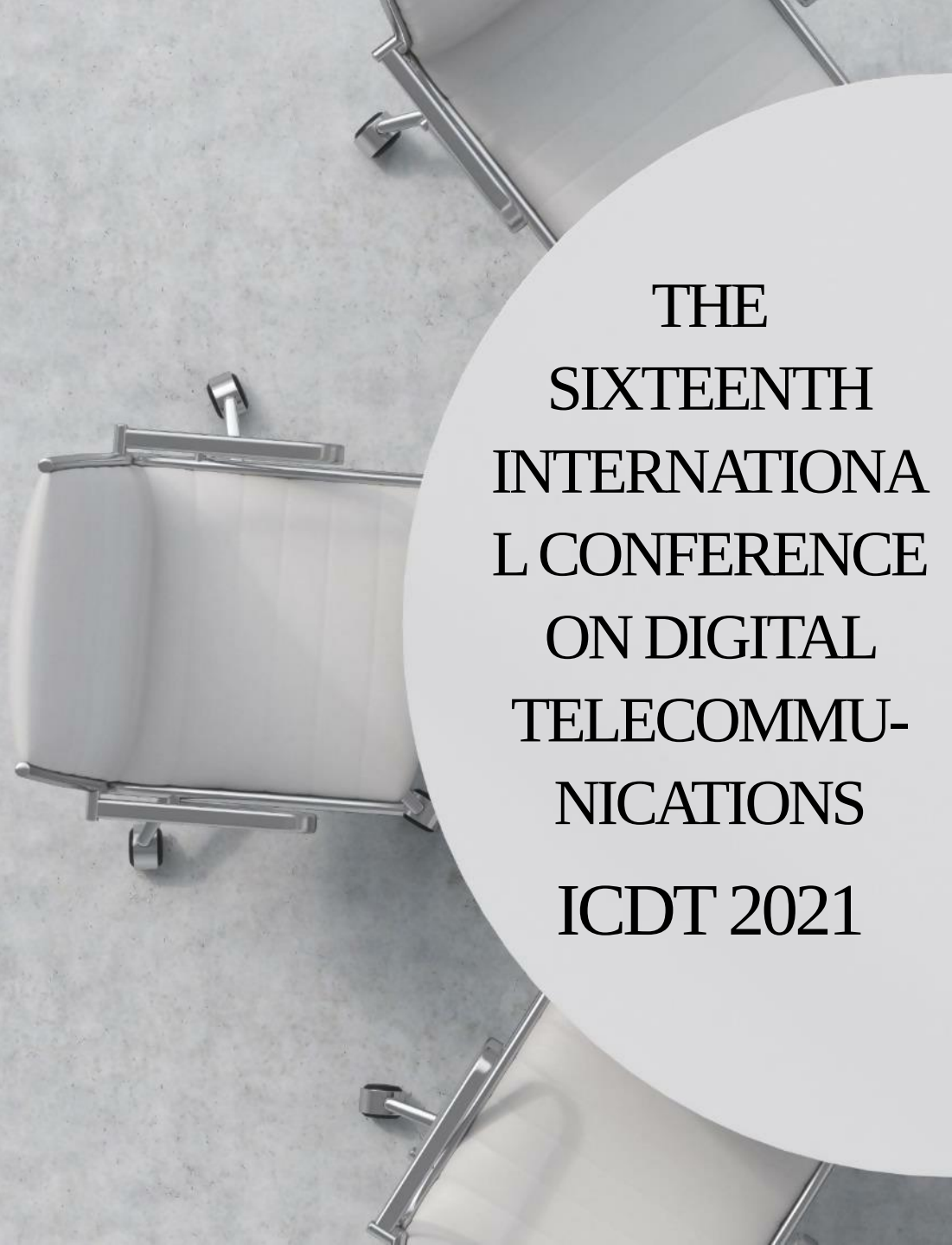
OPTIMIZATION OF SPARSE MATRIX ARITHMETIC OPERATIONS AND PERFORMANCE IMPROVEMENT USING FPGA

Dinesh Kumar Murthy, Semih Aslan
Electrical Engineering, Ingram School of Engineering
Texas State University
San Marcos, Texas, 78666, USA



THE SIXTEENTH INTERNATIONAL CONFERENCE ON DIGITAL TELECOMMUNICATIONS

ICDT 2021

AGENDA

- Introduction
- Problem Statement
- Research Motive
- Sparse Matrices
- FPGA's and EDA Tool
- Sparse Matrix Addition Operation
- Synthesis
- Hardware Implementation
- Performance Analysis
- Results and Discussion
- Conclusion
- Future Work

INTRODUCTION

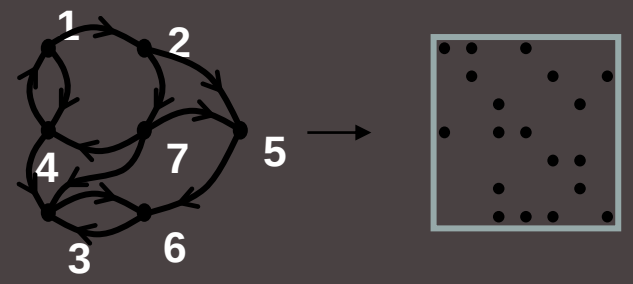
- Big Data collected has become increasingly important and are collected from many real-world applications:
  - Sensors, Social Networks, Portable Devices, Scientific Experiments.

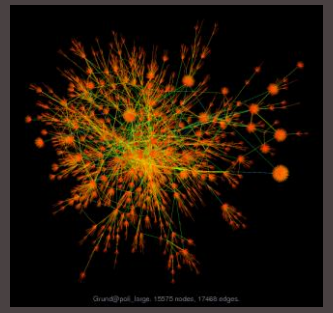
- Graphs are used to model many systems which are of interest to engineers and scientists today, through which useful information is extracted.

