Digital Implementation of LU Decomposition Accelerator

M.Tech Project Report II

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology
Integrated Circuits & Systems

by

Aashish Tamrakar

(Roll No. 193079034)

Under the guidance of

Prof. Sachin Patkar

Department of Electrical Engineering
Indian Institute of Technology, Bombay
2022

Dissertation Approval

This dissertation entitled

Digital Implementation of LU Decomposition Accelerator

by

Aashish Tamrakar
(Roll No. : 193079034)

is approved for the degree of
Master of Technology in Integrated Circuit and Systems specialization

_____	_____
(Examiner)	**(Examiner)**
_____	_____
	Prof. Sachin Patkar
	Dept. of Electrical Engineering
(Chairperson)	**(Supervisor)**

Date: June 2, 2022
Place: IIT, Bombay

Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aashish Tamrakar
Roll No.193079034
IIT Bombay

Date: June 2, 2022

Acknowledgement

I would like to express my deep gratitude to Prof. Sachin Patkar for his kind guidance. I also extend thanks to Sunny Mehta, Anurag Choudhury, Yogesh Mahajan, M Abhijna Anand, Mini K. Namboothiripad, and Mandar Datar for their constant help and support.

Aashish Tamrakar
Electrical Engineering
IIT Bombay

Contents

List of Figures

Abstract

Solving the sparse linear system is one of the most critical steps in many scientific applications such as circuit simulation, training of neural networks, power system modeling, and 5G communication. These operations are iterative and majorly consist of sparse form. In such scenarios, it becomes essential to develop a more efficient way to solve the equations using graph algorithms instead of traditional techniques like Gaussian elimination. The project presents a scalable FPGA-based LU solver system geared towards the matrices that arise in circuit simulations. The LU decomposition approach specified in this project has three main parts. The first part does a symbolic analysis of the matrix. The structure of the sparse system remains the same during the entire simulation and hence can be analyzed symbolically only once to generate a directed acyclic graph. The second part takes this directed flow graph and hardware features such as the number of arithmetic units and BRAMs as inputs and generates a static schedule using the priority list-based ASAP strategy for cross-bar type network. The final software toolchain is a hardware implementation of the cross-bar network.The design was Packed over AXI Protocoal and integrated with Microblaze(Xilinx intellectual property Processor) and SHAKTI (open-source initiative SoCs).
The Digital Physical IP was implemented with frequency of 40MHz on SkyWater Open Source PDK on 130nm technology.The design include SRAM created using OpenRAM,open-source SRAM compiler, and the Processing elements were implemented using OpenLane,OpenLane is an automated RTL to GDSII flow based ASIC implementation. The macro integration was done with the help of Cadence SoC Encounter with satifying generic DRCs, placements, antenna effect and timing constraints.
The future aspect is to achieve better timing performance,Scalable Architecture and implement DFT without design errors.

Chapter 1

Introduction

1.1 Motivation

Solving a system of the equation of the form $Ax = B$ is the most critical and time-consuming operation used in many scientific applications such as circuit simulation, training of neural networks, power system modeling, and 5G communication. The nature of these systems is sparse. These sparse matrices are computationally expensive and challenging to parallelize on traditional processing systems.

There is a need for an efficient algorithm for computation effectively. Extensive analysis has been conducted for expediting the sparse matrix process for various computing platforms, including the FPGA-based systems. Applications can leverage the fact that the underlying matrix structure, the locations of the non-zero elements, remains the same for many iterations. The numerical values are different in each iteration; hence, the data manipulation steps also stay the same. Overall performance can be boosted by sharing these manipulation steps.

LU decomposition is easier to compute the inverse of an upper or lower triangular matrix. By decomposing matrix A into a product of two matrices, i.e., lower triangular matrix L and upper triangular matrix U, we can significantly improve the time required to solve the linear system of an equation $Ax = B$.

It can be mathematically that the cost of factorizing matrix A into L and U is $O(N^3)$. Once the factorization is done, the cost of solving $LUx = b$ is $O(N^2)$. So, if we want to do the simulation for K timesteps, the total cost becomes $O(N^3 + kN^2)$.On the other hand, if we solve the linear equation using Gaussian Elimination, the total cost becomes $O(kN^3)$, which is much larger than $O(N^3 + kN^2)$.

Hence this project intially focuses on accelerating LU decomposition of the sparse matrix using FPGA, where parallelism can exploit the parallelism appropriately. This design was integrating with Xilinx Microblaze futhur it was interconnected by SHAKTI SoC, an open-source initiative by the Reconfigurable Intelligent Systems Engineering.

The Digital IP implementation of design of LU decomposition accelerating hardware. The future scope of this poroject may include interconnecting ASIC Design of SHAKTI SoC.

1.2 Organization of Thesis

The central goal of the project is to implement a scalable LU solver of the sparse matrices. The project is divided into following major divisions:

- Scheduler
- FPGA implemetnation of SoC Design
- Digial IP Implementation of Accelerator

The Pre-processing of the matrix contains approximate minimum degree permutation. Pre-processing would decrease the number of nonzero matrix elements (NNZ) of LU factors, reducing the hardware's memory requirement and pre-processing using Matlab amd functions.
The Scheduler C++ tool accepts the pre-processed matrix symbolic analysis and generates a directed acyclic graph. The directed acyclic graph is scheduled using a priority list-based ASAP strategy under the hardware constraints of following :

- Floating-point Multiply–accumulate operation as MAC unit
- Floating-Point operations as DIV unit
- On-chip block memory units (BRAM)

The collection of MAC units and divider units is referred to as Processing Elements (PE). The number of BRAMs and PEs is configurable according to the need and available FPGA resources. The hardware accepts the data and instruction set generated from the scheduler tool to find the values of the factor matrices.

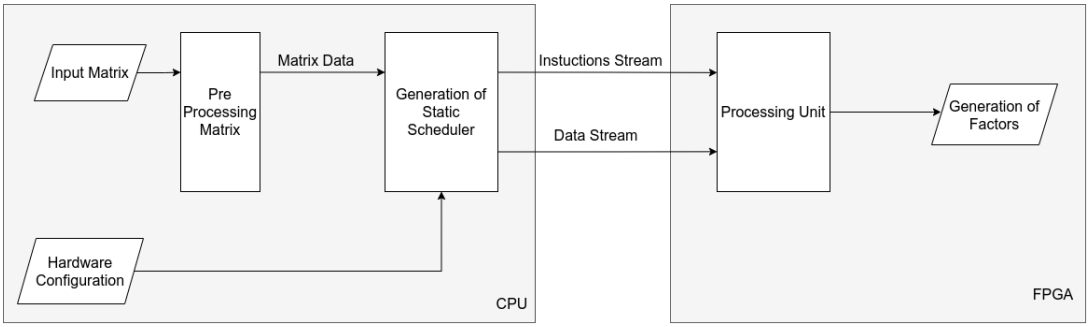


Figure 1.1: The overall flow of FPGA Implementation of IP

The summary of the thesis is given below:

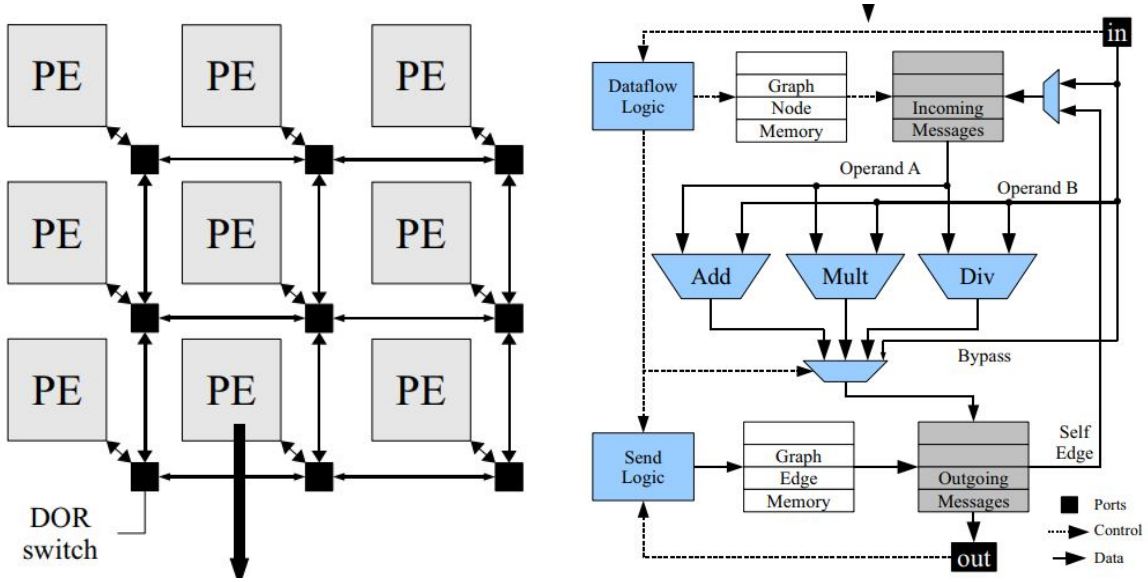
- Preliminaries and literature review
- Scheduler
- FPGA Design Implementation
- ASIC Implementation

Chapter 2

Literature Review

Several researchers have proposed various ideas for implementing an FPGA-based LU solver. These methods vary mainly with the degree of parallelism extracted from the problem and scheduling methods.

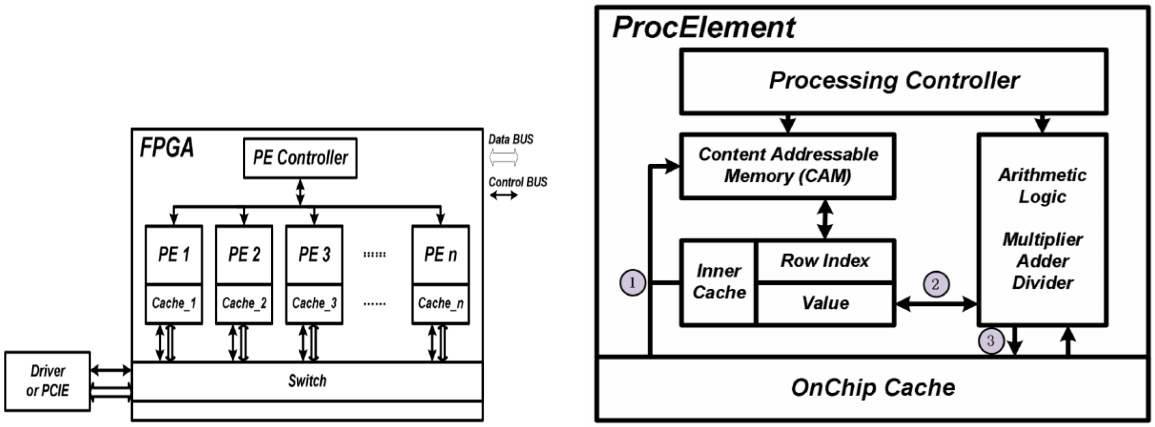
In [1] N. Kapre and A. DeHon proposed a method at the International Conference on Field-Programmable Technology in 2009. He accelerated the LU decomposition of sparse matrices on FPGA using a mesh-grid type architecture in NoC. The significant advantage of his approach is that it's scalable, which is achieved by utilizing the symbolic analysis step of the KLU solver to generate the data flow graph. All the PEs are independent and can make local decisions. This architecture exploits fine-grained parallelism. This particular architecture has obtained a speedup of 1.2-64x using a 250 MHz Xilinx Virtex-5 FPGA compared to the Intel Core i7 965 processor. The mesh architecture and the structure of processing elements is shown in the below figure:-



(a) NOC Configuration (b) Internal Structure of Processing Element

Figure 2.1: NOC based hardware configuration (in [1])

In [2], Wu and Wei have proposed architecture at Reconfigurable Computing: Architectures, Tools, and Applications in 2011 that exploits the coarse-grained parallelism. Each node is connected to an Altera Nios processor attached to a single-precision floating-point unit. Considering coarse-grained parallelism was targeted, the reported speed gain was in the range of 0.5-5.36x, which is not much impressive. Benchmark matrices are smaller and cannot compare results to previous FPGA implementations.



