Matrix Computation Acceleration using FPGAs

M.Tech Project Report I

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
2021

Dissertation Approval

This dissertation entitled

Matrix Computation Acceleration using FPGAs

by

Aashish Tamrakar
(Roll No. : 193079034)

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

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

Date: October 6, 2021
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: October 6, 2021

Acknowledgement

I would like to express my deep gratitude to Prof. Sachin Patkar for his kind guidance all through. I also extend thanks to 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.

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 focuses on accelerating LU decomposition of the sparse matrix using FPGA, where parallelism can exploit the parallelism appropriately.

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 two major divisions:

- Scheduler
- Hardware

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.

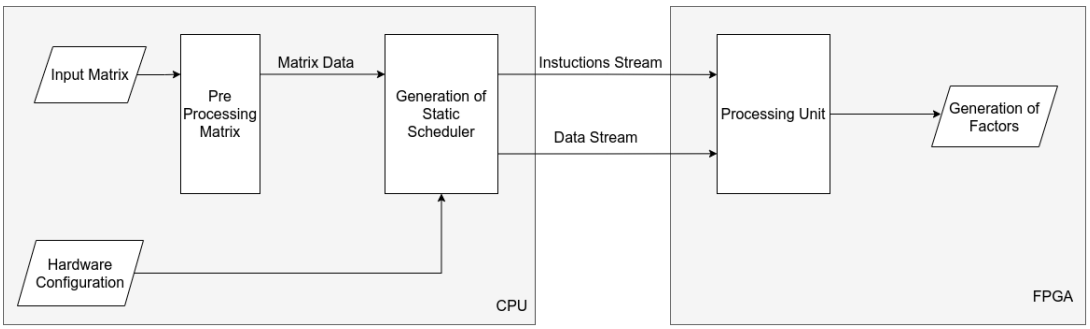


Figure 1.1: The overall flow of the project

The summary of the thesis is given below:

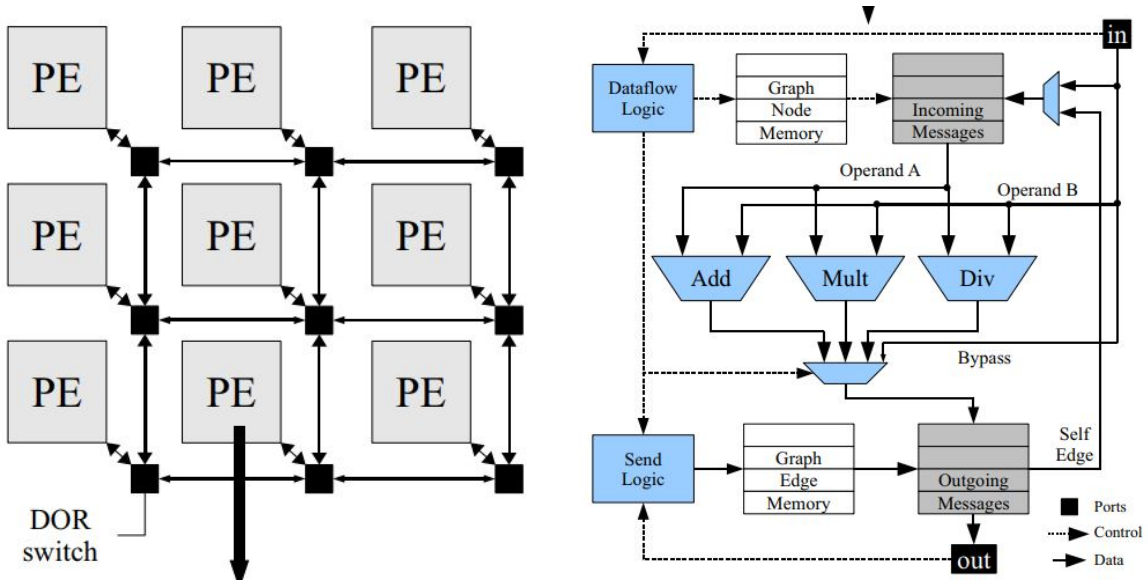
- Preliminaries and literature review
- Scheduler
- Hardware Design Implementation

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.

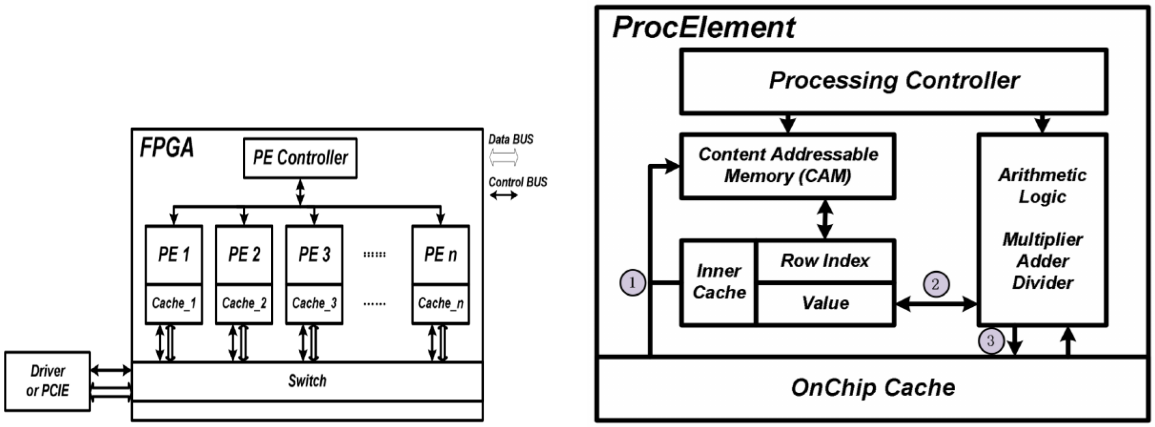
In [1] N. Kapre and A. DeHon proposed a method at the International Conference on Field-Programmable Technology in 2009. He accelerated the LU decomposition of sparse matrices on FPGA using a mesh-grid type architecture in NoC. The significant advantage of his approach is that it's scalable, which is 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: -

