa Reddy (2024H1400123P)

Venkata Kalyan Nalagandla (2024H1230129P)

M L S Nischay (2024H1230308P)

Manne Chanikya (2024H1230157P)

Aim

To generate a layout for the designed 8 x 8 Systolic array matrix multiplier, with the help of Cadence tools like Genus, NCSim, Innovus & Modus for different steps of the ASIC flow.

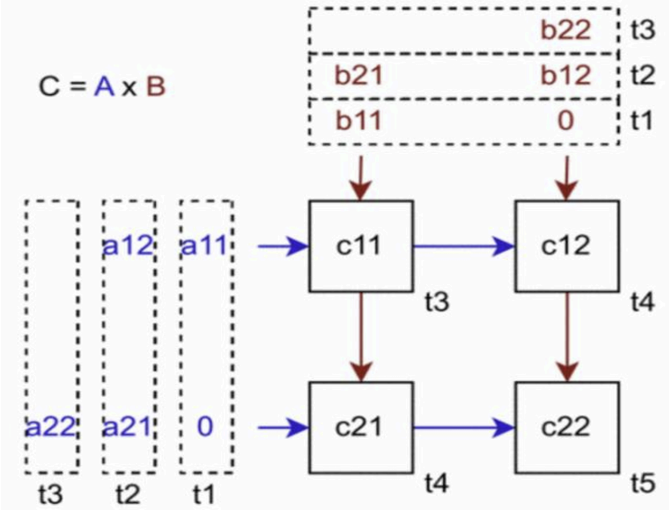
Functional Details – Systolic Array Matrix Multiplier

Systolic arrays represent a specialized form of parallel computing architecture where data flows synchronously across a grid of processing elements (PEs). The typical systolic array for matrix multiplication consists of a 2D grid of PEs. Each PE contains:

1. **Multiplication Unit**: Performs the element-wise multiplication
2. **Accumulation Unit**: Adds the multiplication result to the running sum
3. **Storage Registers**: Hold intermediate values and control signals

Essentially, a PE is a MAC (multiply-and-accumulate) unit. For an m×n by n×p matrix multiplication using an output-stationary approach:

1. Matrix A elements enter from the left edge of the array
2. Matrix B elements enter from the top edge of the array
3. Each PE performs a multiply-accumulate operation
4. Input elements of A and B propagate rightward and downward, respectively
5. Elements of product matrix C are taken from the accumulate registers of each PE



An example of a 2 × 2 matrix multiplier is shown above. Each box indicates a PE, and the above figure shows which element of the product matrix a particular PE corresponds to. For the correct input elements to multiply together, every next row and column entry into the systolic array is delayed by one clock cycle. As the inputs are passed through to neighbouring PEs, input elements need to be supplied only once to the systolic array, maximizing data reuse. Due to the need to provide inputs at designated time slots, a control unit is required. By default, the Genus

synthesis tool uses the CSA tree to implement the multiplication operation. The multiplication is shown for a 2x2 systolic array matrix multiplier.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

The cycle-by-cycle operation in the 2×2 output-stationary systolic array:

Clock Cycle 1:

- PE(1,1): Receives a_{11} and b_{11}, computes $a_{11}\cdot b_{11}$, stores in c_{11}
- Other PEs: Idle

Clock Cycle 2:

- PE(1,1): Receives a_{12} and b_{21}, computes $a_{12}\cdot b_{21}$, adds to c_{11}
- PE(1,2): Receives a_{11} and b_{12}, computes $a_{11}\cdot b_{12}$, stores in c_{12}
- PE(2,1): Receives a_{21} and b_{11}, computes $a_{21}\cdot b_{11}$, stores in c_{21}
- PE(2,2): Idle

Clock Cycle 3:

- PE(1,1): Computation for c_{11} complete ($c_{11} = a_{11}\cdot b_{11} + a_{12}\cdot b_{21}$)
- PE(1,2): Receives a_{12} and b_{22}, computes $a_{12}\cdot b_{22}$, adds to c_{12}
- PE(2,1): Receives a_{22} and b_{21}, computes $a_{22}\cdot b_{21}$, adds to c_{21}
- PE(2,2): Receives a_{21} and b_{12}, computes $a_{21}\cdot b_{12}$, stores in c_{22}

Clock Cycle 4:

- PE(1,2): Computation for c_{12} complete ($c_{12} = a_{11}\cdot b_{12} + a_{12}\cdot b_{22}$)
- PE(2,1): Computation for c_{21} complete ($c_{21} = a_{21}\cdot b_{11} + a_{22}\cdot b_{21}$)
- PE(2,2): Receives a_{22} and b_{22}, computes $a_{22}\cdot b_{22}$, adds to c_{22}

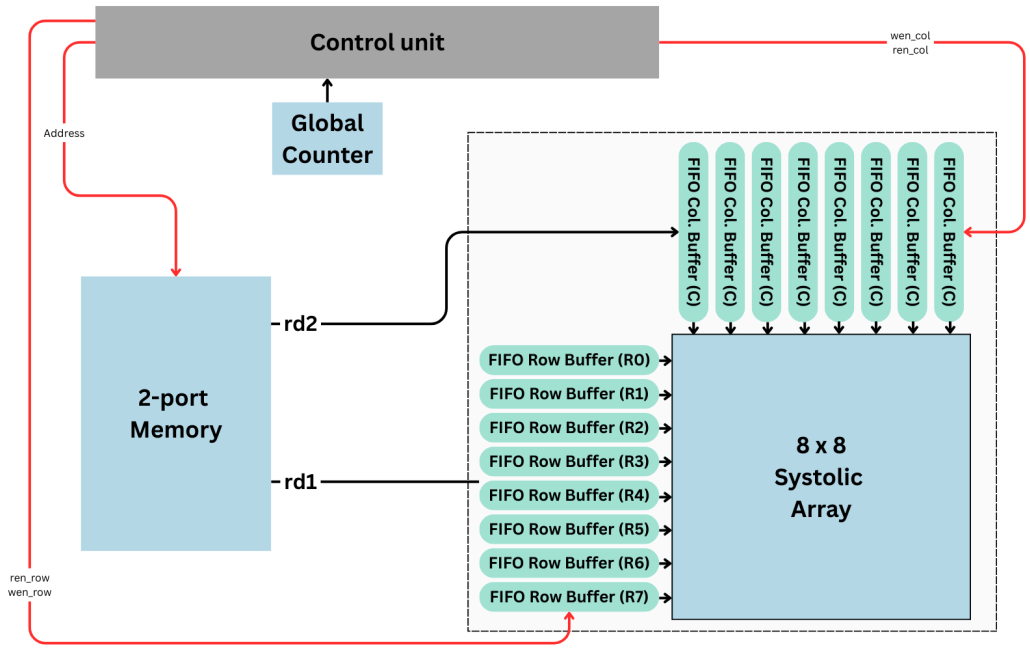
Clock Cycle 5:

- PE(2,2): Computation for c_{22} complete ($c_{22} = a_{21}\cdot b_{12} + a_{22}\cdot b_{22}$)

Thus, the product matrix C is fully computed at the end of the 5th clock cycle. The implemented 8x8 systolic array matrix multiplier computes the product matrix in 23 clock cycles. The same mechanism can be extended to any output-stationary systolic array matrix multiplier of any order. The main principle is that every row and column entry is delayed by one clock cycle with respect to the previous row and column entry. This

Architectural Details

To model a somewhat realistic scenario, we've taken the problem of loading input matrix elements from a 2-port memory, modelled in Verilog, to the Systolic array module. As discussed above, each row/column is delayed by one clock cycle from the previous row/column. To feed matrix elements to the 16 input lines (8 rows and 8 columns) of the systolic array in this timely manner, we use synchronous FIFOs as buffers to temporarily store the values when they are loaded from memory. A control unit is present, which coordinates the loading of data, giving address data to the memory and generating write and read enables of the FIFO buffers during loading and computation. The block diagram is shown below.