(a) FPGA architecture proposed by Wei (b) Internal Structure of Processing Element

Figure 2.2: Crossbar switch based hardware configuration (in [2])

In [3], T. Nechma and M. Zwolinski presented an approach at IEEE Transactions on Computers in 2015 to leverage medium-grain parallelism for LU decomposition based on the column-based parallelism using the Gilbert-Peierls Algorithm. The architecture prepares execution schedule sing symbolic analysis. It maps the computation of each column to processing consist of MAC unit, divider unit, and BRAM, and it is scheduled using the ASAP method. The architecture used is shown in the figure below: -

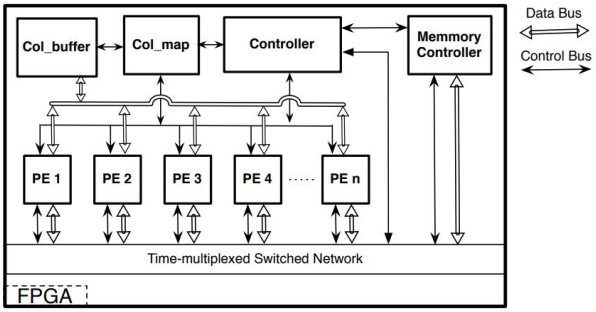


Figure 2.3: Hardware Architecture used by Nechma [3]

The method used in this report is based on the approach similar to the Nechma's [3]. Our scheduler is geared to leverage the fine-grain parallelism using the set of deeply pipelined processing elements and low latency block RAMs available in FPGAs.

Chapter 3

Background

3.1 Sparse Matrix Data Structures

Relatively fewer non-zero elements characterize sparse matrices. These matrices are standard in circuit simulation, power system modeling, computer vision, and 5G Communication. Some data structures can make storage efficient. Hence it is very crucial to select a storage format for using memory efficiently. Following are an example of data formats:-

- Triplet Format:-In this format, we have three arrays consisting in the structure of (A_{ij}, i, j), i.e. (Value, row indices, column indices) of non zero terms on the matrix.
- Compressed Colum Sparse (CCS):-In this format, the three arrays are used with the following logic:-
  - Non-zero records of the same column are listed one after another
  - Row indices of corresponding Non zero records.
  - Column pointer where each column starts
- Compressed Row Sparse (CRS):-This format is similar to CCS. This includes Non-zero records, column indices, and row pointers where each row starts.

$$\begin{bmatrix} 5 & 0 & -5 & 0 & 6 \\ 0 & 4 & 0 & -4 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & -3 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 3 \end{bmatrix}$$

(a) Example Matrix

Values	5	-5	6	4	-4	2	-	-3	-1	-2	3
Column Indices	0	2	4	1	3	0	0	1	3	2	4
Row Indices	0	0	0	1	1	2	3	3	3	4	4

(b) Triplet format

Values	5	2	1	4	-3	-5	-2	-4	-1	6	3
Row Indices	0	2	3	1	3	0	4	1	3	0	4
Column Pointers	0			3		5		7		9	

(c) Compresses Column Sparse

Values	5	-5	6	4	-4	2	-	-3	-1	-2	3
Column Indices	0	2	4	1	3	0	0	1	3	2	4
Row Pointers	0			3		5		6		9	

(d) Compresses Row Sparse

Figure 3.1: Storage formats for sparse matrices

The Gilbert-Peierls' Algorithm uses a columns-based pointing adjacency list, and for solving $Lx = b$ and LU decomposition and hardware should use memory efficiently. The CCS format suits our requirements for creating directed acyclic graphs, symbolic analysis, and scheduling.

3.2 Sparse LU Decomposition

We need to solve equation $A = LU$ where L is the lower triangular matrix and U is the upper triangular matrix. The equation we will get for each element of L and U is given in the following:-

$$U_{(i,j)} = A_{(i,j)} - \sum_{k=1}^{i-1} L_{(i,k)} U_{(k,j)} \tag{3.1a}$$

$$L_{(i,j)} = \frac{A_{(i,j)} - \sum_{k=1}^{j-1} L_{(i,k)} U_{(k,j)}}{U_{(j,j)}} \tag{3.1b}$$

We can use Gauss–Jordan elimination to factorize the matrix, although for sparse nature matrix would not be efficient. Therefore we need to go through Direct methods of sparse LU Decomposition. There are majorly three kinds of Direct Methods solving Left-looking, Right-Looking, and Crout [5]. We will be using Gilbert-Peierls Algorithm: A Column-Oriented LU Factorization with partial pivoting.

3.3 Pre-processing of Matrix

An Approximate Minimum Degree ordering algorithm (AMD) for pre-ordering a symmetric sparse matrix would result in less fill-in of matrix Decomposition. The pre-processing stage is done entirely in MATLAB using built-in function amd of MATLAB. Pre-processing of matrix ensures better performance also provides that there will be no divide by zero error in the computation of Decomposition. AMD Ordering would decrease the floating-point operations, which reduces the compute time. To illustrate the impact following is the matrix before and after AMD ordering on rajat1 Matrix.

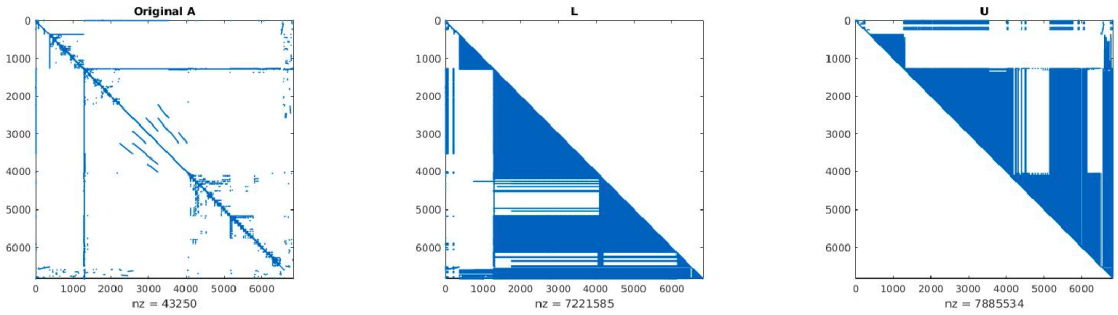


Figure 3.2: Non-zero elements of L and U with original matrix A

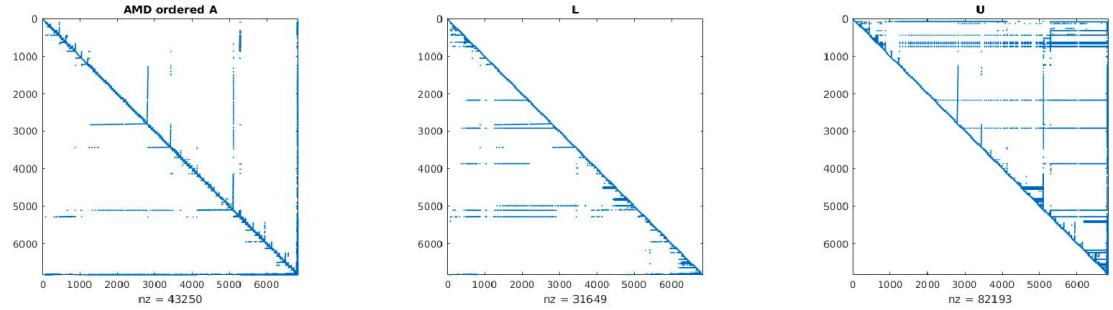


Figure 3.3: Non-zero elements of L and U with AMD ordered matrix permuted A

The given matrix is a sample matrix given by [6]. Circuit simulation matrices from Rajat and Raj.From a company that develops commercial circuit simulation tools. We can solve a linear problem $Ax = b$ by solving the reordered problem where P is the AMD permutation matrix.

$$(P^T AP)(P^T x) = P^T b \tag{3.2}$$

3.4 Gilbert-Peierls' Algorithm

Gilbert's Perierls [7] aimed for LU Decomposition of a matrix with partial pivoting with time proportional to the sum of floating-point operations, i.e., $\mathrm{O}\big(flops(LU)\big)$.The algorithm seems similar to the left-looking algorithm, although partial pivoting gives efficient traversing through the matrix. Following is the algorithm used.

Algorithm 1 Gilbert-Peierls Algorithm: A Column-Oriented LU Factorization

Precondition: A, a $n \times n$ asymmetric matrix

```
1 L := I
2 for j := 1 to n do                    ▷ Compute j^th column of L and U
3     Solve L_j u_j = a_j for u_j
4     b'_j := a'_j − L'j u_j
5     Do Partial Pivoting on b'j
6     u_jj := b_jj
7     l'_j := b'_j / u_jj
```

It computes k^{th} column of L and U using previously computed $(k-1)^{th}$ columns of L matrix. The notation used in the algorithm are as follows: j is the index of the column of L, and U is computed in the A matrix.Furthur A matrix shared to U as a_j and L as a'_j as shown figure after going column 1 to N of A matrix.

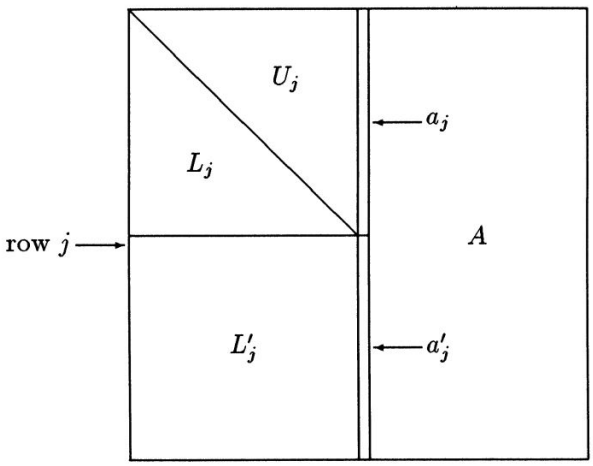


Figure 3.4: Naming conventions used in algorithm

State-of-the-art Gilbert-Peierl's algorithm [7] solve $Lx = b$ with $\mathrm{O}\big(flops(LU)\big).Lx = b$ would iterate this for N columns; therefore, the $\mathrm{O}\big(flops(LU)\big)$ is the efficiency of the algorithm. This method requires locations of all non-zero elements in the sparse column vector x. The creation of a list of non-zero elements is referred in *Symbolic Analysis*.

3.5 ASIC Design Flow

The digital implementation of Desing includes the following steps

1. Design Specification: Design Specification is based on the application provided by the consumer to the designer. This majorly includes the performance requirements under the Budget of area and power.