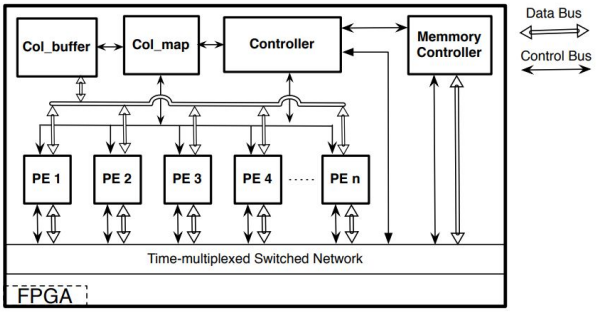


Figure 2.3: Hardware Architecture used by Nechma [3]

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 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.

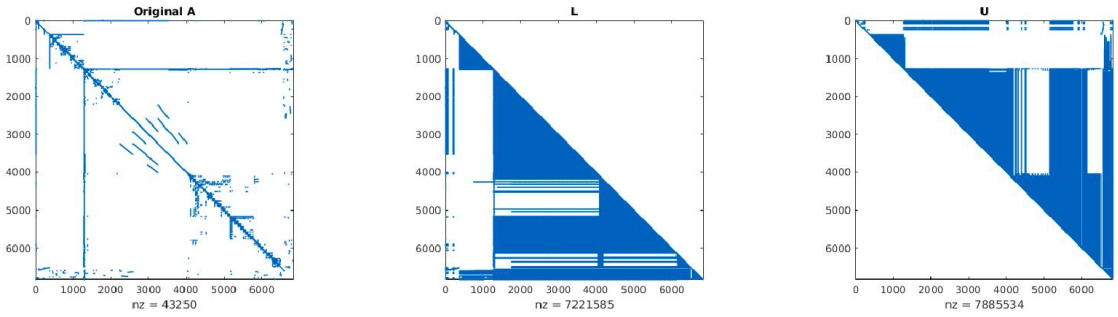


Figure 3.2: Non-zero elements of L and U with original matrix A

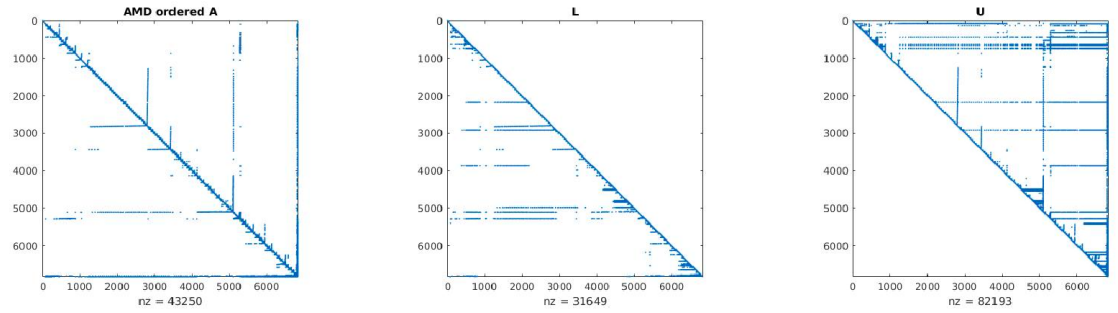


Figure 3.3: Non-zero elements of L and U with AMD ordered matrix permuted A

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.

$$(P^T AP)(P^T x) = P^T b \tag{3.1}$$

3.3 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:-

$$U_{(i,j)} = A_{(i,j)} - \sum_{k=1}^{i-1} L_{(i,k)} U_{(k,j)} \tag{3.2a}$$

$$L_{(i,j)} = \frac{A_{(i,j)} - \sum_{k=1}^{j-1} L_{(i,k)} U_{(k,j)}}{U_{(j,j)}} \tag{3.2b}$$

