Permonace Modeling of Pipelined Linear Algebra Architectures on ASIC

Shubhi Sharma[1], Vidyadhar Jambhale[2], Abhijeet Shinde[3], Sivnantham S[4]

[1]*M.Tech VLSI Design, Vellore Institute of Technology, Vellore*
[2]*M.Tech VLSI Design, Vellore Institute of Technology, Vellore*
[3]*M.Tech VLSI Design, Vellore Institute of Technology, Vellore*
[4]*Professor, Dept. of electronics Engineering, Vellore Institute of Technology, Tamil Nadu, India*

---***---

Abstract— *The elements that characterize execution of a specific usage incorporate the engineering design, number of pipelines and memory bandwidth. Present mathematical model based on above factor is used for calculation of pipelined ASIC accelerators computational time. Linear algebra computation are the main contributors to that of total execution time as they are used for many compute intensive application. Since many combinational implementations are not feasible as their number of operating bits continue to increase so pipelined architecture has been designed for moving data swiftly. We have performed the ASIC implementation comprising of code coverage, floorplan, routing and placement.*

Key Words: **ASIC, Pipelining, MAC, Synthesis**

1.INTRODUCTION

The core of these applications is frequently made out of linear algebra based math calculations, for example, dot product, matrix–matrix multiplication [1], matrix–vector multiplication [2], matrix inverse and matrix decomposition. The model parameters for this architecture is used and defined to calculate time of execution. The mathematical model is going to be based on the factors such as number of pipelined used and memory bandwidth limitations. We have performed the ASIC [4] implementations of the modules using the Verilog [3] code and synthesizing it for obtaining the floorplan.

2. DOT PRODUCT

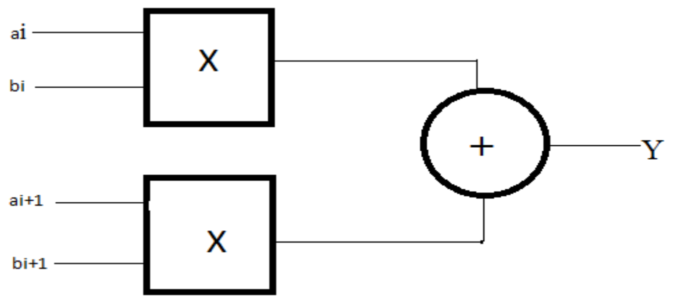


Fig -1: Dot product presentation

Here the pipeline [4] structure is defined as two multipliers and an adder that is able to calculate the dot product of two element vectors. For vectors of size N the number of pipeline used is N/2 and the number of iteration required is 1. For example, a computation using 8 element vectors will require 4 pipelines and will compute the swift operation in one iteration only. Then, the results of the multiplications are accumulated in an adder tree. Although adding more pipelines allows more work to be done in parallel. These additional adders will increase the overall latency of the architecture.

3. MARTRIX – VECTOR MULTIPLICATION

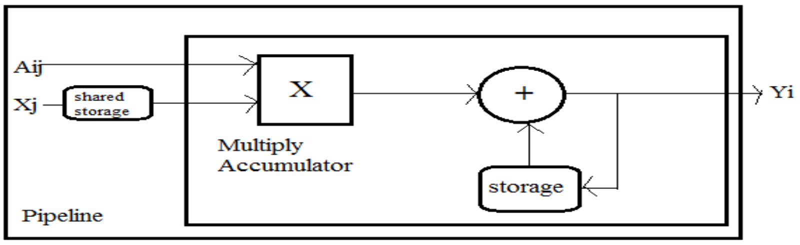


Fig -2: Pipelined Architecture

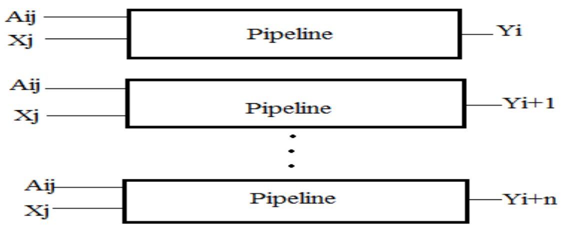


Fig -3: Multiple pipeline structure

The matrix–vector multiplication architecture defines a pipeline as a single multiply accumulate (MAC) unit [2]. The matrix operands be utilized once, yet the vector values can be used again if put away locally. We assume that these vector values are stored in a single on-chip memory accessible to every pipeline. With a specific end goal to spare memory exchange speed the vector value is secured until all multiplication using it are finished as shown in. Since each element in the vector will be multiplied by an element in every row of the matrix, the quantity of Iterations required to finish the figuring of a single value in the resulting vector is M for M x N matrices. This design performs best with one pipeline for each component in the vector to finish the outcomes in parallel. This will require N pipelines, or N Uses of a single pipeline. Shows the organization of multiple pipelines. Every use of a pipeline produces one impetus in the consequent vector. If there are less pipelines than the size of the output vector, some pipelines will compute multiple values in the resulting vector.

4. MATRIX -MATRIX MULTIPLICATION

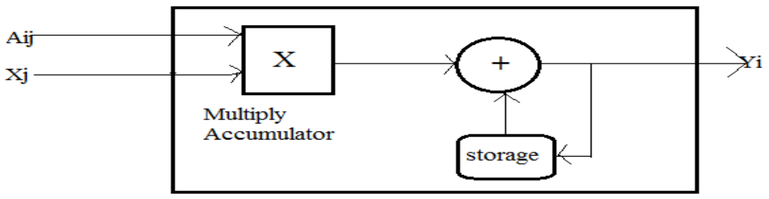
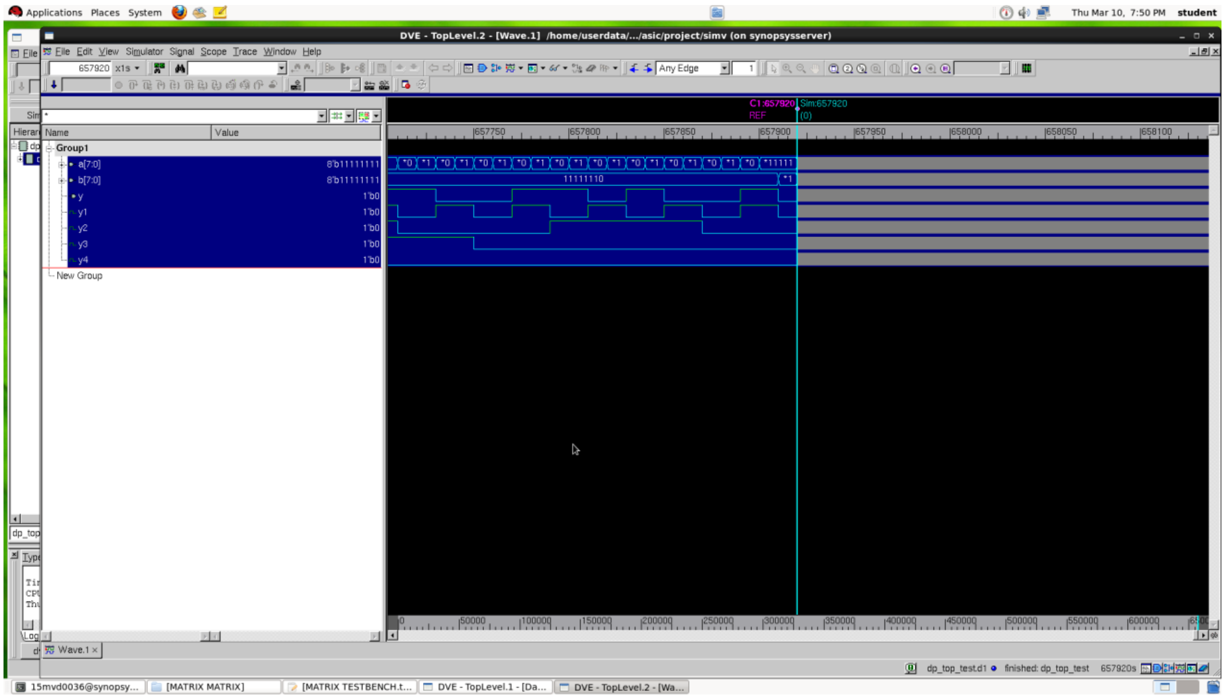