- Once entered a computer, the data from real-world applications no longer looks like a graph.

- Often it is in the form of a sparsely populated matrix with most zeros compared to non-zeros.

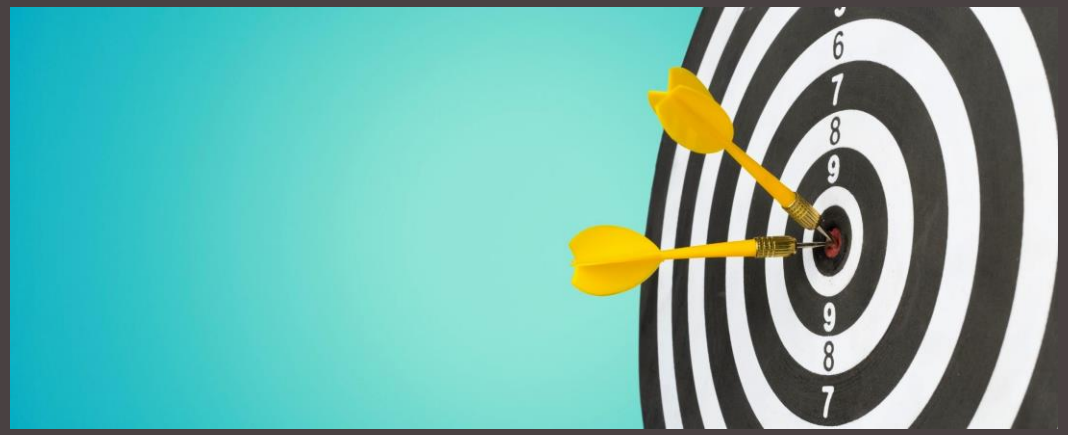
- **Challenge**: 1. Graphs can be complicated.
  2. The nature of data is sparse.





PROBLEM STATEMENT

Indirect Addressing	• Indirect addresses must address the non-zero entries of a sparse matrix in its index array leading to random accesses that require more memory transactions and lower cache hit rate.
Memory Allocation	• The distribution of zero and non-zero entries are not known in advance. Pre-allocating memory blocks of a specific size may waste memory when the intersection of nodes is large.
Low Arithmetic Intensity	• This is caused by the lack of temporal locality in the access to sparse matrices. If the matrix is not structured or blocked, most of the entries in cache line fetched to get an element remain unused causing high memory overhead for sparse matrix operation.



RESEARCH MOTIVE

To determine an algorithm for various sparse matrix arithmetic operations by minimizing gate count, area, computational time, latency, number of addition hardware and to improve throughput.

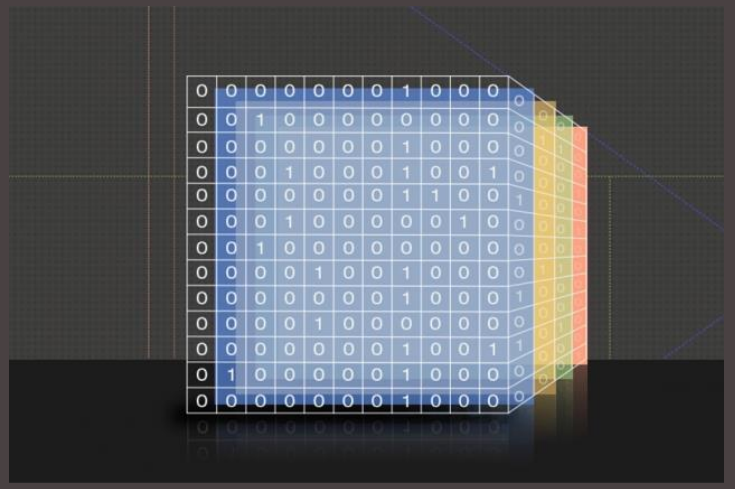
The developed algorithm must be capable of handling matrices of various sizes and should be simple to implement and highly scalable.

The primary goal of the thesis is to improve efficiency and reduce the resources used for the operation.

Comparison of the results and investigate the possible solutions and approaches for scaling up the design for larger matrix more efficiently.

SPARSE MATRICES

- Sparse matrices are at the heart of Linear Algebraic Systems.
- Everything of any significance happening in a sufficiently complex computer system will require lots of Linear Algebraic operations.
- You really cannot represent very large high dimensional matrices (when most of them have zeroes) in memory and do manipulations on them.