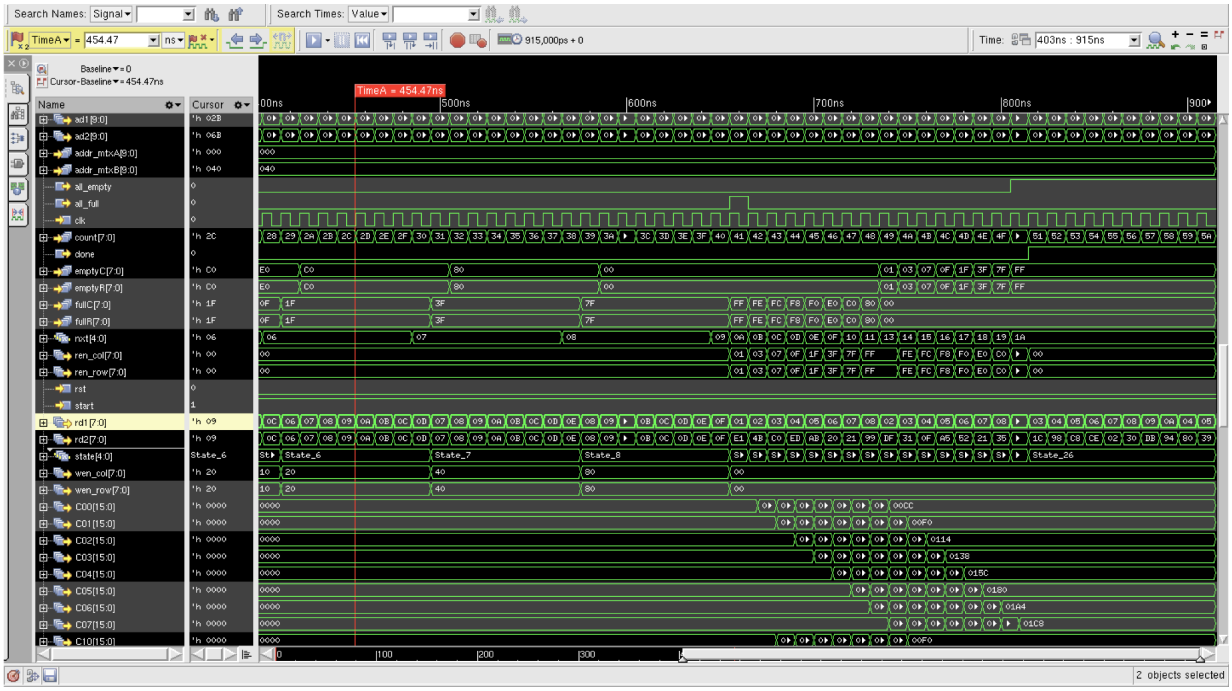
The whole process can be broken down into two phases:

1. Loading phase: Data is loaded into the FIFOs from the memory managed by the control unit. In this phase, the systolic array (the array of PEs) is idle. A counter is used to keep track of the consecutive addresses.
2. Computation phase: The data in the FIFOs is then fed to the systolic array, which is controlled by the control signals from the control unit.

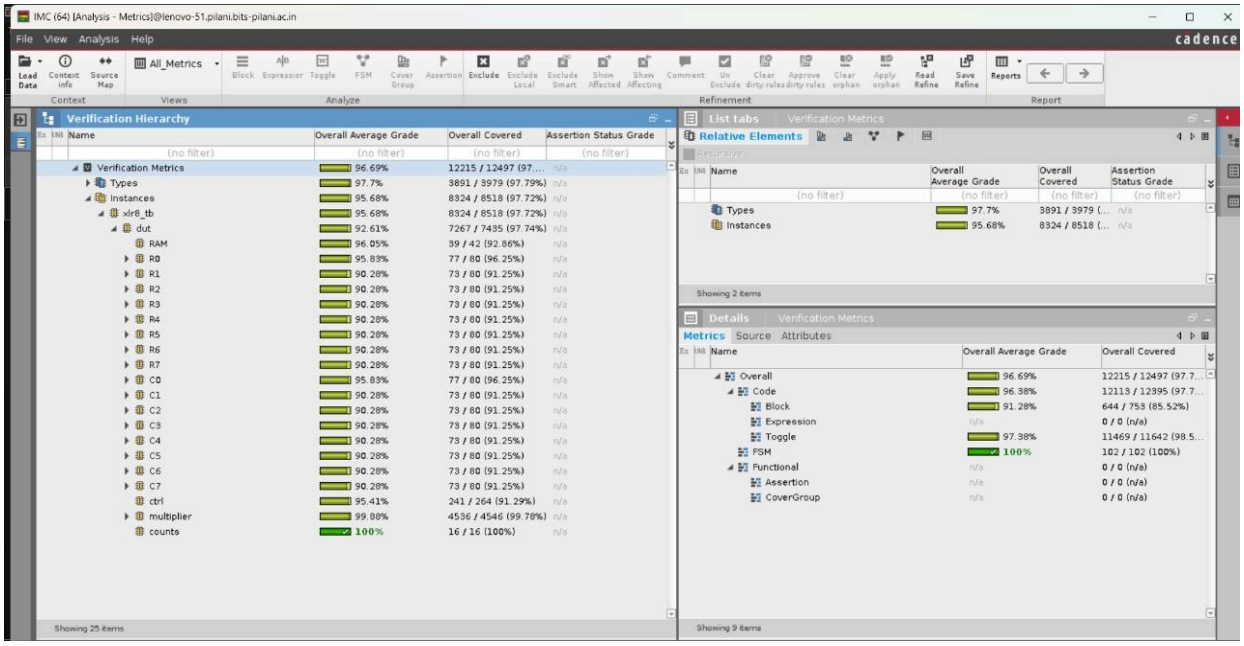
A few measures were taken to keep the design simple, including storing all elements of a single matrix in contiguous memory locations. The first matrix is stored in a row-major fashion, while the second matrix is stored in a column-major fashion. All blocks run on the same clock signal.