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 [7]. We will be using Gilbert-Peierls Algorithm: A Column-Oriented LU Factorization with partial pivoting. 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 1 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
```

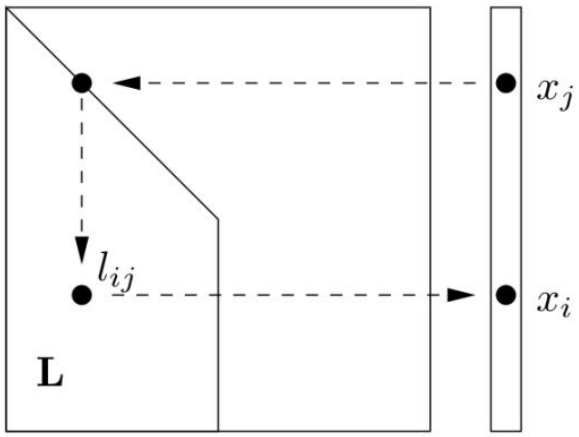


Figure 3.4: Non-zero pattern in LU decomposition

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.

$$(b_i \neq 0) \implies (x_i \neq 0) \tag{3.3a}$$

$$(x_j \neq 0) and \exists i (l_{ij} \neq 0) \implies (x_i \neq 0) \tag{3.3b}$$

Symbolic Analysis can visualize these two implications using the figure 3.4.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.
We can consider following example with 10×10 in the figure with 3.5

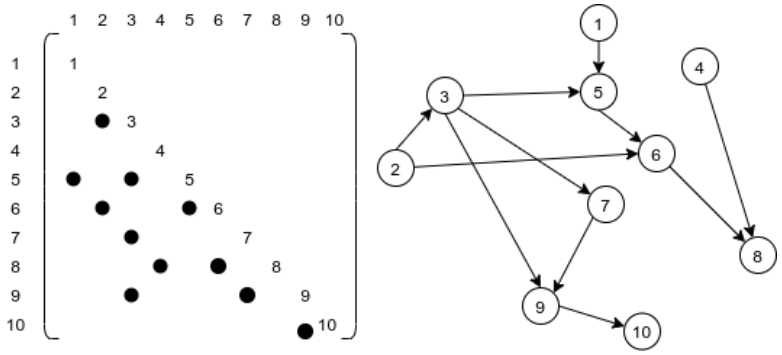


Figure 3.5: DAG generated for 10×10 Sample Matrix

As per the non-zero value obtained from the value of b in $Lx = b$ will be sharing A memory map for the execution of the algorithm [8]

3.4 Gilbert-Peierls' Algorithm

Gilbert's Perierls [9] 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 2 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.

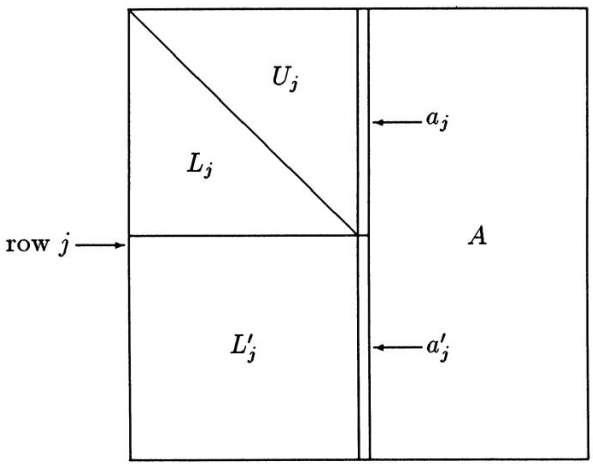


Figure 3.6: Naming conventions used in algorithm

State-of-the-art Gilbert-Peierl's algorithm [9] 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*.

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

The State-of-the-art Gilbert's Transform requires the locations of non-zero elements. Therefore, this includes the need for symbolic analysis.