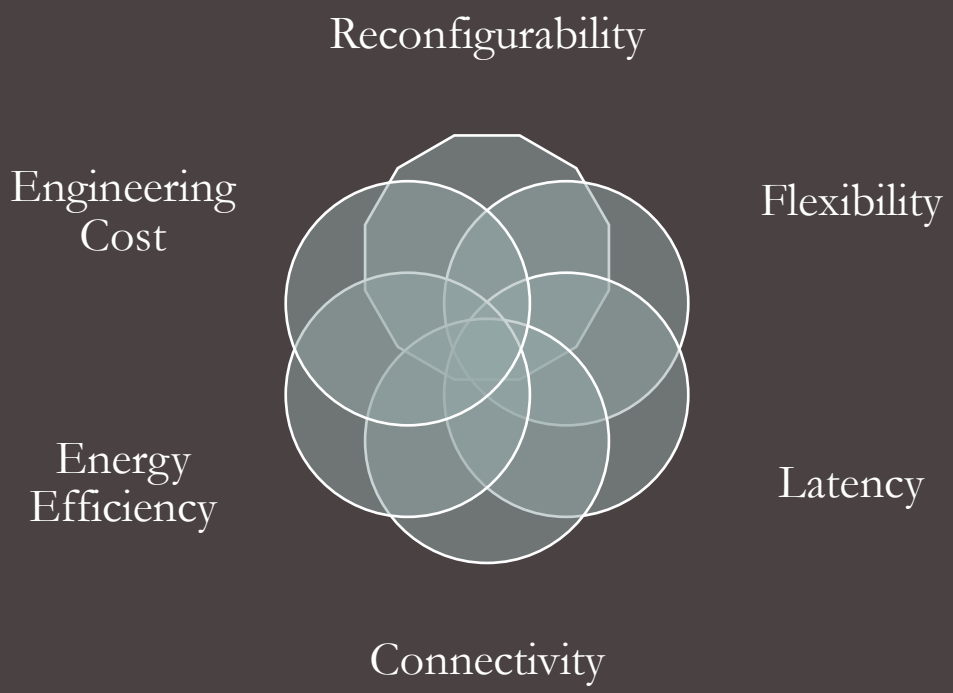


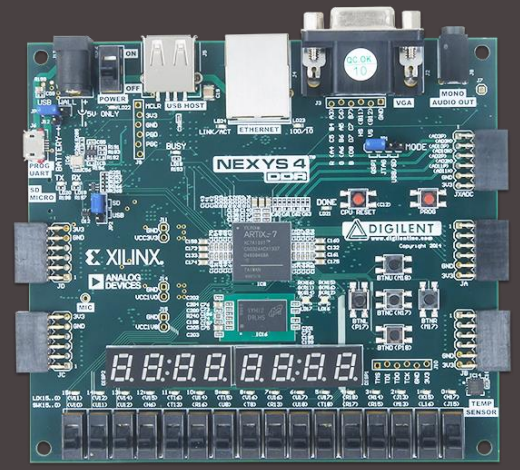
Applications

- Computer Graphics
- Machine Learning
- Information Retrieval
- Social Networks
- Maps
- Graph Based Applications

FPGA'S AND EDA TOOL

The Vivado Design Suite/Xilinx Integrated Software Environment(ISE) is the front-end GUI of the Xilinx tools which are used to program the FPGA devices with the user-defined functionality.



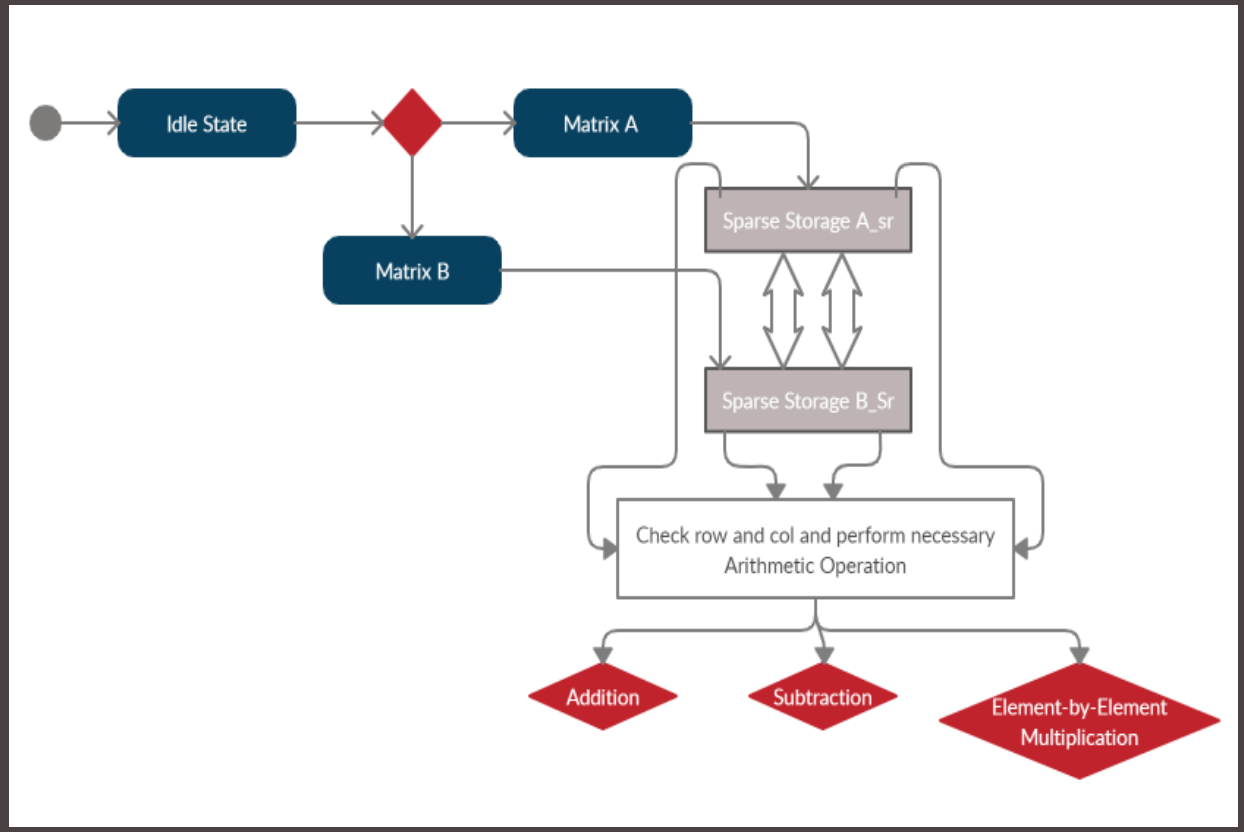


Nexys 4 DDR Artix-7 FPGA

SPARSE MATRIX ADDITION OPERATION

- The proposed architectural algorithm performs sparse matrix addition in which the number of rows and number of columns of two matrices should be equal.
- A parallel implementation of the addition, with enough fast memory algorithm, is proposed.
- The matrix addition performs the operation row-wise and column-wise throughout the matrix only for the nonzero elements present leaving behind the zeros.
- The mathematical representation of the addition operation is given below:

$$c_{i,j} = (a_{i,j}) + (b_{i,j})$$

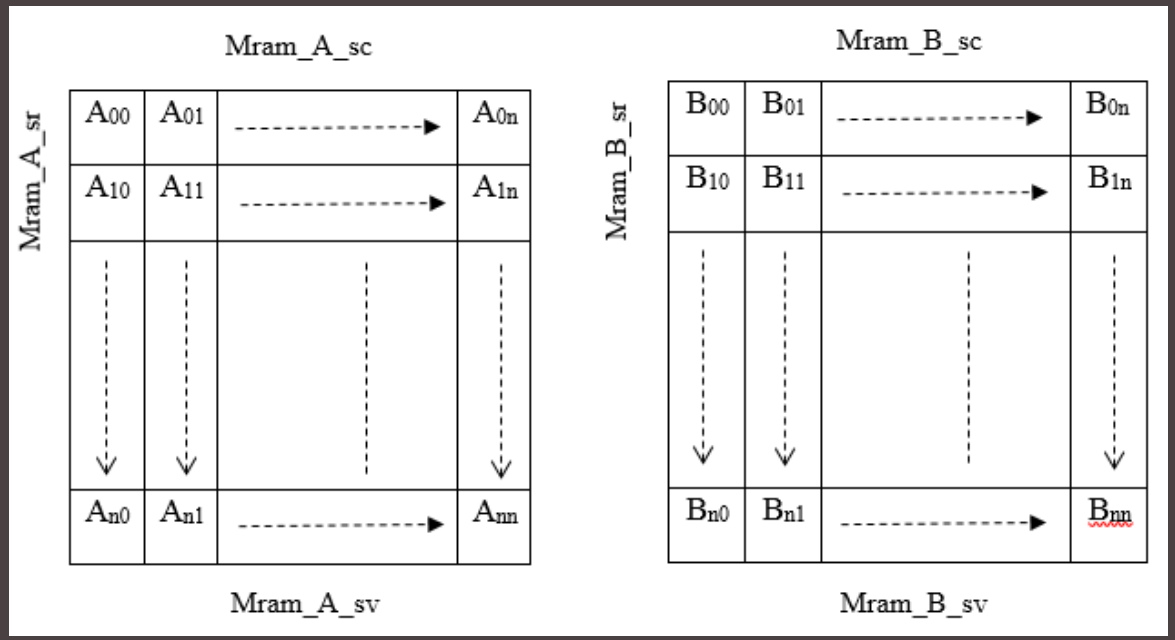


FSM Transition States for Sparse Matrix Arithmetic Operation

HARDWARE IMPLEMENTATION

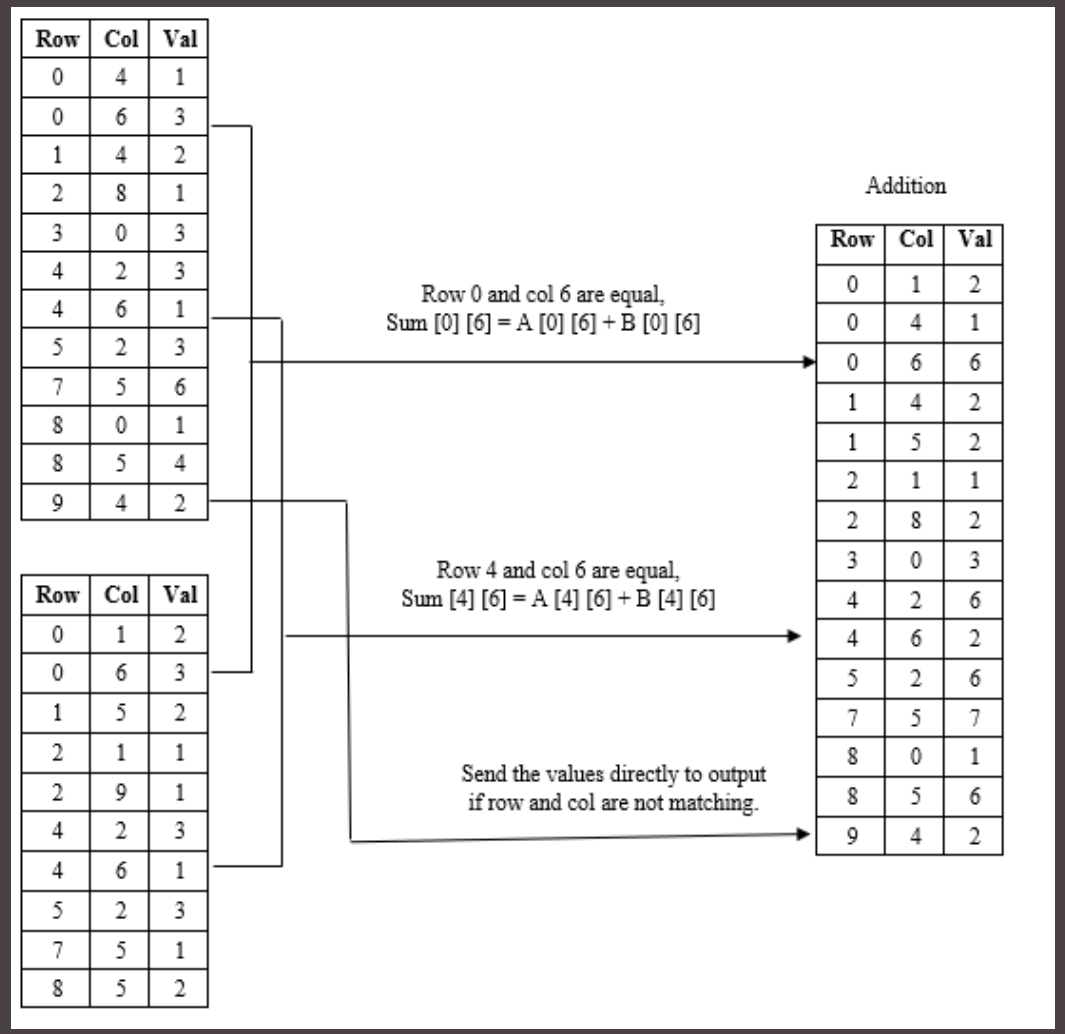
A → *n*×*n* sparse matrix
B → *n*×*n* sparse matrix

```
for i → 0 to MAT_SIZE do
        if (A[i]≠ 0) then
                Indexing row and column = i + 1
                A_sv [i] =A [i]
                A_index = A_count + 1
        end
        if (B[i] ≠ 0) then
                Index2rc = i + 1
                B_index = B_count + 1
                B_sv [ i] = B [ i]
        end
        if((A_sr[A_index]==B_sr[B_index])&&
        (A_sc[A_index] ==B_sc[B_index])) do
                Row <= A_sr [A_index]
                Col <= A_sc [A_index]
                Sum <= A_sv [A_index] + B_sv [B_index]
        end
        if (A_sv [A_index] ≠0) then
                Row <= A_sr [A_index]
                Col <= A_sc [A_index]
                Sum <= A_sv [A_index]
        end
        if (B_sv [B_index] ≠ 0) then
                Row <= B_sr[B_index]
                Col <= B_sc[B_index]
                Sum <= B_sv[B_index]
        end
end
```



