FPGA Implementation of LU decomposition Algorithm on a Crossbar Network

by

Anurag Choudhury
Roll No: 183070032
Email Id: anuragn85@gmail.com

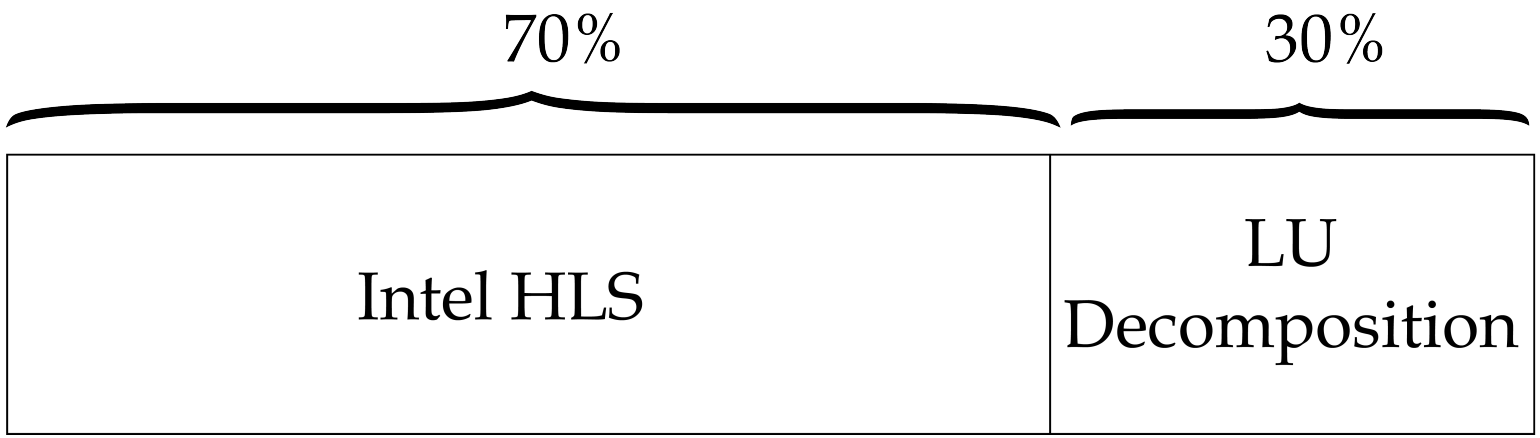
Under the guidance of
Prof. Sachin Patkar

The presentation is divided into following 10 sections

1. Brief discussion on Phase 1 work
2. What is LU decomposition ?
3. Motivation behind the project
4. Overall flow of the project
5. Pre-processing Stage
6. Symbolic analysis and DFG generation Stage
7. Static schedule generation Stage
8. Hardware implementation Stage
9. Results and Observations
10. Future Work

by Anurag Choudhury
Email Id: anuragn85@gmail.com

1. Brief discussion on MTP Phase 1 work



1.1 Intel HLS

- Ported a final PhD project[5] of a student under Prof. Patkar from Vivado HLS to Intel HLS
- Came across many critical concepts not mentioned in official Intel HLS documentation
- All the findings have been clearly documented in my MTP Phase 1 report

by Anurag Choudhury
Email Id: anuragn85@gmail.com

2. What is LU decomposition ?

LU decomposition stands for lower-upper decomposition. In this we factor a square matrix A into product of a Lower Triangular matrix(L) and an Upper Triangular matrix(U). This is depicted in the figure below:-

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & A_{1,5} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & A_{2,5} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} & A_{3,5} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} & A_{4,5} \\ A_{5,1} & A_{5,2} & A_{5,3} & A_{5,4} & A_{5,5} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ L_{2,1} & 1 & 0 & 0 & 0 \\ L_{3,1} & L_{3,2} & 1 & 0 & 0 \\ L_{4,1} & L_{4,2} & L_{4,3} & 1 & 0 \\ L_{5,1} & L_{5,2} & L_{5,3} & L_{5,4} & 1 \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} & U_{1,4} & U_{1,5} \\ 0 & U_{2,2} & U_{2,3} & U_{2,4} & U_{2,5} \\ 0 & 0 & U_{3,3} & U_{3,4} & U_{3,5} \\ 0 & 0 & 0 & U_{4,4} & U_{4,5} \\ 0 & 0 & 0 & 0 & U_{5,5} \end{bmatrix}$$

Note that the diagonal elements of matrix L is always 1.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

2.1 Formula for LU decomposition

If we solve A = LU, we get the following formulae for different elements of L and U:-

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

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

by Anurag Choudhury
Email Id: anuragn85@gmail.com

2. Motivation behind the project

The main benefit of LU decomposition is observed when we have to repeatedly solve a set of linear equations whose structure(non-zero locations) remains same.
Ex. Circuit Simulation
Hence the LU decomposition has to be performed for only once for a single circuit.

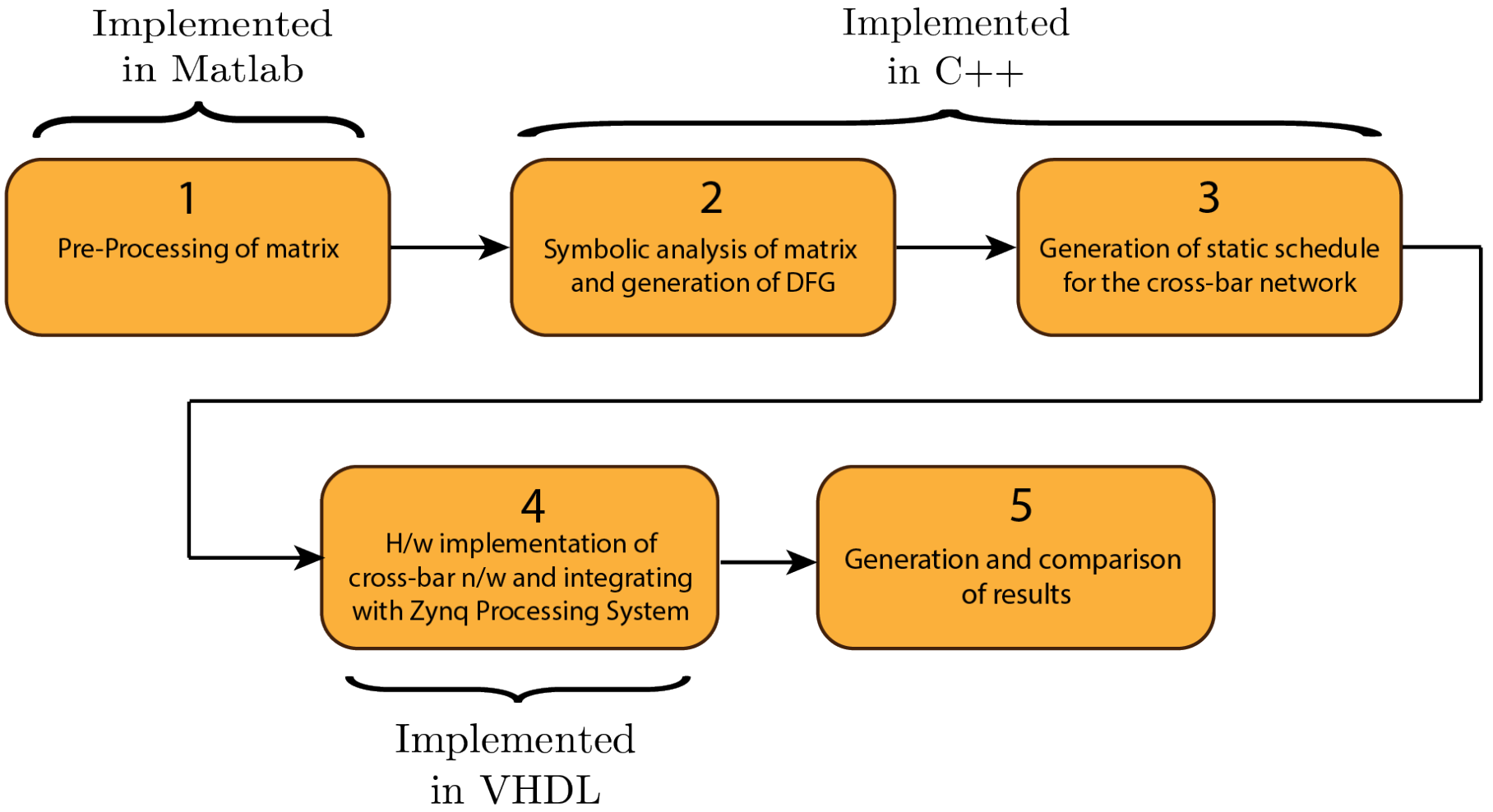
After performing the LU decomposition, instead of solving for $\mathbf{Ax = b}$,
we will be solving for:-

$$\boldsymbol{LUx = b}$$

The cost of factorization of matrix A into L and U is proportional to $\boldsymbol{N^3}$**.** Once the factorization is done, the cost of solving $LUx = b$ is proportional to $\boldsymbol{N^2}$. So, if we want to do the simulation for **k** timesteps, the total cost becomes proportional to $\boldsymbol{(N^3 + kN^2)}$**.**
On the other hand, if we solve the linear equation using **Gaussian Elimination**, the total cost becomes proportional to $\boldsymbol{kN^3}$ which is much larger than $\boldsymbol{(N^3 + kN^2)}$**.**

by Anurag Choudhury
Email Id: anuragn85@gmail.com

4. Overall flow of the project



by Anurag Choudhury
Email Id: anuragn85@gmail.com

5. Pre-Processing of Matrix A

Pre-processing of matrix is done for 2 reasons:-
a) To reduce the number of non zeros in L and U matrix
b) To ensure that none of the diagonal elements of U matrix is 0

The 1st objective is achieved using *Approximate Minimum Degree(AMD)* ordering.
The 2nd objective is achieved using permutation of rows of Matrix A.

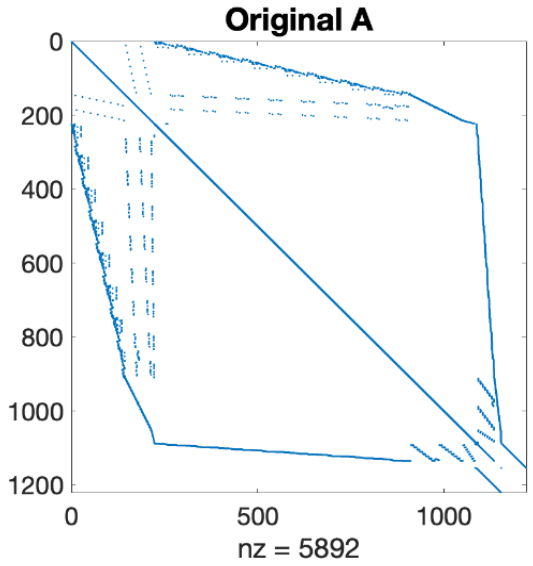
Because of pre-processing, the actual A matrix(i.e. A_{new}) which is being used for the remaining stages of the project is:-

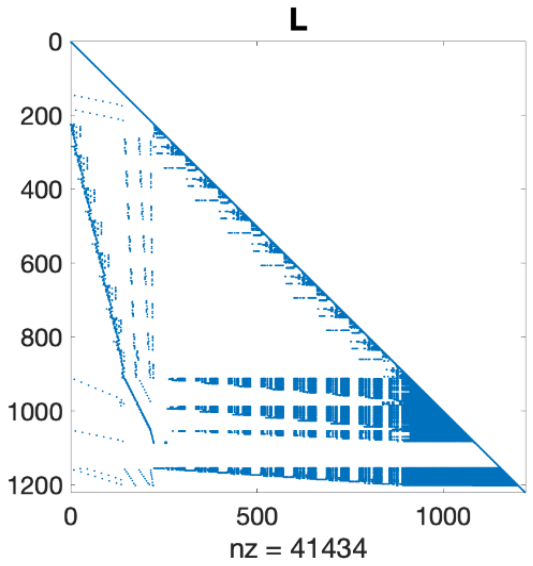
$$\mathbf{A_{new} = P * AMD(A_{original})}$$

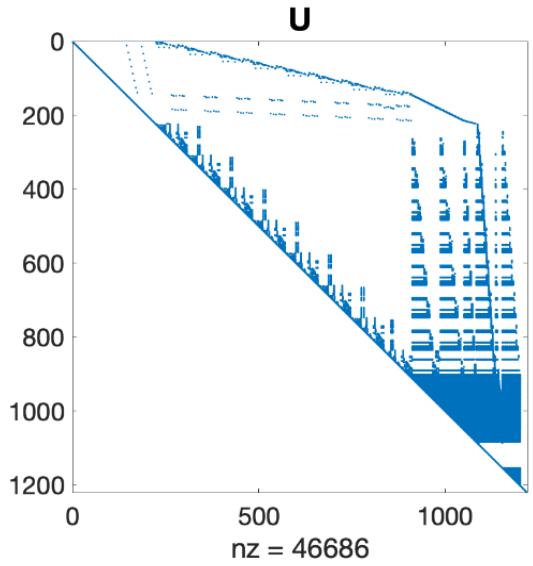
Here P is the permutation matrix.

The effect of AMD ordering on the matrix ***fpga_dcop_01*** which is of size ***1220x1220*** is shown in the next slide.

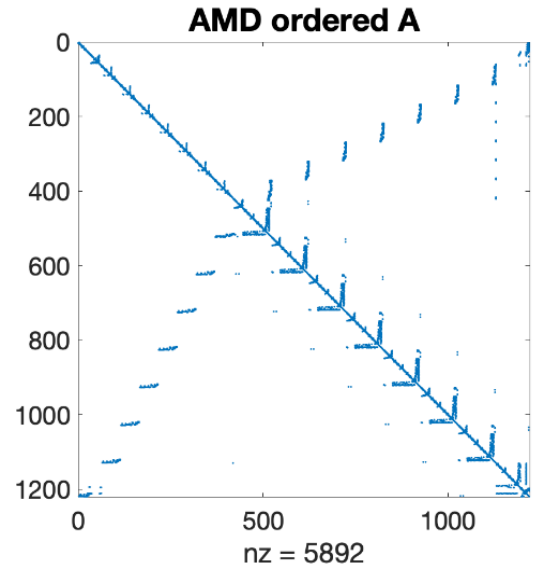
by Anurag Choudhury
Email Id: anuragn85@gmail.com

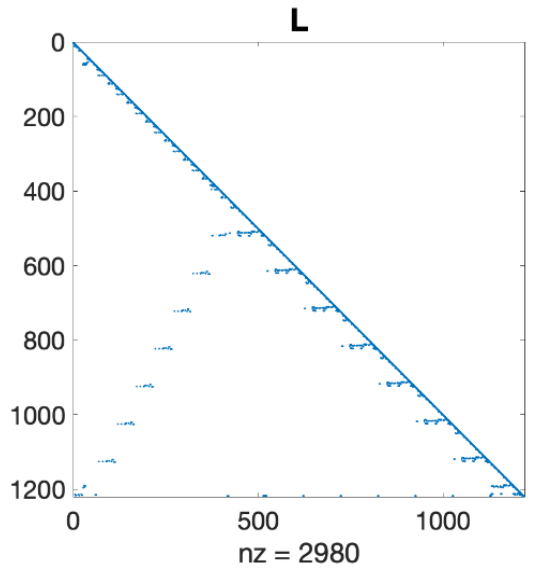


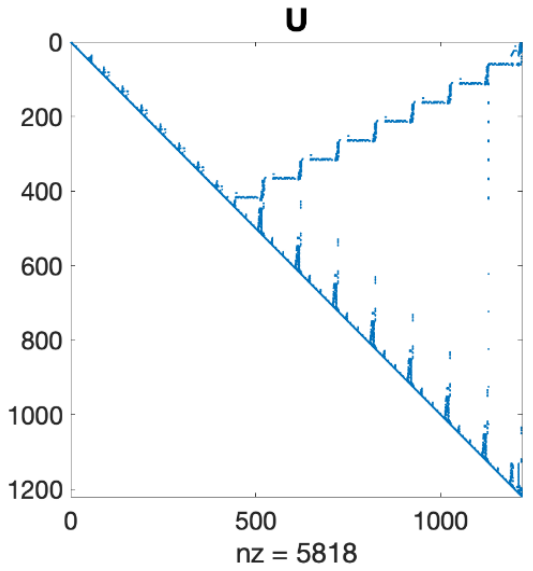




Almost **14x** reduction in non-zero elements of L matrix.





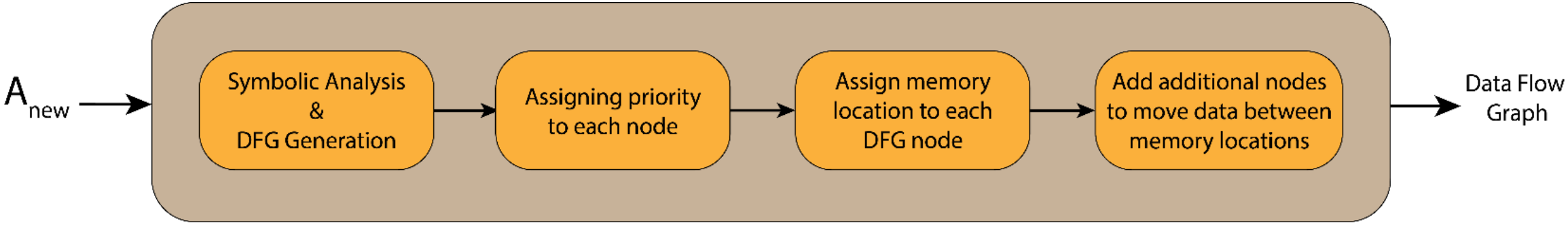


Almost **8x** reduction in non-zero elements of U matrix.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

6. Symbolic analysis and DFG generation Stage

The purpose of this stage is the elimination of unnecessary arithmetic operations encountered while computing L and U matrices. The substages of symbolic analysis stage are shown in the figure below: -



by Anurag Choudhury
Email Id: anuragn85@gmail.com

6.1 Symbolic analysis

In order to explain the symbolic analysis, consider the 5x5 matrix shown in fig (a) and the corresponding ***fil-in-tracking*** matrix shown in fig (b).