Functional verification using Cadence NCSim

To verify the design, we stored sample matrix elements in the memory using a '.mem' file. The top module consisted of the two-port memory, the systolic array, FIFO buffers, a global counter and the control unit. For verification purposes, we take all 64 outputs from the systolic array. A portion of the simulation waveforms is shown below, highlighting the important control signals and a few outputs (C00, C01...). The signals starting with 'wen' & 'ren' are the write and read enables for the row & column FIFOs.



The above figure shows the computation phase, whose start is indicated by the 'all_full' signal going high at approximately 650 ns. The coverage results for this run are shown below.



Synthesis with scan chain insertion using Cadence Genus

The target synthesis library was a 90nm technology node. Since Genus maps constant registers to zero, the 2-port memory was removed from the top module, and the interface was modified accordingly. Synthesis was done with the typical operating conditions (typical.lib). Synthesis was performed with clock periods of 2 ns, 3 ns, and 4 ns. The 4ns implementation gave enough slack margin that would allow to accommodate extra datapath time post-layout. Post-synthesis metrics are tabulated below.

Time Period	Data-path delay	Slack	Setup	Total Area	Total Power	Cell count
4000ps	3004	814	162	189800.241	19.01mW	16548

Normal FFs were converted to scan FFs, and the files required for Modus were generated in the ‘test_scripts’ folder. Muxed scan chain style was selected. The FIFO and systolic array registers were converted into scan flops, resulting in a single scan chain with a length of 3,185 bits.