Fig -4: Matrix multiplication architecture

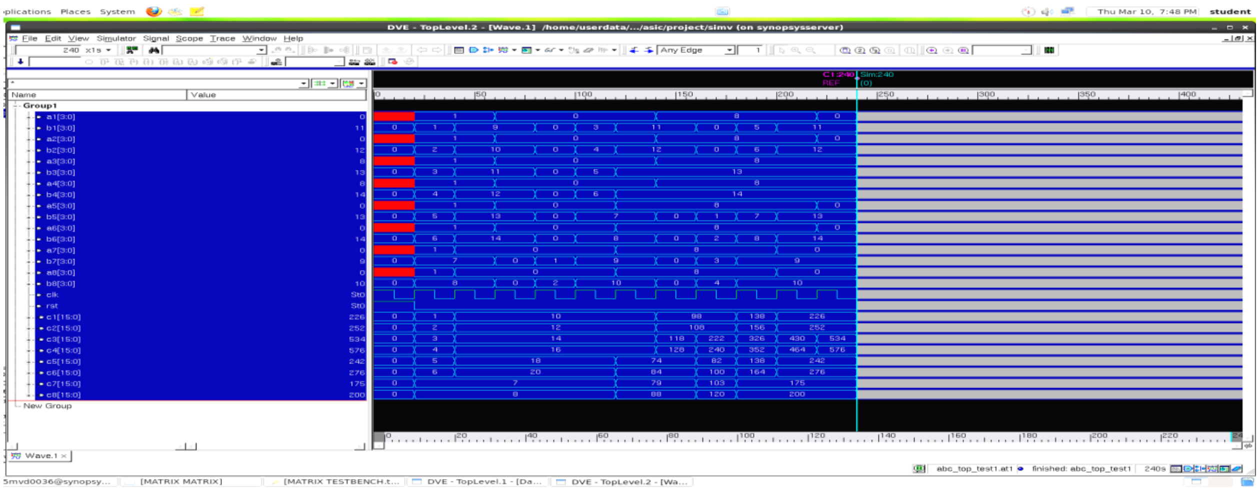
In mathematics, matrix multiplication is a binary operation that takes a pair of matrices, and produces another matrix. Numbers such as the real or complex numbers can be multiplied according to elementary arithmetic. On the other hand, matrices are varieties of numbers, so there is no one of a kind approach to characterize the multiplication of grids. As such, in general the term "matrix multiplication" refers to a number of different ways to multiply matrices. The key components of any matrix augmentation incorporate: the number of rows and columns the original matrices have. Similar to the matrix–vector multiplication architecture, the matrix–matrix multiplication pipeline is composed of a single MAC unit [2]. However, for this computation it is not necessary to store the second input (vector value). Standard matrix– matrix multiplication requires N3 operations. For small matrices it is practical to have N2 MACs and require N iterations for the computation to finish. However as the size of as the matrix develops, this methodology gets to be distinctly unrealistic. We expect that the input matrices can be broken down into smaller, square basic blocks of dimension D _ D. The reference pipeline architecture performs a full matrix multiplication of two of these blocks, A and B, in D iterations using D2 MAC units. During one iteration, each element in a column from matrix A is multiplied by each component in a row from matrix B. This size D is known as the basic block (BB) size. We assume the input matrices are sized M _ N and N _ P, and block-based multiplication can be performed.

