

Rochester Institute of Technology

RIT Digital Institutional Repository

Theses

5-2024

Benchmarking of Matrix Multiplication Acceleration Methods

Ashish Tondwalkar

Follow this and additional works at: <https://repository.rit.edu/theses>

Recommended Citation

Tondwalkar, Ashish, "Benchmarking of Matrix Multiplication Acceleration Methods" (2024). Thesis. Rochester Institute of Technology. Accessed from

This Master's Project is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

BENCHMARKING OF MATRIX MULTIPLICATION ACCELERATION METHODS

by

ASHISH TONDWALKAR

GRADUATE PAPER

Submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE
in Electrical Engineering

Approved by:

Mr. Mark A. Indovina, Senior Lecturer

Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

Dr. Ferat Sahin, Professor

Department Head, Department of Electrical and Microelectronic Engineering

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

MAY, 2024

Dedication

I dedicate this work to my father Saroj Tondwalkar, my mother Priti Tondwalkar, and my brother Anish Tondwalkar, and my colleagues and friends for their love and support during my thesis.

Ashish Tondwalkar

Declaration

I hereby declare that except where specific reference is made to the work of others, that all content of this Graduate Paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This Graduate Project is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Ashish Tondwalkar

May, 2024

Acknowledgements

I would like to thank my advisor Professor Mark A. Indovina for his support, guidance, feedback, and encouragement which helped in the successful completion of my graduate research.

Ashish Tondwalkar

Abstract

With the advent of artificial intelligence (AI), performance and model runtime feasibility poses a challenge to the advancement of AI technology. Novel methods of accelerating the core mathematical functions of AI applications are being explored. The crux of AI computations that would benefit from hardware acceleration is matrix multiplication. This thesis explores the acceleration of matrix multiplication using systolic arrays and the strassen algorithm, methods known for enhancing computational efficiency through parallel processing. The research focuses on the design, implementation, and comprehensive testing of these architectures to expedite matrix multiplication tasks, crucial for applications in deep learning and signal processing. By comparing various design methodologies and evaluating their performance among different scenarios, the thesis aims to identify optimal configurations that maximize processing speed and efficiency as well as determine the circumstances for which method should be deployed. This paper contributes to the advancement of our understanding high-performance computing trade-offs by providing insights into approaches of hardware acceleration.

Contents

Contents	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Research Goals	2
1.2 Contributions	2
1.3 Organization	3
2 Bibliographical Research	5
2.1 A brief history of AI and Hardware	5
2.2 Design Challenges	6
2.3 Matrix Multiplication	7
2.3.1 Blocked Matrix Multiplication	7
2.4 Systolic Execution	9
2.5 Strassen Algorithm	11
3 UVM Environment	16

3.1	Hierarchical Design	16
3.1.1	Configuration Files	17
3.1.2	Agent	17
3.1.3	Reference Model	19
3.1.4	Monitor	20
3.1.5	Scoreboard	20
3.1.6	UVM Phasing	20
3.2	Metrics	20
3.2.1	Verification	21
3.2.2	Performance	21
4	Systolic Array Design	22
4.1	MAC Unit Design	22
4.1.1	Pipelining Methodology	24
4.2	Array of MAC Units	24
4.2.1	Design Features	26
5	Strassen Multiplier Design	27
5.1	Hierarchy	27
5.2	Strassen Unit	28
5.2.1	Pipelining Methodology	31
5.3	Strassen Layers	32
5.3.1	Data Precision	32
5.3.2	Design Features	33
6	Results and Discussion	34

6.1	Immediate Accumulator Reset	34
6.1.1	4×4	34
6.1.2	16×16	36
6.1.3	64×64	37
6.1.4	Trends	39
6.2	No Accumulator Reset	39
6.2.1	4×4	39
6.2.2	16×16	41
6.2.3	64×64	42
6.2.4	Trends	44
6.3	Variable Matrix Input Dimensions	44
6.4	Physical Characteristics	45
6.4.1	Cell Area	45
6.4.2	Timing	47
6.4.3	Power	49
6.5	Area over Relative Performance	52
6.6	Observations	53
7	Conclusion	54
7.1	Future Work	55
	Bibliography	56
I	Source Code	I-1
I.1	Multiply Accumulate Unit	I-1
I.2	Systolic Array RTL Generation Script	I-4

I.3	Shift Register	I-15
I.4	Systolic Array Alignment Module RTL Generation Script	I-17
I.5	Strassen Layer RTL Generation Script	I-24
I.6	Test Config File Generation Script	I-49
I.7	Strassen Accumulator Top File	I-56
I.8	Systolic Array Driver	I-60
I.9	Strassen Matched Bandwidth Driver	I-72
I.10	Strassen Matched Size Driver	I-81
I.11	Sequencer	I-89
I.12	Sequence	I-90
I.13	Agent	I-91
I.14	Input Interface	I-94
I.15	Output Interface	I-96
I.16	Monitor	I-97
I.17	Reference Model	I-102
I.18	C Modeller	I-109
I.19	Packet In	I-118
I.20	Packet Out	I-120
I.21	Environment	I-121
I.22	Comparator	I-125
I.23	Top	I-135
I.24	Test	I-142

List of Figures

2.1	Blocked Matrix Multiplication	8
2.2	Operation of the Systolic Array	10
3.1	UVM Hierarchy	18
4.1	MAC Unit Diagram	23
4.2	MAC Unit Timing Diagram	24
4.3	MAC Array	25
5.1	Recursive Algorithm	28
5.2	Recursive Algorithm	29
5.3	Strassen Unit Diagram	31
5.4	Strassen Unit Timing Diagram	31
5.5	Strassen Layer Diagram	32
6.1	Plot of Performance Data for 4×4	35
6.2	Plot of Performance Data for 16×16	37
6.3	Plot of Performance Data for 64×64	38
6.4	Plot of Performance Data for 4×4	40
6.5	Plot of Performance Data for 16×16	42