Synthesis script

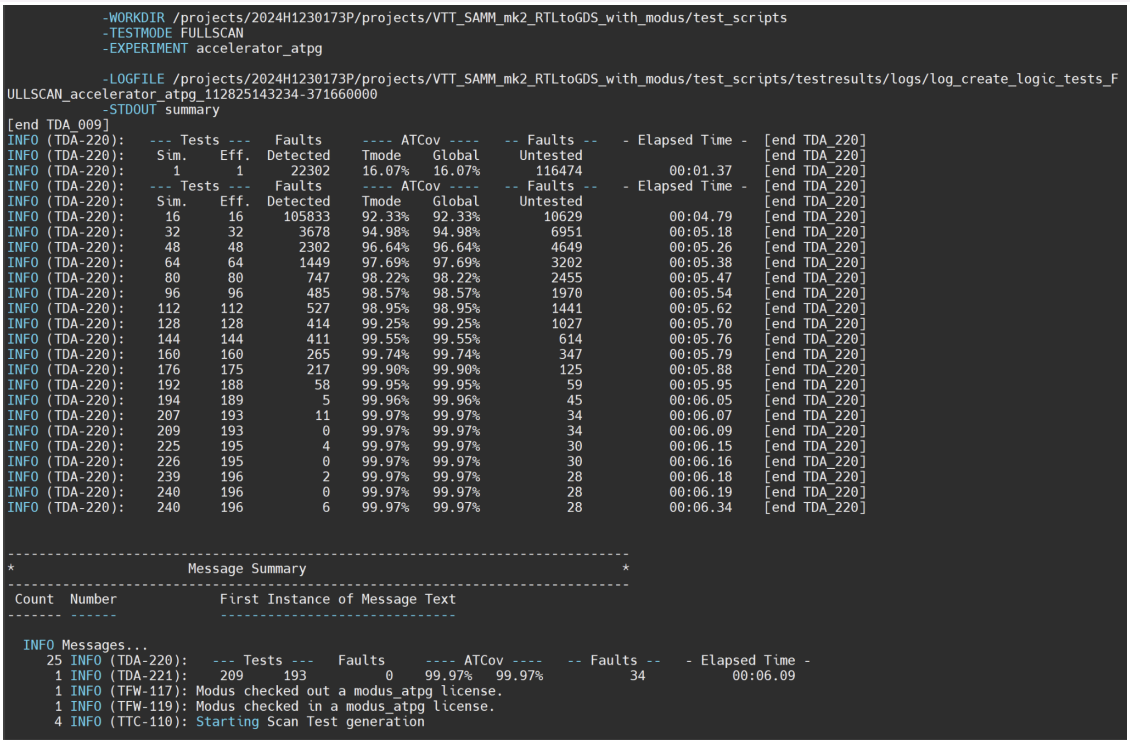
```
read_libs /linuxeda_new/c2scadence/FOUNDRY/digital/90nm/dig/lib/typical.lib
read_hdl xlr.v
read_hdl counter.v
read_hdl synchr_fifo.v
read_hdl fifo_zero_injector.v
read_hdl SA.v
read_hdl PE.v
read_hdl SA_control.v

elaborate

read_sdc sys_array.sdc

#retime -min_area
set_db dft_scan_style muxed_scan
define_shift_enable -name test_enable -active high -create_port test_enable
check_dft_rules

syn_generic
syn_map
map_dft_unmapped_logic accelerator
syn_opt

check_dft_rules
define_scan_chain -name scan_chain -sdi scan_in -sdo scan_out -create_ports
connect_scan_chains -auto_create_chains
gui_show
```

```
write_hdl > sys_Array_netlist_4ns_posedgeSA.v
write_sdc > output_constraints1_4ns_posedgeSA.sdc
write_sdf > output_constraints_4ns_posedgeSA.sdf
write_dft_atpg -library
/linuxeda_new/c2scadence/FOUNDRY/digital/90nm/dig/vlog/typical.v

report timing >> ./reports/SA_timing_4ns_posedgeSA.rpt
report power >> ./reports/SA_power_4ns_posedgeSA.rpt
report area >> ./reports/SA_area_4ns_posedgeSA.rpt
report gates >> ./reports/SA_gates_4ns_posedgeSA.rpt
report dft_chains >> ./reports/SA_dft_chains_4ns_posedgeSA.rpt
report dft_setup >> ./reports/SA_dft_setup_4ns_posedgeSA.rpt
report_scan_chains >> ./reports/SA_scan_chains_4ns_posedgeSA.rpt
```

ATPG and DFT Coverage metrics using Cadence Modus

Genus generates a TCL file that needs to be run after invoking Modus to get coverage results for the inserted scan chain in the design. From the directory where Genus was run, Modus is invoked and source that TCL file present in the ‘test_scripts’ folder.

```
source test_scripts/runmodus.atpg.tcl
```



```
          -WORKDIR /projects/2024H1230173P/projects/VTT_SAMM_mk2_RTLtoGDS_with_modus/test_scripts
          -TESTMODE FULLSCAN
          -EXPERIMENT accelerator_atpg

          -LOGFILE /projects/2024H1230173P/projects/VTT_SAMM_mk2_RTLtoGDS_with_modus/test_scripts/testresults/logs/log_create_logic_tests_F
ULLSCAN_accelerator_atpg_112825143234-371660000
          -STDOUT summary
[end TDA_009]
INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                        [end TDA_220]
INFO (TDA-220):       1       1     22302   16.07%   16.07%       116474          00:01.37    [end TDA_220]
INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                        [end TDA_220]
INFO (TDA-220):      16      16     105833   92.33%   92.33%        10629          00:04.79    [end TDA_220]
INFO (TDA-220):      32      32       3678   94.98%   94.98%         6951          00:05.18    [end TDA_220]
INFO (TDA-220):      48      48       2302   96.64%   96.64%         4649          00:05.26    [end TDA_220]
INFO (TDA-220):      64      64       1449   97.69%   97.69%         3202          00:05.38    [end TDA_220]
INFO (TDA-220):      80      80        747   98.22%   98.22%         2455          00:05.47    [end TDA_220]
INFO (TDA-220):      96      96        485   98.57%   98.57%         1970          00:05.54    [end TDA_220]
INFO (TDA-220):     112     112        527   98.95%   98.95%         1441          00:05.62    [end TDA_220]
INFO (TDA-220):     128     128        414   99.25%   99.25%         1027          00:05.70    [end TDA_220]
INFO (TDA-220):     144     144        411   99.55%   99.55%          614          00:05.76    [end TDA_220]
INFO (TDA-220):     160     160        265   99.74%   99.74%          347          00:05.79    [end TDA_220]
INFO (TDA-220):     176     175        217   99.90%   99.90%          125          00:05.88    [end TDA_220]
INFO (TDA-220):     192     188         58   99.95%   99.95%           59          00:05.95    [end TDA_220]
INFO (TDA-220):     194     189          5   99.96%   99.96%           45          00:06.05    [end TDA_220]
INFO (TDA-220):     207     193         11   99.97%   99.97%           34          00:06.07    [end TDA_220]
INFO (TDA-220):     209     193          0   99.97%   99.97%           34          00:06.09    [end TDA_220]
INFO (TDA-220):     225     195          4   99.97%   99.97%           30          00:06.15    [end TDA_220]
INFO (TDA-220):     226     195          0   99.97%   99.97%           30          00:06.16    [end TDA_220]
INFO (TDA-220):     239     196          2   99.97%   99.97%           28          00:06.18    [end TDA_220]
INFO (TDA-220):     240     196          0   99.97%   99.97%           28          00:06.19    [end TDA_220]
INFO (TDA-220):     240     196          6   99.97%   99.97%           28          00:06.34    [end TDA_220]


------------------------------------------------------------------------
*                     Message Summary                                  *
------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
     25 INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -
      1 INFO (TDA-221):     209     193          0    99.97%   99.97%           34          00:06.09
      1 INFO (TFW-117): Modus checked out a modus_atpg license.
      1 INFO (TFW-119): Modus checked in a modus_atpg license.
      4 INFO (TTC-110): Starting Scan Test generation
```