5. SIMULATION RESULTS

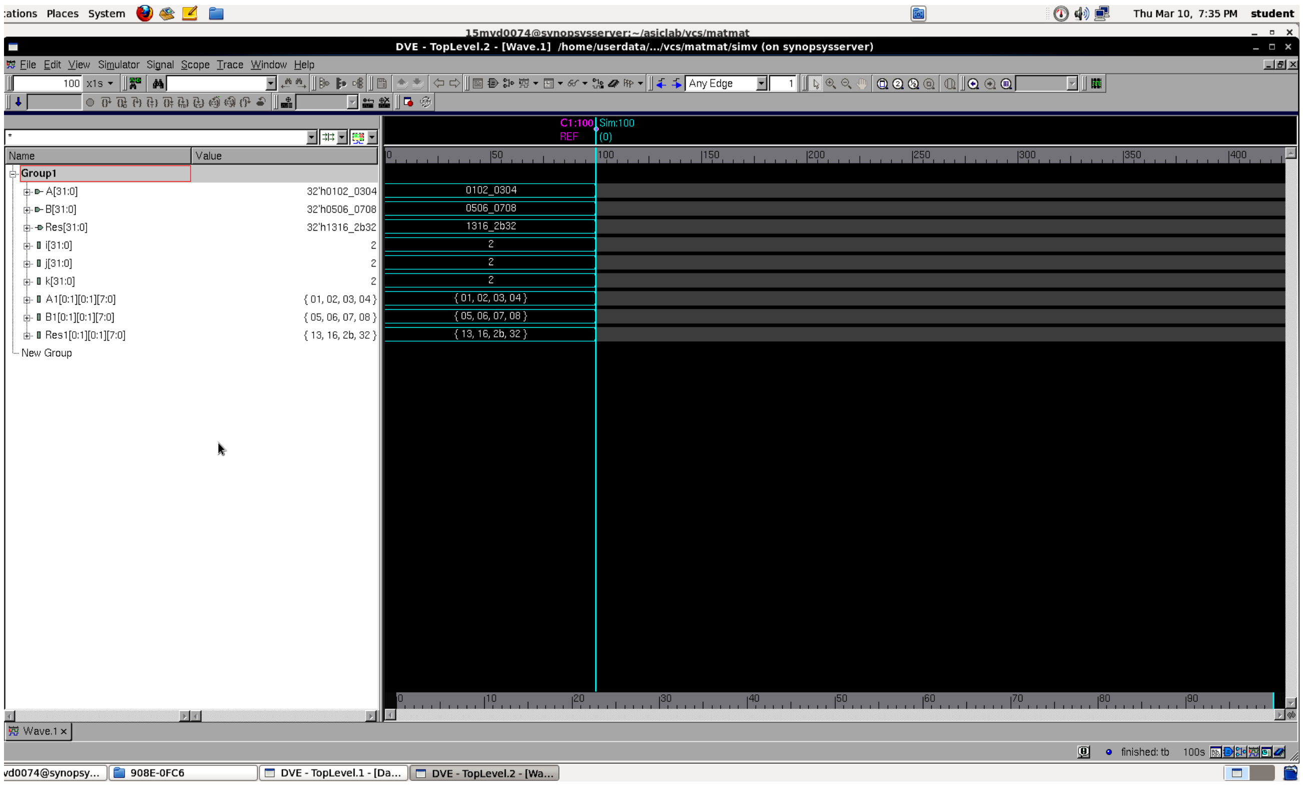
5.1.1 Dot product


DVE - TopLevel.2 - [Wave.1] /home/userdata/.../asic/project/simv (on synopsysserver)
Group1
a[7:0]
b[7:0]
y
y1
y2
y3
y4
New Group
Wave.1

5.1.2 Matrix-vector multiplication



5.1.3 Matrix-matrix multiplication

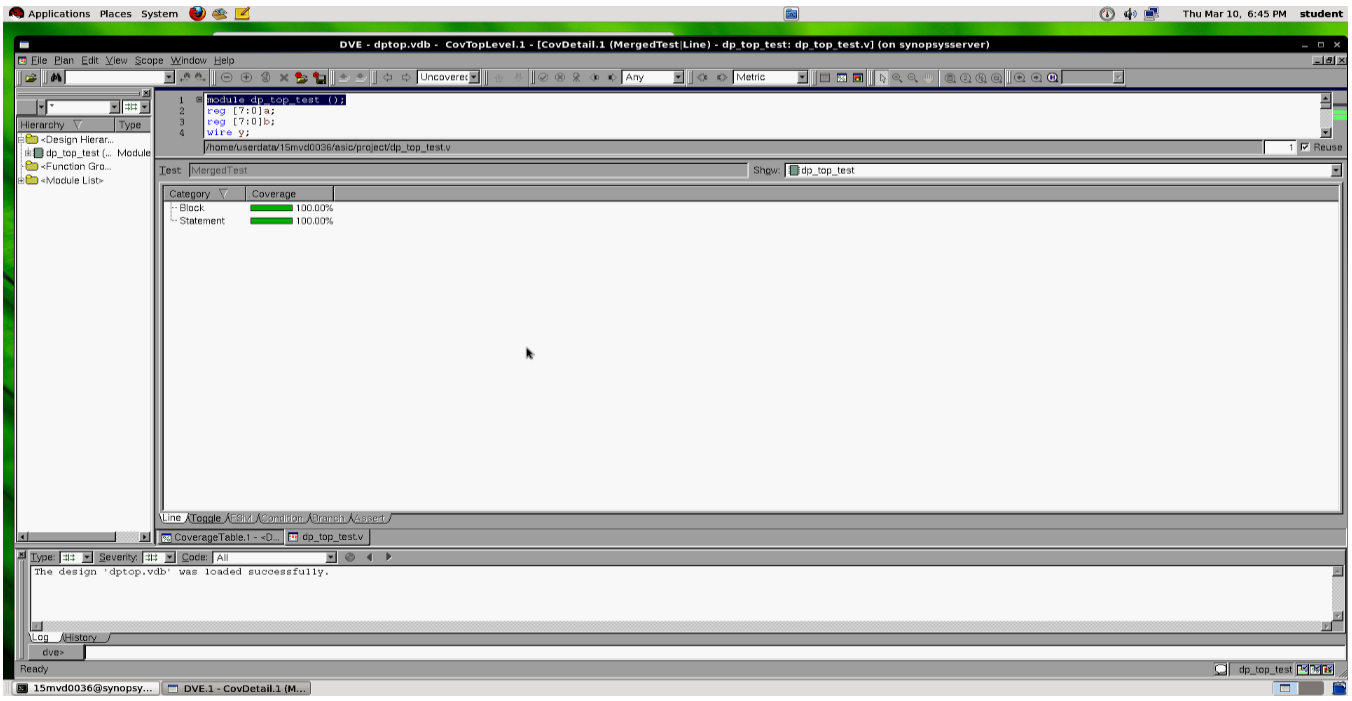


DVE - TopLevel.2 - [Wave.1] /home/userdata/.../vcs/matmat/simv (on synopsysserver)

Name	Value
Group1	
A[31:0]	32'h0102_0304
B[31:0]	32'h0506_0708
Res[31:0]	32'h1316_2b32
i[31:0]	2
j[31:0]	2
k[31:0]	2
A1[0:1][0:1][7:0]	{ 01, 02, 03, 04 }
B1[0:1][0:1][7:0]	{ 05, 06, 07, 08 }
Res1[0:1][0:1][7:0]	{ 13, 16, 2b, 32 }
New Group	

5.2 Code coverage

5.2.1 Dot product



DVE - dptop.vdb - CovTopLevel.1 - [CovDetail.1 (MergedTest|Line) - dp_top_test: dp_top_test.v] (on synopsysserver)

```
module dp_top_test ();
reg [7:0]a;
reg [7:0]b;
wire y;
```

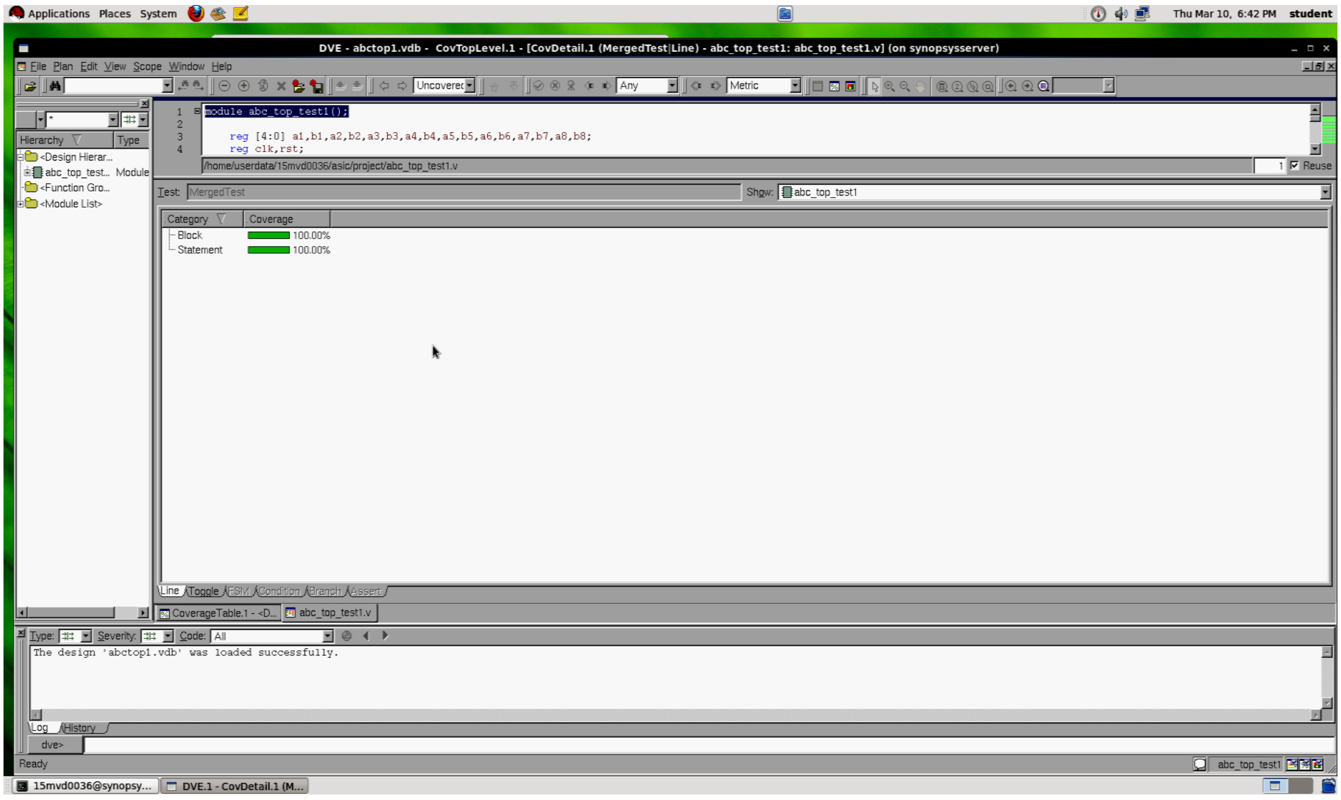
/home/userdata/15mvd0036/asic/project/dp_top_test.v