6.6	Plot of Performance Data for 64×64	43
6.7	Plot of Size vs Cell Area for Pre-scan	47
6.8	Plot of Size vs Cell Area for Pre-scan	48
6.9	Plot of Size vs Dynamic Power	50
6.10	Plot of Size vs Leakage Power	51
6.11	Plot of Size vs Performance over Area	52

List of Tables

4.1	MAC signal details	23
4.2	Systolic Array signal details	25
5.1	Wishbone signal details	31
5.2	Strassen Layer signal details	32
6.1	4×4 Performance	35
6.2	16×16 Performance	36
6.3	64×64 Performance	38
6.4	Synthesis results	39
6.5	4×4 Performance	40
6.6	16×16 Performance	41
6.7	64×64 Performance	43
6.8	Synthesis results	44
6.9	Variable Matrix Input Dimension	44
6.10	Systolic Cell Area Pre-scan	45
6.11	Strassen Cell Area Pre-scan	45
6.12	Systolic Cell Area Post-scan	46
6.13	Strassen Cell Area Post-scan	46

6.14 Timing	48
6.15 Timing	49
6.16 Design Compiler Power	49
6.17 IC Compiler Power	51
6.18 Performance over Area	52

Chapter 1

Introduction

Hardware became a limitation in AI development as the complexity and size of models grew faster than the processing capabilities of traditional computing systems. These models require extensive computational power and memory, pushing conventional processors to their limits and leading to bottlenecks in training and inference times [1]. This gap necessitated the development of specialized hardware accelerators like graphics processors, designed to handle the parallel processing demands of AI algorithms more efficiently. With further advancements, tensor processing became a necessity for accelerating core computations of AI models.

Hardware acceleration is essential for AI due to the immense computational demands of training and running deep learning models. It enables faster processing of complex algorithms and massive datasets, significantly reducing the time required for AI tasks [2, 3]. This acceleration not only improves efficiency but also allows for more sophisticated models and real-time applications, advancing the applications and possibilities of what AI can achieve [4, 5].

This paper delves into the realm of AI acceleration through the lens of matrix multiplication hardware, a cornerstone operation in numerous AI algorithms and deep learning models. Through the use of a Universal Verification Methodology (UVM) environment, the hardware

designs are comprehensively benchmarked through various scenarios of data sets. Randomization of data sets include randomized data, matrix sizes, and matrix pipelines that mimic possible realities of AI applications [6]. Through approaches in hardware design and testing techniques, this paper aims to contribute to the efficiency and performance of AI acceleration. The sections below describe the motivation, research goals and the organization of this paper.

1.1 Research Goals

The primary intent of setting up this paper is to research and implement a self-checking test-bench using the UVM framework to observe the correctness of matrix multiplication hardware and to run various tests for benchmarks. Shown below is a summary of the leading research goals.

- To understand the hardware implementation of matrix multiplication acceleration methods and verify their operation
- To benchmark the acceleration methods with randomized testing data and scenarios
- To compare the effectiveness of acceleration methods and observe trade-offs for various scenarios

1.2 Contributions

The significant contributions to the projected are listed below.:

1. The systolic array and Strassen algorithm are implemented in Verilog
2. A fully featured UVM environment is constructed to contain the multipliers and run benchmarks

3. A script was written to generate various scenarios for matrix size and pipelines to mimic real applications of AI
4. Drivers were created to propagate randomized data to the multipliers with matrix formatting provided by the script
5. The effectiveness of the random sequences and the test-plan is measured using functional coverage metrics such as cover-groups and assertions.
6. The obtained information is analyzed and is presented using graphs and charts.

1.3 Organization

The structure of the paper is as follows:

- Chapter 2: This chapter discusses the rationale of the research and the necessary background information.
- Chapter 3: This chapter explains the verification components used in the paper's UVM framework, their importance and hierarchy.
- Chapter 4: The chapter goes into the details and design choices of the systolic array multiplier.
- Chapter 5: This chapter explains the details and design choices of the Strassen algorithm multiplier.
- Chapter 6: This chapter discusses about the obtained results in detail and the drawn observations from the recorded results.

- Chapter 7: This chapter outlines the conclusion of the study and possible ways of extending it.

Chapter 2

Bibliographical Research

2.1 A brief history of AI and Hardware

AI computation on hardware has transitioned through several phases, starting with CPUs which were general-purpose but lacked efficient parallel processing for complex AI tasks. GPUs then became popular due to their ability to perform many operations concurrently, greatly accelerating deep learning tasks. Google's introduction of TPUs, tailored for neural network computations, represented a leap in efficiency and processing speed for AI workloads [7]. These evolution's reflect the industry's response to the escalating computational demands of AI, emphasizing specialized hardware to unlock new levels of algorithm complexity and application potential [8]. The introduction of the TPU laid a great foundation for the future of domain specific architectures for AI acceleration [9]. The industry observes a number of newer, smaller organization designed processing systems for various AI sub-fields including large language models and natural language processing [10]. As these organizations develop newer technologies for this emerging niche of processing, it is important to understand the trade-offs for creating accelerators. Below is a list of characteristics for trade-offs in AI hardware

accelerators:

- Area
- Power
- Performance
- Cache Size
- Flexibility

With design complexity rapidly growing, the pressure falls on the design engineer to analyze and ensure that the design is optimized for its application, balancing these characteristics [11, 12]. It is crucial to understand the design choices and what hardware can accommodate for them.

2.2 Design Challenges

Balancing trade-offs in AI acceleration involves optimizing between power consumption, speed, memory bandwidth, and latency [13]. Achieving high performance may increase power usage and heat, while optimizing for low power might reduce computational capabilities [14]. Additionally, memory bandwidth must be sufficient to feed data to processing units without causing bottlenecks, which is often the greatest challenge for creating a highly parallelized architecture such as an AI accelerator. Innovations in architecture, algorithm efficiency, and hardware-software co-design are key to managing these trade-offs effectively. Thus, a theoretical understand of the approaches to matrix multiplication hardware is important.