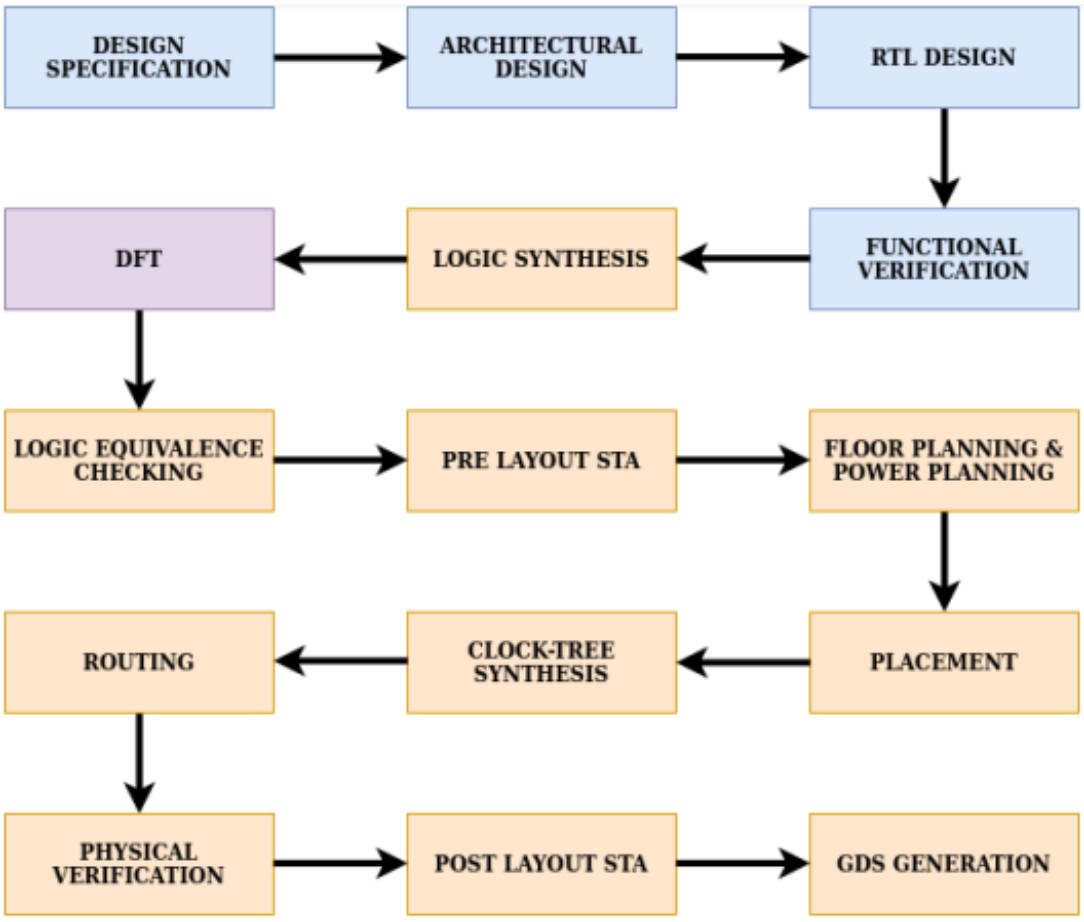


Figure 3.5: ASIC Design Flow[4]

2. Architecture Design: the Architecture Designer create the architectural view of microarchitecture and protocols from the design specification as per the requirement.

3. RTL Design: The RTL designer codes the microarchitecture under the specification using Hih Level Description Langauge. This includes the behavioral and data flow modeling of Design.

4. Functional Verification: The functional Verification is the essential part where the design specifications, i.e., the architecture integrations and protocols, are verified over one head. The RTL designer and verification team work resemblance for Architecture demands as per the Design Specification. The process here would include the functional part, i.e., i.ee. They matched the requirements without considering the area, power, or timing conditions.

5. Logical Synthesis: The logical Synthesis is the process of RTL code to map with the Standard cells to form a gate-level netlist. Foundries provide the standard cells. The performance is decided by the technology nodes used to synthesize the RTL, and the optimum value is synthesis is done as per the timing and power budget of the Design.

6. Design for Test: The testability done after synthesis that would be used after manufacturing the Design. This step includes the generation of test vectors to find the Design's testability.

7. Logical Equivilance Checking: The process of logical equivalence of RTL with the gate level in the stage of pre and post-PnR.The Gate level simulation of generated netlist would delay the process. Therefore, the Logical Equivalence is used as an alternative where reduced boolean and their equivalence is checked.

8. Pre-Layout STA: After Logical equivalence, the Timing analysis is as per the design requirement. This would be the end of Front end Design.

9. Floor planning and Power Planning: The Aim is to define the Aspect ratio and Utilization Area of Design over the chip. This step includes Partitioning of the Design, Macro Placements, and Power planning of the Design.

10. Placements: The Netlist generated by the Synthesis tool is used and placed under the constraints of floor planning. Filler Cells are filled to ensure that all power nets are connected. It does take place in stages

    - Pre Placement Optimization contains the downsizing of cells
    - In Placement: performs cell bypassing, gate duplications
    - Post Placement optimization for Timing violations based on the routings

11. Clock tree Synthesis(CTS): CTS is to minimize the sket and insertion of delay. Clock Tree analyzed the structures under timing and power constraints.

12. Routing: This step is interconnecting the Macros. There are two kinds of routing:

    - Global Routing: The routing is loose routed is generated to estimate the delay for final routing
    - Detail routing: The actual geometry layout of the net is done, including the substantial delay with optimization for timing, DRC, and Antenna effect is done.

13. Physical Verification: The physical verification is done to check the foundry based Rules for proper functioning

    - DRC: Design as per the required rules given by Foundry
    - LVS: Layout versus netlist generated by synthesis tools verification
    - ARC: Design to flag any open-ended track/arc primitive, or open-ended track / arc that is terminated with a via checks

14. RC Extractions and Post Layout STA: extraction is done, and final Post Layout Timing Is verified with interconnects delas

15. GDSI generation: Final binary file representing the planar geometric shapes and other information that foundries would use for tape-out

Chapter 4

Scheduler

The Scheduler is one of the widely used list scheduling techniques for commencing operations under hardware constraints. It takes input from the Matlab script after the permutation of the matrix. The operation of the scheduler tool is divided into the following major phases:

1. Matrix Data and Hardware Constraints
2. Symbolic analysis
3. Scheduling

4.1 Matrix Data and Hardware Constraints

The Scheduler tool accepts the input matrix data in Compressed Column Sparse (CCS) format, i.e., generated from MATLAB Scripts. The following hardware constraints are taken into account, which is as follows:

1. Number of Data BRAM blocks
2. Number of MAC(Fused Multiplier) units
3. Number of DIV(Division) units
4. Latency of all the units
5. Number of Data BRAM Ports
6. Address depth of Data & Instruction BRAMs

The preprocessing process ensures that the on-chip memory is sufficient to operate.

4.2 Symbolic analysis

Symbolic analysis is a process to determine the set of non-zero locations (χ) in solving the lower triangular system $L_j x = b$, where J_j is a unit diagonal lower triangular matrix representing only first $(j-1)$ columns. Listing all the non-zero locations χ gives the numerical computation time proportional to the number floating point operations i.e. $\mathrm{O}(f)$,

Algorithm 2 Gilbert-Peierls Algorithm: Solving Triangular System $L_j x = b$

Precondition: L_j is a lower triangular matrix, x, b are sparse column vectors

```
1 x := b
2 for each j ∈ χ do
3     for each i > j for which l_ij ≠ 0 do
4         x_i := x_i − l_ij x_j
```

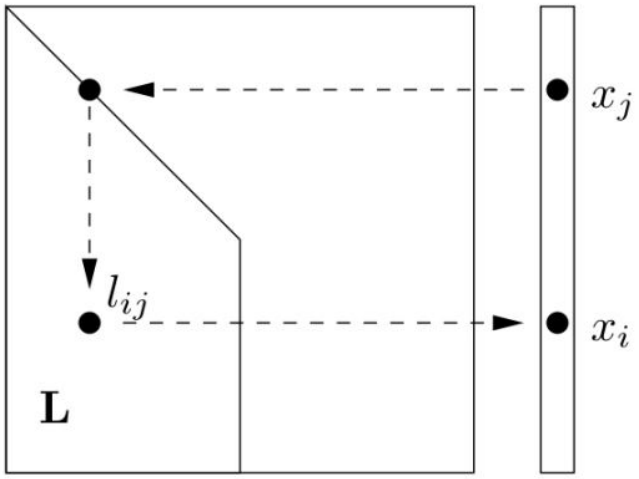


Figure 4.1: Non-zero pattern in LU decomposition

The algorithm suggests that the element of the result vector (x_{ij}) can become non-zero only of the corresponding element in the vector b (b_i) is non-zero or there exits a non-zero element $l_{i,j}$ where j is less than i and x_j is non zero.

$$(b_i \neq 0) \implies (x_i \neq 0) \tag{4.1a}$$

$$(x_j \neq 0) and \exists i (l_{ij} \neq 0) \implies (x_i \neq 0) \tag{4.1b}$$

Symbolic Analysis can visualize these two implications using the figure 4.1.In the column factorization algorithms like Gilbert-Peierls, we know the locations of all the non-zero

elements for the columns with indices lower than j we can determine the non-zero sites (χ) before solving for that column and the entire lower triangular system.
The State-of-the-art Gilbert's Transform requires the locations of non-zero elements. Therefore, this includes the need for symbolic analysis.

4.3 Priority Ordering

Proper priority assignment is beneficial for optimal schedule generation. For resolving contention, scheduling should be assigned the highest priority to the specific nodes. Priority assignment starts from root nodes, and we gradually move towards the leaf nodes. The memory read should be given more priority to the floating-point operations. The priority is defined as:

$$Priority(n) = \sum_{i \in Parents(n)} Priority(i) + \sum_{x \in tasks(n)} Delay(x)$$

where $Parents(n)$ is the set of node which are dependent on the node n and $tasks(n)$ is the set of operations to evaluate the node n.This definition nodes will prioritized by the greedy scheduling algorithm.
The priority calculation formula for node of type "/"(DIV) is given below: -

$$''/''nodepriority = \sum_{i \in Parents(n)} Priority(i) + latency_{DIV}$$

The priority calculation formula for node of type "mac_sub"(MAC) is given below: -

$$''mac\_sub''nodepriority = \sum_{i \in Parents(n)} Priority(i) + \#MAC\_operations \times latency_{MAC}$$

4.4 Priority List Based Scheduling

The basic idea of list scheduling is to make an ordered list of processes by assigning them priorities and then repeatedly execute the following steps until a valid schedule is obtained :

- Select from the list the process with the highest priority for scheduling.
- Select a resource to accommodate this process.
- If no resource can be found, we select the following process in the list.

The important steps in the algorithm are as follows:

Algorithm 3 Priority List based Scheduling Process

Precondition: G, a computation flow graph for LU decomposition

```
1  scheduledNodes := 0
2  readyNodes := G.leaves()
3  while scheduledNodes < G.size() do
4      Update status of nodes retired in previous cycle
5      Update scheduledNodes
6      Update the ready nodes list
7      Assign memory port to retiring nodes
8      Calculate the free BRAM ports for reading and writing
9      Select the most prior set of ready nodes which can be schedules in current cycle
10     Assign the memory memory operations
```

The scheduling algorithm uses three tables, namely the assignment table, retirement table, and memory operation table, to track all assigned operations and determine the next set of assignments. The scheduler cycle-accurate simulation and stores the assignment as a set of instructions for hardware.

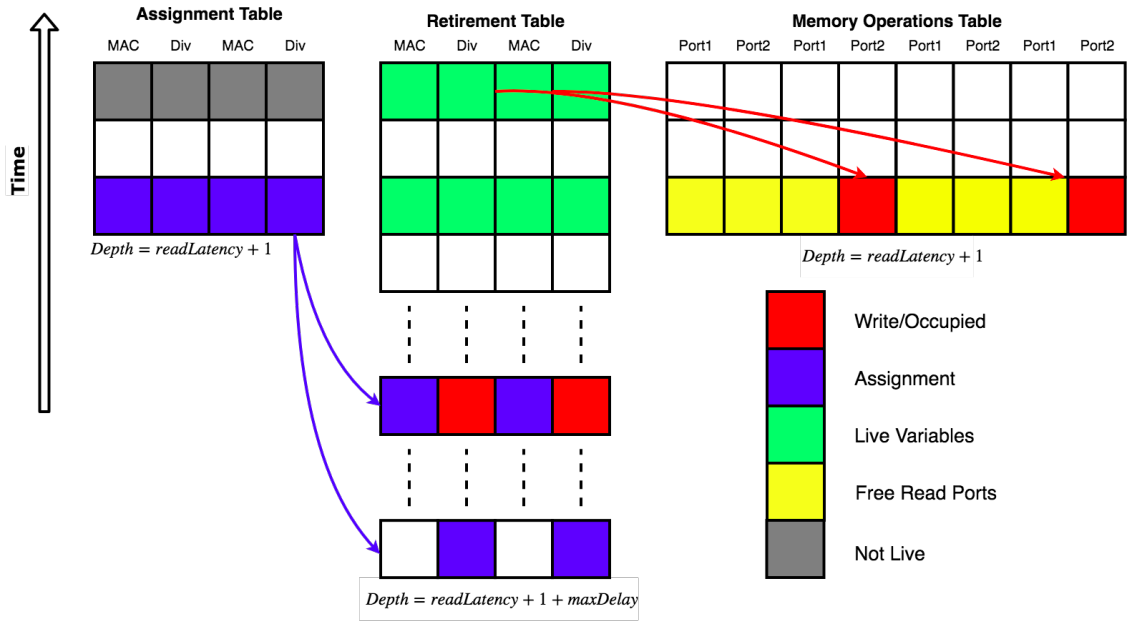


Figure 4.2: Allocation tables used by scheduling algorithm

Assignment Table:
All the operations scheduled in a cycle must store in the same cycle they return because of the unavailability of the output buffer—the corresponding retirement table used for removing duplication of entries in the cell. The size of the Assignment Table is equal to the number of MAC & DIV units with reading latency. The read latency can be two as it will be virtual if Quad-port Data BRAM is used; else, it would be one.