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

Fig (a)

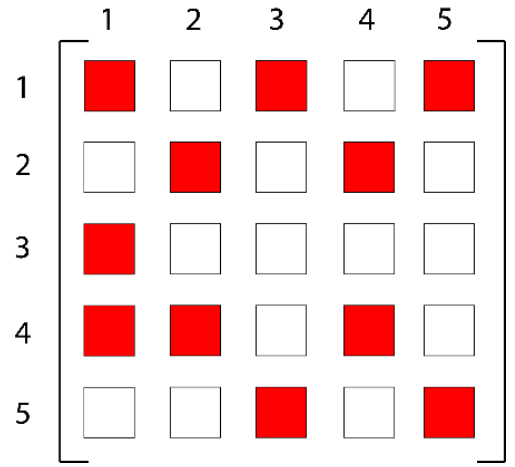
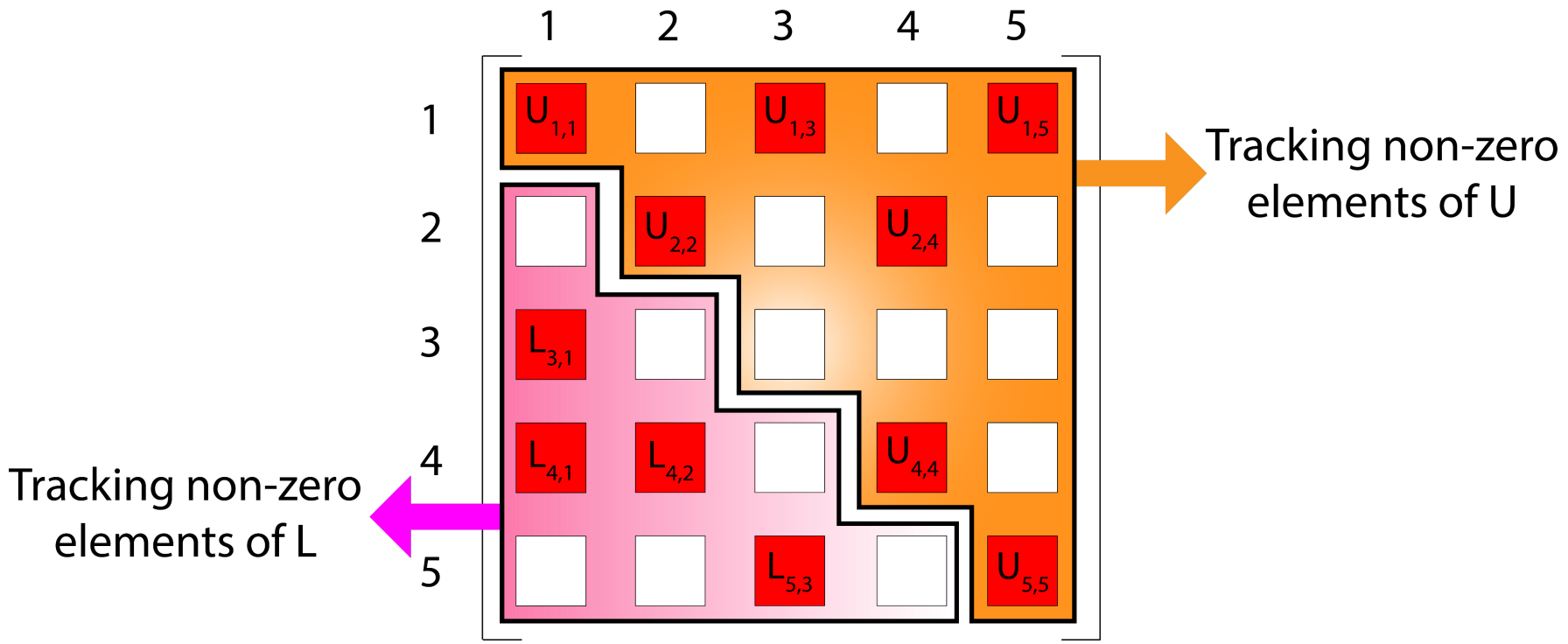


Fig (b)

The initial non zero locations are marked with red colour. The new non zero locations will be marked with yellow colour as we proceed with symbolic analysis column by column.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Understanding the *fill-in-tracking* matrix



The filled in blocks in upper part of the matrix indicates non-zero U elements.
The filled in blocks in the lower part of the matrix indicates non-zero L elements.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

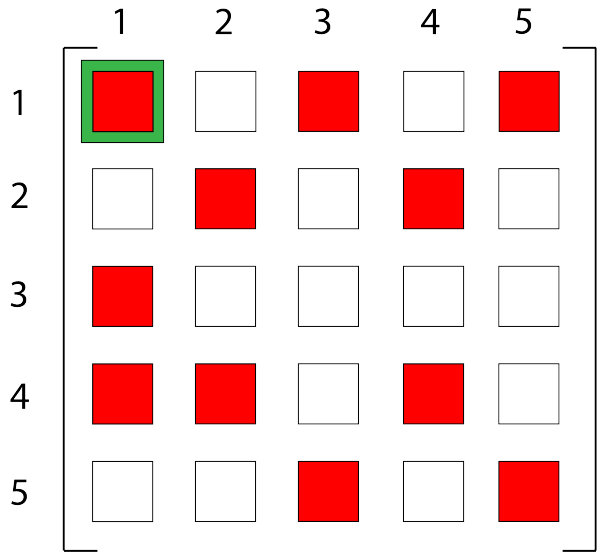
Revisiting the formula for calculating L and U

The formula for calculating L and U as mentioned in section 2.1 is given below:-

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

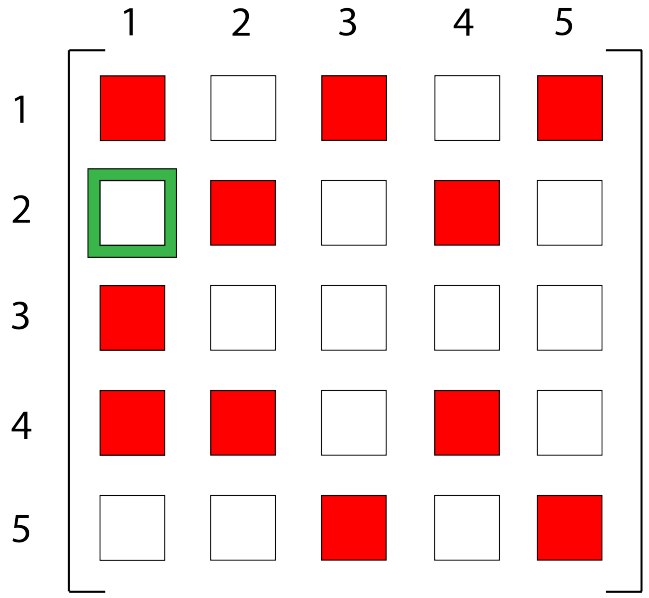
Starting with actual symbolic analysis

Column 1:-



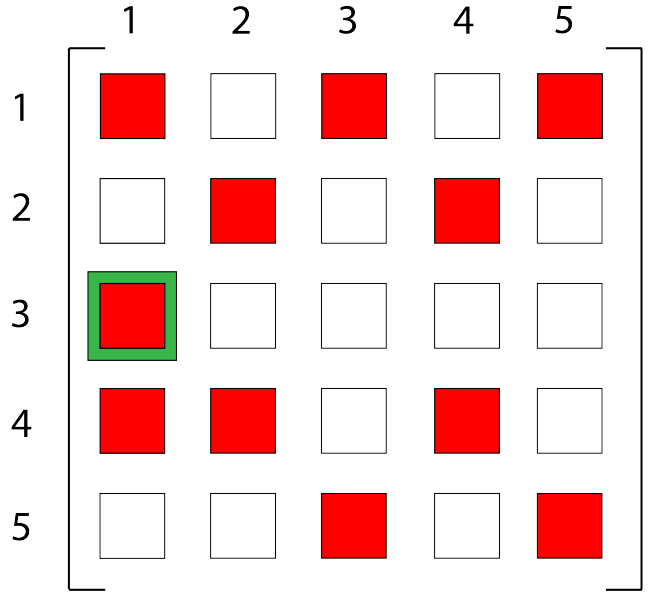
$$U_{1,1} = A_{1,1}$$

by Anurag Choudhury
Email Id: anuragn85@gmail.com



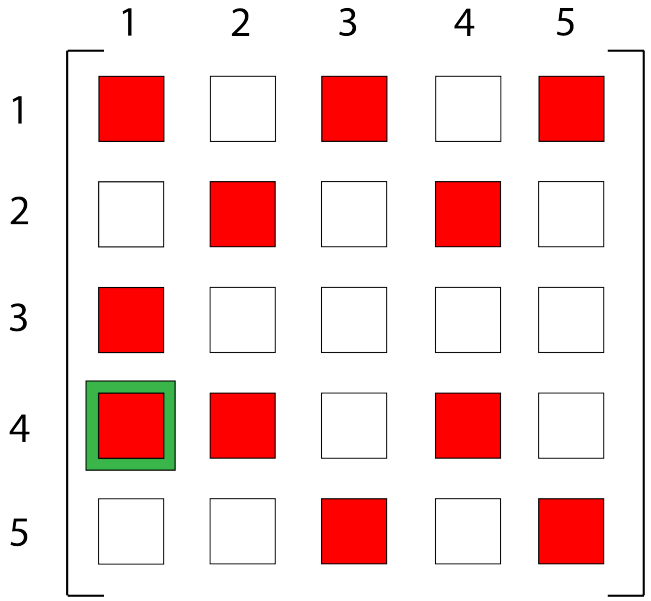
$$L_{2,1} = A_{2,1}/U_{1,1} = 0$$

Since $A_{2,1}$ is 0, the entire $L_{2,1}$ equation will be neglected

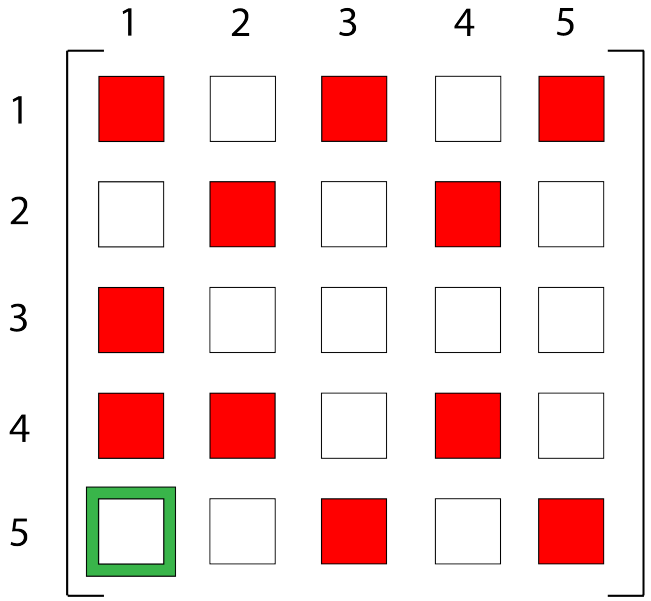


$$L_{3,1} = A_{3,1}/U_{1,1}$$

by Anurag Choudhury
Email Id: anuragn85@gmail.com



$$L_{4,1} = A_{4,1}/U_{1,1}$$

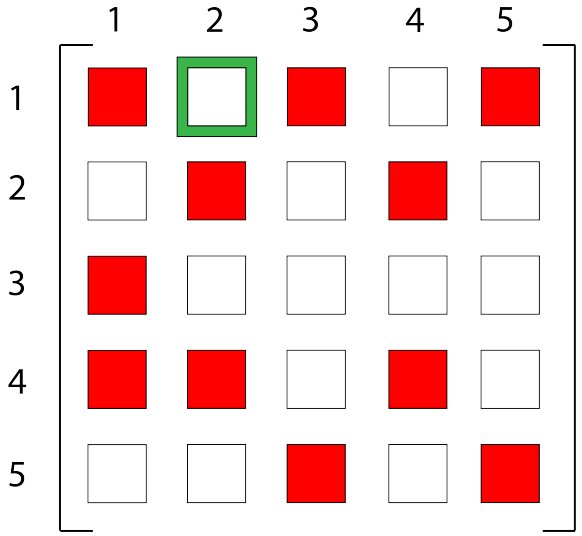


$$L_{5,1} = A_{5,1}/U_{1,1} = 0$$

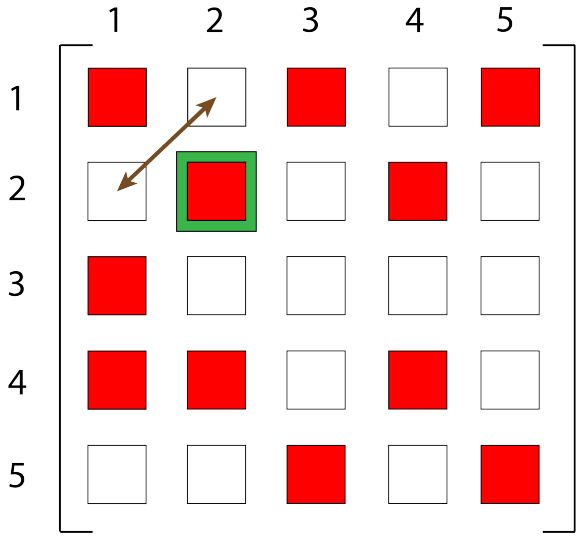
Since $A_{5,1}$ is 0, the entire $L_{5,1}$ equation will be neglected.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Column 2:-

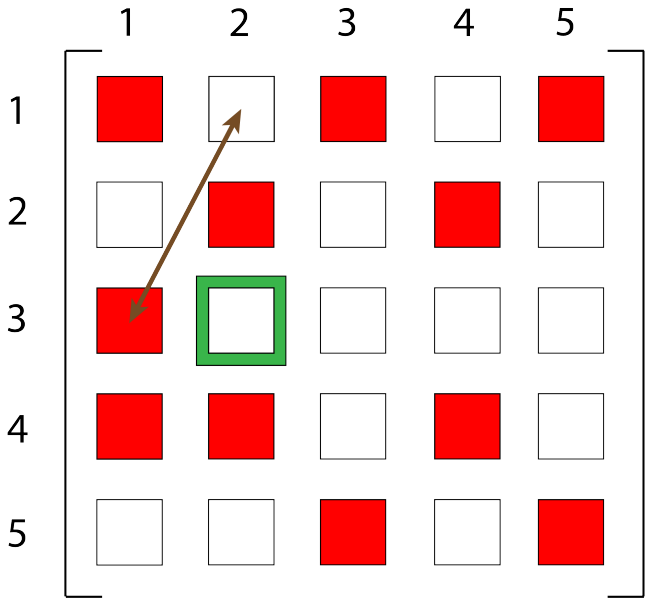


$U_{1,2} = A_{1,2} = 0$
Since $A_{1,2}$ is 0, $U_{1,2}$ will be neglected.



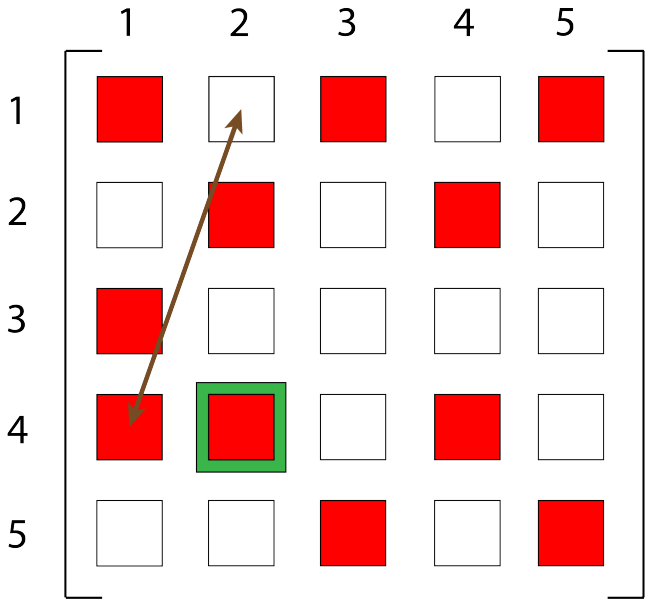
$U_{2,2} = A_{2,2} - L_{2,1}*U_{1,2} = A_{2,2}$
Since both $L_{2,1}$ and $U_{1,2}$ terms are 0, the multiplication operation can be neglected.

by Anurag Choudhury
Email Id: anuragn85@gmail.com



$L_{3,2} = (A_{3,2} - L_{3,1}*U_{1,2})/U_{2,2} = 0$

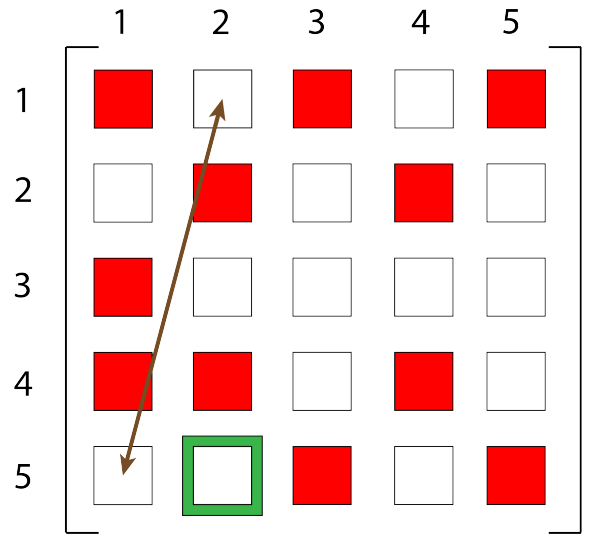
Since $A_{3,2}$ and $U_{1,2}$ are 0, the entire $L_{3,2}$ equation is neglected.



$L_{4,2} = (A_{4,2} - L_{4,1}*U_{1,2})/U_{2,2} = A_{4,2}/U_{2,2}$

Since $U_{1,2}$ is 0, the multiplication operation can be neglected.

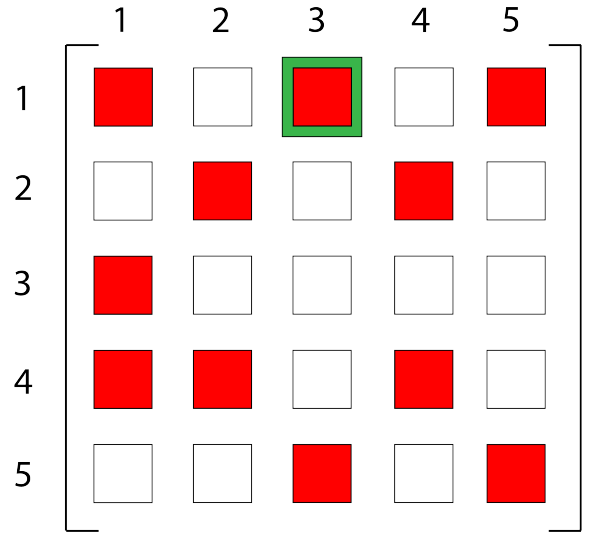
by Anurag Choudhury
Email Id: anuragn85@gmail.com



$L_{5,2} = (A_{5,2} - L_{5,1}*U_{1,2})/U_{2,2} = 0$

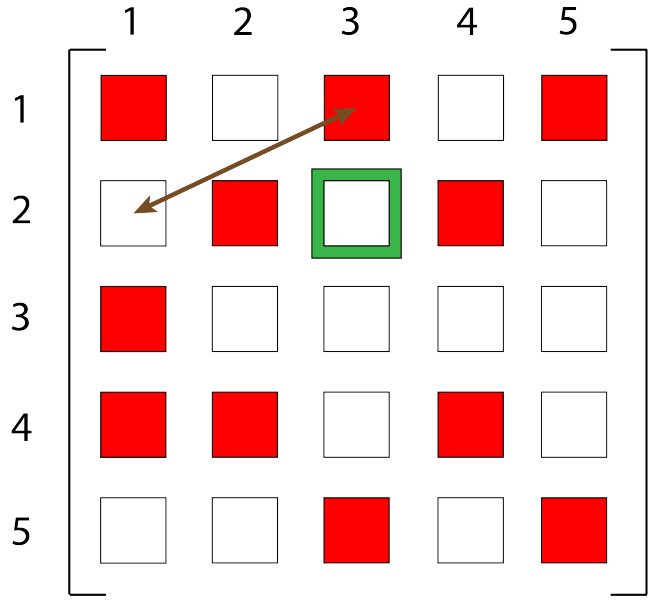
Since $A_{5,2}$ and$U_{1,2}$ are 0, the entire $L_{5,2}$ equation is neglected.