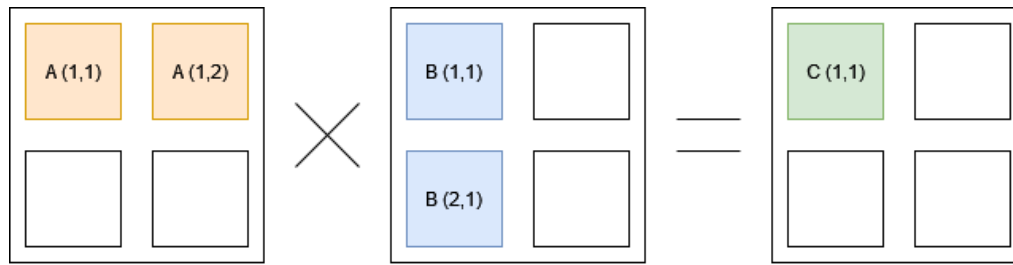
2.3 Matrix Multiplication

Multiplying matrices is a fundamental operation in AI applications, offering a compact way to represent and manipulate linear transformations of neural networks [13]. To multiply two matrices, you need to follow these steps. Let's say we have two matrices, A and B, and we want to find the product $C = AB$.

- The number of columns in matrix A must equal the number of rows in matrix B. If A is an $m \times n$ matrix, then B needs to be an $n \times p$ matrix, and the resulting matrix C will be an $m \times p$ matrix.
- Perform Element-wise Multiplication and Summation: To find the element C_{ij} in the resulting matrix C (i.e., the element in the i^{th} row and j^{th} column), you multiply each element in the i^{th} row of matrix A by the corresponding element in the j^{th} column of matrix B and sum these products. Mathematically, if A is $m \times n$ and B is $n \times p$, then $C = AB$ is $m \times p$ where: $C_{ij} = \sum_{k=1}^n A_{ik}B_{kj}$
- Traditionally, each element of C is computed by taking the dot product of the corresponding row of A and the corresponding column of B

2.3.1 Blocked Matrix Multiplication

Blocked matrix multiplication is an optimization technique used especially for large matrices to improve cache usage and reduce the number of cache misses during the computation. This approach involves subdividing the matrices into smaller blocks or sub-matrices, and then applying the standard matrix multiplication algorithm to these smaller blocks as seen in Figure 2.1. The key steps in blocked matrix multiplication are:



$$A(1,1) \times B(1,1) + A(1,2) \times B(2,1) = C(1,1)$$

Figure 2.1: Blocked Matrix Multiplication

- **Divide Matrices into Blocks:** You divide each matrix into smaller sub-matrices or blocks. The size of the blocks is typically chosen based on the architecture of the computer to make efficient use of the cache memory.
- **Multiply Blocks as Units:** Treat each block as an element in a "larger matrix" of blocks. Perform multiplication on these blocks using the standard matrix multiplication method, where you now multiply and sum blocks instead of scalar elements.
- **Combine Block Results:** After computing the products of the blocks, you combine them to form the final matrix product.

A great advantage of blocked multiplication is by working on small blocks that fit into the cache, the algorithm reduces cache misses. This is because once a block is loaded into the cache, it can be reused multiple times before being evicted [15]. Choosing a block size is usually dependent of the architecture of the CPU to take advantage of the cache utilization. In the context of matrix multiplication accelerators, however, block size is determined by the maximum size of the multiplier hardware and use a large buffer to localize spatial data instead of a cache [1]. This will maximize use of the multiplier pipeline and give the highest theoretical throughput.

2.4 Systolic Execution

Systolic execution using the systolic array is the most traditional approach to perform a parallelized matrix multiplication in hardware [16]. A systolic array is a network of simple cores that rhythmically process and pass the data through the system as seen in Figure 2.2. Each core, known as multiply-accumulate (MAC) unit, accumulates the multiplication on a pair of elements. Then it passes each element to its neighboring row and column MAC unit. This structure allows for high-throughput and efficient parallel processing, making systolic arrays effective for matrix multiplication and other operations common in AI and deep learning tasks [17]. The regular data flow minimizes memory access delays, enhancing computational speed and efficiency. This approach is also deterministic, allowing the processing system to only process the data for the minimum number of cycles necessary to compute the matrix product.

The figure below shows the operation of a 3x3 systolic array where the product of matrices A and B is performed. The elements represented as circles are passed into the array of MAC units in a staggered fashion, where the first element of each row or column is delayed by a multiply-accumulate cycle from the preceding row/column. The input matrix A is shown on the right, while the transpose of matrix B is entered through the top. During each cycles, the elements that enter then MAC are multiplied then accumulate within the MAC. Afterwards, the elements are passed to the next MAC units, where data from matrix A are passed to the right and the data from matrix B is passed below. After the multiplication is complete, each MAC unit contains an element of the product matrix.

The operation of the systolic array is deterministic, as long as the processor has access to the input matrix dimensions. Given matrix dimensions of $I \times J$ and $J \times K$, the number of MAC cycles for the array to complete multiplication is $I + J + K - 2$. To use the figure as an example, multiplying two 3×3 matrices will require 7 MAC cycles. The ideal scenario to populate the