Test: MergedTest Show: dp_top_test

Category	Coverage
Block	100.00%
Statement	100.00%

```
The design 'dptop.vdb' was loaded successfully.
```

5.2.2 Matrix-vector multiplication


DVE - abctop1.vdb - CovTopLevel.1 - [CovDetail.1 (MergedTest|Line) - abc_top_test1: abc_top_test1.v] (on synopsysserver)

```
module abc_top_test1();

    reg [4:0] a1,b1,a2,b2,a3,b3,a4,b4,a5,b5,a6,b6,a7,b7,a8,b8;
    reg clk,rst;
```

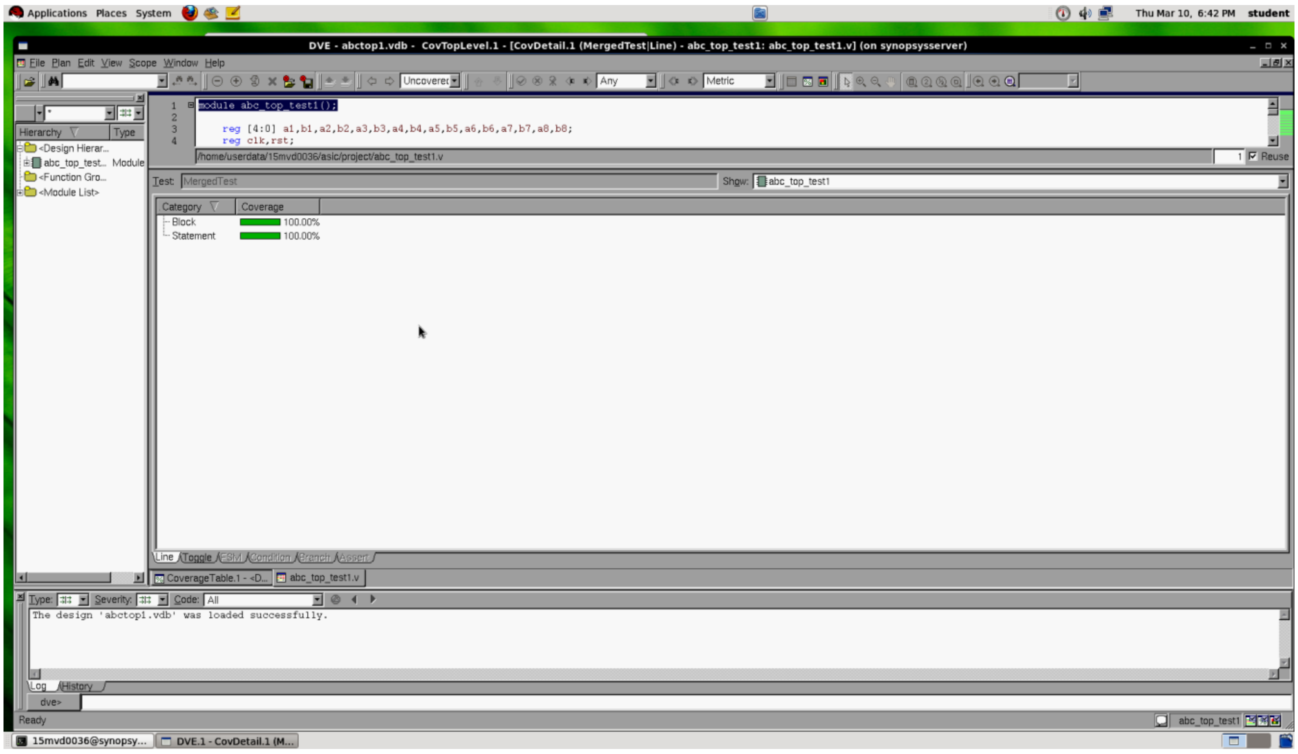
/home/userdata/15mvd0036/asic/project/abc_top_test1.v

Test: MergedTest Show: abc_top_test1

Category	Coverage
Block	🟩 100.00%
Statement	🟩 100.00%

```
The design 'abctop1.vdb' was loaded successfully.
```

5.2.3 Matrix-matrix multiplication


DVE - abctop1.vdb - CovTopLevel.1 - [CovDetail.1 (MergedTest|Line) - abc_top_test1: abc_top_test1.v] (on synopsysserver)

```
module abc_top_test1();

    reg [4:0] a1,b1,a2,b2,a3,b3,a4,b4,a5,b5,a6,b6,a7,b7,a8,b8;
    reg clk,rst;
```