Retirement Table:
The operations are retiring in the current cycle, i.e., operations whose results are available on the output ports of PEs. These results must be written to the memory except in the case of write cancellation.

Memory Operation Table:
Memory Operation Table serves to maintain a record of available ports and assigned operations. Operation is assigned to particular Data BRAM from the time-multiplexed ports.

Selecting the set of assignments: All the schedulable nodes are stored in a priority list in decreasing order. All combinations of nodes are checked availability of BRAM ports for both reading and retirement. A valid combination with the highest sum of priorities is selected as an assignment for the current cycle.

The required number of readings and writes corresponding to each node for each Data RAM will be listed in the adjoining tables. A good group of operations must have the sum of ports required, but the selected node should be less than the total available nodes. The set with a maximum sum of priority would be assigned to Processing elements in the current cycle.The additional required data values are set to be read from corresponding memory locations read allocation table.

The Scheduling process will be recursively working till the dummy node of the sequential graph is not reached. The final allocations of resources and processes are recorded for every timestamp, and instructions are generated. These instructions are in the form of bits that will be pass through instruction BRAM to the crossbar switch for executions.

Chapter 5

FPGA Implementation

The General hardware design is explained using following the hierarchy graph.

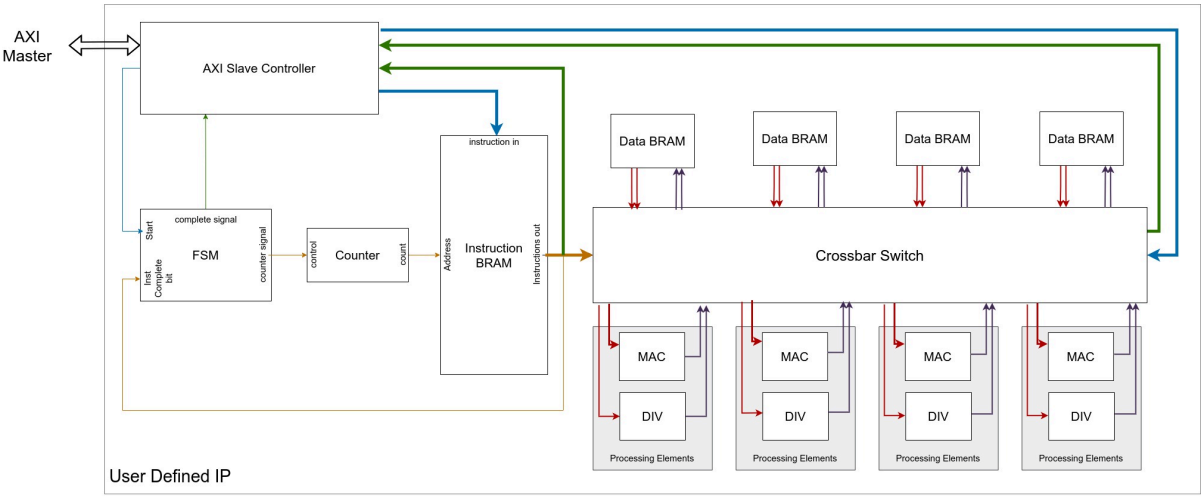


Figure 5.1: FPGA Implementation architecture

The design uses Xilinx optimized IP, and a Crossbar switch box is used to interconnect the overall design as per the requirement for the accelerator. Following are the IPs used:-

- Block Memory Generator (BRAM Unit) [8]
  - Instruction BRAM (custom bit size)
  - Data BRAM (float / double type)
- Floating-point [9]
  - Fused Multiply-Add/Sub (MAC Unit)
  - Division (DIV Unit)

5.1 Block Memory Generator (BRAM Unit) IP

The Xilinx LogiCORE IP Block Memory Generator replaces the Dual Port Block Memory and Single Port Block Memory LogiCOREs but is not a direct drop-in replacement. It should be used in all new Xilinx designs. The core supports RAM and ROM functions over a wide range of widths and depths. Use this core to generate block memories with symmetric or asymmetric read and write port widths and centers that can perform simultaneous write operations to separate locations and simultaneous read operations from the exact location. For more information on differences in interface and feature support between this core and the Dual Port Block Memory and Single Port Block Memory LogiCOREs, please consult the datasheet.

The BRAM Units are used for instructions produced by the scheduler and storing A and LU Matrix values. The instructions BRAM used to select change the connection of crossbar switch in the design, which would lead to floating-point operations as per the schedule for Decomposition. The instructions BRAM is custom input/output bit as per the scheduler, and Data BRAM should be 32bit / 64bit as per the data type.

5.2 Floating-point IP

The Xilinx Floating-Point Operator is capable of being configured to provide a range of floating-point operations. The core offers addition, subtraction, accumulation, multiplication, fused multiply-add, division, reciprocal, square-root, reciprocal-square-root, absolute value, logarithm, exponential, compare, and conversion operations. High-speed, single-cycle throughput is provided at a wide range of word lengths, including half, single and double precision. DSP48 slices can be used with specific operations.

The floating-point IP is used to Multiply and Accumulate Unit and Divider Unit.

The LU factorization requires only a floating-point multiplier and subtract operation ($Result = C - AB$).The Xilinx's MAC units utilize on-chip DSP Slices to achieve higher performance.

The Xilinx Floating Point Divider IP utilizes some radix-2 SRT division algorithm variants and can not use DSP slices. These units are bulkier and therefore have to be deeply pipelined to operate. The speed grade of the target FPGA is one of the most dominating factors in determining the depth of the pipeline.

5.3 Crossbar switch box

The Crossbar switch box should connect any output port to an input port for Data BRAMs, MAC, and DIV units. This can be achieved with a multiplexer. The select signals are provided by instructions BRAMs, which were produced by the scheduler. The number of multiplexers depends on Processing elements, the Number of Data BRAMs, and the number of ports at each BRAM.

The top-level design is developed into IP via AXI slave wrapper to connect soft-core and/or hard-core processors. The Crossbar switch box should instruct BRAM and Data BRAM to upload data/instruction and receive Matrix Decomposed Matrix.

5.4 Shakti Board Integration (PARSHU)

SHAKTI is an open-source initiative by the Reconfigurable Intelligent Systems Engineering (RISE) group at IIT-Madras. The SHAKTI initiative aims to build open-source production grade processors, complete System on Chips (SoCs), development boards, and a SHAKTI-based software platform. SHAKTI support on different development boards is crucial as this expands the hardware choice of FPGAs.The core has been completely developed using BSV (Bluespec System Verilog). As part of this effort, initially, two varieties of FPGA boards are being supported. They are Xilinx's Arty7 35T and Arty7 100T.
In This Design, the targeted target is PARASHU[10] with the following specification.

- PARASHU is an SoC based on SHAKTI E-class [11]
- PARASHU is supported on Artix 7 100T board
- It has an abridged version of the 32-bit E-class
- Targeted frequency is 200 MHz
- Storage 4 KB of ROM and 256 MB of DDR

This is the embedded class processor, built around a 3-stage in-order core. It is aimed at low-power and low compute applications and can run basic RTOSs like FreeRTOS and Zephyr (Chronos is also being ported and will be released soon). Typical market segments include: smart-cards, IoT sensors, motor controls, and robotic platforms.Based on the Following Map, it is configured.
The Integration of the accelerator is wrapped around AXI Slave protocol. Using the User manual[12] . The Basic structure of Shakti FPGA Project has the following directory structure:-

S.No.	Peripeheral	Base Address Start	Base address End
1	Memory DDR	0x8000_0000	0x8FFF_FFFF
2	Debug	0x0000_0010	0x0000_001F
3	UART0	0x0001_1300	0x0001_1340
4	UART1	0x0001_1400	0x0001_1440
5	UART2	0x0001_1500	0x0001_1540
6	I2C0	0x0004_0000	0x0004_00FF
7	GPIO	0x0004_0100	0x0004_01FF
8	CLINT	0x0200_0000	0x020B_FFFF
9	PLIC	0x0C00_0000	0x0C01_001F
10	PWM0	0x0030_0000	0x0030_00FF
11	PWM1	0x0030_0100	0x0030_01FF
12	PWM2	0x0030_0200	0x0030_02FF
13	PWM3	0x0030_0300	0x0030_03FF
14	PWM4	0x0030_0400	0x0030_04FF
15	PWM5	0x0030_0500	0x0030_05FF
16	SPI0	0x0002_0000	0x0002_00FF
17	SPI1	0x0002_0100	0x0002_01FF
18	I2C1	0x0004_1400	0x0004_14FF
19	XADC	0x0004_1000	0x0004_13FF
20	PinMux	0x0004_1500	0x0004_15FF
21	Bot Rom	0x0000_1000	0x0004_15FF
22	Custom Module	0x0005_0000	0x0005_FFFF

```
Shakti_project
├─ bootcode : Boot related codes
├─ bsv_build : BSV related object files for Integration
├─ common_bsv : Common BSV files for integration
├─ common_verilog : Verilog File generated using BSV
├─ devices : Devices that would be connected like BRAM PWM
├─ e-class : E-class core of the SHAKTI Processor family
├─ fabrics : interconnects based BSV files
├─ fpga_project : Final Project and User-defined IPs
├─ tcl : Automated scripts for fpga_project
├─ verilog : Final Build Verilog files and Cutsom Verilog Files
├─ Soc.bsv : Top Module Definition
├─ Soc.defines : Memory Maps of Design
└─ spi_cluster and uart_cluster : BSV files for SPI and UART Integration
```

The alteration is done for the integration UserDefined IP to the SHAKTI Processor by specifying the interconnecting cluster and protocol.

5.5 Shakti SDK

It is the open-source software development platform for SHAKTI. Clean separation between drivers, boot, core, and application layers.Driver support SPI, QSPI, PLIC, CLINT, UART, I2C, and PWM. Multiple sensors are connected and proven with SHAKTI-SDK. Standalone and Debug mode supported. Multilevel logging, Flash programming & Dynamic memory management are supported. A single place for bare-metal application development, projects, and benchmarks.

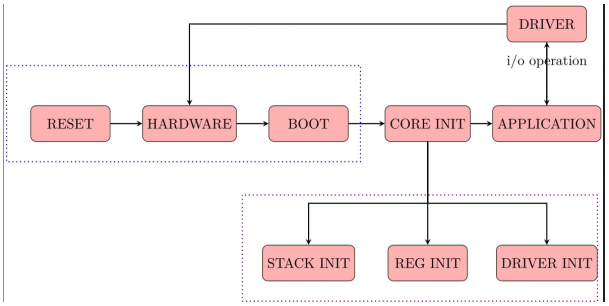


Figure 5.2: Software program flow

The SHAKTI-SDK is a C/C++ platform for developing applications over SHAKTI. The SDK has the necessary firmware code and framework to develop newer applications on the hardware. The framework is lightweight and customizable.

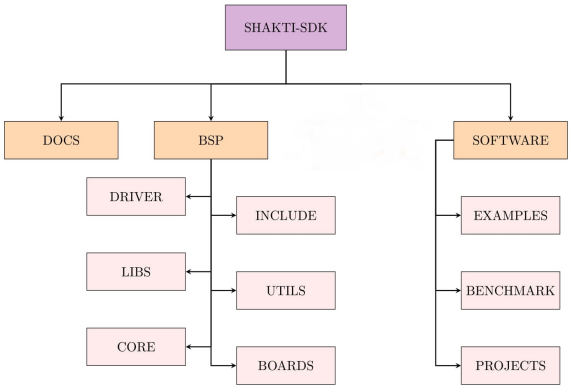


Figure 5.3: SDK architecture

Board Support Package

The BSP consists of system files and driver files for various devices. It contains specific platform-dependent definitions for each board. Essentially, the BSP is the layer above the hardware. It includes the following.