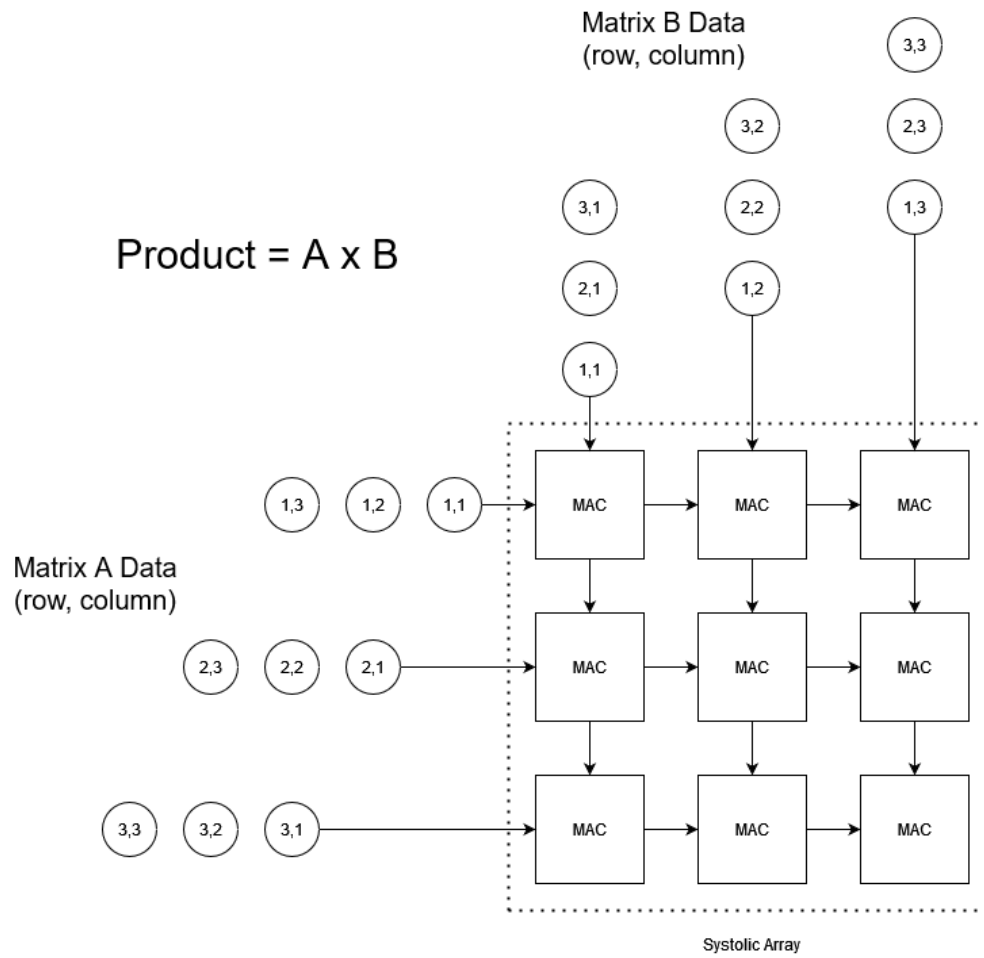


Figure 2.2: Operation of the Systolic Array

full two dimensional pipeline of the array is to continuously pass square matrices matching the size of the array [17]. Smaller matrices will not occupy each MAC, but will have a shorter operation duration. In most deployments of processors using systolic arrays, the array is too small to accommodate for the matrices being multiplied and results in blocked multiplication, which leads to a fully populated array [18].

Systolic Arrays can be easily scaled to match design requirements. Adding or reducing the amount of MAC units in the array is linearly proportional to the computational capacity of the array. The repetitive structure's simplicity and scalability makes it especially easy to accommodate various applications such as edge computing or super computing [19]. However, the high throughput is reliant on regular delivery of data to the array's inputs, which creates a dependency on data access and memory bandwidth.

2.5 Strassen Algorithm

The Strassen algorithm is a sophisticated method for matrix multiplication that improves on the traditional approach's computational complexity. Instead of using the conventional method, which has a time complexity of $O(n^3)$ for multiplying two matrices, the Strassen algorithm reduces the complexity to approximately $O(n^{2.8074})$ [20]. It accomplishes this by decreasing the number of recursive multiplications needed to compute the product of two matrices. Let's consider two matrices A and B, with the goal of computing the matrix $C = AB$.

Partition each matrix into four sub-matrices. For matrices A and B, each of size 2×2 , this is straightforward:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad (2.1)$$

First, seven intermediate products (as opposed to the usual eight) are computed using specific combinations of the elements from matrices A and B:

$$1. P_1 = a(f - h)$$

$$2. P_2 = (a + b)h$$

$$3. P_3 = (c + d)e$$

$$4. P_4 = d(g - e)$$

$$5. P_5 = (a + d)(e + h)$$

$$6. P_6 = (b - d)(g + h)$$

$$7. P_7 = (a - c)(e + f)$$

The elements of the resulting matrix C can then be computed using these intermediate products:

$$1. C_{11} = P_5 + P_4 - P_2 + P_6$$

$$2. C_{12} = P_1 + P_2$$

$$3. C_{21} = P_3 + P_4$$

$$4. C_{22} = P_1 + P_5 - P_3 - P_7$$

For matrices larger than 2×2 , the Strassen algorithm applies a recursive divide-and-conquer strategy [21]. The large matrix is divided into sub-matrices, and the algorithm is applied to these sub-matrices, reducing the problem size at each step until reaching the base case of 2×2 matrices. The intermediate results are then combined to form the final product. This can be understood by the following example with the goal of computing the matrix $Z = XY$, where matrices X and Y represent 4×4 matrices:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad (2.2)$$

First, seven intermediate products (as opposed to the usual eight) are computed using specific combinations of the elements from matrices A and B:

1. $P_1 = A(F - H)$
2. $P_2 = (A + B)H$
3. $P_3 = (C + D)E$
4. $P_4 = D(G - E)$
5. $P_5 = (A + D)(E + H)$
6. $P_6 = (B - D)(G + H)$
7. $P_7 = (A - C)(E + F)$

Before the result matrix Z can be formed, the multiplications of the intermediate products are performed using the Strassen Algorithm at the base case. Here, the first intermediate product is calculated:

$$A = \begin{bmatrix} i & j \\ k & l \end{bmatrix}, (F - H) = \begin{bmatrix} v & b \\ n & m \end{bmatrix} \quad (2.3)$$