Representation of Row and Column Access of Input Matrices

SPARSE MATRIX ADDITION OPERATION



Row	Col	Val
0	4	1
0	6	3
1	4	2
2	8	1
3	0	3
4	2	3
4	6	1
5	2	3
7	5	6
8	0	1
8	5	4
9	4	2

Row	Col	Val
0	1	2
0	6	3
1	5	2
2	1	1
2	9	1
4	2	3
4	6	1
5	2	3
7	5	1
8	5	2

Addition

Row	Col	Val
0	1	2
0	4	1
0	6	6
1	4	2
1	5	2
2	1	1
2	8	2
3	0	3
4	2	6
4	6	2
5	2	6
7	5	7
8	0	1
8	5	6
9	4	2

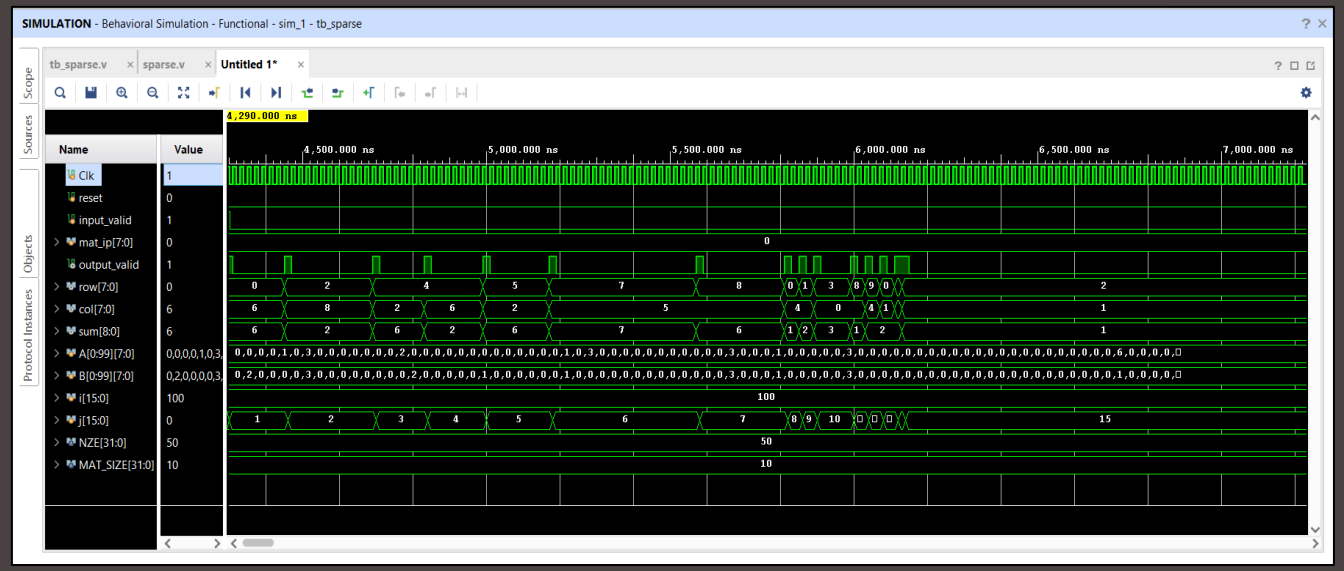
0	0	0	0	1	0	3	0	0	0
0	0	0	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
0	0	3	0	0	0	1	0	0	0
0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	6	0	0	0	0
1	0	0	0	0	4	0	0	0	0
0	0	0	0	2	0	0	0	0	0