- Drivers: The drivers are a set of software constructs that help software applications access the SoC devices. They are generally low-level APIs that execute a particular task in the hardware.
- Include: The board independtn varabole/macro definitios adn declarations.
- Libs: The library utilities and the boot code.
- Core: The core usually has functions related to the startup codes, trap handlers and interrupt vectors. The code is related to memory initialization.
- Utils: This contains the code related to standalone mode features of the Software

Software
The software directory provides a platform for developing various applications independent of the underlying BSP. All the applications/projects developed in SHAKTI-SDK reside in this directory.
Shaki tools
SHAKTI uses the RISC-V toolchain. A software toolchain to create assembly instructions & sequences for execution in both a simulator and target FPGA.

Chapter 6

ASIC Implementation

The VLSI design flow can be divided into two parts

- Frontend design flow
  - RTL Design/Coding
  - Synthesis
  - Fuctional Verificaion
  - DFT
- Backend design flow.
  - Floor Planning
  - Placement
  - Clock Tree Synthesis
  - Routing
  - Static timing analysis
  - Physical Verification

Both together allow the creation of a functional chip from scratch to production.
There is a need to analyze the accelerator for optimum use of Hardware. The Optimum value would be 8 SRAM and 4 Processing Elements as per the graph.
Following is the target specification:-

- Number of Data RAM:- 8
- Size of Each Data SRAM:- 8 kbits (width 32-bit depth 256)
- Size of Instruction SRAM required:-410 kbits (width 205-bit depth 2048)
- Number of Processing Elements:- 4 (4 MAC and 4 DIV units)
- Targetted Frequency:- 40 MHz

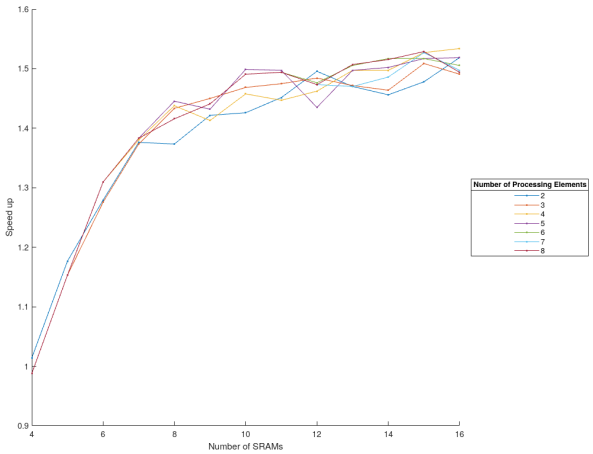


Figure 6.1: Speed up of Design vs Number of SRAM with varying Processing Element

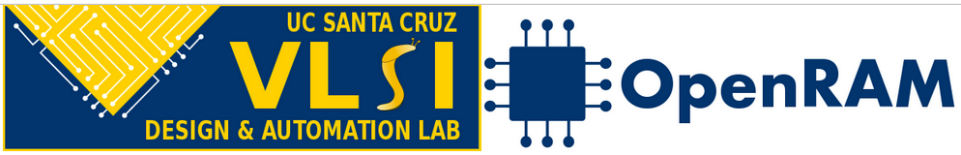
OpenRAM[13] will create the SRAM, and OpenLane[14] RTL to GDS converter will process the design elements. Final Integration will be using Cadence SoC Encounter. The design will be implemented using The Skywater 130nm PDK. This PDK doesn't have support for Propriorty software, although a GitLab configure file helped for Integration[15].

6.1 OpenRAM

OpenRAM[13] is an open-source Python framework for creating the layout, netlists, timing, and power models. It is a Memory Compiler and is used for creating SRAMas macro. This SRAM is used to store Instructions and Matrix Data. Some Common configure files that need to Provide like Supply voltages, Temperature, Process corner, and Number of Ports. The Output Files includes followings:-

- GDS (.gds)
- SPICE (.sp)
- Verilog (.v)
- PnR Abstract (.lef)
- Liberty (multiple corners .lib)
- Datasheet (.html)
- Log (.log)
- Configuration (.py) for replication of creation

Example of Datasheet Given in Below :-



DATA_32_256_sky130A.html

Compiled at: 2022-05-02

DRC errors: skipped

LVS errors: skipped

Git commit id: 47690e0076ffd1adb4678e03609e152e27baabce

Ports and Configuration

Type	Value
WORD_SIZE	32
NUM_WORDS	256
NUM_BANKS	1
NUM_RW_PORTS	1
NUM_R_PORTS	0
NUM_W_PORTS	0
Area (μm^2)	303413

Operating Conditions

Parameter	Min	Typ	Max	Units
Power supply (VDD) range	1.8	1.8	1.8	Volts
Operating Temperature	25	25	25	Celsius
Operating Frequency (F)			122	MHz

Figure 6.2: OpenRAM Condition

Operating Conditions

Parameter	Min	Typ	Max	Units
Power supply (VDD) range	1.8	1.8	1.8	Volts
Operating Temperature	25	25	25	Celsius
Operating Frequency (F)			122	MHz

Timing Data

Using analytical model: results may not be precise

Parameter	Min	Max	Units
din0[31:0] setup rising	0.009	0.009	ns
din0[31:0] setup falling	0.009	0.009	ns
din0[31:0] hold rising	0.001	0.001	ns
din0[31:0] hold falling	0.001	0.001	ns
dout0[31:0] cell rise	1.807	2.432	ns
dout0[31:0] cell fall	2.008	2.211	ns
dout0[31:0] rise transition	0.007	0.027	ns
dout0[31:0] fall transition	0.007	0.027	ns
csb0 setup rising	0.009	0.009	ns
csb0 setup falling	0.009	0.009	ns
csb0 hold rising	0.001	0.001	ns
csb0 hold falling	0.001	0.001	ns
addr0[7:0] setup rising	0.009	0.009	ns
addr0[7:0] setup falling	0.009	0.009	ns
addr0[7:0] hold rising	0.001	0.001	ns
addr0[7:0] hold falling	0.001	0.001	ns
web0 setup rising	0.009	0.009	ns
web0 setup falling	0.009	0.009	ns
web0 hold rising	0.001	0.001	ns
web0 hold falling	0.001	0.001	ns

(a) OpenRAM Timing details

Power Data

Pins	Mode	Power	Units
!csb0 & clk0 & !web0	Read Rising	331123	mW
!csb0 & clk0 & !web0	Read Falling	331123	mW
!csb0 & !clk0 & web0	Write Rising	331123	mW
!csb0 & !clk0 & web0	Write Falling	331123	mW
csb0	leakage	0.008606	mW

Characterization Corners

Transistor Type	Power Supply	Temperature	Corner Name
TT	1.8	25	_TT_1p8V_25C.lib
SS	1.8	25	_SS_1p8V_25C.lib
FF	1.8	25	_FF_1p8V_25C.lib

Deliverables

Type	Description	Link
.gds	GDSII layout views	DATA_32_256_sky130A.gds
.html	This datasheet	DATA_32_256_sky130A.html
.lef	LEF files	DATA_32_256_sky130A.lef
.lib	Synthesis models	DATA_32_256_sky130A_TT_1p8V_25C.lib
.lib	Synthesis models	DATA_32_256_sky130A_SS_1p8V_25C.lib
.lib	Synthesis models	DATA_32_256_sky130A_FF_1p8V_25C.lib
.log	OpenRAM compile log	DATA_32_256_sky130A.log
.py	OpenRAM configuration file	DATA_32_256_sky130A.py
.sp	SPICE netlists	DATA_32_256_sky130A.sp
.v	Verilog simulation models	DATA_32_256_sky130A.v

(b) OpenRAM Characteristic and Power details

Figure 6.3: OpernRAM Analysis report

```
techname
├── __init__.py : Sets up PDK environment
├── tech : Contains technology configuration
│   ├── __init__.py : Loads all modules
│   └── tech.py : SPICE DRC GDS and layer config
├── gds_lib : Contains .gds files for each lib cell
├── sp_lib : Contains .sp file for each lib cell
├── models : Contains SPICE device corner models
├── tf : May contain some PDK material
└── mag_lib : May contain other layout formats
```

These files should have

- Bitcell (and dummy and replica bitcell)
- Sense amplifier
- DFF (from a standard cell library)
- write driver

VSD Corp. Pvt. Ltd [16] had to share the Technology files of SKYWater 130nm PDK to create OpenRAM. The Macro provided is used for the creation of Physical implementation of the design. The files have an outdated version of Skywater PDK rules for DRC and LVS. Therefore the OpenRAM could produce a design with DRC faults.

6.2 OpenLane

OpenLane[14] is an automated RTL to GDSII flow based. The components include open-source tools, which include followings:-

- YosysHQ
- OpenROAD
- Open Circuit Design

The MAC unit has a 9(multiplication) latency +7(Addition)=16 clocks. It is cascading of simple floating-point multiplication and addition units of the design. The Division module has a latency of 36 clocks. It includes the restoring division method.

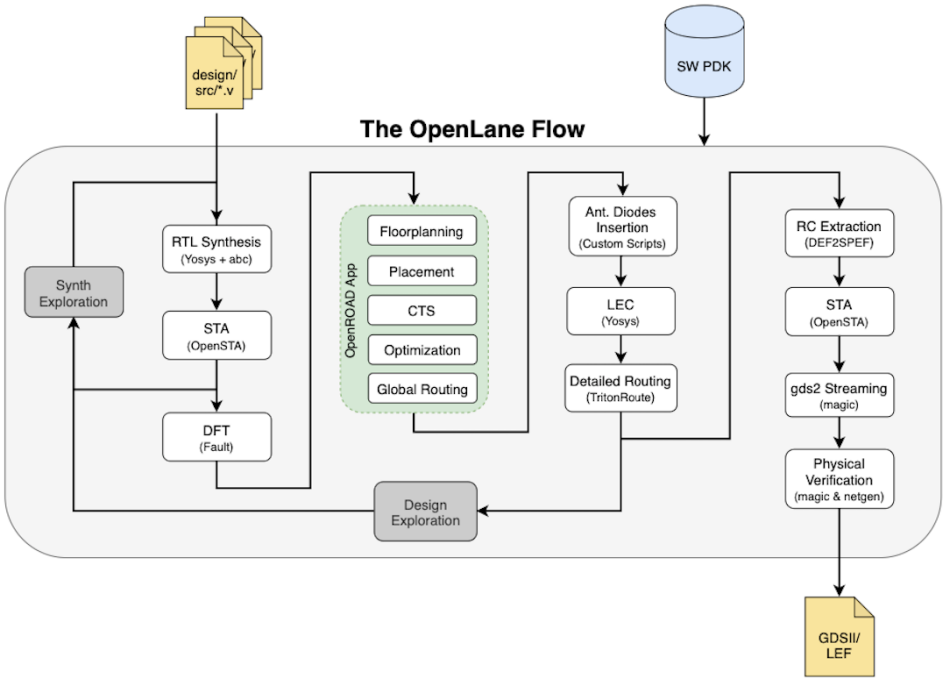


Figure 6.4: OpenLane Flow

(a) Fused Multiplication Add Unit

(b) Division Unit

Figure 6.5: Processing Element Synthesis Analysis Report

The design was converted from RTL to GDS using OpenLane.Processing elements, i.e., pipelined version of Fused Multuiplicnation Unit and DIV unit, are created using the OpenLane.based on Synth Exploration feature. Design Synthesis Exploration performed the design more importance given to Performance over the area.
OpenLane indeed is a good package for automated RTL to GDS. Although it had the following issues:-

- It was not able to give process antenna clean for every design.
- Hardening macro is not robust enough, specifically with the OpenRAM module.

For the final hardening of the design, there is a need to use Cadence SoC Encounter.

6.3 Fount-end design flow

The frontend flow is responsible for determining a solution for a given problem or opportunity and transforming it into an RTL circuit description. The process of ASIC Desing with RTL generation. We have used Cadence RC-Compiler for synthesis. We have used Skywater 130nm PDK.