1. $Q_1 = i(b - m)$
2. $Q_2 = (i + j)m$
3. $Q_3 = (k + l)v$

4. $Q_4 = l(n - v)$
5. $Q_5 = (i + l)(v + m)$
6. $Q_6 = (j - l)(n + m)$
7. $Q_7 = (i - k)(v + b)$
8. $P_{1,11} = Q_5 + Q_4 - Q_2 + Q_6$
9. $P_{1,12} = Q_1 + Q_2$
10. $P_{1,21} = Q_3 + Q_4$
11. $P_{1,22} = Q_1 + Q_5 - Q_3 - Q_7$

All intermediate products are computed by applying the Strassen Algorithm. Finally, the result matrix can be computed by combing the intermediate products:

1. $Z_{11} = P_5 + P_4 - P_2 + P_6$
2. $Z_{12} = P_1 + P_2$
3. $Z_{21} = P_3 + P_4$
4. $Z_{22} = P_1 + P_5 - P_3 - P_7$

In the context of a general purpose processor, the Strassen algorithm is faster than the conventional matrix multiplication method for large matrices due to its lower asymptotic complexity and reduced count of multiplications [22]. However, It can be less numerically stable and requires more memory for storing intermediate results. The performance gain over the conventional method depends on the implementation and the matrix sizes involved. The Strassen algorithm represents a significant step in the study of computational complexity, showing that

the intuitive $O(n^3)$ bound for matrix multiplication is not optimal. It opened the door for further research into even more efficient algorithms, such as the Coppersmith-Winograd algorithm and others that have an even lower asymptotic complexity.

In a highly parallelized context, the Strassen Algorithm can pipeline full matrix computations with a very high throughput. However, the silicon area cost of performance and the complexity of implementation plays into question the practicality of this algorithm.

Chapter 3

UVM Environment

UVM is a class library based on transaction-level modeling which will be used for functional verification and performance metrics of matrix multiplier implementations [6, 23]. The UVM hierarchy will feature 3 multipliers or designs under test that will be stimulated through their respective drivers and compared to with a reference model. The environment also features a methodology to calculate performance or clock cycles required to complete calculations.

3.1 Hierarchical Design

The UVM hierarchy encompasses a monitor and an agent that contains drivers along with virtual interfaces to and from the matrix multipliers [24]. The agent uses three drivers to send data provided by a sequencer to the multipliers and a reference model. The monitors and collects outputs of the designs and sends them to the scoreboard. The reference model uses the input data to create golden output data using a C model and sends those to the scoreboard. The scoreboard will compare the reference and output data to determine whether the designs are functionally valid. Specifics for simulation length and data parameters are controlled by

external files that are read by the environment. The environment and the generation of data is constructed by a simple test that is run at the top level. The hierarchy can be understood by Figure 3.1

3.1.1 Configuration Files

The configuration files are generated by scripts and contain the amount of matrix multiplications before resetting accumulators as well as the dimensions of matrix data for each set of matrix multiplications

3.1.2 Agent

The driving agent houses three drivers, each are responsible for sending stimulus to their respective design through a virtual interface. Each driver will receive randomized data using the sequencer. Drivers are also responsible for enabling performance counters.

Systolic Array Driver

The systolic array driver reads from configuration files that detail the amount of multiplications that will take place before resetting accumulators along with the matrix data dimensions. Using the dimension data, the driver modifies the sequencer data to clear unused rows and columns and propagates the data through the virtual interface. This is an important feature as the execution time is deterministic and can be reduced for smaller input matrices. The systolic driver also controls the enable signals of the systolic array's pipeline.