The algorithm can be easily understandable by considering the analysis step using the following example:-

Consider the following matrix with the non-zero Locations red in color whereas zeros' as white color.

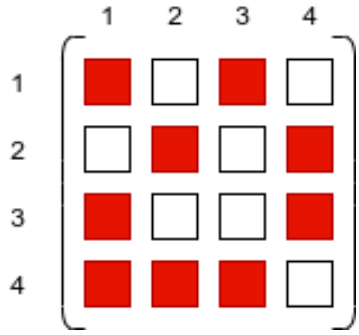


Figure 4.1: A matrix non-zero location

As per Gilbert's algorithm, We will be analyzing column by column. The new fill-in Matrix will be indicated using yellow color.Symbolic Analysis is analyzing data and creating the directed acyclic graph for the floating-point operations.We will be considering **??**

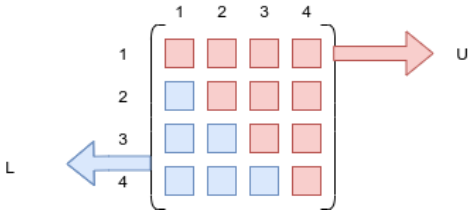
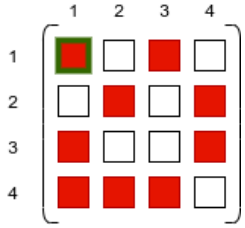
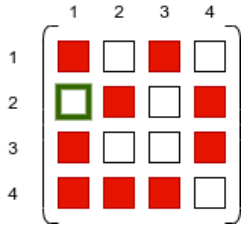


Figure 4.2: Convention of LU Matrix Position

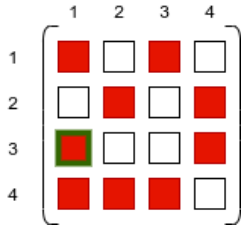
Column 1: -
It is the only column that is guaranteed to have no other additional non-zero elements.



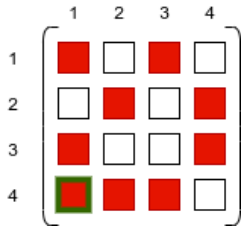
According to equation 3.2a,
$U_{1,1} = A_{1,1}$



According to equation 3.2b,
$L_{2,1} = 0$

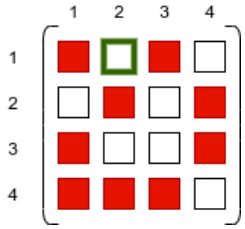


According to equation 3.2b,
$L_{3,1} = A_{3,1}/U_{1,1}$

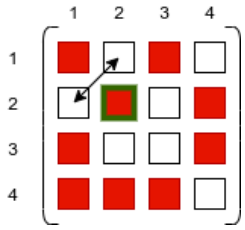


According to equation 3.2b,
$L_{4,1} = A_{4,1}/U_{1,1}$

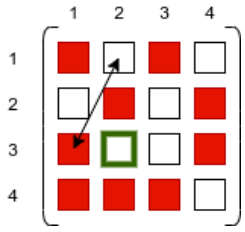
Column 2: -



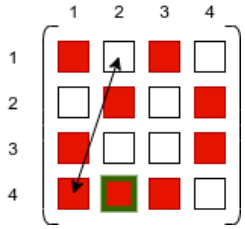
According to equation 3.2a,
$U_{1,2} = 0$



According to equation 3.2a,
$U_{2,2} = A_{2,2}$

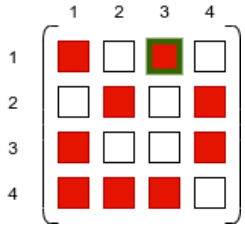


According to equation 3.2b,
$L_{3,2} = 0$