/home/userdata/15mvd0036/asic/project/abc_top_test1.v

Test: MergedTest Show: abc_top_test1

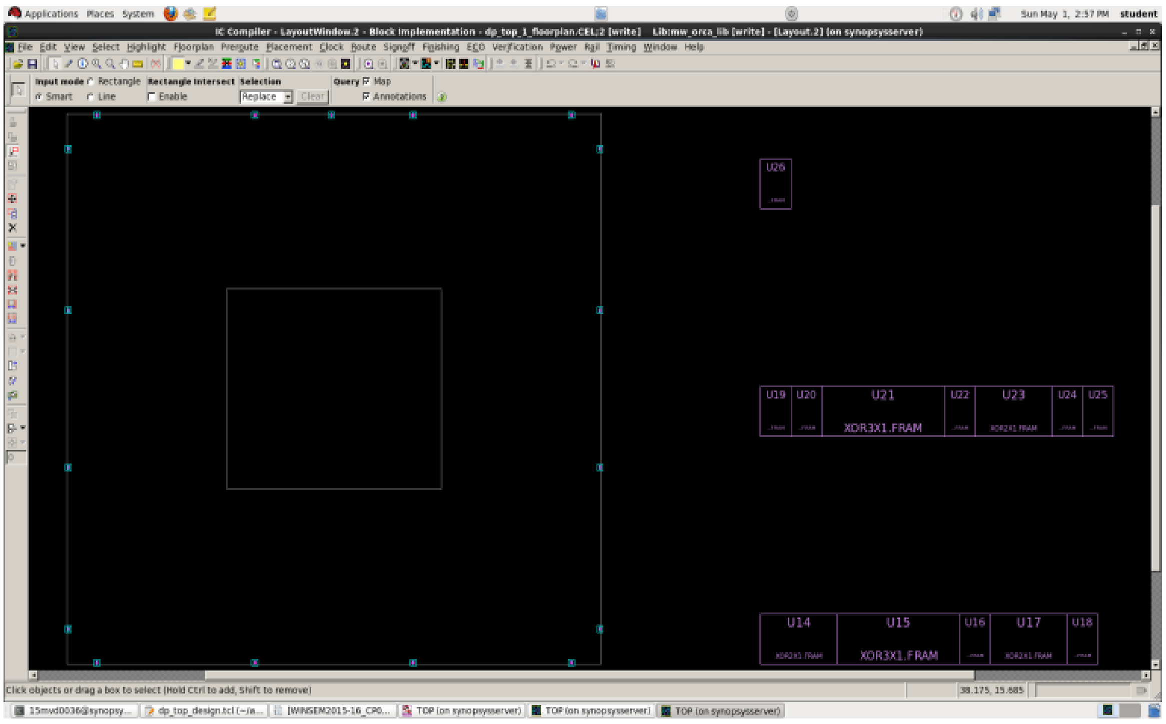
Category	Coverage
Block	🟩 100.00%
Statement	🟩 100.00%

```
The design 'abctop1.vdb' was loaded successfully.
```

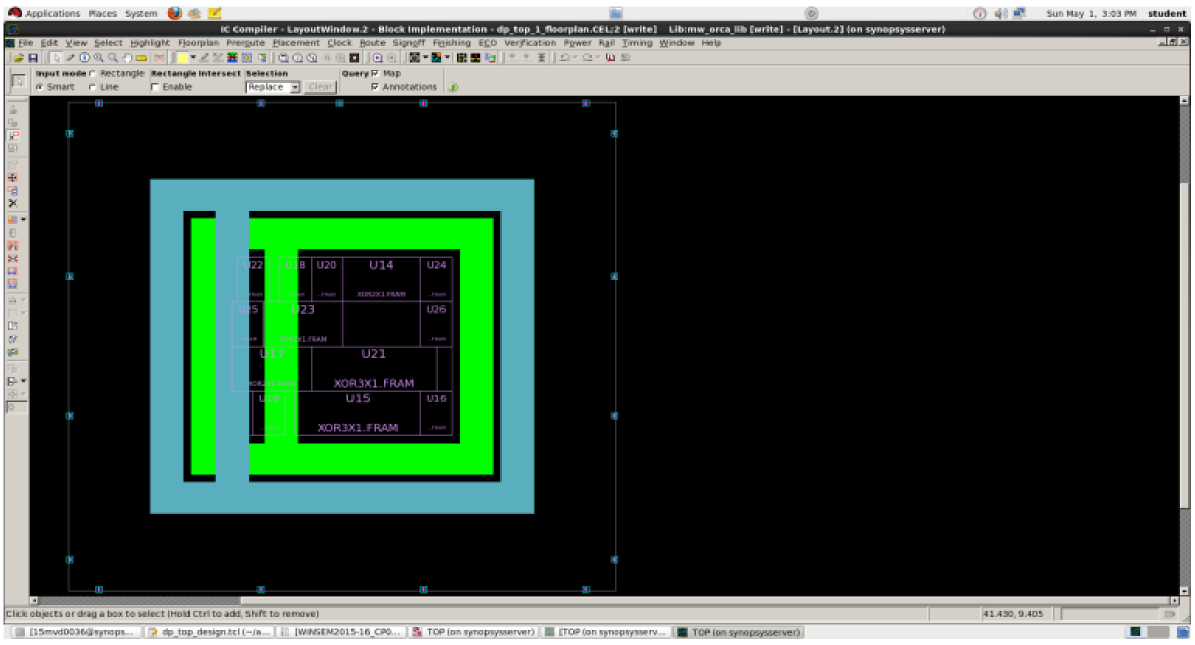
5.3 ICC SNAPSHOT

5.3.1 Dot product

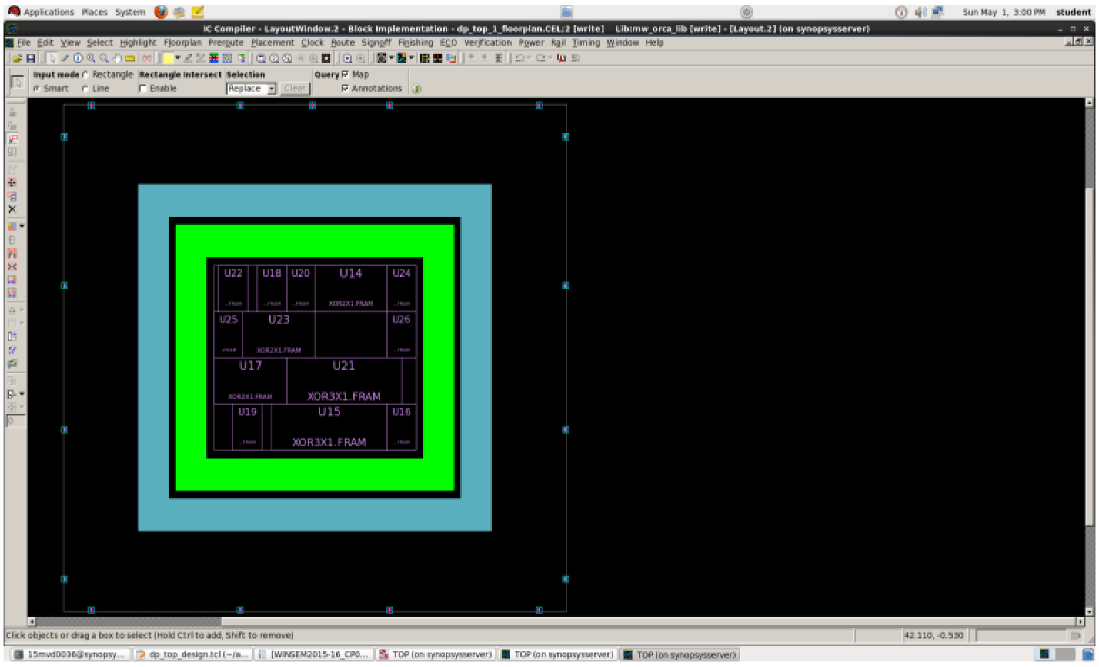
5.3.1.1 Floor plan


IC Compiler - LayoutWindow.2 - Block Implementation - dp_top_1_floorplan.CEL;2 [write] Lib:mw_orca_lib [write] - [Layout.2] (on synopsysserver)
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
U26
U19
U20
U21
XOR3X1.FRAM
U22
U23
U24
U25
U14
U15
XOR3X1.FRAM
U16
U17
U18
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
38.175, 15.685