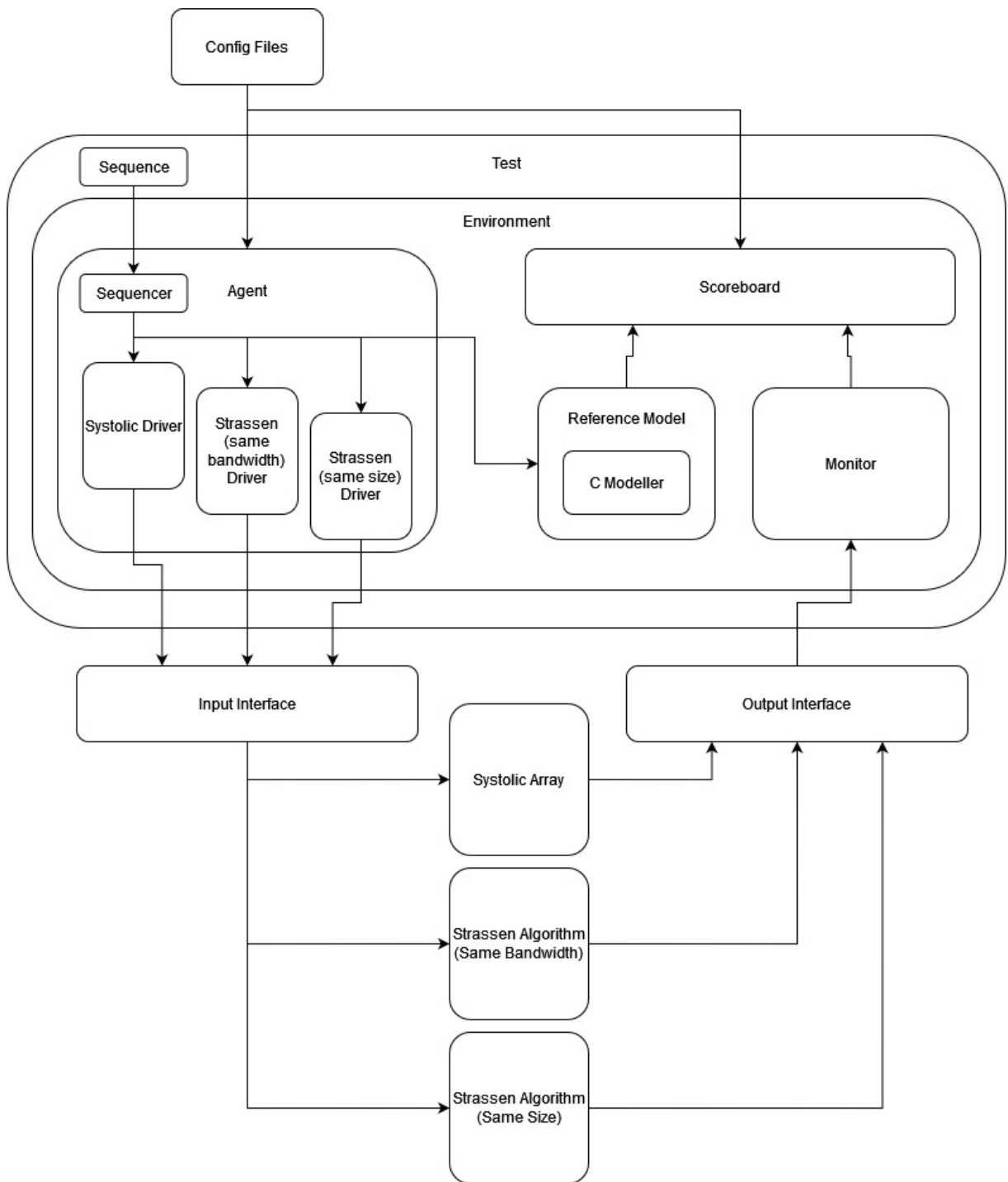


Figure 3.1: UVM Hierarchy

Strassen Algorithm Drivers

The Strassen algorithm reads from a file that contains data detailing the amount of multiplications that will take place before resetting the accumulators. The algorithm execution time is not deterministic and is equal for all input matrix sizes, so it will not need to modify sequencer data. The Strassen drivers control the enable signal for the pipeline. The drivers also keep track of cycles the pipeline to ensure the minimum number is achieved to receive all valid data.

3.1.3 Reference Model

The reference model collects all transaction data being sent to the designs. The model uses this data to generate the golden output data generated by a model written in C through DPI functions. The reference model will collect golden output data and send them to the scoreboard through FIFO's.

C Model

The DPI functions from the C model perform matrix multiplications on the sequencer data for each multiplier. The multiplication is performed using traditional matrix multiplication and uses the appropriate matrix size for the respective multiplier. For example, if matrix A is $m \times n$ and matrix B is $n \times p$, then the module computes matrix where: $C = AB$ is $m \times p$ where: $C_{ij} = \sum_{k=1}^n A_{ik}B_{kj}$. Each multiplier will achieve the same result as this method, so it is not necessary to implement each multiplier's algorithm. The systolic multiplier and Strassen multiplier with the same size will have results of identical sizes, while the Strassen multiplier with the same bandwidth will have a resulting matrix with smaller dimensions.

3.1.4 Monitor

The monitor collects output data for each multiplier through the virtual interface and creates output packets when the respective multiplier's driver asserts that an output is valid. The monitor sends these packets to the scoreboard through FIFO's.

3.1.5 Scoreboard

The scoreboard collects design output and reference data for each multiplier. Upon receiving both a packet from each, the scoreboard compares the data to verify the correctness of the design. Upon mismatched data, the test ends. Experimentally, the designs function without fail. The scoreboard reads a configuration file that details the number of multiplications, so it can determine the end of the run phase when all designs have completed the specified amount of multiplications.

3.1.6 UVM Phasing

Phasing is an essential feature of UVM methodology where different phases collect, run and process data to avoid run-time conflicts. Phases are a group of callback methods which could be tasks or functions. All the phases in UVM can be grouped into three main categories which are discussed below [24].

3.2 Metrics

The UVM environment keeps track of metrics during the run phase and generates a metrics report during the report phase

3.2.1 Verification

For the verification of the multiplier designs. The scoreboard keeps track of the number of matched and mismatched outputs. During the report phase, these metrics are reported.

3.2.2 Performance

Each multiplier has an associated performance counter in the UVM environment that is enabled and counted by the respective driver. The counters are increment for each clock cycle that the multiplier is active and completing operations. Upon finishing of the test, the environment reports all performance counter values.

Chapter 4

Systolic Array Design

The design of the systolic array starts by creating the MAC unit, which is the fundamental building block of the array. After which, the array can be designed to accommodate for matrix sizes of the application. This paper creates systolic arrays with dimension sizes 2, 4, 16, 64, and 128. The design features a parameterized data width and matrix size.

4.1 MAC Unit Design

The MAC unit's purpose is to compute a dot product, which can be divided into stages that store the element data, compute the product, and accumulate the result as seen in Figure 4.1. Each stage of the pipeline is controlled by a corresponding enable signal. The diagram in Figure 4.1 shows the implementation of the MAC unit for this paper.

The pin level details of all the MAC signals are outlined in Table 4.1.

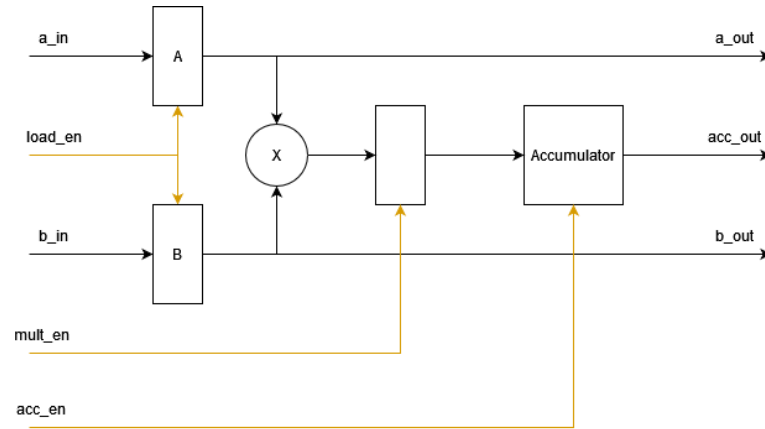


Figure 4.1: MAC Unit Diagram

Table 4.1: MAC signal details

Port	Width	Direction	Description
clk	1	Input	Clock
reset	1	Input	Active high synchronous reset
a_in	8	Input	Element from Matrix A
b_in	8	Input	Element from Matrix B
load_en	1	Input	Enable loading elements
mult_en	1	Input	Enable multiplication
acc_en	1	Input	Enable accumulation
acc_out	32	Output	Output of accumulator
a_out	8	Output	Output of element from Matrix A
b_out	8	Output	Output of element from Matrix A