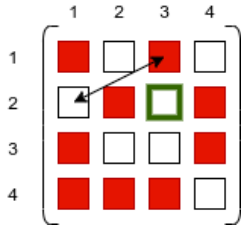


According to equation 3.2b,
$L_{4,2} = A_{4,2}/U_{2,2}$

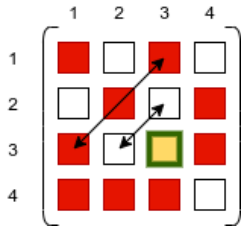
Column 3: -



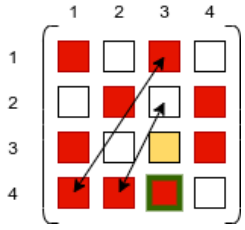
According to equation 3.2a,
$U_{1,3} = A_{1,3}$



According to equation 3.2a,
$U_{2,3} = 0$

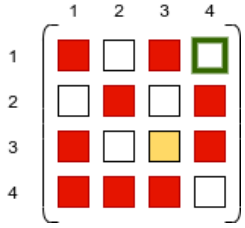


According to equation 3.2a,
$U_{3,3} = -(L_{3,1} \cdot U_{1,2})$

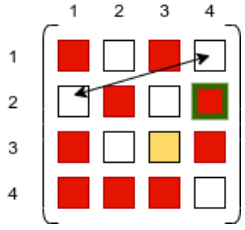


According to equation 3.2b,
$L_{4,3} = A_{4,3} - (L_{4,1} \cdot U_{1.3})/U_{3,3}$

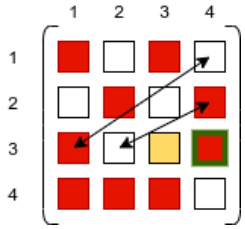
Column 4: -



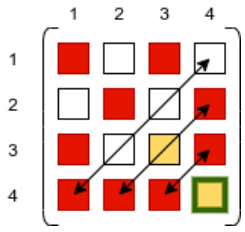
According to equation 3.2a,
$U_{1,4} = 0$



According to equation 3.2a,
$U_{2,4} = A_{2,4}$



According to equation 3.2a,
$U_{3,4} = A_{3,4}$



According to equation 3.2a,
$U_{4,4} = -(L_{4,2} \cdot U_{2,4})$

This would be resulting in the **??** in terms of non-zero matrix locations. As we can see that there are new locations (colored yellow) while decomposing the matrix.

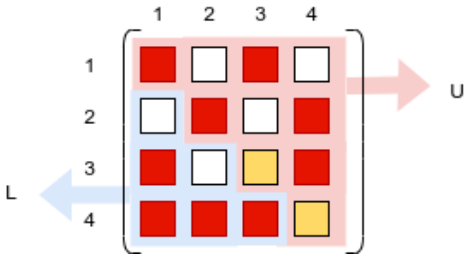


Figure 4.3: Non-zero location of Matrix

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: -

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

Using The Priority ordering and Symbolic analysis in 4.1, the Following DAG is generated, 4.4

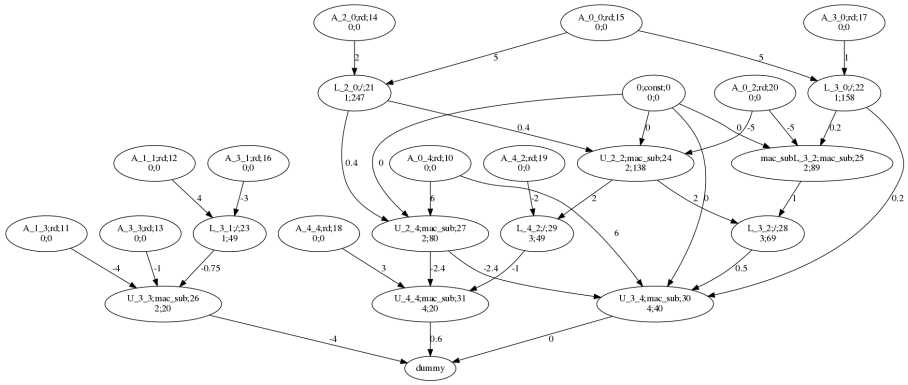


Figure 4.4: Generated DAC

The meaning of terms inside each node of the data flow graph is described in the figure 4.5 below: -

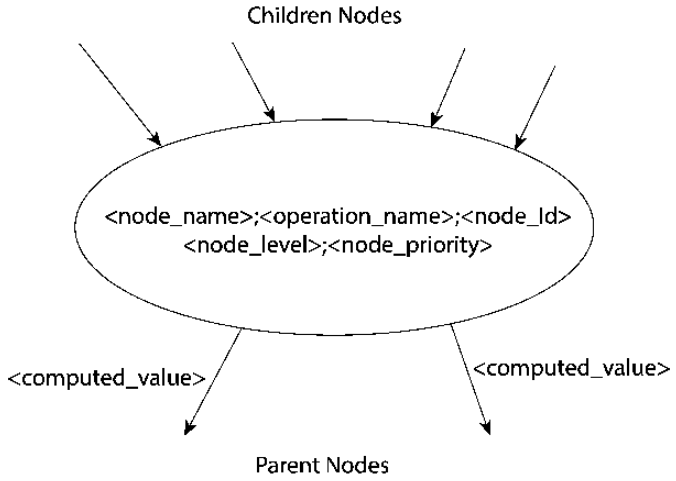


Figure 4.5: DAC node convention