Matrix A

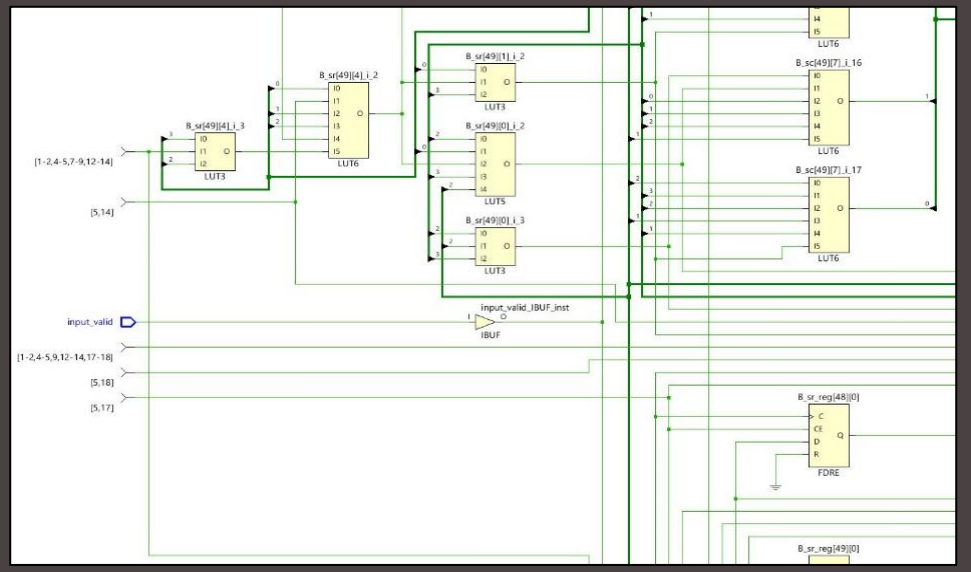
0	2	0	0	0	0	3	0	0	0
0	0	0	0	0	2	0	0	0	0
0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	3	0	0	0	1	0	0	0
0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Matrix B