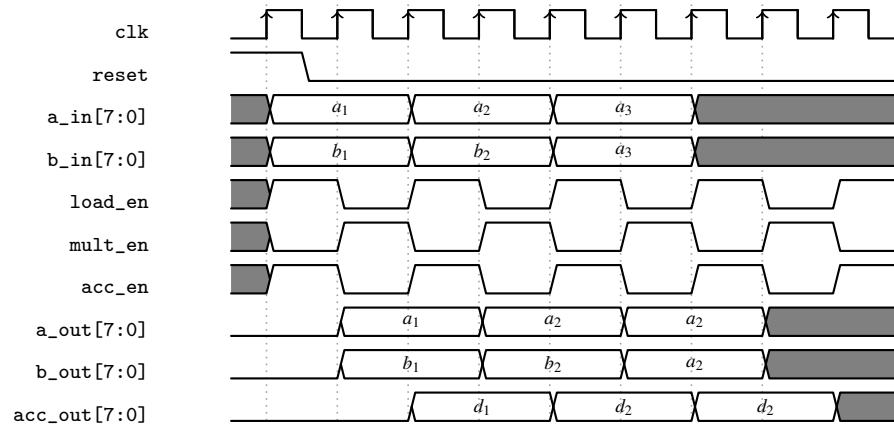


Figure 4.2: MAC Unit Timing Diagram

4.1.1 Pipelining Methodology

The pipelining design separates each operation to a flop to reduce the critical path delay. The MAC unit has 3 stages, where the element loading stage and the accumulation stage can be performed in parallel. Thus, the load and accumulator enable signals are asserted identically. This can be visualized in the timing diagram Figure 4.2

4.2 Array of MAC Units

The systolic array is comprised of a 2 dimensional array of the MAC units. Each MAC unit propagates data in the East and South direction, where the data from the first matrix operand is sent East and the data from the second matrix operand is sent South. In this fashion, it can be visualized that the multiplication being performed for each MAC unit follows the traditional method of matrix multiplication. Each MAC unit computes the dot product of the row from the first matrix and the column of the second matrix. This can be seen in Figure 4.3

All MAC unit enable signals are controlled together by the top level inputs to the systolic array. Data from the first matrix (matrix A) is provided to the array by supplying an element

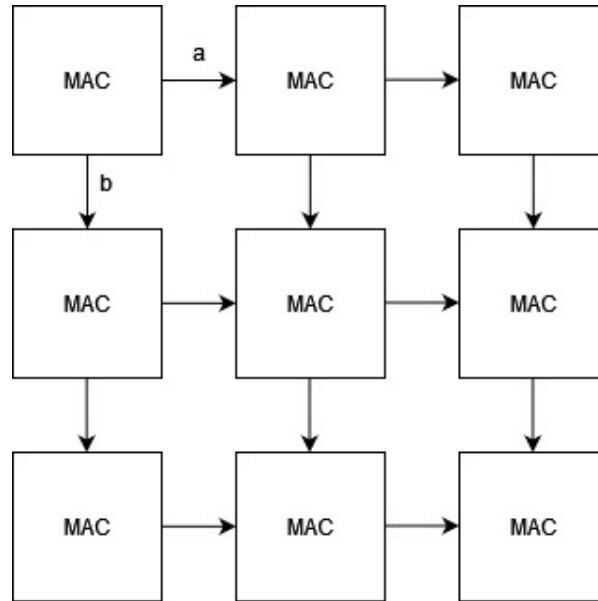


Figure 4.3: MAC Array

from each row, likewise the data from the second matrix (matrix B) contains an element from each column of the matrix. The output of the systolic array is a collection of each accumulator from all the MAC units, allowing the hardware to deliver each element of the product matrix in parallel.

The pin level details of all the Systolic Array signals are outlined in Table 4.2.

Table 4.2: Systolic Array signal details

Port	Width	Direction	Description
clk	1	Input	Clock
reset	1	Input	Active high synchronous reset
a_in	8*Size	Input	Elements from each row of Matrix A
b_in	8*Size	Input	Elements from each column of Matrix B
load_en	1	Input	Enable loading elements
mult_en	1	Input	Enable multiplication
acc_en	1	Input	Enable accumulation
d_out	32*Size ²	Output	Output of all accumulators

4.2.1 Design Features

- The design is fully parameterized for matrix size and data width
- Signed arithmetic is supported
- Flattened array data to the design's inputs
- The clock frequency can be easily programmed by software.
- The design is completely synthesizable.

Chapter 5

Strassen Multiplier Design

The design of the Strassen multiplier is hierarchical, because of the recursive and divide and conquer nature of the Strassen Algorithm. The multiplier is made up of layers that perform the algorithm on matrices of different sizes. The atomic unit of this hierarchy performs the multiplications at the word level, while the layers recursively spread the multiplications to these atomic units. The layers perform the larger matrix level additions and subtractions. All operations of the Strassen Multiplier are pipelined.