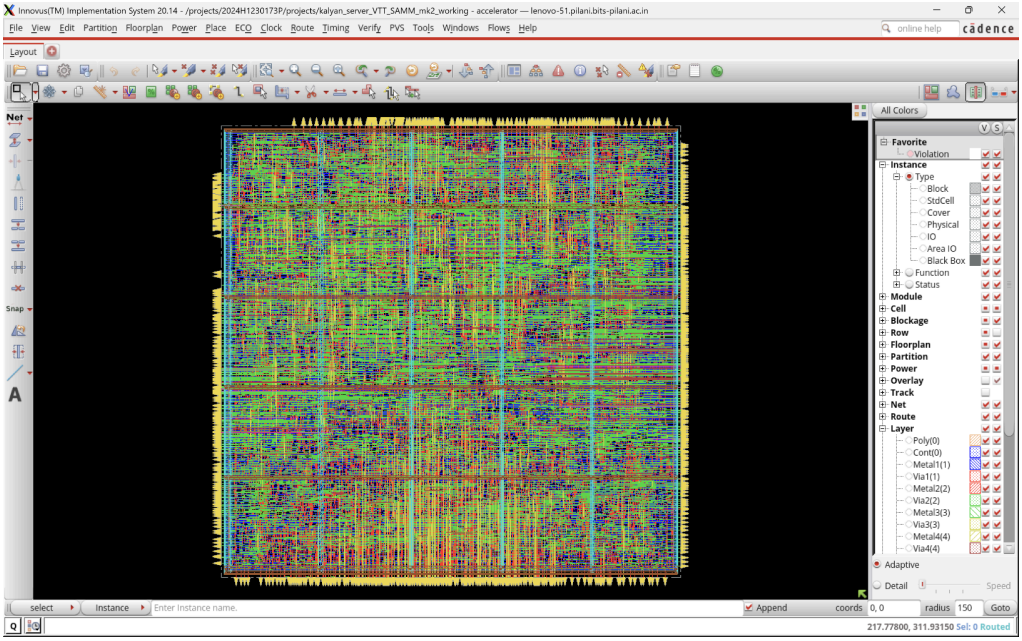
The results of ATPG and coverage are shown above. We see that 99.97% of fault coverage is achieved.