6.3.1 RTL Design/Coding

The RTL code must be generated for the design, although our method includes SRAM and Processing Element Macros. Our method would be a down-to-up approach. We need The Memory elements to be generated using OpenRAM. To consider a black box and give database files is provided as output from OpenRAM.We would use Harden Macro generated by OpenLane of the Processing Elements, i.e., Floating point, Pipelined version of Fused Multiplication, and Add unit and Division unit. The rest of the interconnects are coded for the simulation and synthesis process. The interconnecting logic, i.e., specialized crossbars coding, is done for interconnecting.

6.3.2 Synthesis

Synthesis is responsible for converting the RTL description into a structural gate-level-based netlist. This netlist instantiates every element (standard cells and macros) that compose the circuit and its connections.

$$Synthesis = Translation + Optimization + Mapping$$

Cadence RTL Compiler requires the following inputs:-

- RTL Description
- The standard cell and Macro library
- The defined constraints: synthesis goals regarding timing, area, capacitance, max transition, and fanout. Delivered by the Frontend team
- Design Environment: The operating conditions (Libraries corners), wire models.

6.3.3 Fuctional Verification

Functional verification was done in the FPGA board implementation. This functional verification is an essential step for adequately verifying the RTL Codes. This will be the end of the Front-end of Design if no Design-for testing module is considered.

6.3.4 DFT

Testing of every manufactured chip is an essential aspect. We need to add the methodology in the chip to improve this time which is called design for testability. For larger Designs, every net is not controllable or observable; hence we need to add circuitry to increase the testability of the circuit. we need to add the scan chains to improve the testability

To add the scan flip-flops in the design, we have used Genus synthesis to Scan check the Processing elements and overall design. After successfully adding the scan chains, check the dft_drc error; if there are no violations, we have successfully added the scan chains. The Scan Chain flip-flop can verify the details of the scan chains added through various reports.

The testing requires finding out the test vectors which can test the chip after its manufacturing. There are so many tools that can analyze the design and output the test vectors to be used for testing on the manufactured chip. Cadence Modus exports the ATPG pattern for the design, which could serve as the test vectors for the procedure.

The design is not optimized for testability because of less register and macro-based design. The overall controllability and observability are very low. Therefore design did not include scan chains for testability of the circuit.

6.4 Backend design flow

The backend process is responsible for the physical implementation of a circuit. It transforms the RTL circuit description into a physical design composed of gates and interconnections. The main phases of the backend process are Synthesis and Place & Route.

6.4.1 Floor Planning

The initial phase of the Backend starts with Floorplan. The physical shape, i.e., Aspect ratio, Utilisation Area, and row structure for placement of a standard cell, is defined. It also has information like boundaries, the I/O pin location, and blockages in the Design. In the Bottom-up approach, We need to define the area of Macros in the Design and the level of the blockage, i.e., Metal restrictions over the macro. This step includes the Power planning of Design.

6.4.2 Placements

In the Placements and Routeing stage, convert the gate-level netlist produced during synthesis into a physical design. Placement involves placing all macros and cells into a specific and predefined space. It is done in three phases.

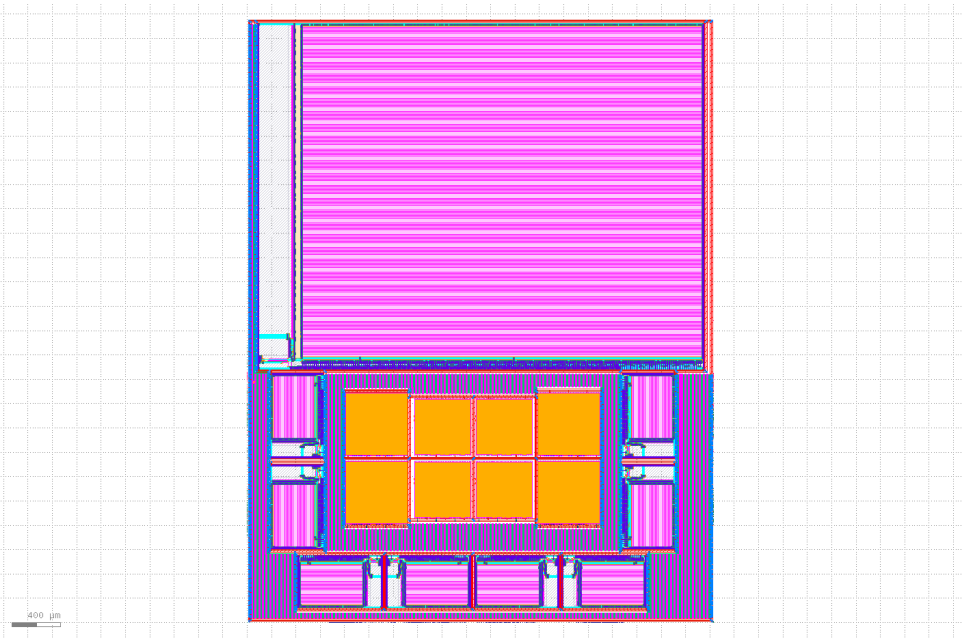


Figure 6.6: Floorplan of Design

- Pre-placement optimizations are performed before placements and contain downsizing of the cells. It places the standard cells to optimize timing and congestion but not take into account overlapping prevention
- In placement optimization, perform cell bypassing, gate duplication, and buffer insertion. It includes legalizing placement and eliminating overlap problems in the closest available space.
- Post-placement optimization fixes set up and hold violations and are based on global routing. The clock consideration till the stage was ideal, and hence with clock skew, a post-placement optimization after CTS optimization is required.

6.4.3 Clock Tree Synthesis

Clock tree synthesis creates a balanced buffer tree in all high fanout clock nets to avoid violations regarding clock skew, max transition time, capacitance, and setup and hold times. This process includes 3 phases.

- Pre-CTS: The first step in the physical design is floor-planning. However, it can be manual by placing blocks. Accordingly, it's good to let the tool decide on a floor plan for you, and later, based on the analysis, Pre-CTS can instruct a proper floor plan. All the wire load models are removed before the placement, and absolute RC values from Virtual Route are considered to calculate timings. It performs different optimization at different stages. The downsizing of the cells is accomplished in Pre-placement optimizations. While at the time of placement optimization, cell

bypassing, gate duplication, and buffer insertion. Post-placement optimization aims to fix setup and hold violations after the global routing. The clock is considered ideal till the stage.

- CTS: The analysis used an ideal clock with zero skews. The software uses the FE-CTS commands to complete clock tree synthesis. In this mode, if present, the FE-CTS specification file is translated into a CCOpt clock tree specification Tcl script, and the clock tree synthesis is performed using the CCOpt engine.
- Post-CTS: After successfully implementing clock tree synthesis in the design, we need to verify the design rule violations and the hold time. The command optDesign post-CTS will fix the error. The detailed routing for all the nets is performed.

6.4.4 Routing

Routing is about interconnecting the pins as per the netlist. This takes place in 2 phases.

- Global Routing: A loose route is generated for each net with estimated values in this type of routing. Global routing is further divided into Line Routing and Maze Routing.
- Detailed Routing: In detailed routing, the actual geometry layout of each net is calculated, i.e., substantial delays of wire are calculated. Several optimizations like Timing optimization can obtain the exact delays, CTS, etc.

6.4.5 Static timing analysis

Static timing analysis is an integral part of Placement, Clock Synthesis, and Routing. A wire load model cannot guarantee accurate analysis and timing constraints. The difference between the accuracy of the analysis by both the tools can be examined when we use the parasitic extracted after the Place and Route. The buffers are used to improve the analysis and avoid violations.

6.4.6 Physical Verification

Physical verification ensures a design's layout works as intended and does not violate the design constraints provided by the foundries. The increase in complex design with shrinking process geometries needs highly productive physical verification.

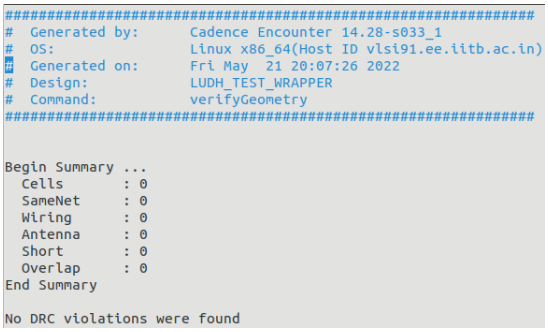
- Antenna Rule Check is includes the maximum tolerance for the ratio of a metal line area to the area of connected gates.

```
#####################################################################
#  Generated by:      Cadence Encounter 14.28-s033_1
#  OS:                Linux x86_64(Host ID vlsi91.ee.iitb.ac.in)
#  Generated on:      Fri May  20 18:31:38 2022
#  Design:            LUDH_TEST_WRAPPER
#  Command:           timeDesign -postRoute -outDir ./reports/5.postRouteOpt_LUDH_TEST_WRAPPER
#####################################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.524  |  0.524  |  1.359  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|  1064   |   385   |  1053   |
+--------------------+---------+---------+---------+

+----------------+--------------------------------+------------------+
|                |              Real              |       Total      |
|    DRVs        +------------------+-------------+------------------|
|                |  Nr nets(terms)  | Worst Vio   |  Nr nets(terms)  |
+----------------+------------------+-------------+------------------+
|   max_cap      |    120 (463)     |   -0.385    |    121 (465)     |
|   max_tran     |    138 (515)     |   -1.305    |    139 (516)     |
|   max_fanout   |      1 (1)       |     -1      |      1 (1)       |
|   max_length   |      0 (0)       |     0       |      0 (0)       |
+----------------+------------------+-------------+------------------+

Density: 100.000%
Total number of glitch violations: 0
------------------------------------------------------------
```

Figure 6.7: Post Route QoR report

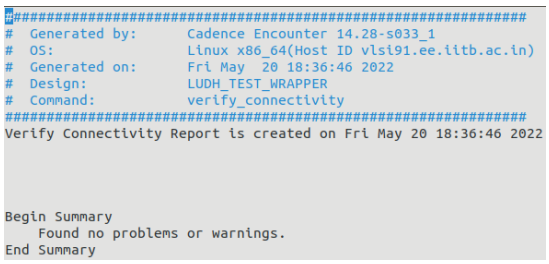
- Connectivity Checks is the Automated routing tools fail to check using connecting checks. This is a generic check.
- Design rule checking (DRC) verifies if a chip layout satisfies the rules defined by the manufacturer. Each semiconductor process has its own set of rules, and we need to ensure sufficient margins so that normal variability in the process will not fail the chip.Geometry check is more like overlapping of cells, Same net, loop formation, etc. checks are done using geometry Check
- Layout vs. Synthesis (LVS) verifies if the generated structure is functionally the same as the schematic/netlist of the design we synthesized. It checks if we have transferred the geometry into the layout correctly. There should not be any missing nets or ports, or there should not be any shorts. In the case of semi-custom design, we do not have any schematic; hence we need to create a spice netlist to verify it with our generated layout.

The proper sign-off tools, Cadence ASSURA or Mentor Graphics Calibre, files are not provided by Skywater 130 nm Foundry. Generic Tests are successfully passed in Cadene Soc Encounter.