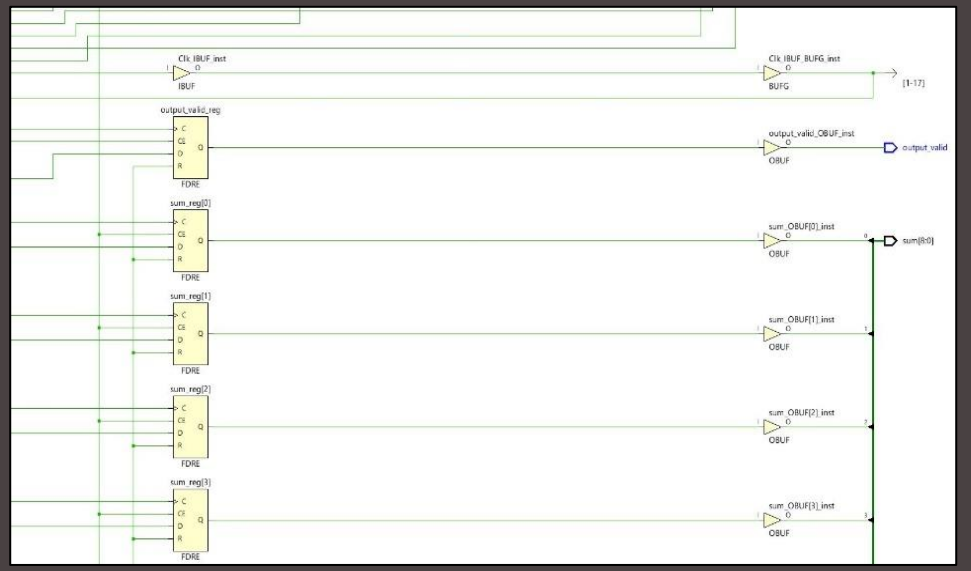
SIMULATION AND IMPLEMENTATION



Design Simulation: Sparse Matrix Addition

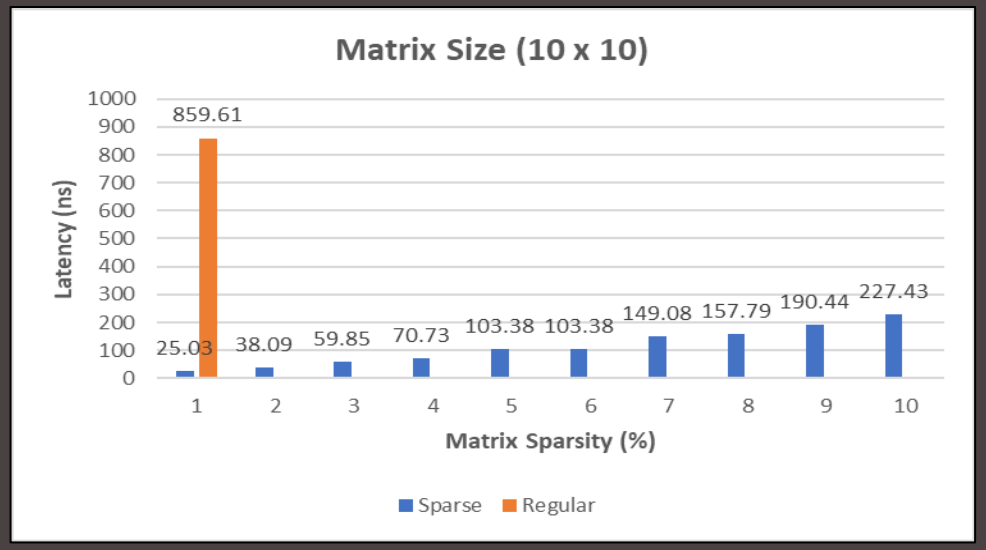


Input Valid Schematic

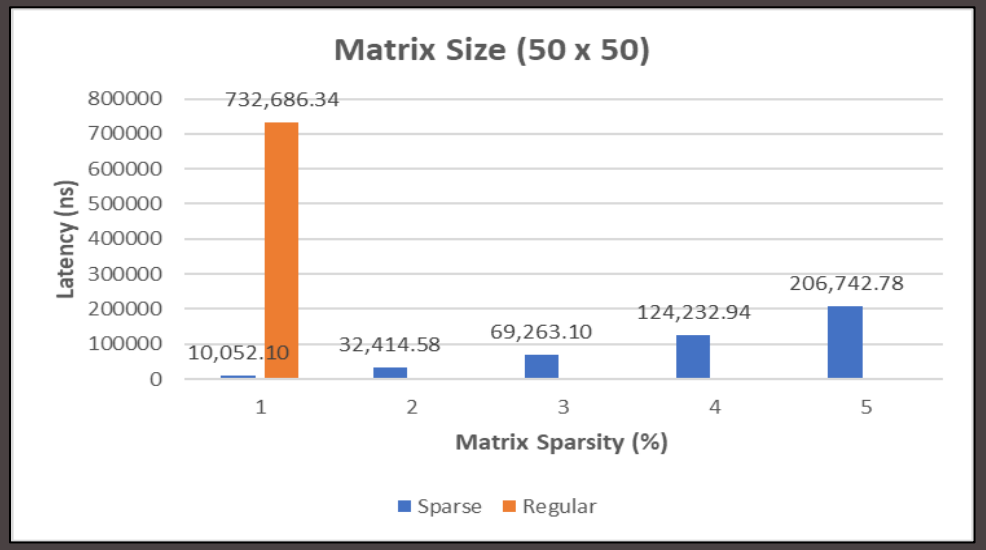


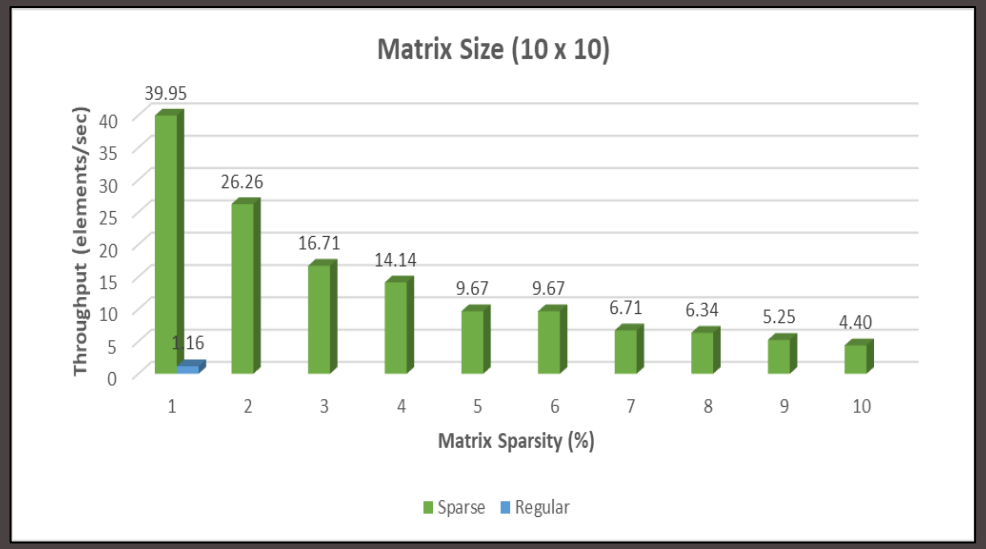
Output Valid Schematic

RESULT ANALYSIS

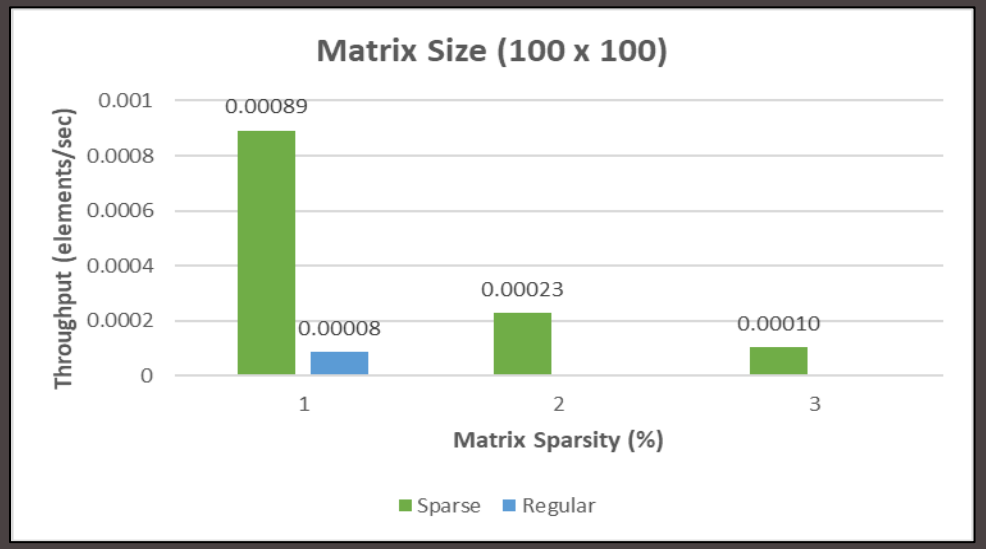


Significant decrease in Latency for the proposed sparse addition compared to regular matrix addition