Column 3:-

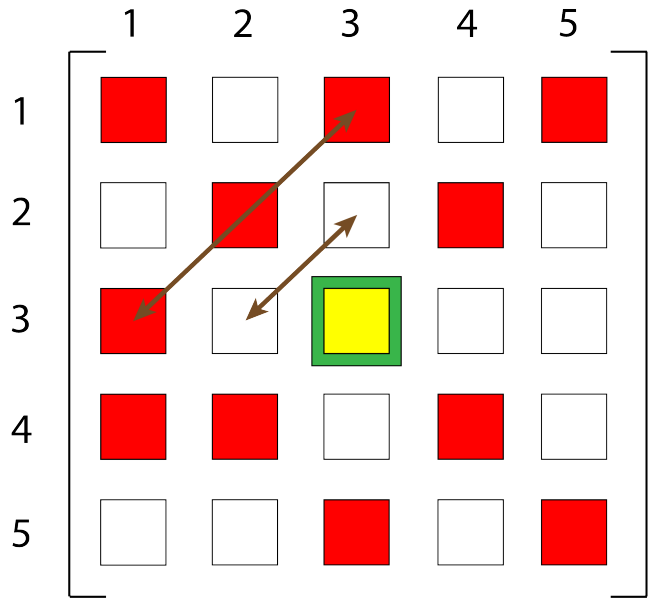


$U_{1,3} = A_{1,3}$

by Anurag Choudhury
Email Id: anuragn85@gmail.com

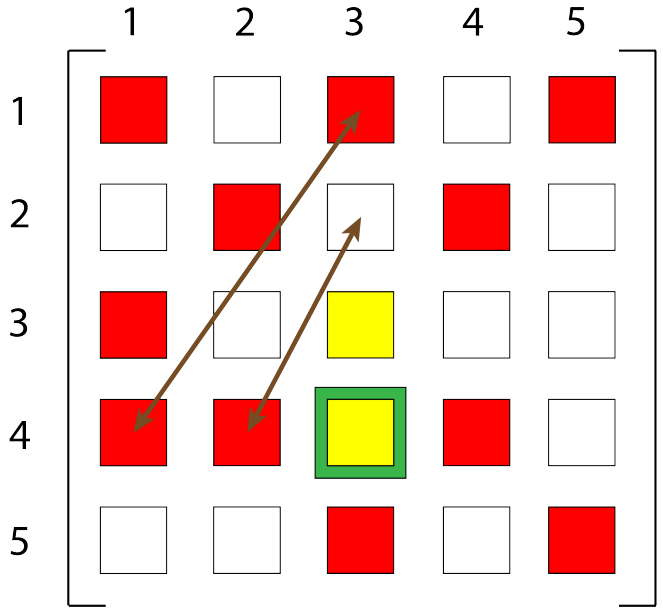


$U_{2,3} = A_{2,3} - L_{2,1}*U_{1,3} = 0$
Since $A_{2,3}$ and $L_{2,1} = 0$, the entire $U_{2,3}$ equation can be neglected.

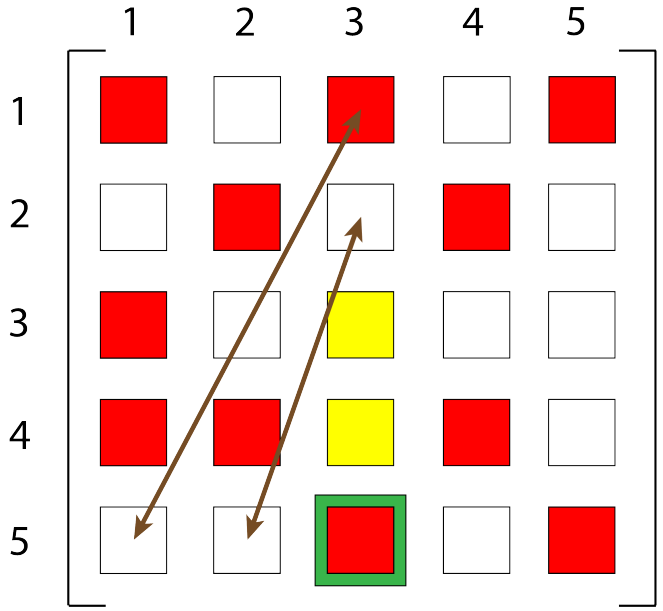


$U_{3,3} = A_{3,3} - L_{3,1}*U_{1,3} - L_{3,2}*U_{2,3} = - L_{3,1}*U_{1,3}$
Since both $L_{3,2}$ and $U_{2,3}$ are zero, the second multiplication operation is neglected.

by Anurag Choudhury
Email Id: anuragn85@gmail.com



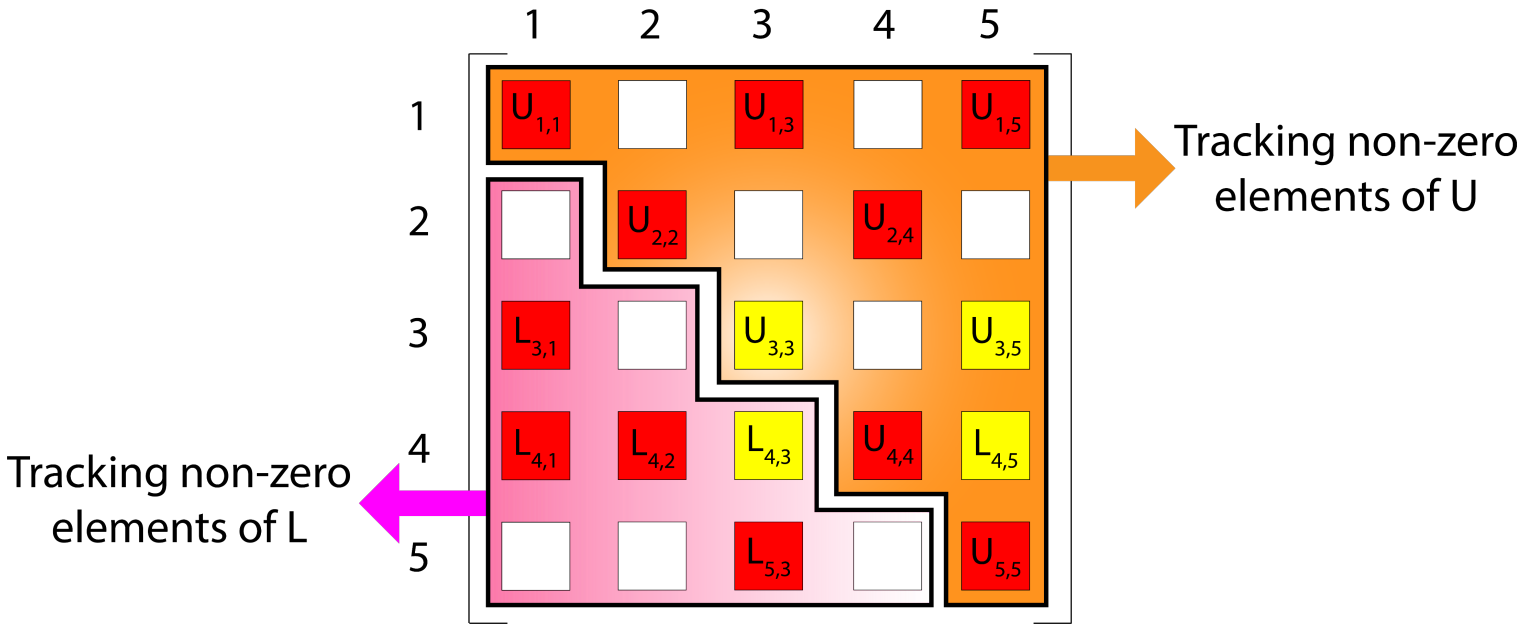
$L_{4,3} = (A_{4,3} - L_{4,1}*U_{1,3} - L_{4,2}*U_{2,3})/U_{3,3} = (- L_{4,1}*U_{1,3})/U_{3,3}$
Since $U_{2,3}$ is zero, the second multiplication operation is neglected.



$L_{5,3} = (A_{5,3} - L_{5,1}*U_{1,3} - L_{5,2}*U_{2,3})/U_{3,3} = A_{5,3}/U_{3,3}$
Since $L_{5,1}$ is 0 the 1^{st} multiplication operation is neglected.
Since both $L_{5,2}$ and $U_{2,3}$

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Same procedure can be followed for 4 and 5. The final *fill-in-tracking* matrix is show below:-



From the above matrix it's clear that 4 additional elements have become non-zero.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Col. No.	Equations before symbolic analysis	MAC ops	DIV ops	Equations after symbolic analysis	MAC ops	DIV ops
1	$U_{1,1} = A_{1,1}$	0	0	$U_{1,1} = A_{1,1}$	0	0
	$L_{2,1} = A_{2,1}/U_{1,1}$	0	1	-	0	0
	$L_{3,1} = A_{3,1}/U_{1,1}$	0	1	$L_{3,1} = A_{3,1}/U_{1,1}$	0	1
	$L_{4,1} = A_{4,1}/U_{1,1}$	0	1	$L_{4,1} = A_{4,1}/U_{1,1}$	0	1
	$L_{5,1} = A_{5,1}/U_{1,1}$	0	1	-	0	0
2	$U_{1,2} = A_{1,2}$	0	0	-	0	0
	$U_{2,2} = A_{2,2} - L_{2,1}*U_{1,2}$	1	0	$U_{2,2} = A_{2,2}$	0	0
	$L_{3,2} = (A_{3,2} - L_{3,1}*U_{1,2})/U_{2,2}$	1	1	-	0	0
	$L_{4,2} = (A_{4,2} - L_{4,1}*U_{1,2})/U_{2,2}$	1	1	$L_{4,2} = A_{4,2}/U_{2,2}$	0	1
	$L_{5,2} = (A_{5,2} - L_{5,1}*U_{1,2})/U_{2,2}$	1	1	-	0	0
3	$U_{1,3} = A_{1,3}$	0	0	$U_{1,3} = A_{1,3}$	0	0
	$U_{2,3} = A_{2,3} - L_{2,1}*U_{1,3}$	1	0	-	0	0
	$U_{3,3} = A_{3,3} - L_{3,1}*U_{1,3} - L_{3,2}*U_{2,3}$	2	0	$U_{3,3} = - L_{3,1}*U_{1,3}$	1	0
	$L_{4,3} = (A_{4,3} - L_{4,1}*U_{1,3} - L_{4,2}*U_{2,3})/U_{3,3}$	2	1	$L_{4,3} = (- L_{4,1}*U_{1,3})/U_{3,3}$	1	1
	$L_{5,3} = (A_{5,3} - L_{5,1}*U_{1,3} - L_{5,2}*U_{2,3})/U_{3,3}$	2	1	$L_{5,3} = A_{5,3}/U_{3,3}$	0	1
4	$U_{1,4} = A_{1,4}$	0	0	-	0	0
	$U_{2,4} = A_{2,4} - L_{2,1}*U_{1,4}$	1	0	$U_{2,4} = A_{2,4}$	0	0
	$U_{3,4} = A_{3,4} - L_{3,1}*U_{1,4} - L_{3,2}*U_{2,4}$	2	0	-	0	0
	$U_{4,4} = A_{4,4} - L_{4,1}*U_{1,4} - L_{4,2}*U_{2,4} - L_{4,3}*U_{3,4}$	3	0	$U_{4,4} = A_{4,4} - L_{4,2}*U_{2,4}$	1	0
	$L_{5,4} = (A_{5,4} - L_{5,1}*U_{1,4} - L_{5,2}*U_{2,4} - L_{5,3}*U_{3,4})/U_{4,4}$	3	1	-	0	0
5	$U_{1,5} = A_{1,5}$	0	0	$U_{1,5} = A_{1,5}$	0	0
	$U_{2,5} = A_{2,5} - L_{2,1}*U_{1,5}$	1	0	-	0	0
	$U_{3,5} = A_{3,5} - L_{3,1}*U_{1,5} - L_{3,2}*U_{2,5}$	2	0	$U_{3,5} = - L_{3,1}*U_{1,5}$	1	0
	$U_{4,5} = A_{4,5} - L_{4,1}*U_{1,5} - L_{4,2}*U_{2,5} - L_{4,3}*U_{3,5}$	3	0	$U_{4,5} = - L_{4,1}*U_{1,5} - L_{4,3}*U_{3,5}$	2	0
	$U_{5,5} = A_{5,5} - L_{5,1}*U_{1,5} - L_{5,2}*U_{2,5} - L_{5,3}*U_{3,5}$	3	0	$U_{5,5} = A_{5,5} - L_{5,3}*U_{3,5}$	1	0
Total		**29**	**10**	**Total**	**7**	**5**

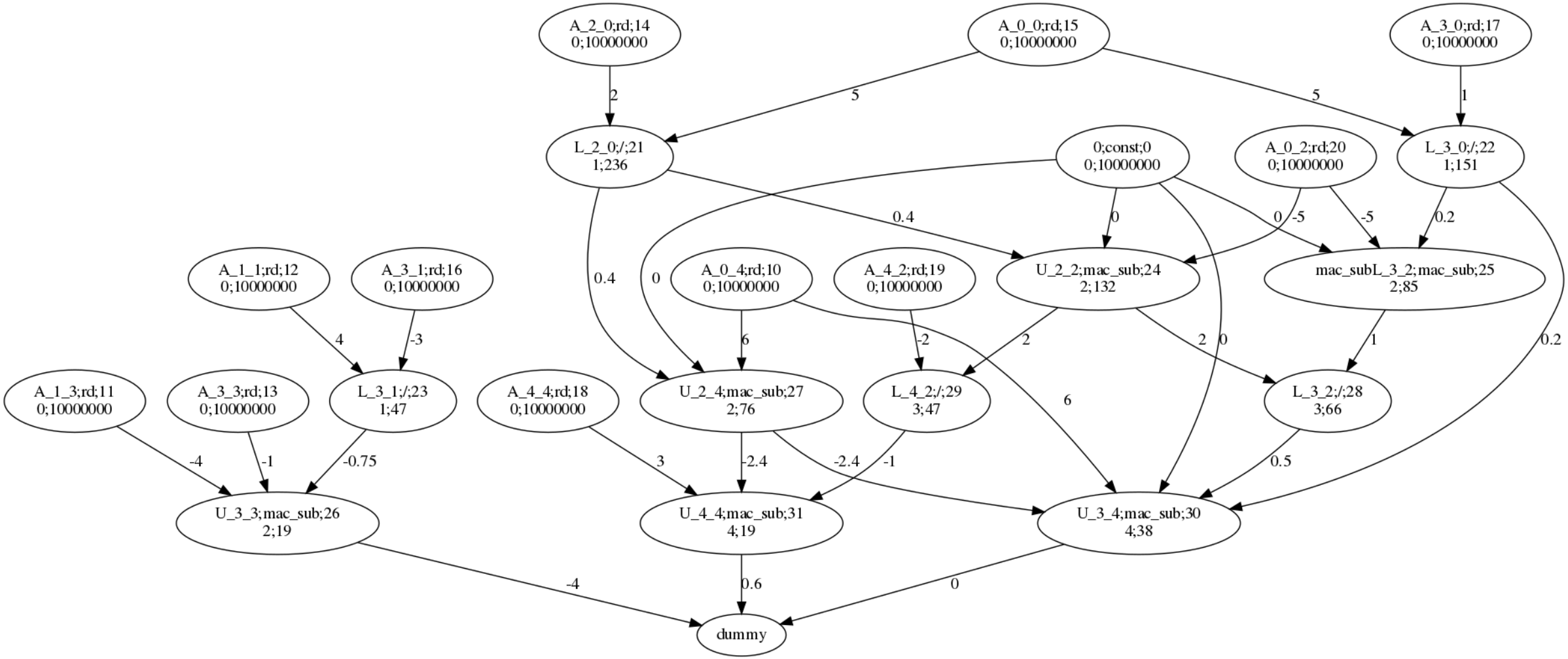
From the previous table it's clear that the number of MAC operations reduced by almost 76% and number of DIV operations reduced by 50%.
It's also important to note that the set of equations obtained after symbolic analysis is only valid for those matrices whose non-zero locations is same as the non-zero locations of A matrix

Data Flow Graph(DFG) generation

DFG can be generated using the equations obtained after symbolic analysis. The DFG is shown in the next slide.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

The generated DFG



by Anurag Choudhury
Email Id: anuragn85@gmail.com

6.2 Assigning priority to each node

For assigning priority, we start with the root nodes and mode towards the leaf nodes. The priority calculation formula for DIV("/") node is:-

$$\text{"/" node priority} = \sum \textit{Child Node priority} + \textit{DIV latency}$$

The priority calculation formula for MAC("mac_sub") node is:-

$$\text{MAC node priority} = \sum \textit{Child Node priority} + \textit{No. of MAC operations} * \textit{MAC latency}$$

For the Data Flow Graph in the previous slide, the **MAC latency is 19 cycles** and **DIV latency is 28 cycles**

by Anurag Choudhury
Email Id: anuragn85@gmail.com

6.3 Assigning memory location to each DFG node

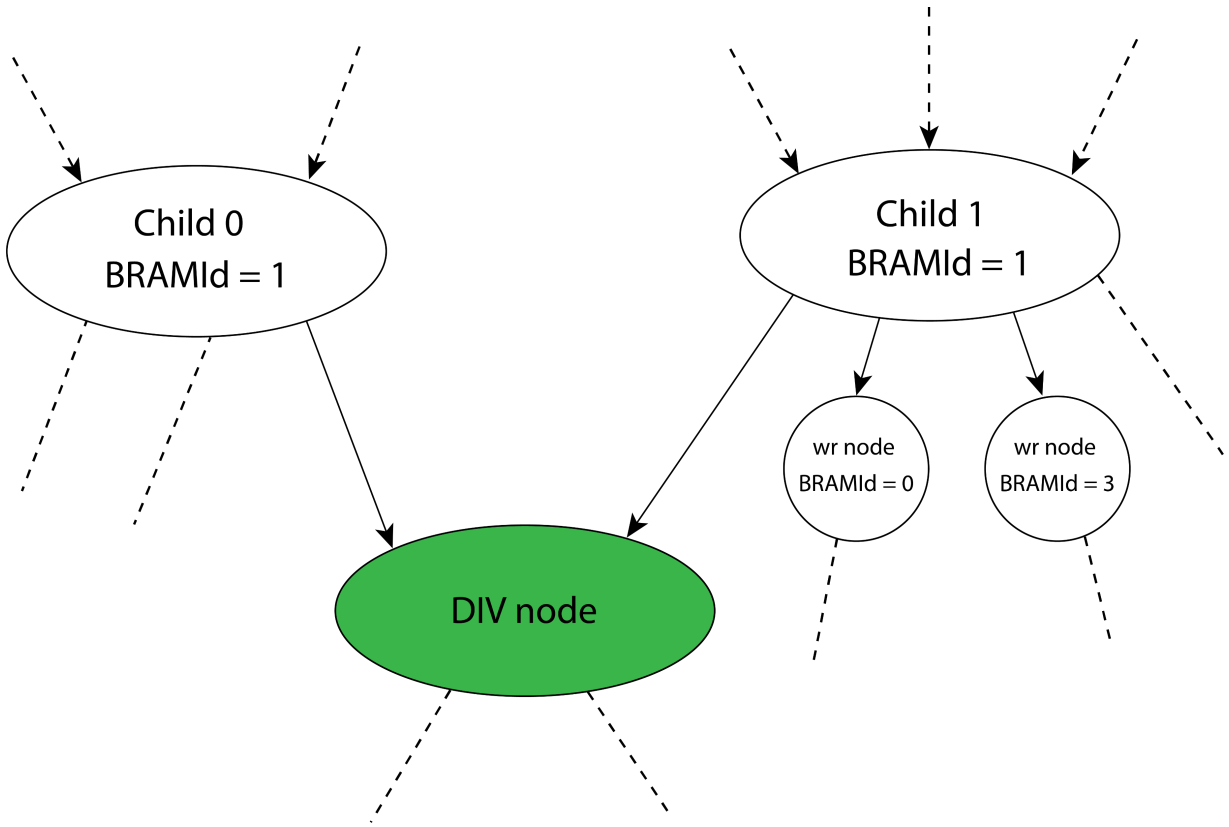
The memory locations have been assigned randomly to each graph node. While assigning the memory locations, it's ensured that all the BRAMs are equally utilized.