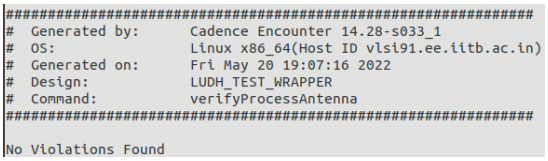
```
#################################################################
#  Generated by:      Cadence Encounter 14.28-s033_1
#  OS:                Linux x86_64(Host ID vlsi91.ee.iitb.ac.in)
#  Generated on:      Fri May  21 20:07:26 2022
#  Design:            LUDH_TEST_WRAPPER
#  Command:           verifyGeometry
#################################################################


Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

No DRC violations were found
```

(a) DRC Report



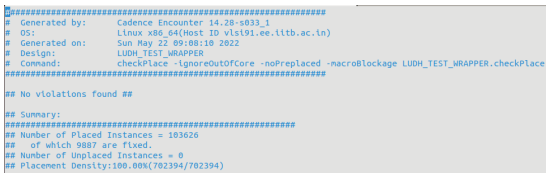
```
#################################################################
#  Generated by:      Cadence Encounter 14.28-s033_1
#  OS:                Linux x86_64(Host ID vlsi91.ee.iitb.ac.in)
#  Generated on:      Fri May  20 18:36:46 2022
#  Design:            LUDH_TEST_WRAPPER
#  Command:           verify_connectivity
#################################################################
Verify Connectivity Report is created on Fri May 20 18:36:46 2022



Begin Summary
    Found no problems or warnings.
End Summary
```

(b) Connectivity



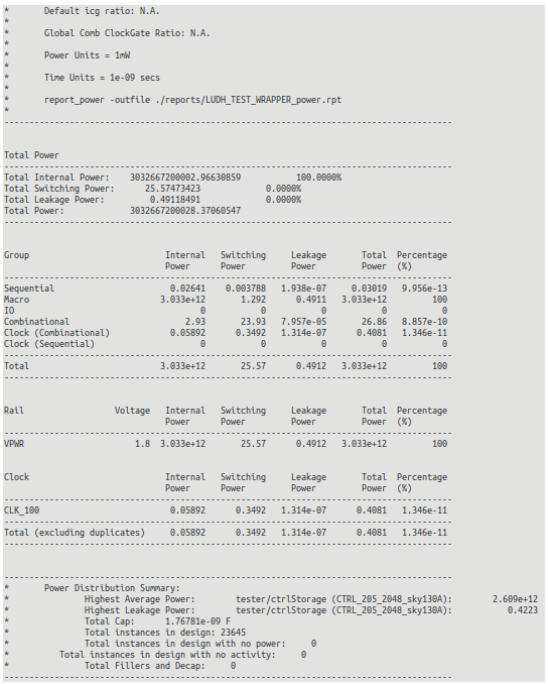
```
#################################################################
#  Generated by:      Cadence Encounter 14.28-s033_1
#  OS:                Linux x86_64(Host ID vlsi91.ee.iitb.ac.in)
#  Generated on:      Fri May 20 19:07:16 2022
#  Design:            LUDH_TEST_WRAPPER
#  Command:           verifyProcessAntenna
#################################################################

No Violations Found
```

(c) Process Antenna Report



```
#########################################################
#  Generated by:      Cadence Encounter 14.28-s033_1
#  OS:                Linux x86_64(Host ID vlsi91.ee.iitb.ac.in)
#  Generated on:      Sun May 22 09:08:10 2022
#  Design:            LUDH_TEST_WRAPPER
#  Command:           checkPlace -ignoreOutOfCore -noPreplaced -macroBlockage LUDH_TEST_WRAPPER.checkPlace
#########################################################

## No violations found ##

## Summary:
#########################################################
## Number of Placed Instances = 103626
##   of which 9887 are fixed.
## Number of Unplaced Instances = 0
## Placement Density:100.00%(702394/702394)
```

(d) Placement Report



```
*       Default icg ratio: N.A.
*
*       Global Comb ClockGate Ratio: N.A.
*
*       Power Units = 1nW
*
*       Time Units = 1e-09 secs
*
*       report_power -outfile ./reports/LUDH_TEST_WRAPPER_power.rpt
*
-------------------------------------------------------------------------------


Total Power
-------------------------------------------------------------------------------
Total Internal Power:    3032667200002.96630859          100.0000%
Total Switching Power:      25.57473423            0.0000%
Total Leakage Power:         0.49118491            0.0000%
Total Power:             3032667200028.37060547
-------------------------------------------------------------------------------


Group                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-------------------------------------------------------------------------------
Sequential                      0.02641    0.003788  1.938e-07     0.03019   9.956e-13
Macro                        3.033e+12       1.292     0.4911  3.033e+12         100
IO                                   0           0          0           0           0
Combinational                     2.93       23.93  7.957e-05       26.86   8.857e-10
Clock (Combinational)          0.05892      0.3492  1.314e-07      0.4081   1.346e-11
Clock (Sequential)                   0           0          0           0           0
-------------------------------------------------------------------------------
Total                        3.033e+12       25.57     0.4912  3.033e+12         100
-------------------------------------------------------------------------------


Rail                 Voltage   Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-------------------------------------------------------------------------------
VPWR                     1.8  3.033e+12       25.57     0.4912  3.033e+12         100


Clock                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-------------------------------------------------------------------------------
CLK_100                         0.05892      0.3492  1.314e-07      0.4081   1.346e-11
-------------------------------------------------------------------------------
Total (excluding duplicates)    0.05892      0.3492  1.314e-07      0.4081   1.346e-11
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
*       Power Distribution Summary:
*            Highest Average Power:       tester/ctrlStorage (CTRL_205_2048_sky130A):       2.609e+12
*            Highest Leakage Power:       tester/ctrlStorage (CTRL_205_2048_sky130A):         0.4223
*            Total Cap:      1.76781e-09 F
*            Total instances in design: 23645
*            Total instances in design with no power:     0
*         Total instances in design with no activity:     0
*            Total Fillers and Decap:     0
-------------------------------------------------------------------------------
```

(e) Power Report

Figure 6.8: DRC, Placement, Connectivity, Antenna Check report and Power Summarry

Chapter 7

Conclusion

The complete workflow consists of FPGA Implementation and scheduler mentioned is given in the following figure:-

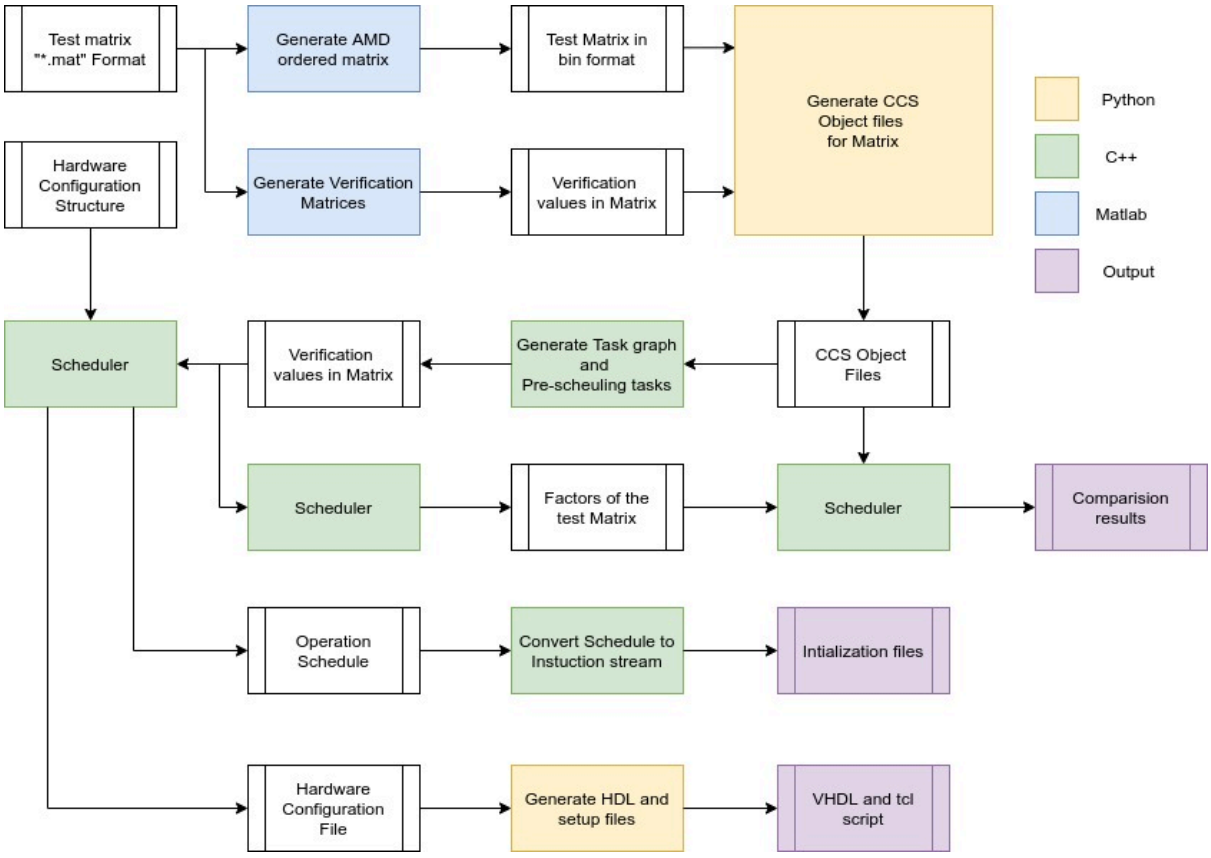


Figure 7.1: Workflow of Software

The hardware is generated to an AXI wrapper IP such that it can be connectable with the Microblaze (soft-core processor) or ZynQ (hard-core processor) is generated. The following is the report of Dual Port IP hardware.

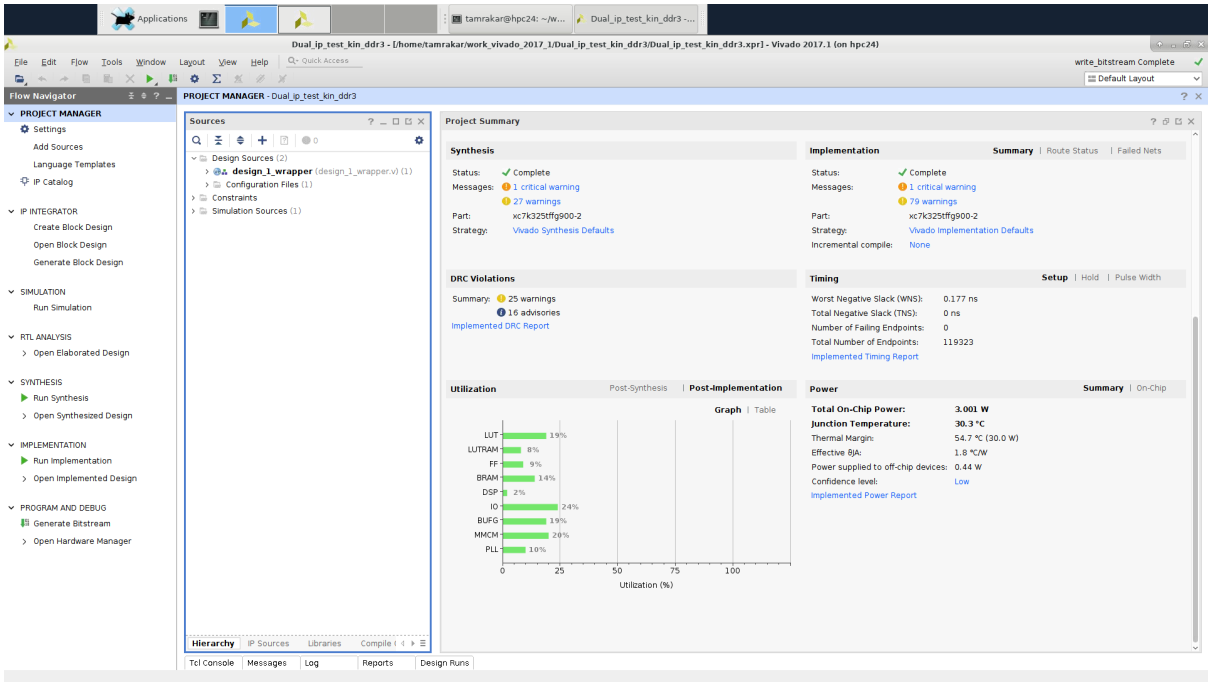


Figure 7.2: IP generation report

The AXI wrapper hardware is connected to MicroBlaze and the necessary components to properly function on Kintex KC-705 Xilinx Board. A cache is attached to increase the efficiency of the executions. Following is the block-level design of the interconnections of hardware for Testing.