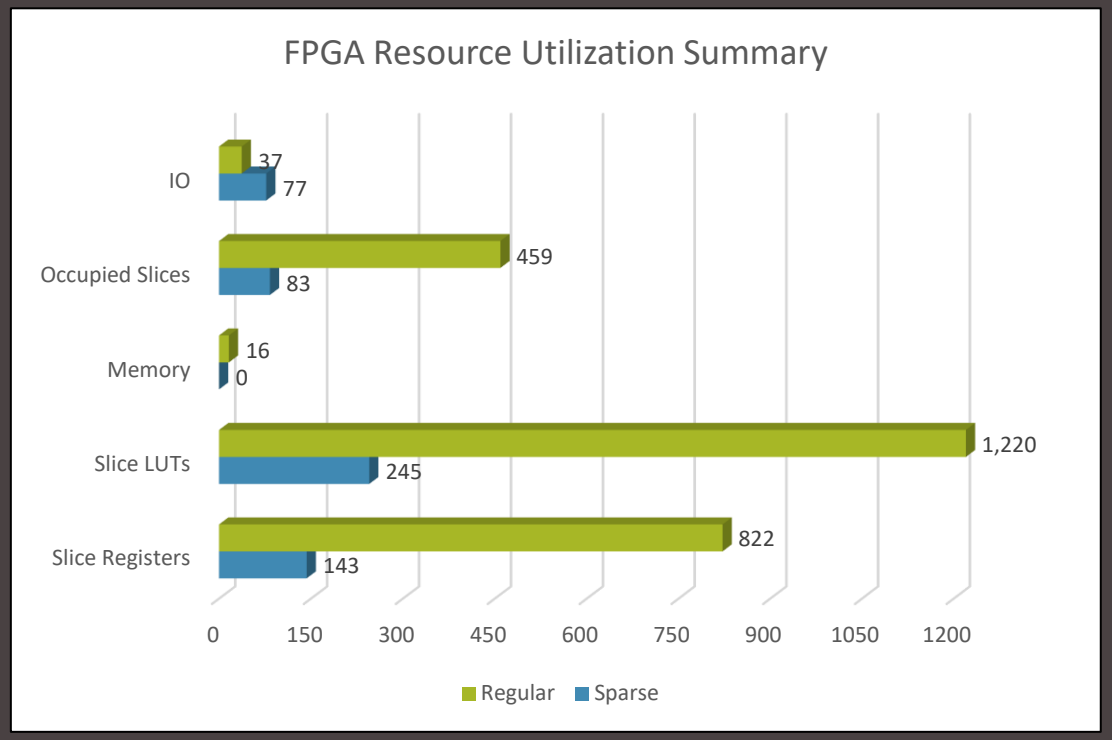




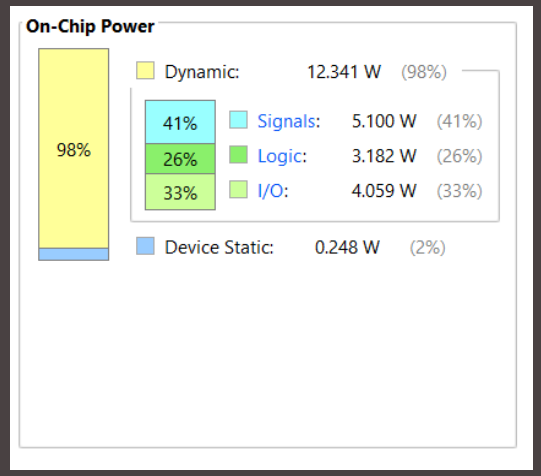
Significant increase in Throughput for the proposed sparse addition compared to regular matrix addition



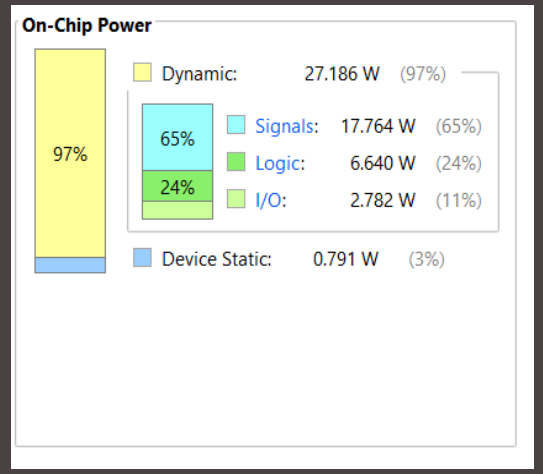
RESULT COMPARISON



FPGA Resource Utilization: Regular vs Sparse for Matrix Addition Operation



Proposed Method



Regular Method

Power Analysis: Sparse Matrix Arithmetic Operation

Hardware Implementation: Best- and worst-case Delays

	Input (ns)	Output (ns)
Best Case Delay (min)	3.76	4.82
Worst Case Delay (max)	4.34	5.96

KEY TAKEAWAYS

- Today's applications require higher computational throughput and a distributed memory approach for real-time applications.
- This research is primarily focused on designing an optimized architecture for sparse matrix operations, allowing for more efficiency than standard operations.
- The functionality of the design is verified by different sets of test cases under a specific size.
- The system contains a memory control which fetches the data from memory and passes it on for various arithmetic operations.

FUTURE WORK

- Research improvement in this area is needed to increase logic resources by a comparable increase in I/O bandwidth and on-chip memory capacity, especially when the matrix sparsity is unstructured and randomly distributed.
- Opportunities for future work includes increasing the simulations for a variety of benchmark matrices from all application domains and exploring further optimization.

THANK YOU