Note the convention used by children nodes and parent nodes in this project. Nodes which are providing a value to a given node are **children** of the given node. And the nodes which are accepting value from a given node are **parents** to the given node

<node_name>: Indicates the name associated with the node. he node name is of the form “L_x_y”,“U_x_y”, “mac_subL_x_y” etc. The “node_name” is unique.

<operation_name>: Indicates the operation performed by the node. The operations can be “rd”(memory read), “wr”(memory write), “mac_sub” (MAC), “/”(DIV) or “const” (indicates that the node has a constant value of 0).

<node_Id>: Indicates a unique Id associated with each node.

<node_level>: Indicates the level of the node i.e. the worst-case distance from a leaf node.

<node_priority>: It indicates the priority level of the node. This parameter is extremely useful while scheduling the graph.

<computed_value>: This parameter indicates the value computed by the node. Ex. $U_{1,1} = 5$, $U_{2,3} = -2.65$ etc.

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.

Considering an example.We have following which are pipeline in nature mentioned in 4.6 2 DIV units(latency = 2) and 2 MUL units(latency = 1)

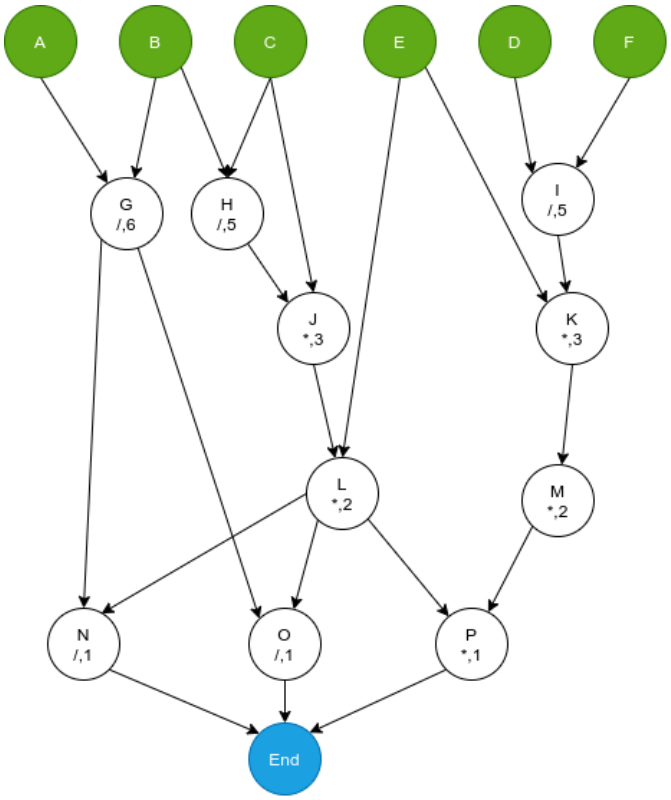
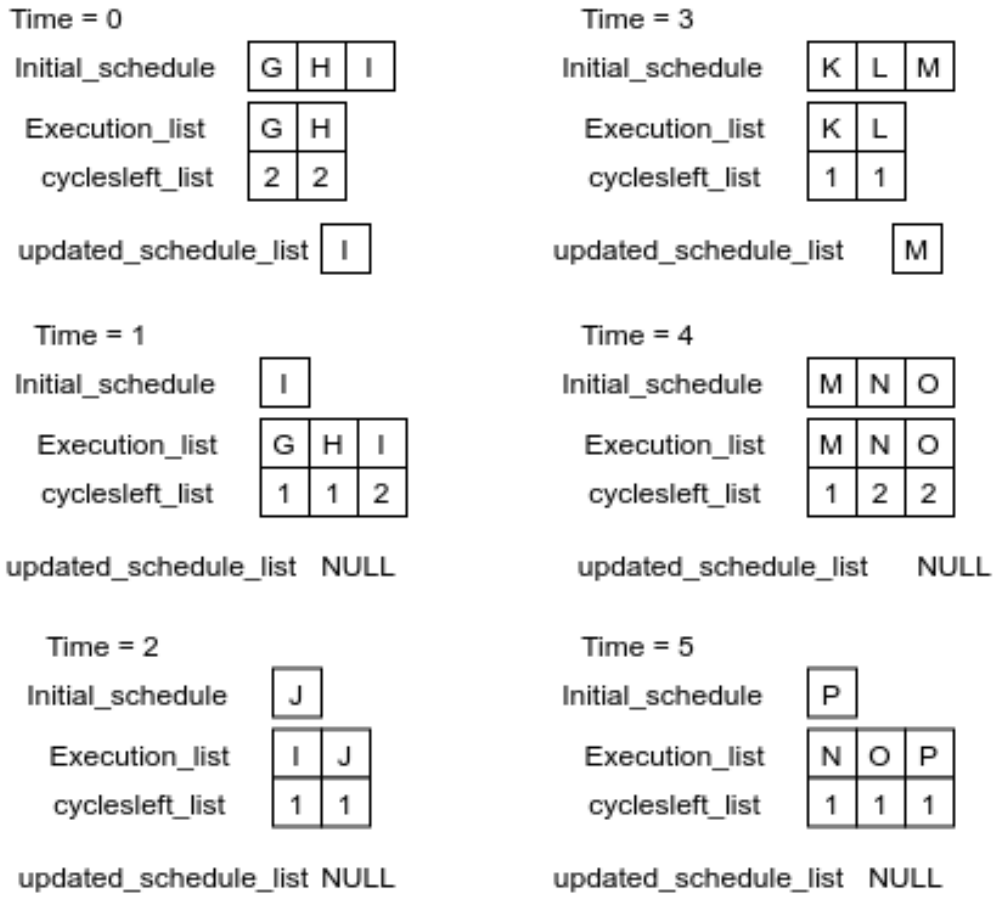


Figure 4.6: Priority-Based Algorithm

For demonstrating list scheduling with this graph, I have considered four lists "schedulable_list," "execution_list" & "cyclesLeft_list". The last two lists go hand in hand. The contents of these lists in each cycle is given below: -