6.4 Adding *wr* (memory write) nodes

These nodes are used to move data from one BRAM to another.
If we use Single Port BRAMs, then ***wr*** nodes might have to be added before both MAC and DIV nodes.
If we use Dual port BRAMs, then ***wr*** nodes might have to be added only before MAC nodes.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

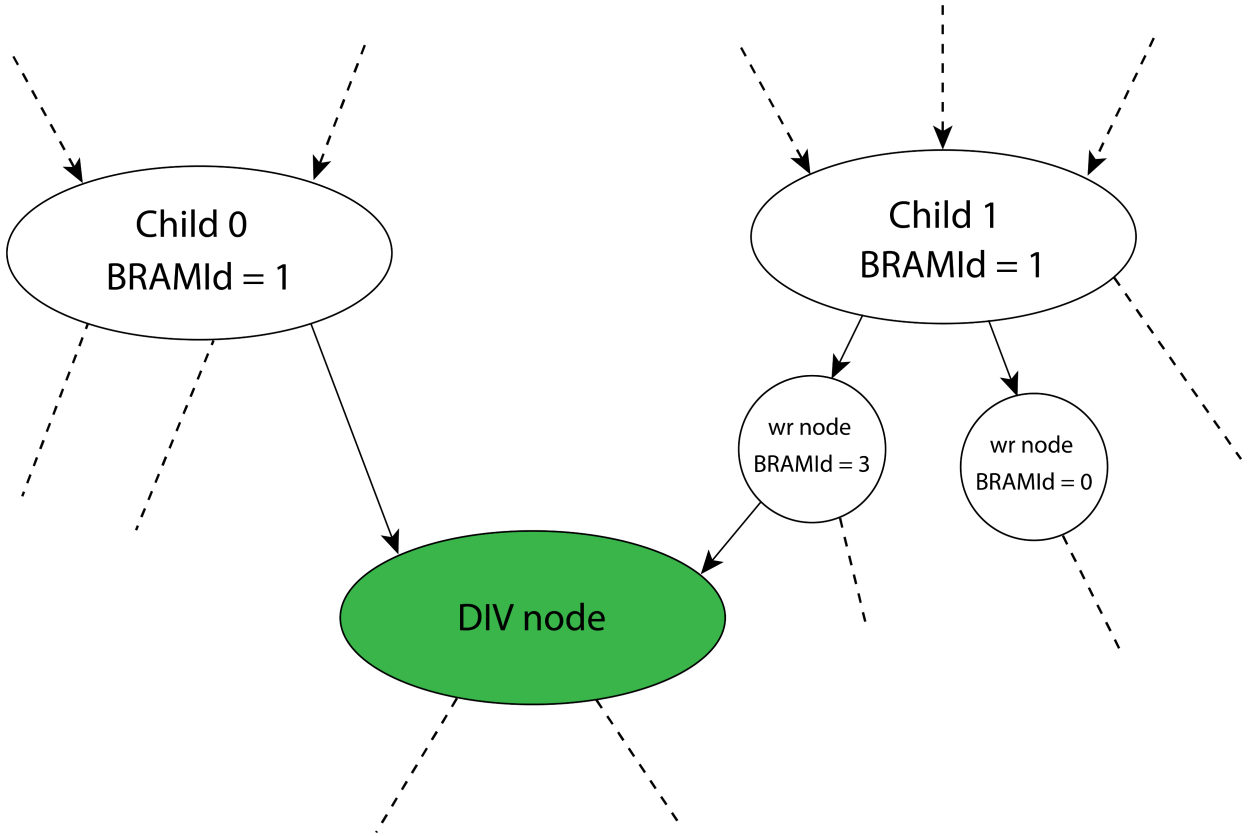
Assume that we are using **4 Single Port BRAMs**. Consider the DIV node shown below:-



Possible candidates for new BRAM location are BRAM Id **0, 2 and 3.**
So a ***wr*** node with any of the above BRAM Id can be added after any 1 of the children.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

As ***wr*** node with BRAM Id 3 and 0 already exist for Child 1, we do not add a new ***wr*** node. The new node connections are as follows:-



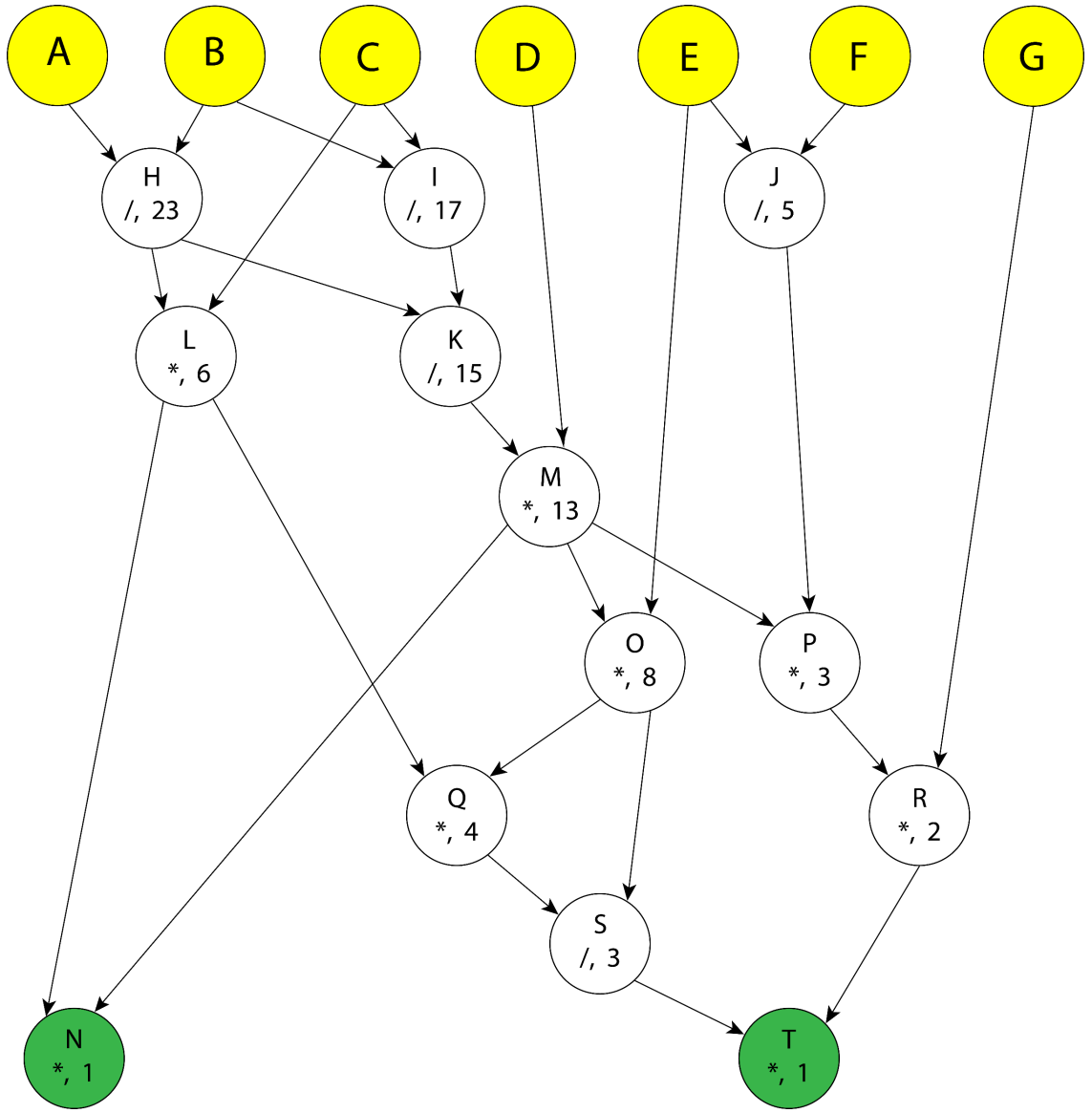
by Anurag Choudhury
Email Id: anuragn85@gmail.com

7. Static Schedule Generation Stage

The scheduling algorithm used for this project is ***list scheduling***. The basic idea behind list scheduling is that operations are scheduled continuously based on availability of operands and resources (number of adders, multipliers etc.).

In the list scheduling example explained in the next few slides, I have considered **2 MUL units(latency = 1)** and **2 DIV units(latency = 2)** which are **pipelined.**