5.3.1.2 Power rings & straps


IC Compiler - LayoutWindow.2 - Block Implementation - dp_top_1_floorplan.CEL;2 [write] Lib:mw_orca_lib [write] - [Layout.2] (on synopsysserver)
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
U20
U14
U24
XOR2X1.FRAM
U23
U26
U21
XOR3X1.FRAM
U15
U16
XOR3X1.FRAM
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
41.430, 9.405

5.3.1.3 Placement

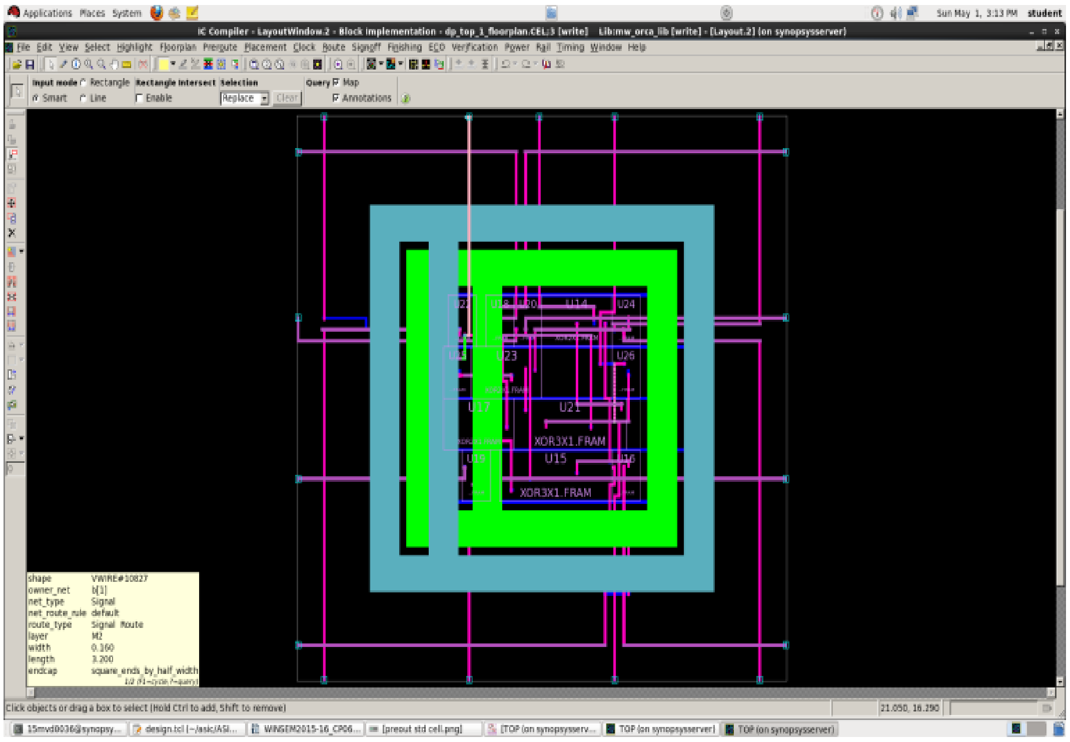

IC Compiler - LayoutWindow.2 - Block Implementation - dp_top_1_floorplan.CEL;2 [write] Lib:mw_orca_lib [write] - [Layout.2] (on synopsysserver)

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

42.110, -0.530

Applications Places System
Sun May 1, 3:00 PM student
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
Input mode Rectangle Smart Line
Rectangle Intersect Enable
Selection Replace Clear
Query Map Annotations
U22
U18
U20
U14
U24
XOR2X1.FRAM
U25
U23
U26
XOR2X1.FRAM
U17
U21
XOR2X1.FRAM
XOR3X1.FRAM
U19
U15
U16
XOR3X1.FRAM
15mvd0036@synopsy...
dp_top_design.tcl (~/a...
[WINSEM2015-16_CP0...
TOP (on synopsysserver)
TOP (on synopsysserver)
TOP (on synopsysserver)

5.3.1.4 cts and routing


IC Compiler - LayoutWindow.2 - Block Implementation - dp_top_1_floorplan.CEL;3 [write] Lib:mw_orca_lib [write] - [Layout.2] (on synopsysserver)

```
shape            VWIRE#10827
owner_net        b[1]
net_type         Signal
net_route_rule   default
route_type       Signal Route
layer            M2
width            0.160
length           3.200
endcap           square_ends_by_half_width
```

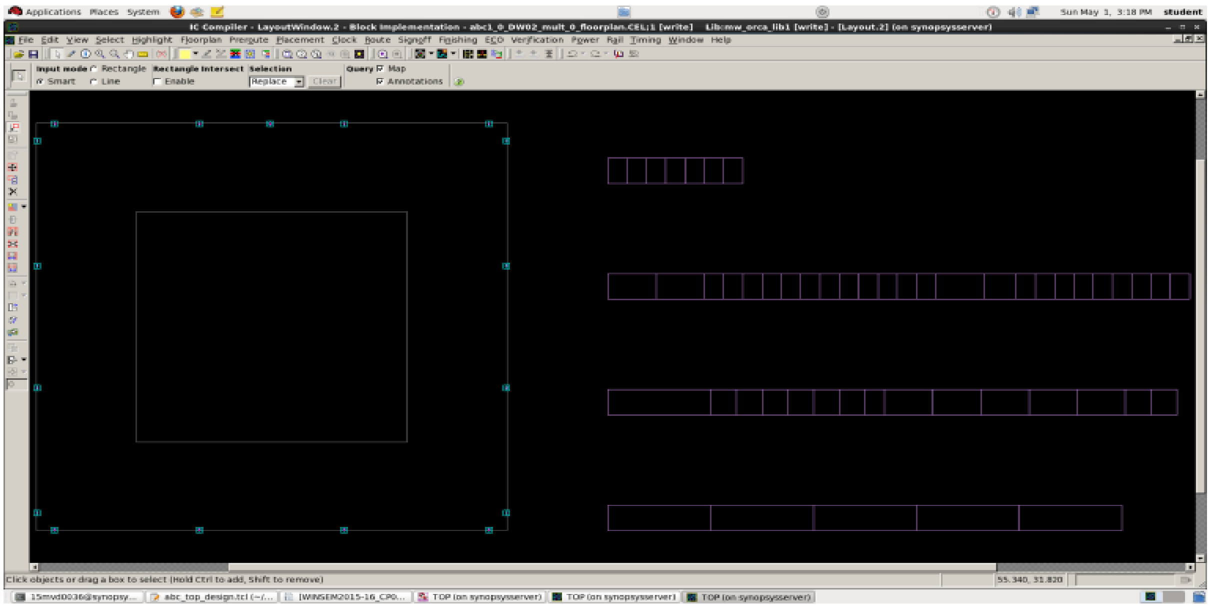
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