The concept behind list scheduling used in this project is same as the above example although included some of the real constraints

- All the operand are fetch from memory i.e. limited number of ports
- Handling Live of operands
- Resource Binding of Processing elements

Therefore following are the inputs to the Scheduler

- Number of BRAM blocks
- Number of ports per BRAM block
- Number of MAC and DIV units
- Latencies of MAC and DIV units

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 above Concept can be explainable by following flow chart figure 4.7

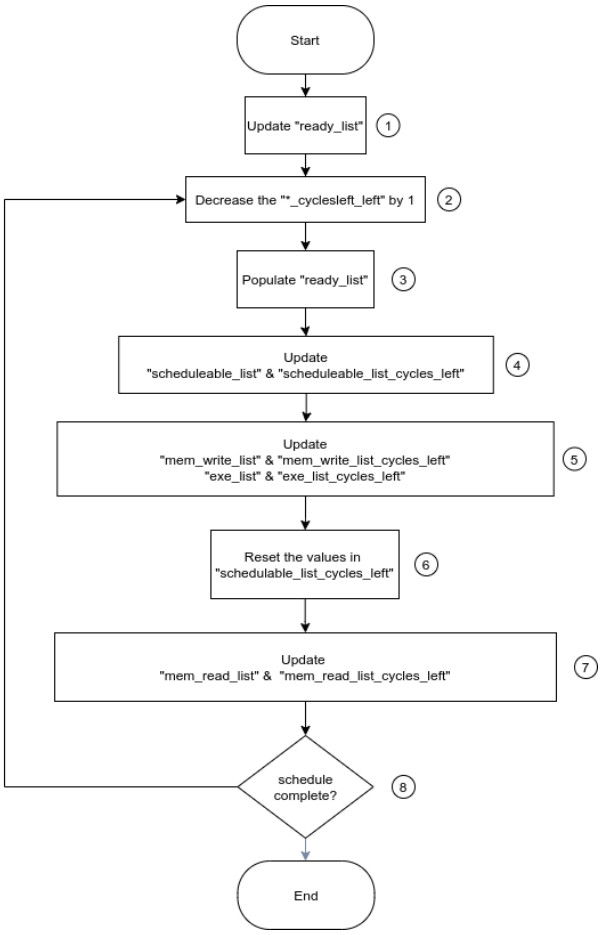


Figure 4.7: Flowchart of Priority List based Scheduling Process [4]

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.[4]

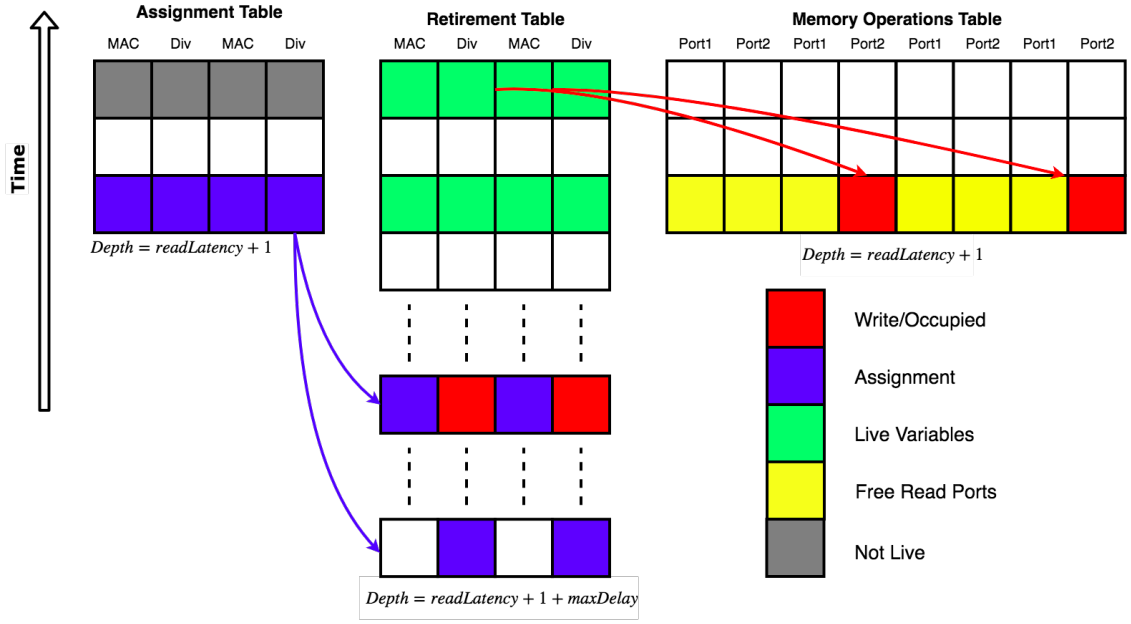


Figure 4.8: Allocation tables used by scheduling algorithm [5]

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

Hardware Design

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

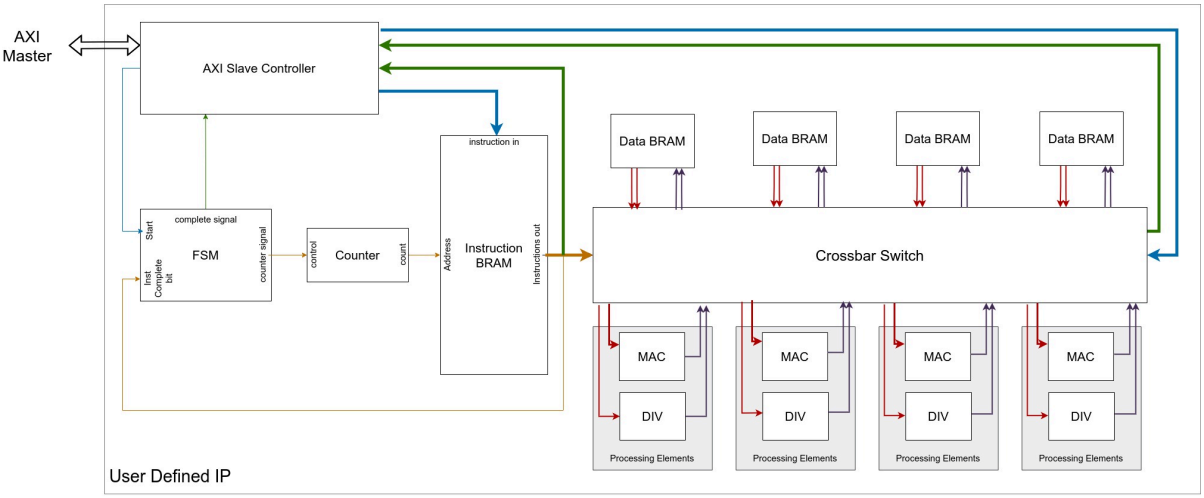


Figure 5.1: Hardware architecture

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) [10]
  - Instruction BRAM (custom bit size)
  - Data BRAM (float / double type)
- Floating-point [11]
  - 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.

Chapter 6

Conclusion

The complete workflow consists of hardware and scheduler mentioned in the project are given in the following figure:-

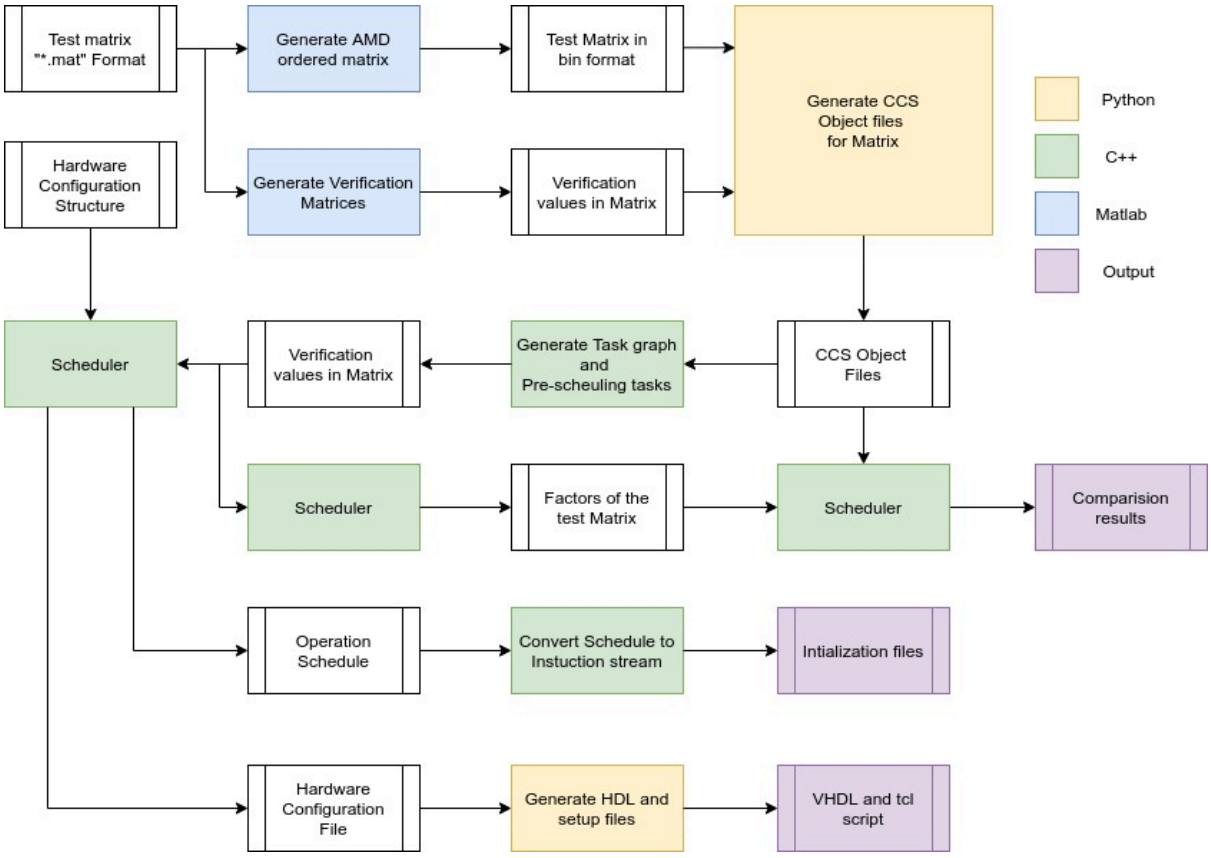