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 0

initial schedulable_list = | H | I | J |

execution_list = | H | I |

cyclesLeft_list = | 2 | 2 |

updated schedulable_list = | J |

Time = 1

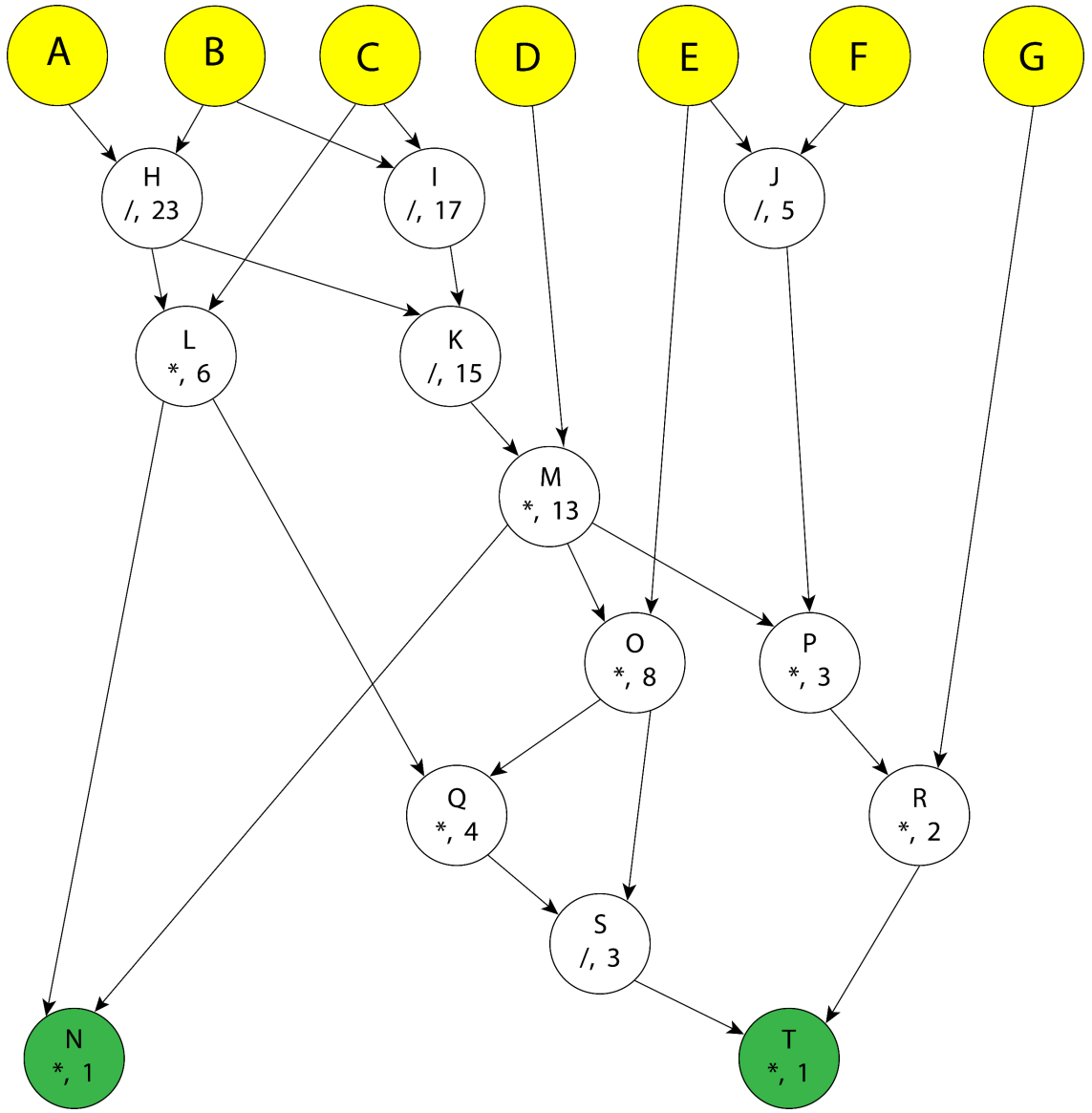
initial schedulable_list = | J |

execution_list = | H | I | J |

cyclesLeft_list = | 1 | 1 | 2 |

updated schedulable_list = Null

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 2

initial schedulable_list = | L | K |

execution_list =

cyclesLeft_list =

J	L	K
1	1	2

updated schedulable_list = Null

Time = 3

initial schedulable_list = Null

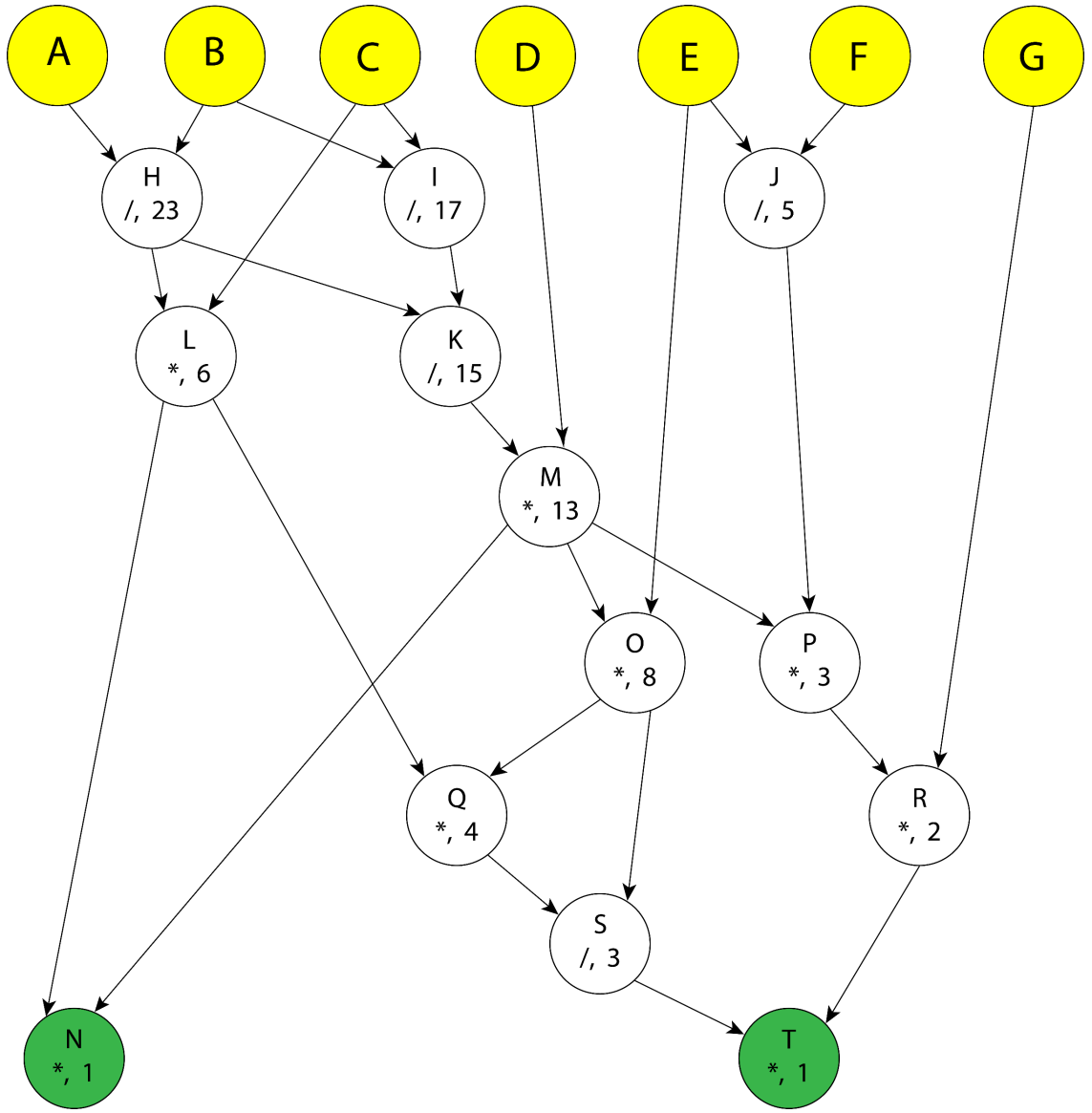
execution_list =

cyclesLeft_list =

K
1

updated schedulable_list = Null

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 4

initial schedulable_list = | M |

execution_list = | M |

cyclesLeft_list = | 1 |

updated schedulable_list = Null

Time = 5

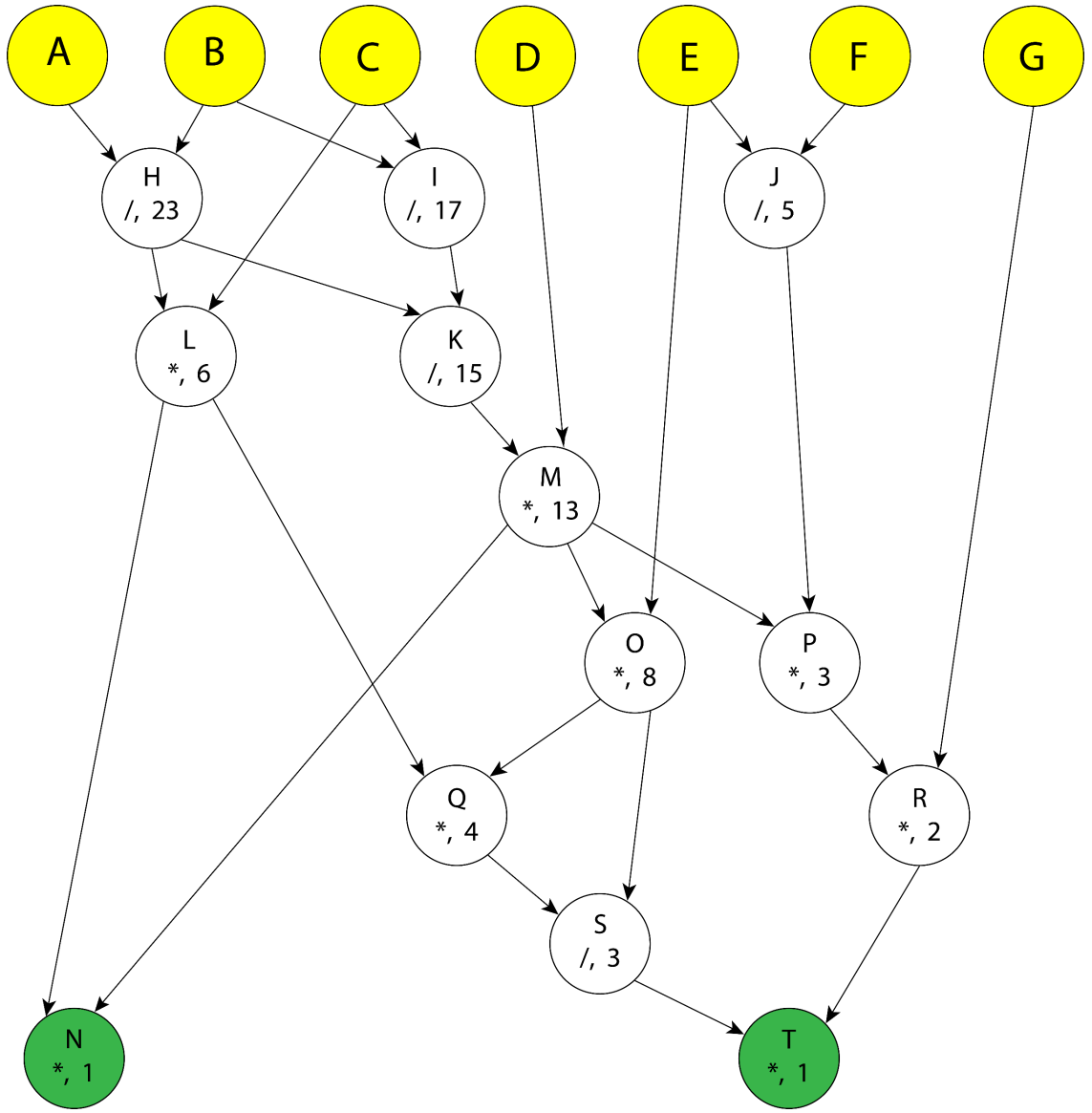
initial schedulable_list = | N | O | P |

execution_list = | O | P |

cyclesLeft_list = | 1 | 1 |

updated schedulable_list = | N |

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 6

initial schedulable_list = | N | Q | R |

execution_list = | Q | R |

cyclesLeft_list = | 1 | 1 |

updated schedulable_list = | N |

Time = 7

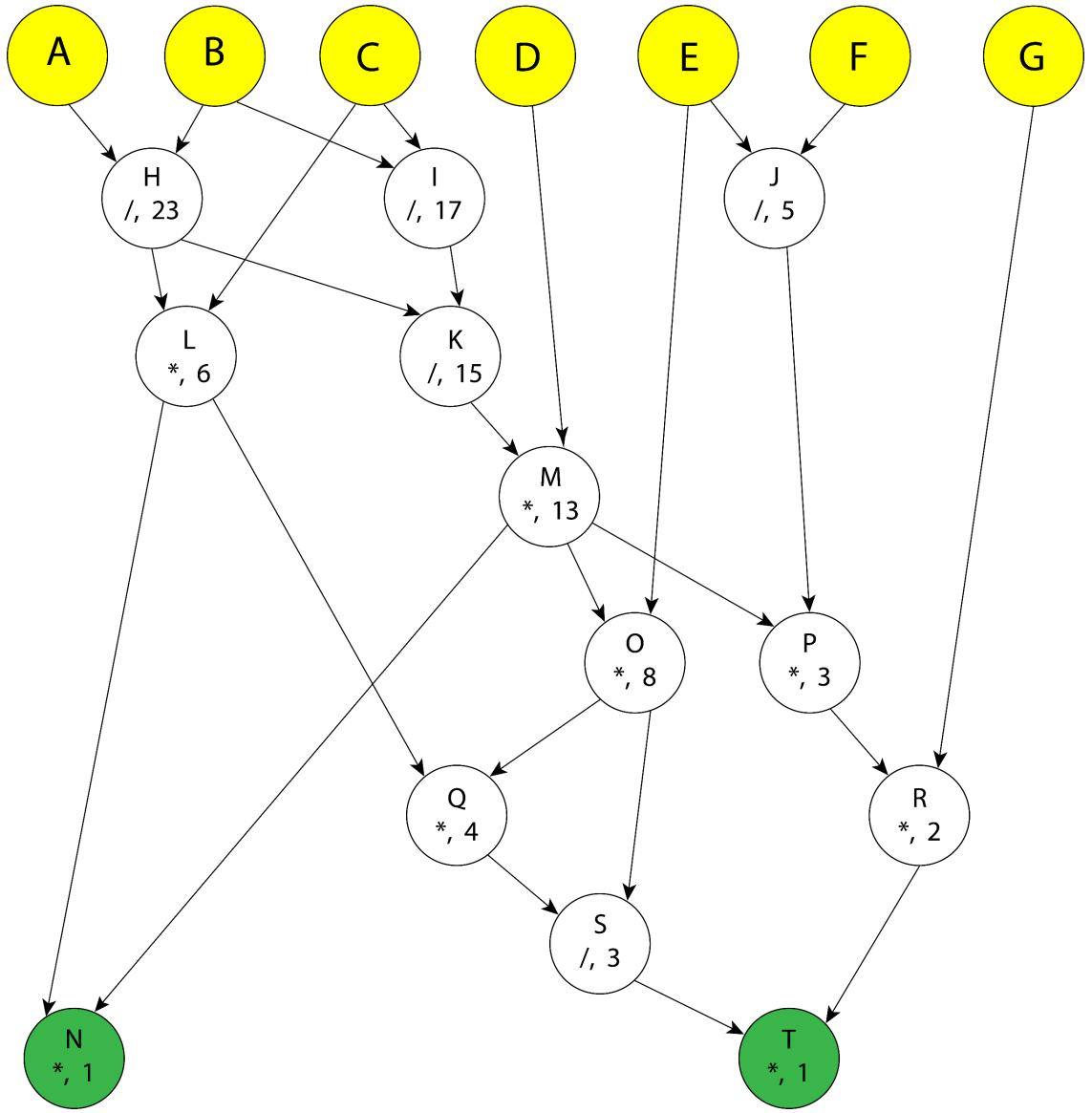
initial schedulable_list = | N | S |

execution_list = | N | S |

cyclesLeft_list = | 1 | 2 |

updated schedulable_list = Null

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 8

initial schedulable_list = Null

execution_list = | S |

cyclesLeft_list = | 1 |

updated schedulable_list = Null

Time = 9

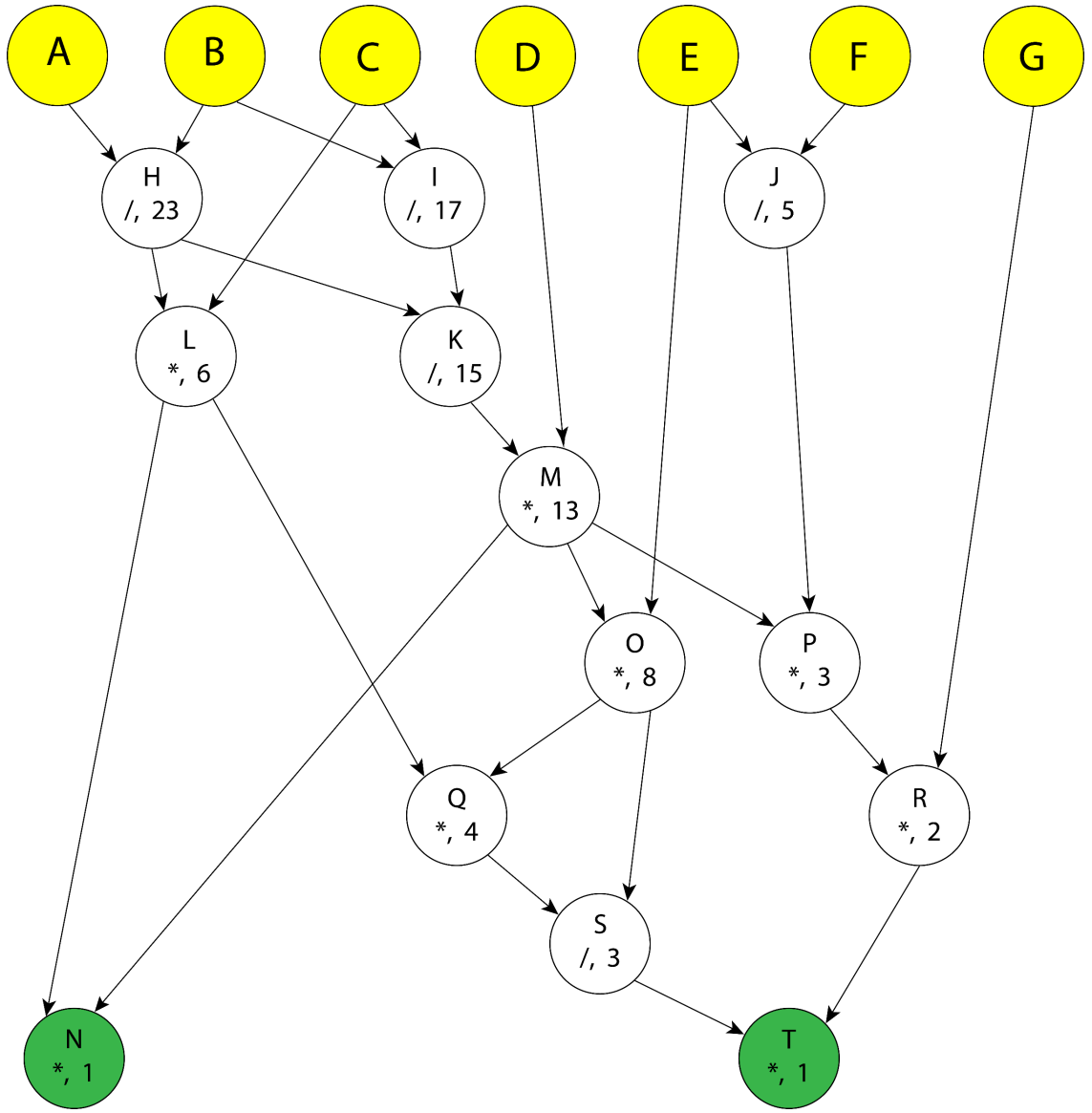
initial schedulable_list = | T |

execution_list = | T |

cyclesLeft_list = | 1 |

updated schedulable_list = Null

by Anurag Choudhury
Email Id: anuragn85@gmail.com



Time = 10

initial schedulable_list = Null

execution_list = Null

cyclesLeft_list = Null

updated schedulable_list = Null

by Anurag Choudhury
Email Id: anuragn85@gmail.com

The concept behind list scheduling used in this project is same as the above example with **2 main differences**: -

1. The operands have to be fetched from memory, and the results have to be stored in memory.
2. The MAC type nodes used in this project can internally have multiple MAC operations. So, whichever MAC operation is ready in a MAC node, will be scheduled.

The scheduler implemented in this project takes the following inputs: -

1. Number of BRAM blocks
2. Read and write latency of BRAM blocks
3. Number of ports per BRAM block
4. Number of MAC and DIV units
5. Latencies of MAC and DIV units

by Anurag Choudhury
Email Id: anuragn85@gmail.com

What are live operands?

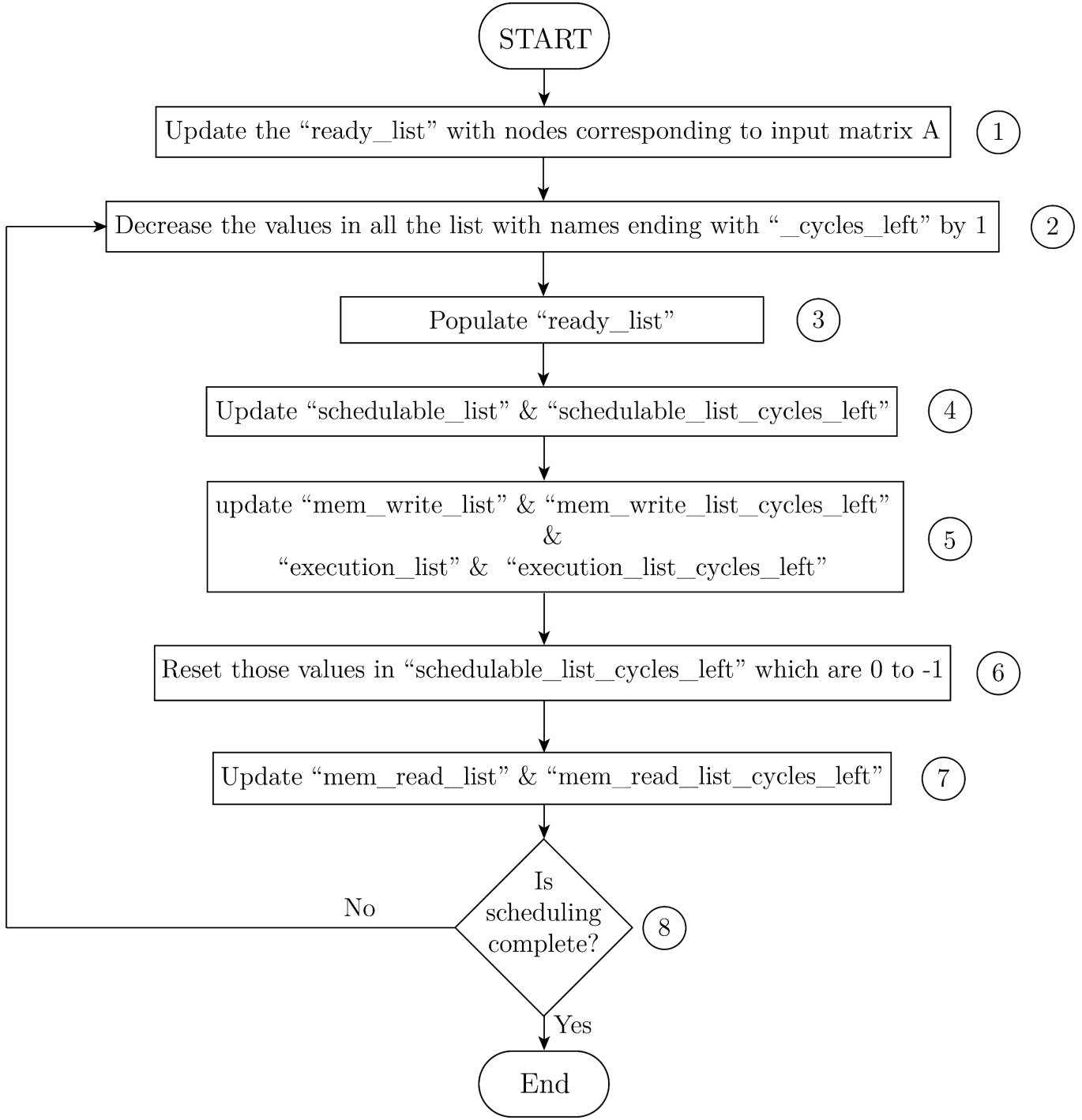
These are the operands that are present at the output of some BRAM or Arithmetic Unit and are valid for that particular cycle only.

Important Lists that have been used in Scheduling Algorithm:-

1. *ready_list*
2. *schedulable_list* and *schedulable_list_cycles_left*
3. *execution_list* and *execution_list_cycles_left*
4. *mem_read_list* and *mem_read_list_cycles_left*
5. *mem_write_list* and *mem_write_list_cycles_left*

The *ready_list* is the only list that is cleared as we move from one cycle to next. Remaining lists retain their values.

by Anurag Choudhury
Email Id: anuragn85@gmail.com



by Anurag Choudhury
Email Id: anuragn85@gmail.com

Stage 5

a) Updating *execution_list* and *execution_list_cycles_left*

- Generate the *live_list*
- Prepare a temporary list of nodes that can be put to *execution_list*
- Put those nodes from the temporary list to *execution_list* which satisfy the following conditions:-
  - Enough arithmetic units are available in the current cycle
  - Enough BRAMs will be available to store the result when the result becomes ready

b) Updating *mem_write_list* and *mem_write_list_cycles_left*

Not every calculation that has finished evaluation need to be saved in memory.

Eg. In case of MAC nodes, the intermediate result is not saved in memory if it's possible to put the next MAC operation from the same MAC node in *execution_list*.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Stage 7:-

a) Updating *mem_read_list* and *mem_read_list_cycles_left*

In this stage we fetch operands so that new nodes can be scheduled, memory read latency cycles later.

First priority is given to scheduling those new nodes whose one of the operands will finish execution memory read latency cycles later. This is done because that operand will be there in *live_list* without having to be read from memory.