21.050, 16.290

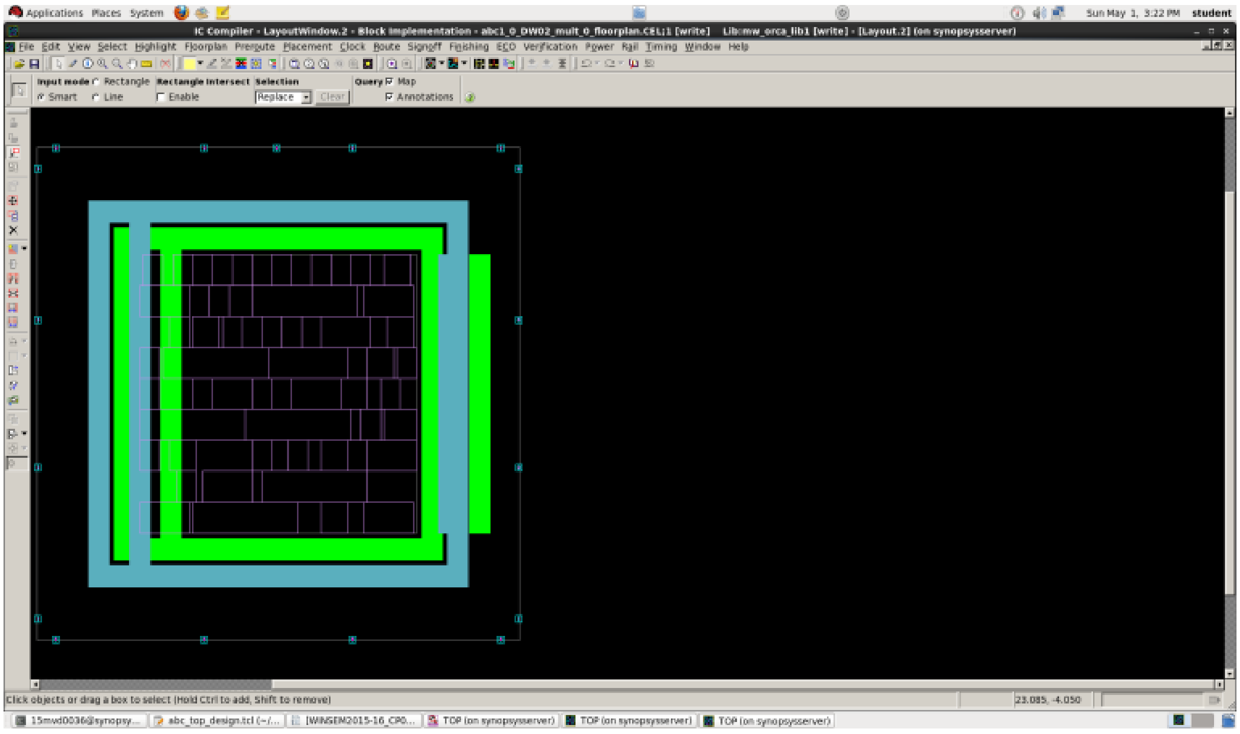
Applications Places System
Sun May 1, 3:13 PM student
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
Input mode Rectangle Smart Line
Rectangle Intersect Enable
Selection Replace Clear
Query Map Annotations
U22
U18
U20
U14
U24
U23
U26
U17
U21
XOR3X1.FRAM
U19
U15
U16
XOR3X1.FRAM
15mvd0036@synopsy...
design.tcl (~/asic/ASI...
WINSEM2015-16_CP06...
[preout std cell.png]
[TOP (on synopsysserv...
TOP (on synopsysserver)
TOP (on synopsysserver)

5.3.2 Matrix vector

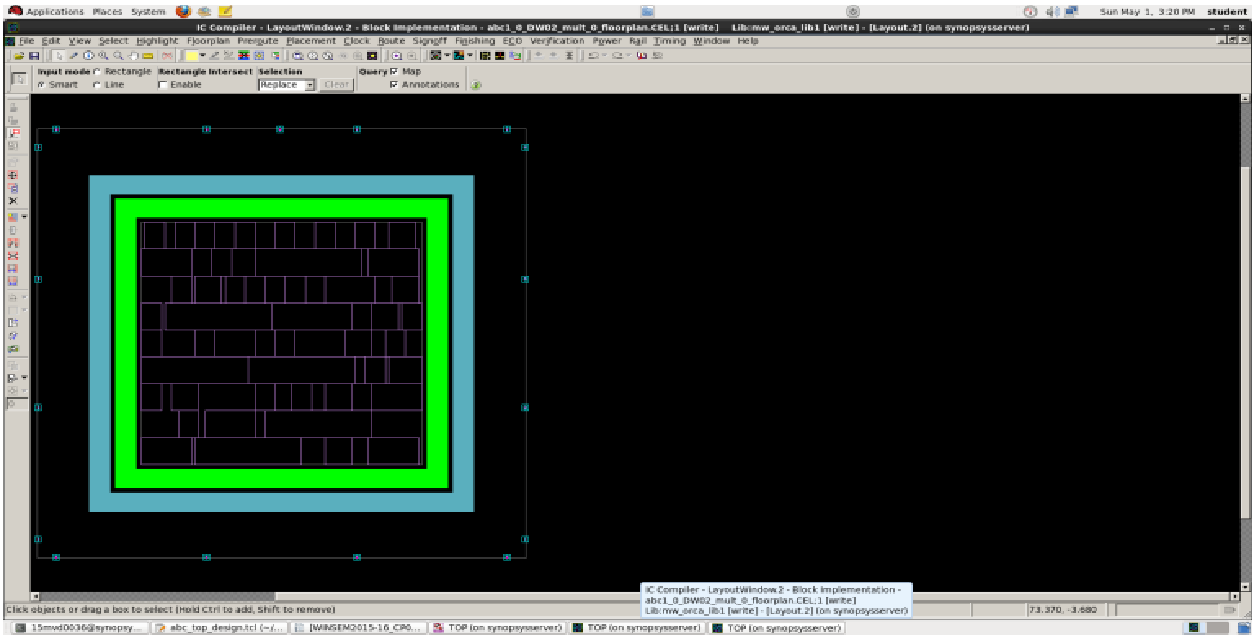
5.3.2.1 Floor plan


IC Compiler - LayoutWindow.2 - Block Implementation - abc1_0_DW02_mult_0_floorplan.CEL;1 [write] Lib:mw_orca_lib1 [write] - [Layout.2] (on synopsysserver)
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
55.340, 31.820