Layout generation using Cadence Innovus

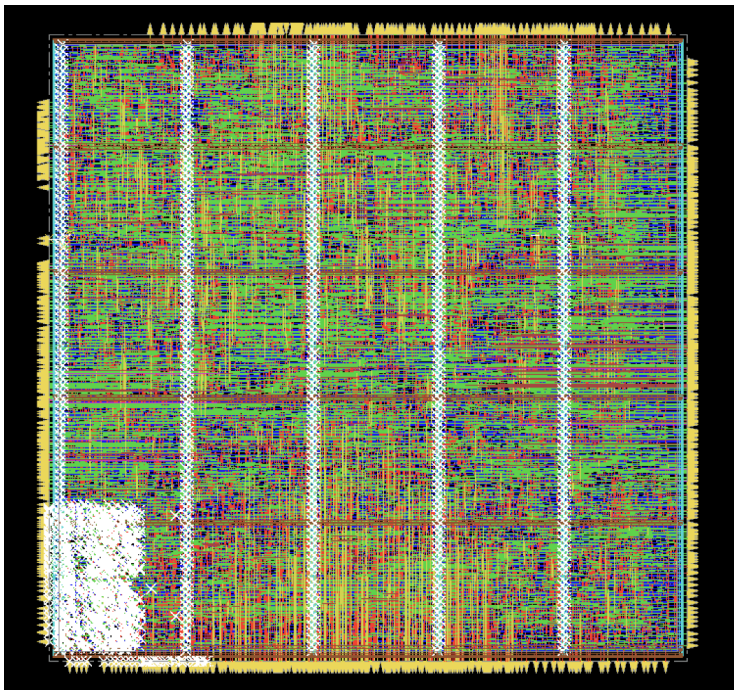
After synthesis, Genus generates a netlist file (.v) and an output constraints file (.sdc). These files are required by Innovus to map standard cells & interconnections to their layout and provide timing constraints to adhere to. After loading the files and setting up the analysis window, power rails and stripes were laid down. Then, full floor planning, placement, and routing were completed. Before placement, the ‘create scan chain’ command is used to map FFs to scan flops in the layout.

```
createScanChain scan_chain -start scan_in -stop scan_out
```

All cells were placed and no antenna violations were found. However, power via and DRC errors were found, mostly in the lower left section of the design. The layout is shown below.



In the figure below, the DRC error points are shown in white.