Second priority is given to scheduling those new nodes which are present in *schedulable_list*.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Three important conditions are checked before we can start putting the nodes in *mem_read_list*:-

- There should be enough BRAM ports available in the current cycle to fetch the operands from memory
- There should be enough arithmetic units available memory_read latency cycles after the current cycle so that the node that is planned to be scheduled can be put in execution_list
- Enough BRAM ports should be available when the node that is planned to be scheduled finishes execution so that it can be saved in memory

by Anurag Choudhury
Email Id: anuragn85@gmail.com

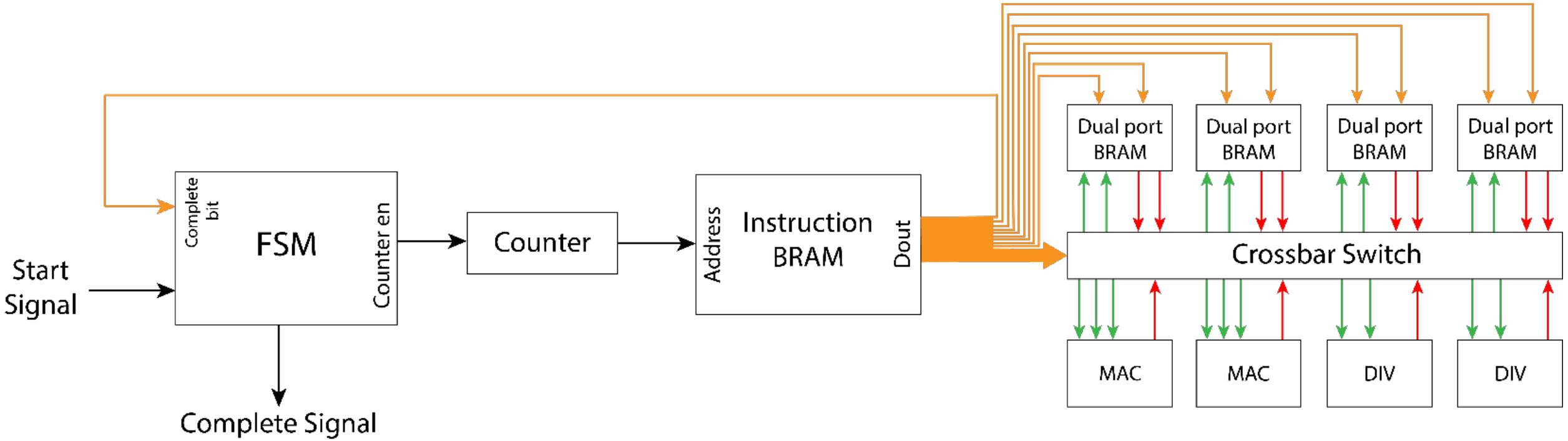
8. Hardware Implementation Stage

Two major problems I faced during implementation in FGPA:-

1. If clock gating is used, FPGA synthesis will most likely fail in meeting timing constraints. So, if we want to use clock gating, we have to specify false paths to the synthesis tool.
2. If we are using the Zynq Processing System to generate 2 clocks, the phase between the two clocks cannot be guaranteed.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

LU decomposition Crossbar Network Block Diagram

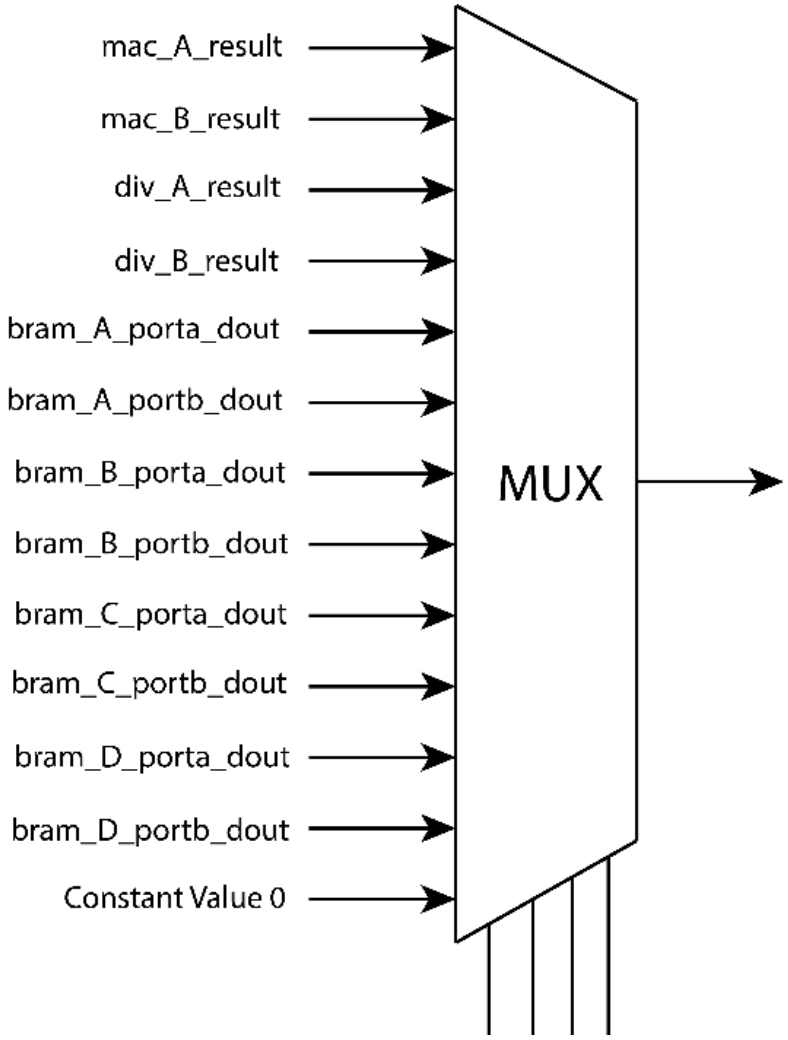


Each word in the Instruction BRAM contain 3 types of data:-

1. The complete bit
2. BRAM addresses
3. MUX select pin values

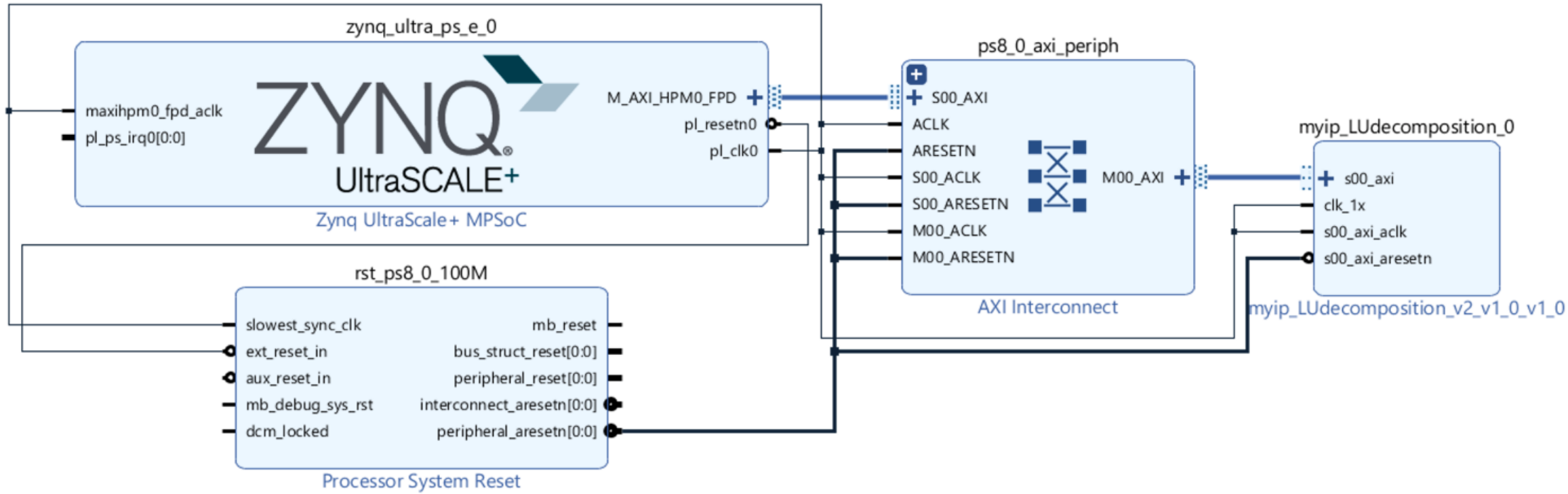
by Anurag Choudhury
Email Id: anuragn85@gmail.com

The MUX used at the input of BRAM and Arithmetic Unit Data ports



by Anurag Choudhury
Email Id: anuragn85@gmail.com

Block diagram after Interfacing LU decomposition h/w with Zynq PS



by Anurag Choudhury
Email Id: anuragn85@gmail.com

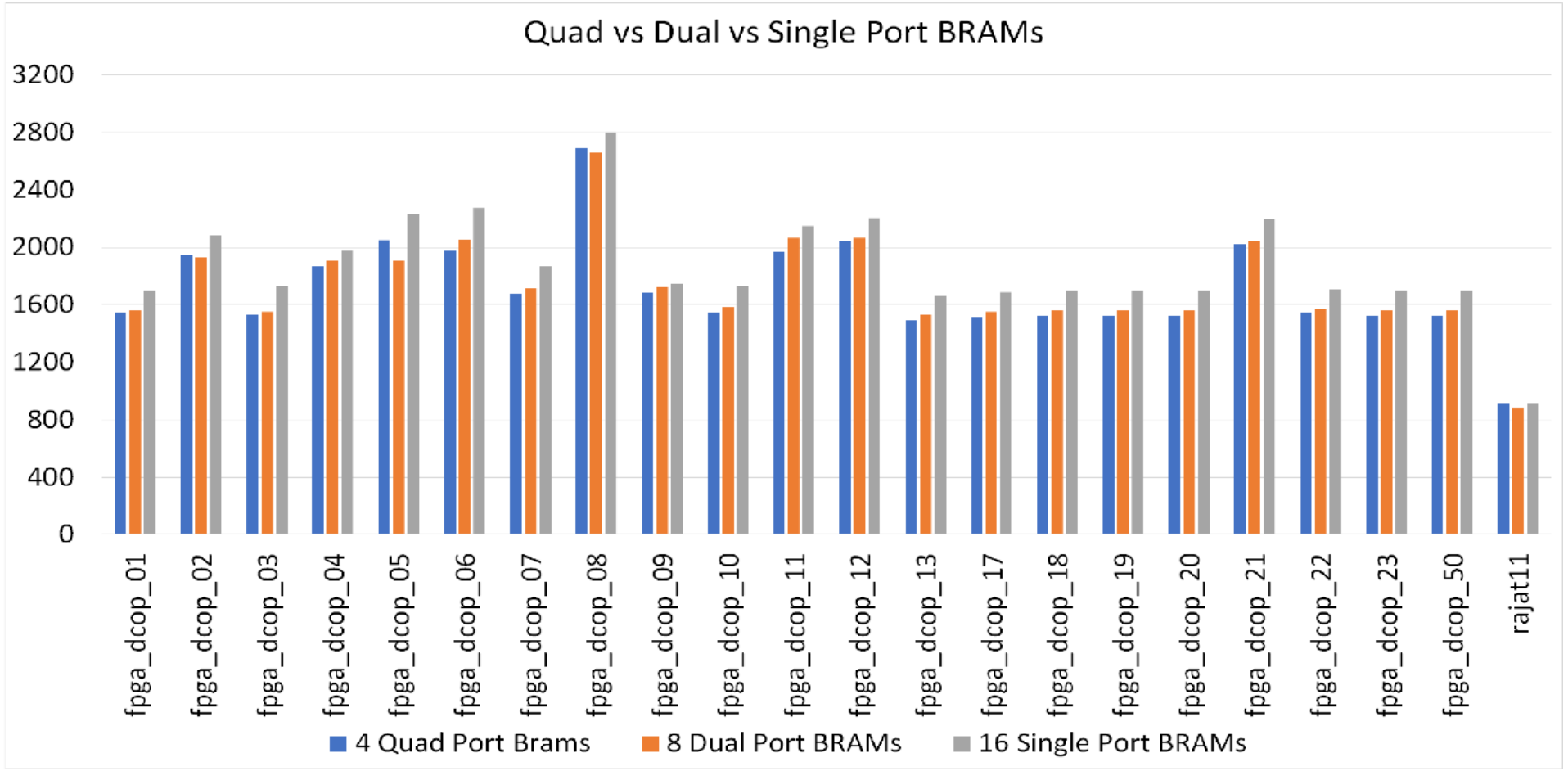
9. Results and Observations

The result is divided into 2 sections:-
1. Quad port vs Dual port vs Single port BRAMs
2. Performance comparison with Nechmas' implementation

The results discussed in section **9.1** have used **4 MACs** and **4 DIVs** along with either **4 Quad Port BRAMs**, or **8 Dual Port BRAMs** or **16 Single Port BRAMs.**

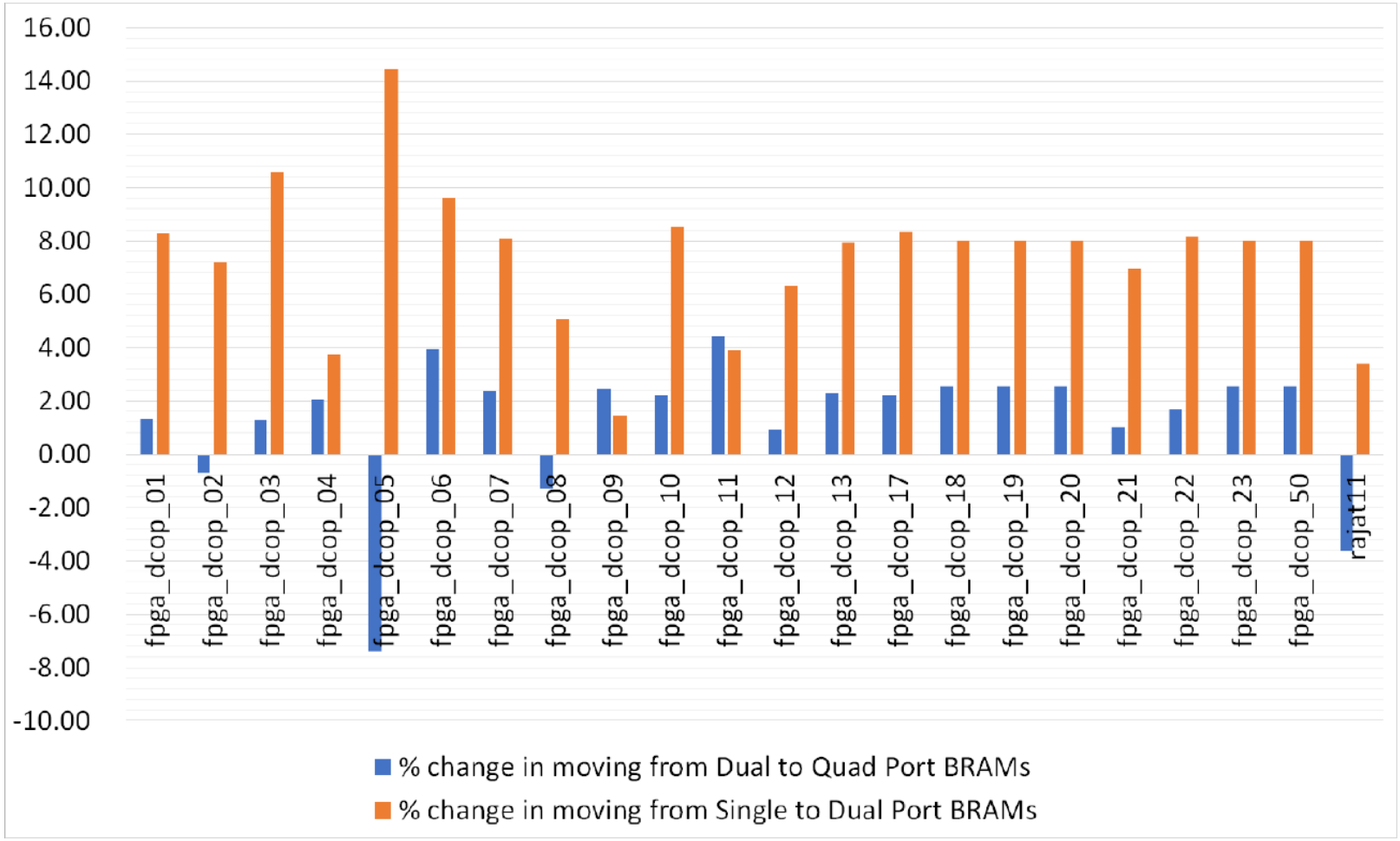
by Anurag Choudhury
Email Id: anuragn85@gmail.com

9.1 Quad Port vs Dual Port vs Single BRAMs



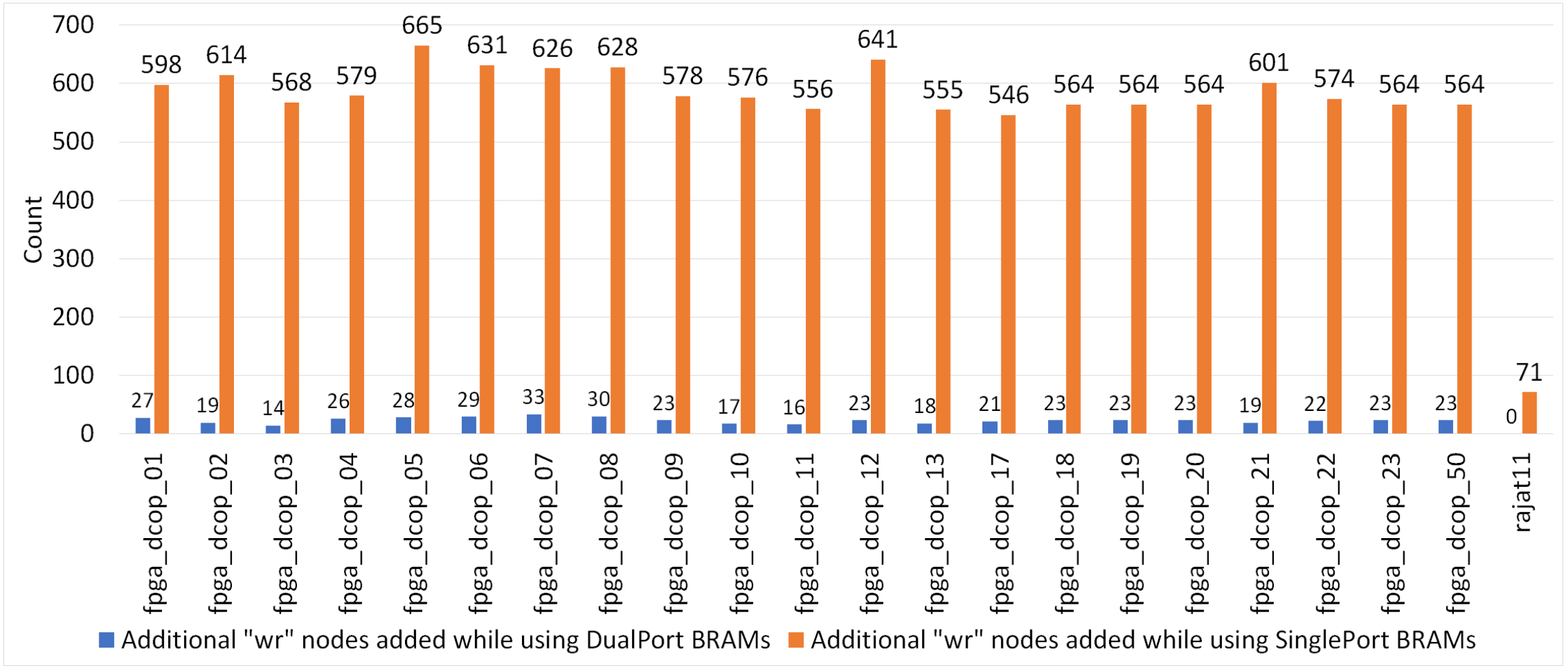
by Anurag Choudhury
Email Id: anuragn85@gmail.com