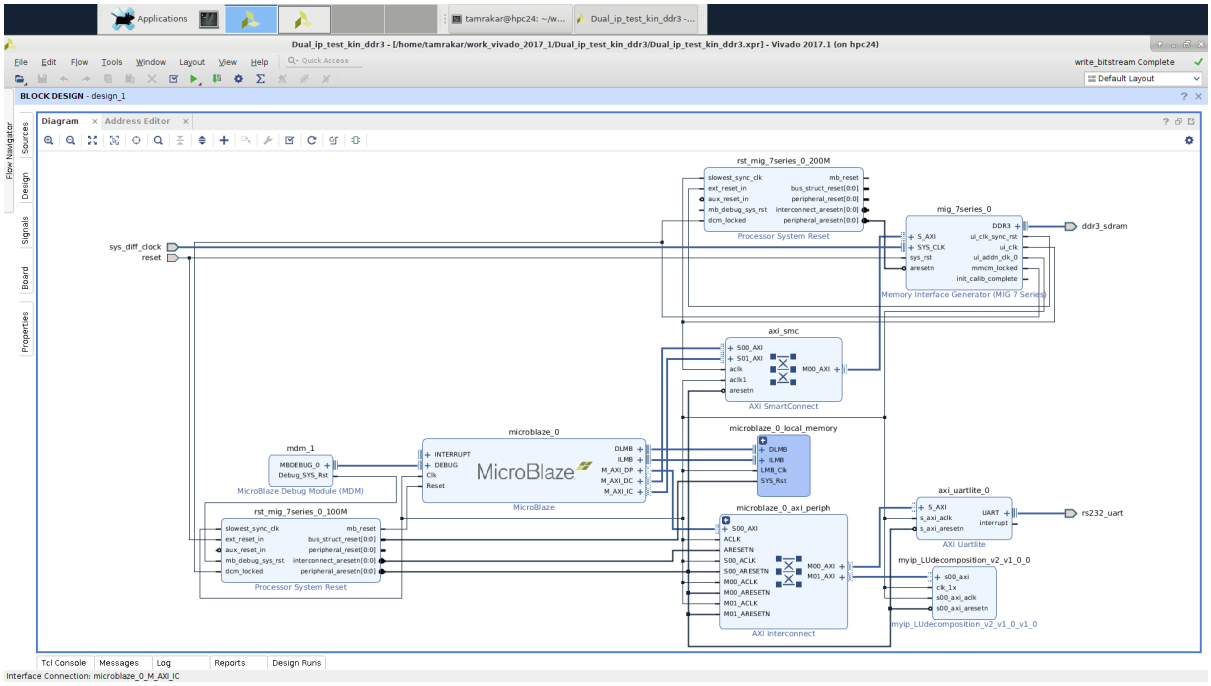


Figure 7.3: MicroBlaze interconnect

Microblaze code generated by the schedule for uploading instructions and data stream is

used for interfacing IP and MicroBlaze. This was implemented and tested successfully on the Kintex KC-705 board, and the output is taken from UART and verified with golden reference generated by MATLAB script.

Shakti-based architecture Design is successfully constructed, integrated, and implemented on the Xilinx Arty-100T board. Some linguistic edition was required for ShaktiSDK because of different ISA, and accelerator tested results successfully

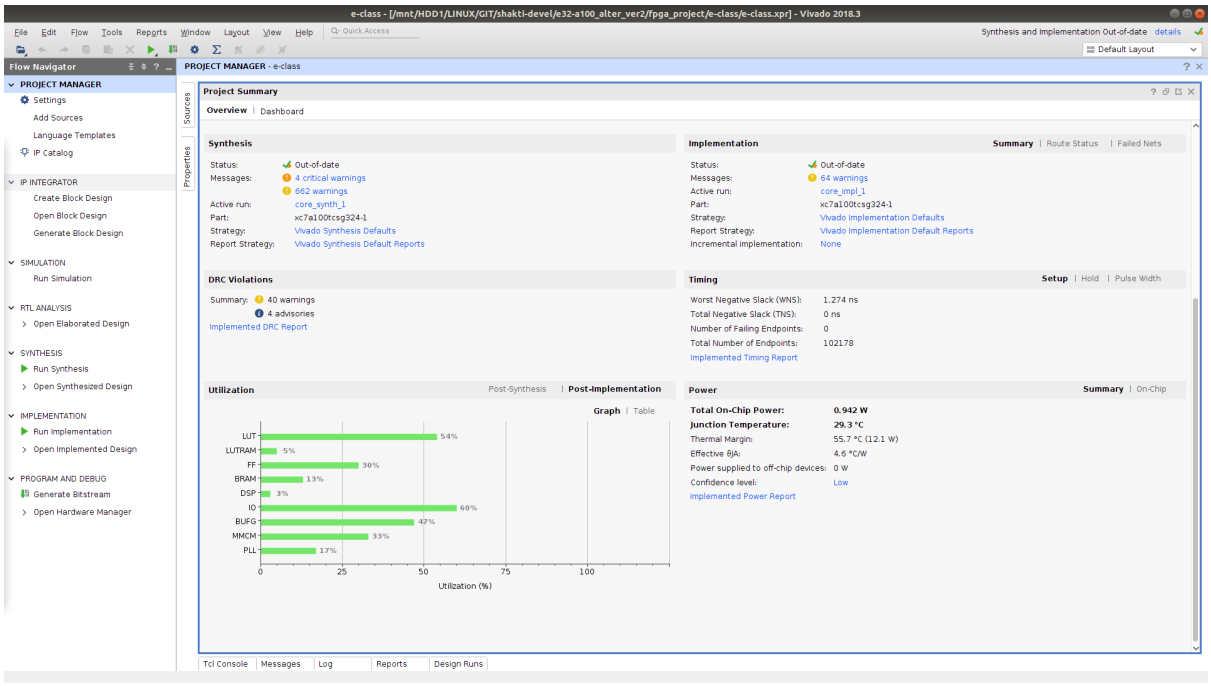


Figure 7.4: Shakti interconnect report

Accomplished final ASIC Design successfully with the help of OpenRAM(for SRAM), OpenLane(for Processing Elements), and Cadence SOC encounter(Hardening the Macros). The Proper sign-off tool decks were absent for Physical verification, although the design achieved the generic physical verification and Timing constraints.The overall Chip Area is $3796.630 \times 4944.565 \mu m^2$ i.e $18.772 mm^2$

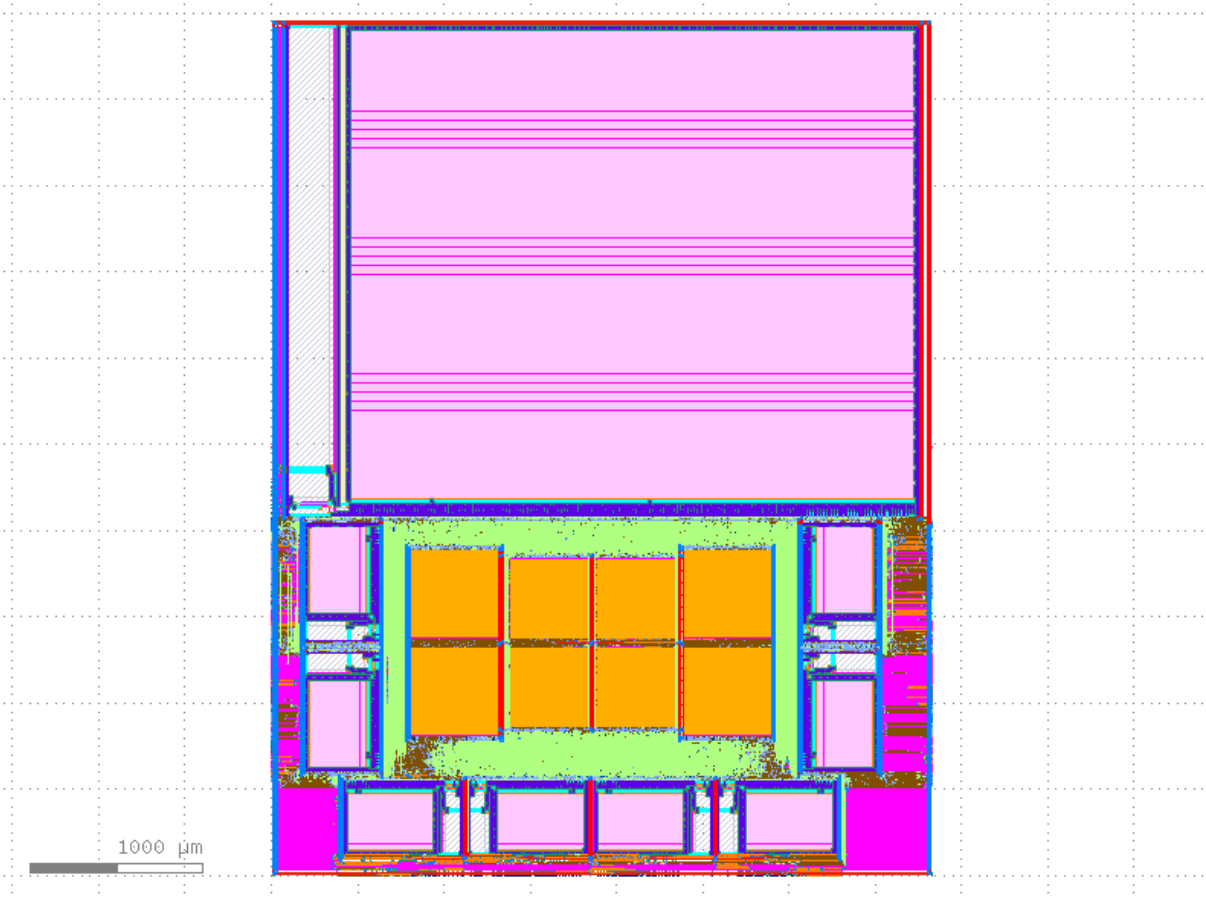


Figure 7.5: ASIC Implementation with 8 SRAMs and 4 Processing Elements

Currently following are the scope for further improvements and implementation in the projects:-

- ILP Based Scheduler
- Scheduler for QR Decomposition
- Uniform Channel Decomposition for MIMO communication
- Scalable Architecture

Bibliography

[1] N. Kapre and A. DeHon, "Parallelizing sparse matrix solve for spice circuit simulation using fpgas," in *2009 International Conference on Field-Programmable Technology*, pp. 190–198, Dec 2009.

[2] W. Wu, Y. Shan, X. Chen, Y. Wang, and H. Yang, "Fpga accelerated parallel sparse matrix factorization for circuit simulations," in *Reconfigurable Computing: Architectures, Tools and Applications* (A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, eds.), (Berlin, Heidelberg), pp. 302–315, Springer Berlin Heidelberg, 2011.

[3] T. Nechma and M. Zwolinski, "Parallel sparse matrix solution for circuit simulation on fpgas," *IEEE Transactions on Computers*, vol. 64, pp. 1090–1103, April 2015.

[4] S. Mehta, "Digital asic implementation of length-73 low density parity check decoder," 2021.

[5] S. T. W.H. Press, "Crout's method," in *Numerical Recipes 3rd edition: The Art of Scientific Computing*, Cambridge Univ. Press, 2007.

[6] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," vol. 38, (New York, NY, USA), Association for Computing Machinery, Dec. 2011.

[7] J. Gilbert and T. Peierls, "Sparse partial pivoting in time proportional to arithmetic operations," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 5, pp. 862–874, 1988.

[8] Xilinx, "Block memory generator v8.3 logicore ip product guide vivado design suite."

[9] Xilinx, "Floating-point operator v7.1 logicore ip product guide vivado design suite."

[10] "Shakti parashu board configure." https://gitlab.com/shaktiproject/sp2020/tree/master/e32-a100.

[11] "Shakti e-class manual." https://gitlab.com/shaktiproject/cores/e-class.

[12] "Shakti soc user manual." https://shakti.org.in/docs/shakti-soc-user-manual.pdf.

[13] M. Guthaus, "Openram,an open-source static random access memory (sram) compiler." https://github.com/VLSIDA/OpenRAM.

[14] "Openlane,an open-source automated rtl to gdsii." https://github.com/The-OpenROAD-Project/OpenLane.

[15] "Skywater 130 nm pdk for digital flow." https://code.stanford.edu/ee272/skywater-130nm-adk.git.

[16] K. Ghosh, "Vsdsram, openram for skywater 130nm." https://github.com/vsdip/vsdsram_sky130.