Figure 6.1: Workflow of Software

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.

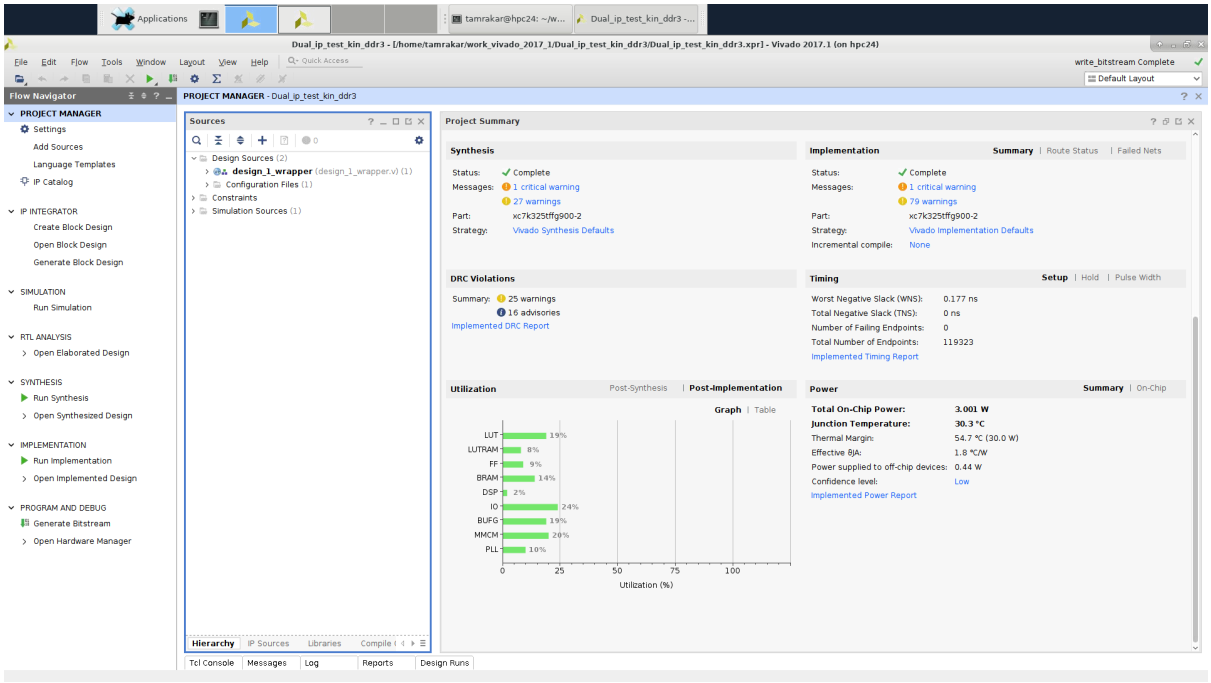


Figure 6.2: IP generation report

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.

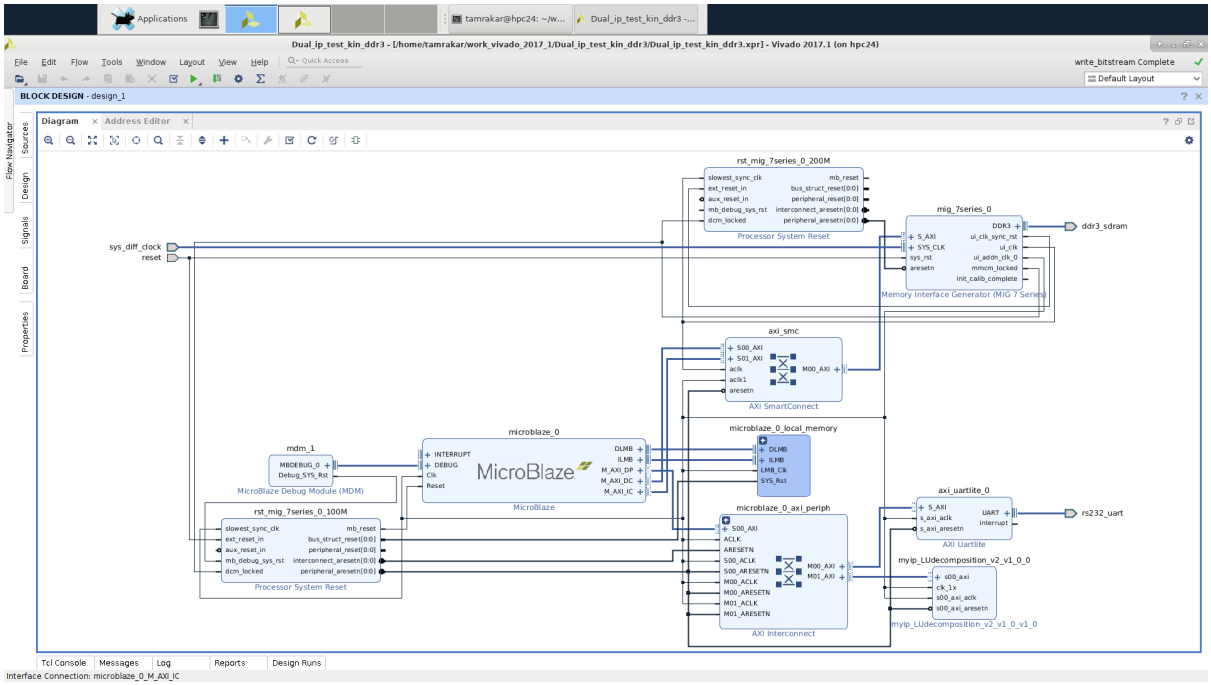


Figure 6.3: MicroBlaze interconnect

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.

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

- Dynamic Scheduler
- ILP Based Scheduler
- Scheduler for QR Decomposition
- Uniform Channel Decomposition for MIMO communication
- Generating ASIC Design
- Integration with generic processor

Phase II of the Project will converge to some of the improvements and implementation mentioned above.

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] A. Choudhury, "Fpga implementation of lu decomposition algorithm on crossbar network," vol. Thesis Indian Institute of Technology Bombay, 2020.

[5] Y. Mahajan, S. Obla, M. K. Namboothiripad, M. J. Datar, N. N. Sharma, and S. B. Patkar, "Fpga-based acceleration of lu decomposition for analog and rf circuit simulation," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pp. 131–136, 2020.

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

[8] T. A. Davis, *Direct methods for sparse linear systems*. SIAM, 2006.

[9] 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.

[10] Xilinx, *Block Memory Generator v8.3 LogiCORE IP Product Guide Vivado Design Suite*.

[11] Xilinx, *Floating-Point Operator v7.1 LogiCORE IP Product Guide Vivado Design Suite*.