The % change in performance obtained while moving from **Single port to Dual port** and **Dual port to Quad port** is shown below: -



by Anurag Choudhury
Email Id: anuragn85@gmail.com

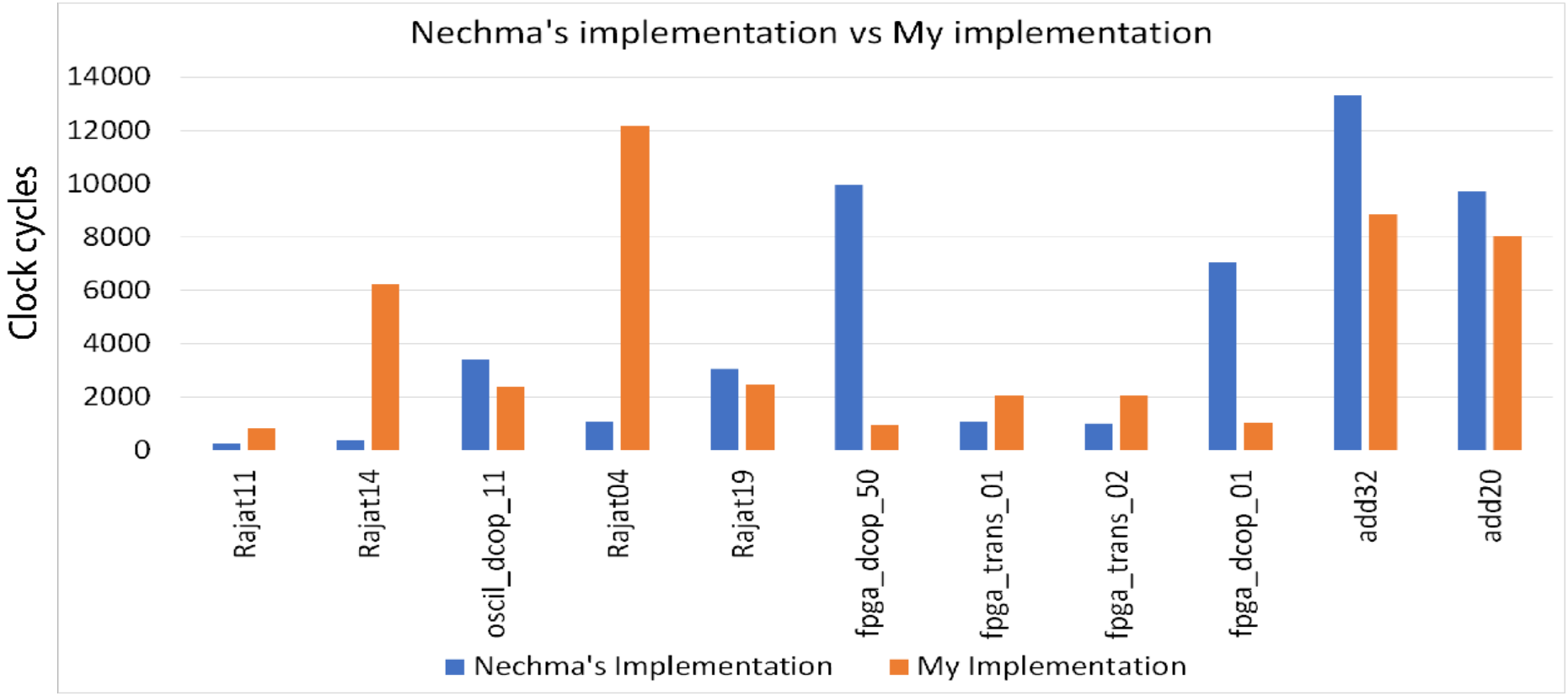
Comparison of additional ***wr*** nodes added is shown below:-



by Anurag Choudhury
Email Id: anuragn85@gmail.com

9.2 Performance comparison with Nechmas' implementation

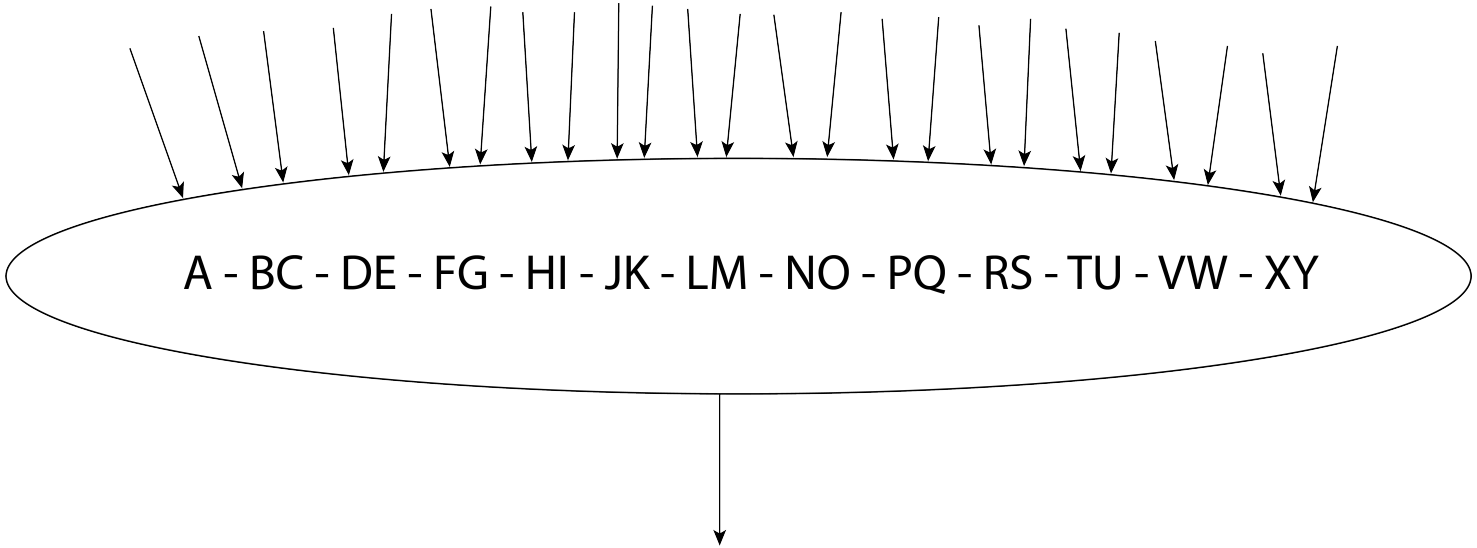
For this section I have used **16 dual port BRAMs**, **16 MACs** and **16 DIVs**. **BRAM rd/wr latency = 1**, **MAC latency = 19** and **DIV latency = 28**



by Anurag Choudhury
Email Id: anuragn85@gmail.com

Analysis of poor performance in *Rajat14* and *Rajat04* matrices

Consider the present implementation of MAC node:-

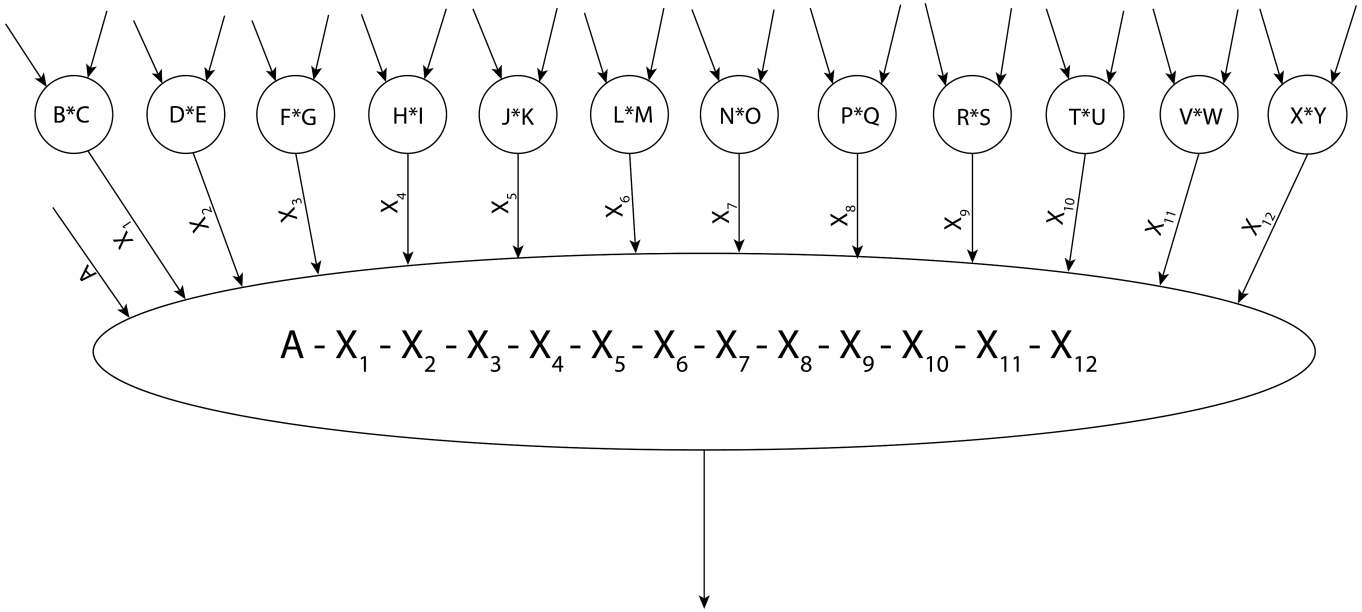


Assume MAC latency is **19 cycles**. If all the operands are available at the same time, the time taken by the above MAC node is: -

$$\textbf{Time} = \textbf{12*19} = \textbf{228 cycles}$$

by Anurag Choudhury
Email Id: anuragn85@gmail.com

Now consider the MAC node implementation where MAC operation has been divided into MUL and AND operations:-

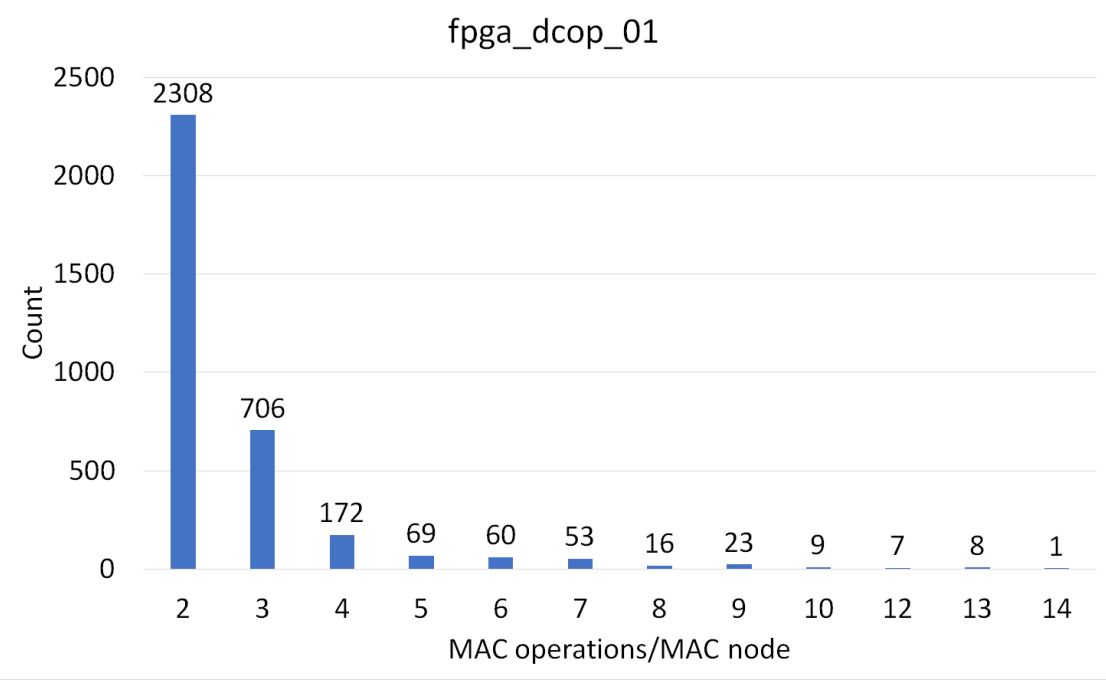


Assume **MUL** latency is **8 cycles** and **ADD** latency is **11 cycles**. If all the operands Are available at the same time, the time taken to perform the above operation will be:-

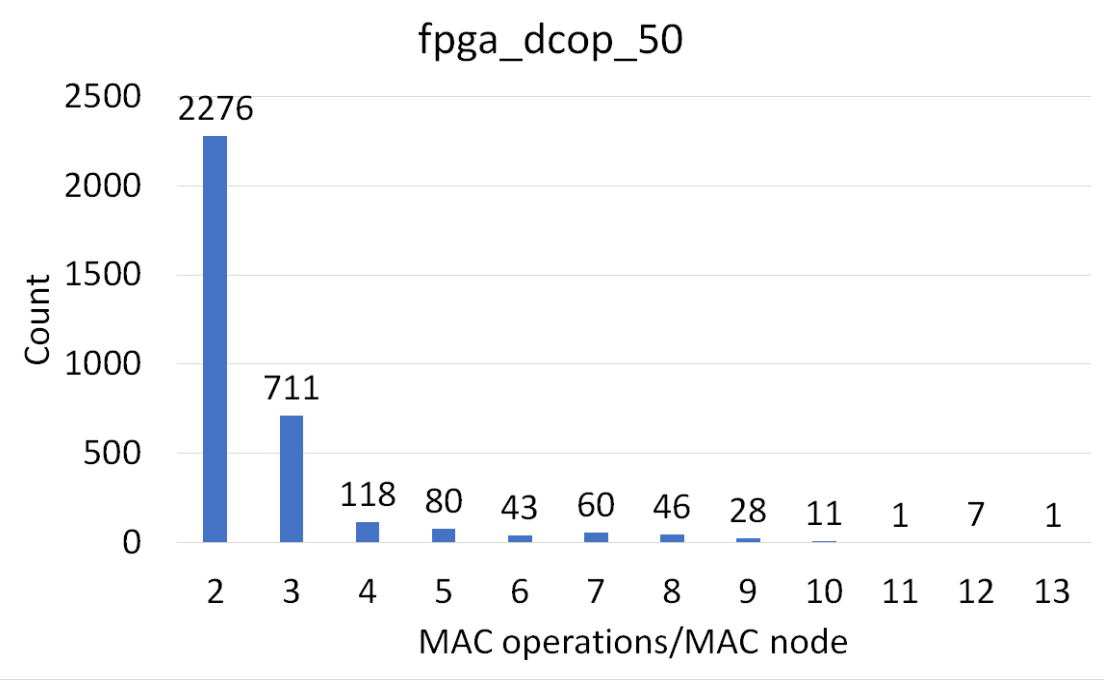
$$\textbf{Time} = 8 + 11*4 = \textbf{52 cycles}$$