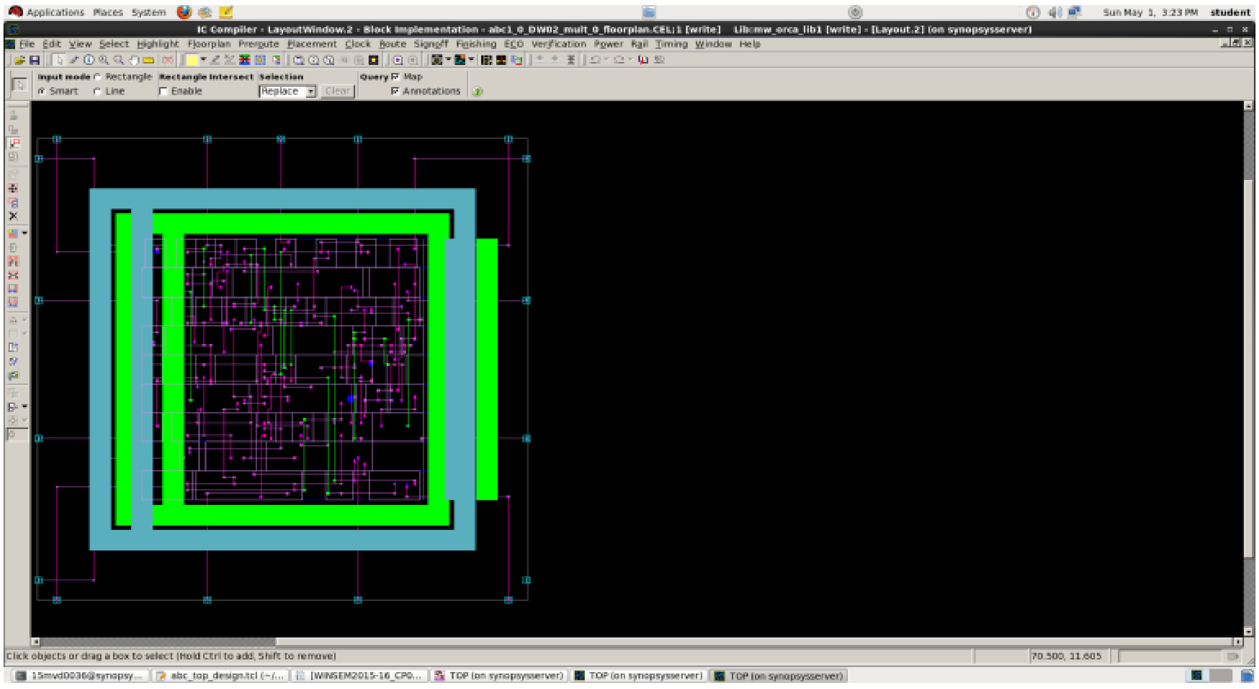
5.3.2.2 Power rings & straps


IC Compiler - LayoutWindow.2 - Block Implementation - abc1_0_DW02_mult_0_floorplan.CEL;1 [write] Lib:mw_orca_lib1 [write] - [Layout.2] (on synopsysserver)
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing ECO Verification Power Rail Timing Window Help
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
23.085, -4.050

5.3.2.3 Placement



5.3.2.4 cts and routing



6. CONCLUSIONS

Since many combinational implementations are not feasible as their number of operating bits continue to increase so we have designed pipelined architecture for moving data swiftly. Synthesize outputs along with the 100% code coverage is carried out, also the placement and routing of cell is implemented.

ACKNOWLEDGEMENT

We acknowledge the help of Prof. Sivanatham S who guided us helped a lot to get idea and methodologies to do this project. We are also thanking our program chair Prof.K Sivasnskaran. We are indebted to our lab assistant Prof.Karthikeyan and all other faculties of VLSI System department. Above all we thank god almighty. Also we are thanking to our classmates and family who supported us.

REFERENCES

[1] Guo Li, Wang Miao-feng, Qiu Tian, Liu Lu, Luo Feng, "Bit-Level Parallel Array Algorithms of Vector-Vector and Matrix-Matrix Multiplication", 2006.

[2] Levent Aksoy, Paulo Flores and Jos´e Monteiro, "A Novel Method for the Approximation of Multiplierless Constant Matrix Vector Multiplication", IEEE 13th International Conference on Embedded and Ubiquitous Computing, 2015.

[3] Dipu P, Naveen S Hamp annavar, P Jayakrishnan, "Implementation of Non-linear pipelined Floating Point Adder", International Conference on Green Computing, Communication and Conservation of Energy (ICGCE), 2013.

[4] Sudip Ghosh, Nachiketa Das,Subhajit Das,Santi P. Maity and Hafizur Rahaman, "Digital Design and Pipelined Architecture for Reversible Watermarking Based on Difference Expansion using FPGA", International Conference on Information Technology, 2014.

[5] DANIEL D. GAJSKI," An Algorithm for Solving Linear Recurrence Systems on Parallel and Pipelined Machines", IEEE TRANSACTIONS ON COMPUTERS, 1981.

[6] KR. Santha and V. Vaidehi, "Parallel-Pipelined Architecture for the Kalman Based Adaptive Equalizer", IEEE – ICSCN, 2007.

[7] Zaher Merhi, Milad Ghantous, Mohammad Elgamel, Magdy Bayoumi, And Ayman El-Desouki, "A Fully Pipelined Parallel Architecture for Kalman Tracking Filter", The International Workshop on Computer Architecture for Machine Perception and Sensing, 2006.

[8] Yun Qu, Viktor K. Prasanna, "High-performance Pipelined Architecture for Tree-based IP lookup Engine on FPGA", IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum, 2013.

[9] M. Garrido and J. Grajal, "Continuous-flow variable-length memoryless linear regression architecture", Electronics letter, 2013.

[10] Georgios Lilis(a), Theodoros Kyriakidis(a), Guillaume Lanz, Rachid Cherkaoui, Maher Kayal, "Pipelined Numerical Integration on Reduced Accuracy Architectures for Power System Transient Simulations", International Conference on Computer Modelling and Simulation, 2014.

[11] Mu-Chu Lee, Wei-Lin Chiang, Chih-Jen Lin, "Fast Matrix-vector Multiplications for Large-scale Logistic Regression on Shared-memory Systems", IEEE International Conference on Data Mining, 2015.

[12] Jongwook Sohn, Earl E. Swartzlander, "A Fused Floating-Point Four-Term Dot Product Unit", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, 2016.

[13] Miao Hu, John Paul Strachan, Zhiyong Li, R. Stanley, Williams, "Dot-Product Engine as Computing Memory to Accelerate Machine Learning Algorithms", ieee Int'l Symposium on Quality Electronic Design, 2016.

[14] Ryan Eberhardt, Mark Hoemmen, "Optimization of Block Sparse Matrix-Vector Multiplication on Shared-Memory Parallel Architectures", IEEE International Parallel and Distributed Processing Symposium Workshops, 2016.

[15] Steena Monteiro, Forrest Iandola, Daniel Wong, "STOMP: Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication", IEEE 28th International Symposium on Computer Architecture and High Performance Computing, 2016.

[16] Jiwu Peng, Zheng Xiao, Cen Chen, Wangdong Yang, "Iterative Sparse Matrix-Vector Multiplication on In Memory Cluster Computing Accelerated by GPUs for Big Data", 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, 2016.

[17] Shmuel Wimer , Israel Koren, "Energy efficient deeply fused dot-product multiplication architecture", IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2016.