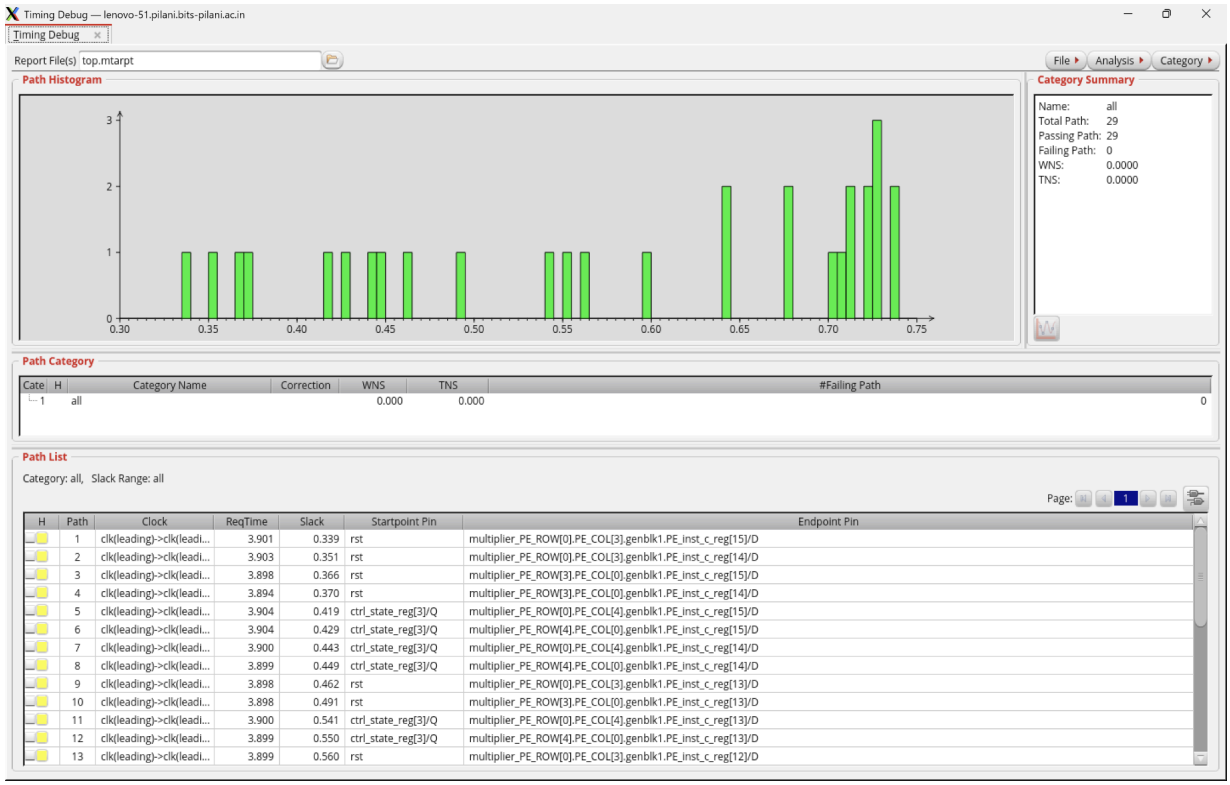


The power via and DRC error logs are shown below.

```
innovus 3>
******** Start: VERIFY POWER VIA ********
Start Time: Thu Nov 27 14:32:01 2025

Check all 2 Power/Ground nets
*** Checking Net vdd
Found missing via(s) on net vdd.
*** Checking Net vss
Found missing via(s) on net vss.
Actually Checked 2 Power/Ground nets with physical connectivity

Begin Summary
    930 Problem(s) Found between Layers: Metal1 and Metal8.
    930 total info(s) created.
End Summary

End Time: Thu Nov 27 14:32:02 2025
******** End: VERIFY POWER VIA ********
  Verification Complete : 930 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.6  MEM: 75.004M)


******** Start: VERIFY POWER VIA ********
Start Time: Thu Nov 27 14:32:06 2025

Check all 2 Power/Ground nets
*** Checking Net vdd
Found missing via(s) on net vdd.
*** Checking Net vss
Found missing via(s) on net vss.
Actually Checked 2 Power/Ground nets with physical connectivity

Begin Summary
    930 Problem(s) Found between Layers: Metal1 and Metal8.
    930 total info(s) created.
End Summary

End Time: Thu Nov 27 14:32:07 2025
******** End: VERIFY POWER VIA ********
  Verification Complete : 930 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.5  MEM: 1.004M)
```

```
  VERIFY DRC ..... Deleting Existing Violations
  VERIFY DRC ..... Creating Sub-Areas
  VERIFY DRC ..... Using new threading
  VERIFY DRC ..... Sub-Area: {0.000 0.000 128.640 124.800} 1 of 16
**WARN: (IMPVFG-1103):  VERIFY DRC did not complete: Number of violations exceeds the Error Limit [1000]

  Verification Complete : 1000 Viols.

  Violation Summary By Layer and Type:

                  Mar outOfDie   AdjCut   Totals
       Metal2     895       39        0      934
       Via2         0        0       14       14
       Metal3       0       44        0       44
       Metal4       0        6        0        6
       Metal5       0        1        0        1
       Metal6       0        1        0        1
       Totals     895       91       14     1000

 *** End Verify DRC (CPU: 0:00:00.4  ELAPSED TIME: 0.00  MEM: 8.0M) ***

innovus 3> #-check_ndr_spacing auto               # enums={true false auto}, default=auto,  user setting
#-report accelerator.drc.rpt           # string, default="", user setting
 *** Starting Verify DRC (MEM: 1872.3) ***

  VERIFY DRC ..... Starting Verification
  VERIFY DRC ..... Initializing
  VERIFY DRC ..... Deleting Existing Violations
  VERIFY DRC ..... Creating Sub-Areas
  VERIFY DRC ..... Using new threading
  VERIFY DRC ..... Sub-Area: {0.000 0.000 128.640 124.800} 1 of 16
**WARN: (IMPVFG-1103):  VERIFY DRC did not complete: Number of violations exceeds the Error Limit [1000]

  Verification Complete : 1000 Viols.

  Violation Summary By Layer and Type:

                  Mar outOfDie   AdjCut   Totals
       Metal2     895       39        0      934
       Via2         0        0       14       14
       Metal3       0       44        0       44
       Metal4       0        6        0        6
       Metal5       0        1        0        1
       Metal6       0        1        0        1
       Totals     895       91       14     1000

 *** End Verify DRC (CPU: 0:00:00.4  ELAPSED TIME: 0.00  MEM: 0.0M) ***
```

There were zero paths that failed timing, with minimum slack being about 330ps. The timing debug histogram is shown below.



Summary

The post-layout metrics are tabulated below for the implemented systolic array matrix multiplier.

Clock Period	4 ns
Datapath Delay	3.562 ns
Setup Time	0.089 ns
Slack	0.339 ns
Cell Count	16548
Die Area	~2,51,904.4143um^2 (497.53um × 506.31um)
Internal Power	22.23mW
Switching Power	3.556mW
Leakage Power	0.8762mW
Total Power	26.66mW
PDP	9.49E-11 Ws