by Anurag Choudhury
Email Id: anuragn85@gmail.com

MAC operations/MAC node for *fpga_dcop_01* matrix:-

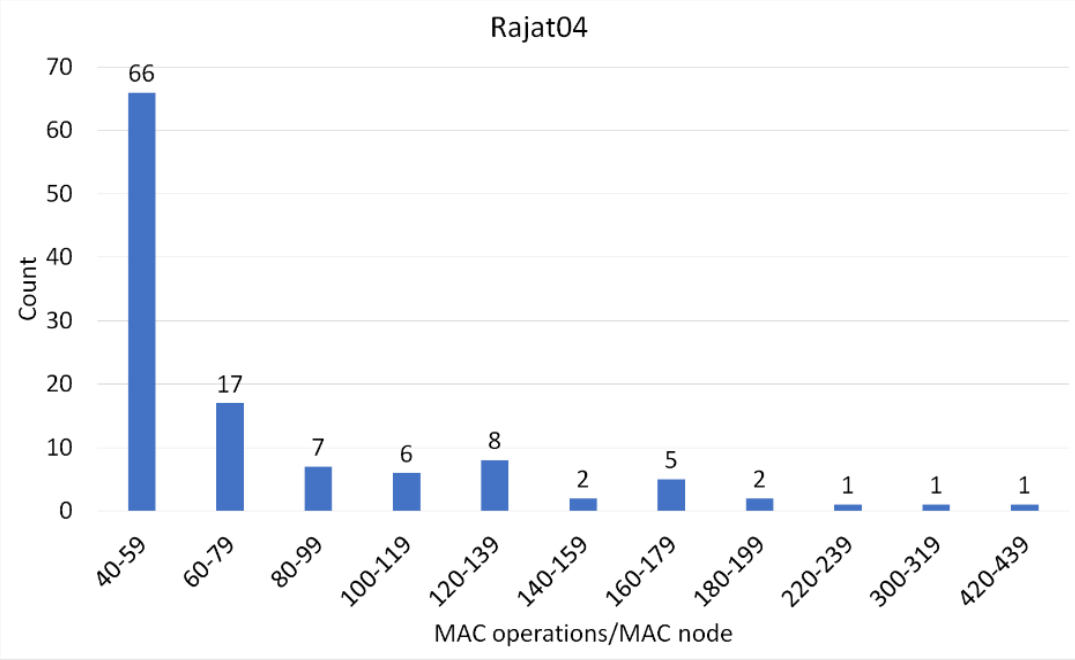


MAC operations/MAC node for *fpga_dcop_50* matrix:-

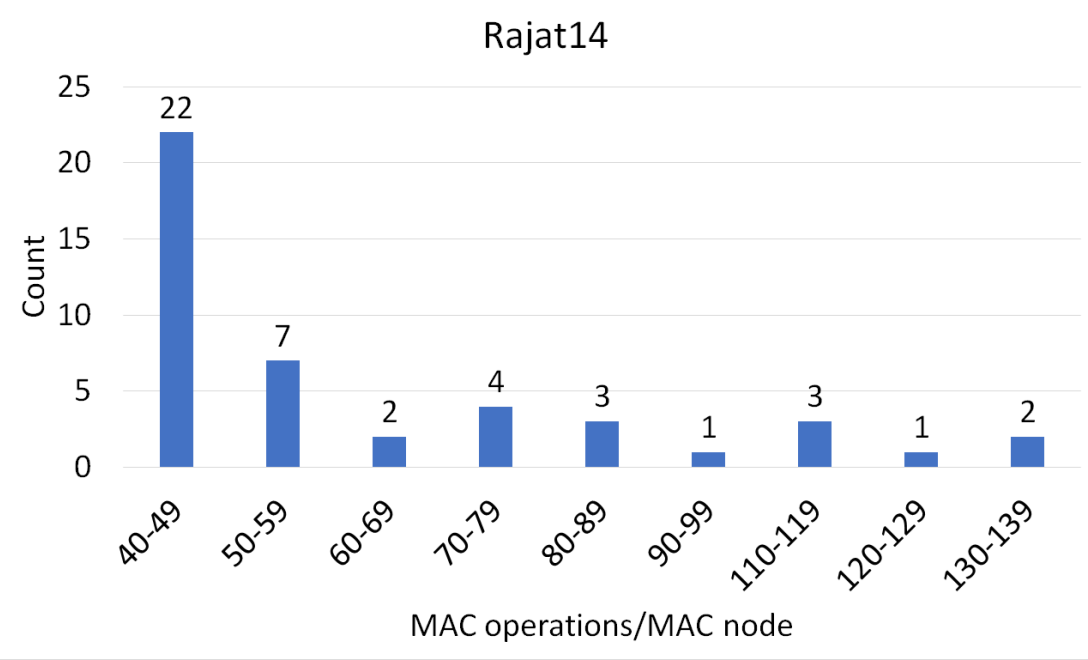


by Anurag Choudhury
Email Id: anuragn85@gmail.com

MAC operations/MAC node for *Rajat04* matrix:-



MAC operations/MAC node for *Rajat14* matrix:-



by Anurag Choudhury
Email Id: anuragn85@gmail.com

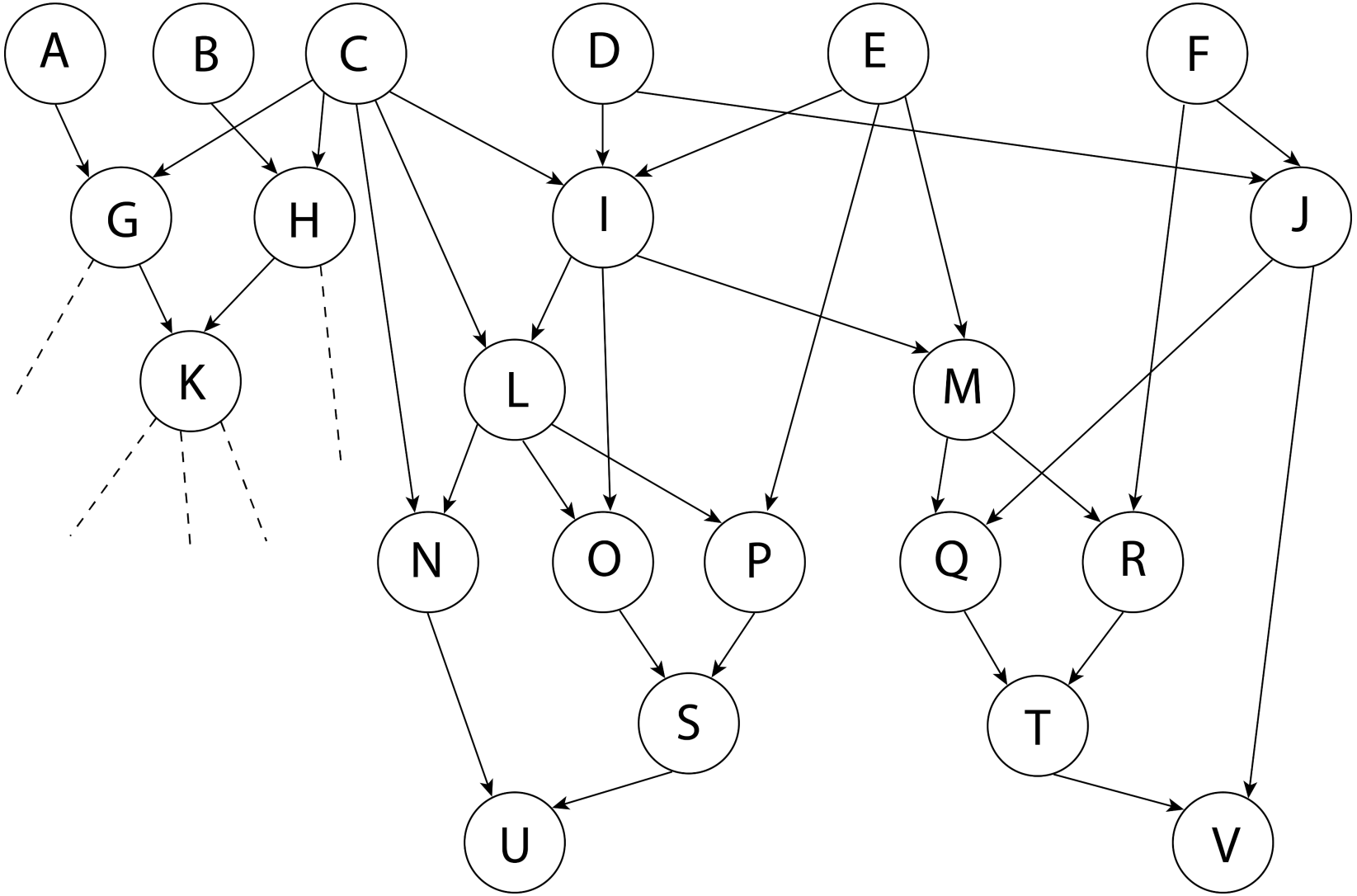
10. Future Work

10.1 Separating the MAC Arithmetic Unit into MUL and ADD AU

10.2 Splitting the DFG and assigning each graph to a separate LUD Hardware block

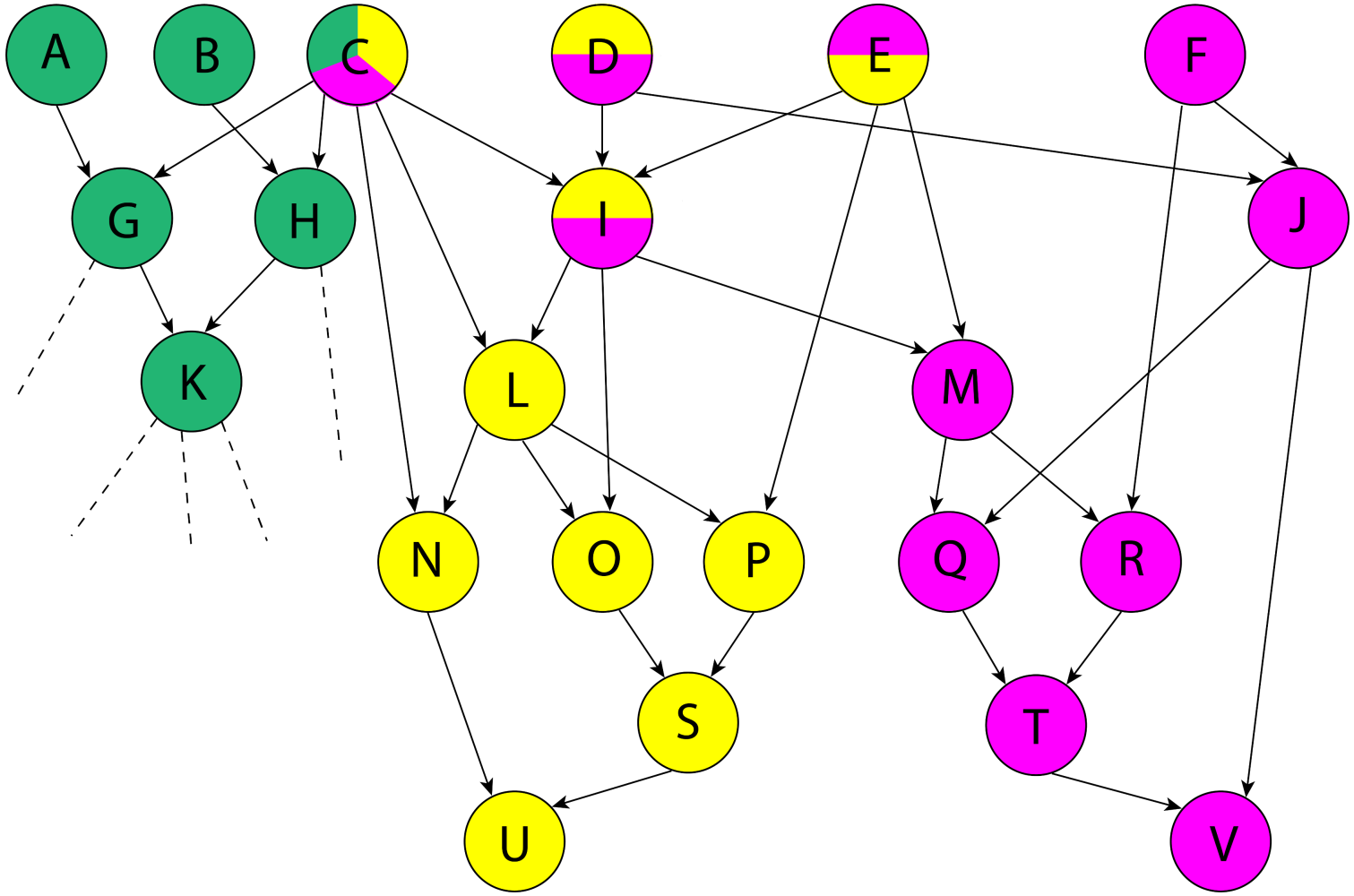
This approach will be helpful when we know that adding additional AU and BRAM to the cross-bar network would significantly reduce the clock cycles required for LU decomposition of the matrix, but adding any more blocks to the cross-bar network is making the design un-synthesizable due to routing congestions.
As an example, consider the graph given below: -

by Anurag Choudhury
Email Id: anuragn85@gmail.com

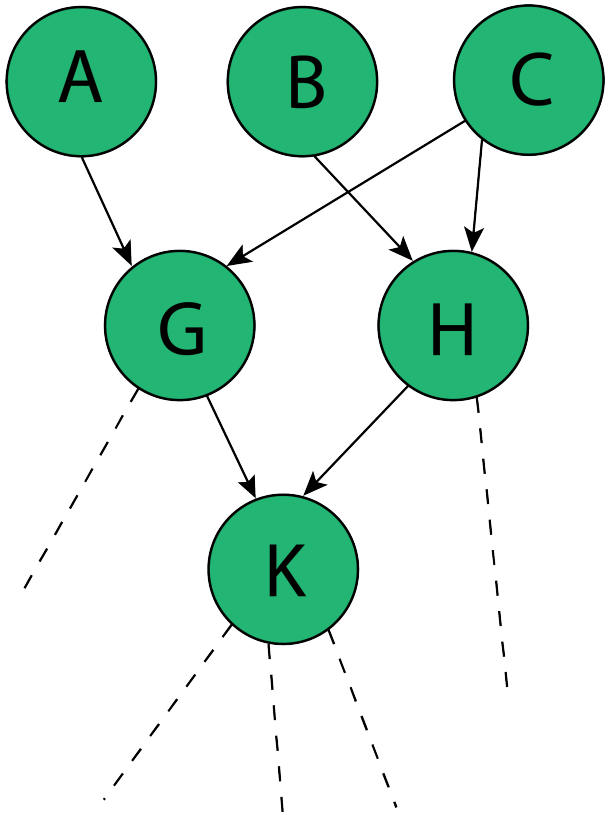


by Anurag Choudhury
Email Id: anuragn85@gmail.com

Suppose we want to divide it into 3 graphs. The colour coded graph is shown below:-

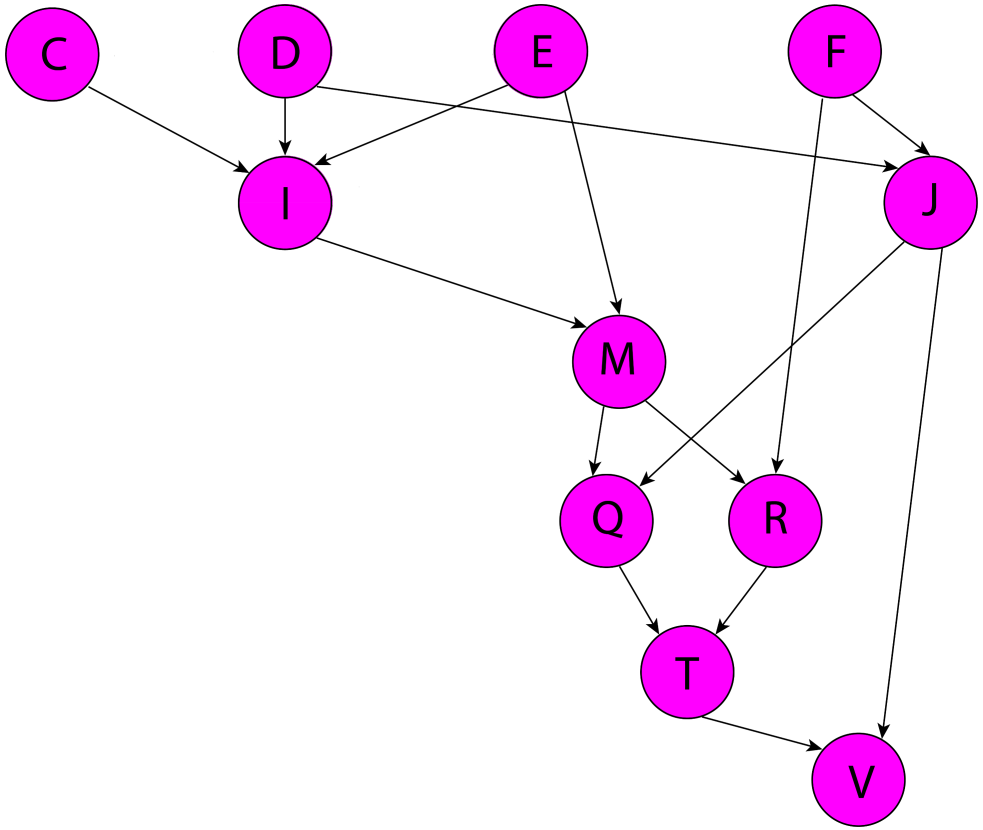


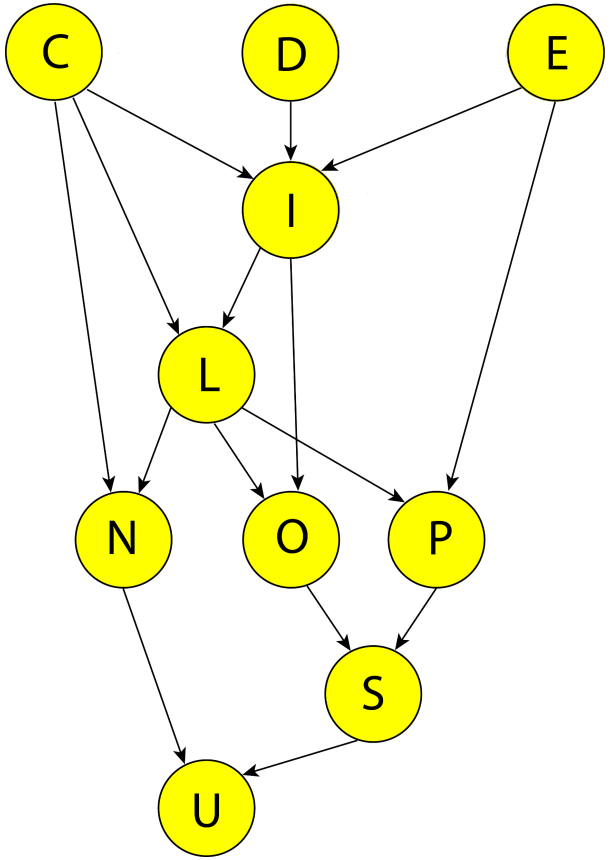
by Anurag Choudhury
Email Id: anuragn85@gmail.com



Graph 1

Graph 2

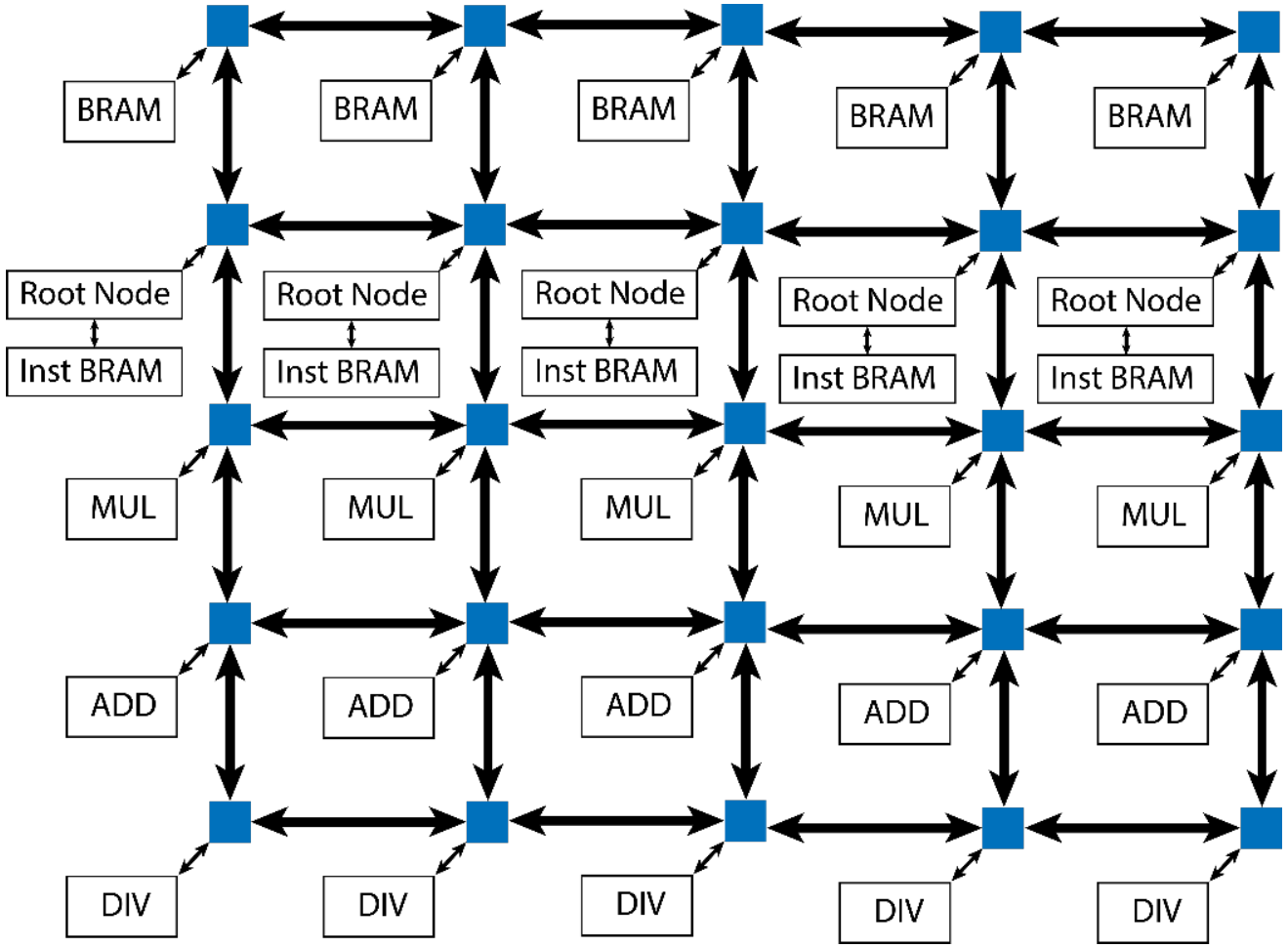




Graph 3

by Anurag Choudhury
Email Id: anuragn85@gmail.com

10.3 Implementing the hardware on an NOC (Network on Chip)



by Anurag Choudhury
Email Id: anuragn85@gmail.com

References

[**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**] 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.

[**3**] 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.

[**4**] 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.

by Anurag Choudhury
Email Id: anuragn85@gmail.com

[**5**] Engineer, Pinalkumar, Rajbabu Velmurugan, and Sachin Patkar. "Scalable implementation of particle filter-based visual object tracking on network-on-chip (NoC)." *Journal of Real-Time Image Processing* (2019): 1-18.

by Anurag Choudhury Email Id: anuragn85@gmail.com

Thank You

by Anurag Choudhury
Email Id: anuragn85@gmail.com