5.1 Hierarchy

The algorithm divides the matrix into 4 quadrants and performs arithmetic operations on the quadrants. For a large matrix, a quadrant will be a submatrix, that will be recursively subdivided further until each quadrant is an element. Then the multiplications are performed, and the larger matrices are reformed. The process of the recursive algorithm can be seen with Figure 5.1

This algorithm can be implemented by using a hierarchy of the Strassen algorithm, where each level in the hierarchy is responsible for a different matrix size. This can be seen in Figure

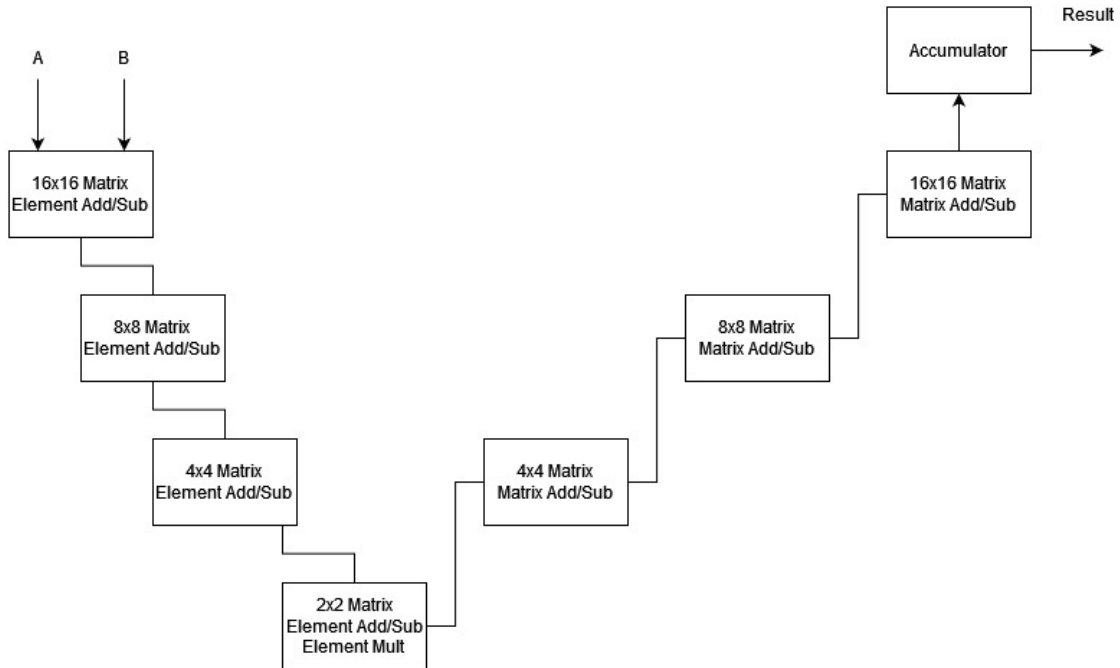


Figure 5.1: Recursive Algorithm

5.3. Each upper layer has four instances of a smaller layer, where each smaller layer is given a quadrant of the upper layers input matrix. The lowest layer being the atomic module that performs the element multiplications and returns the result upward.

5.2 Strassen Unit

The Strassen Unit performs the Strassen algorithm for a 2x2 matrix and is the only module where multiplication is performed. The unit performs the following element level operations
 Partition each matrix into four sub-matrices. The algorithm can be seen by the multiplication of matrices A and B:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad (5.1)$$

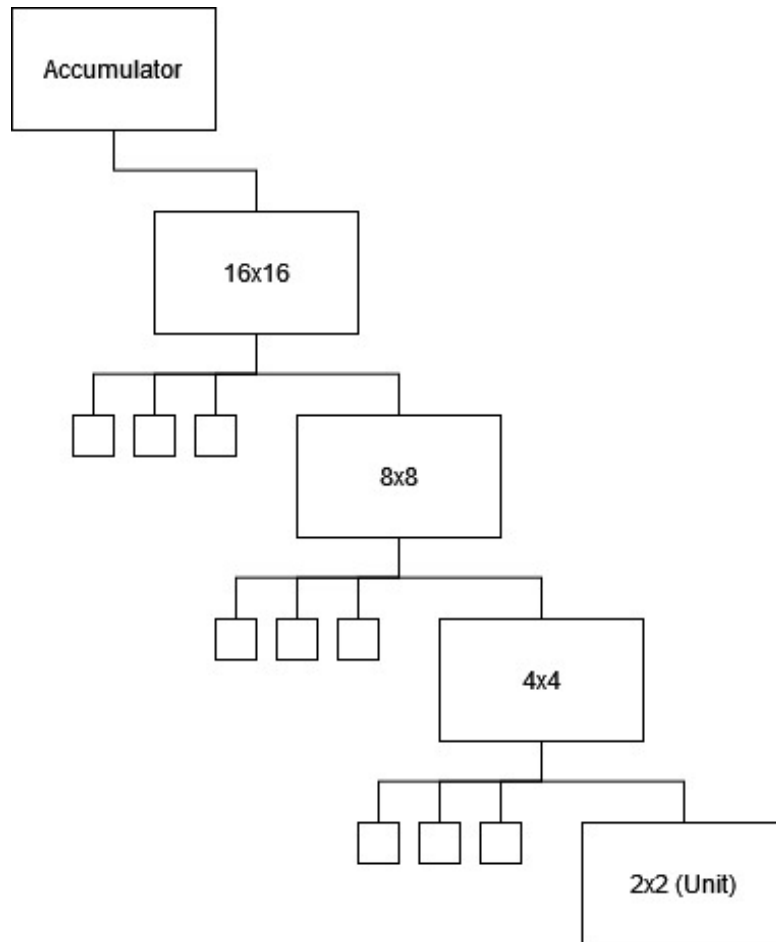


Figure 5.2: Recursive Algorithm

First, seven intermediate products (as opposed to the usual eight) are computed using specific combinations of the elements from matrices A and B:

1. $P_1 = a(f - h)$

2. $P_2 = (a + b)h$

3. $P_3 = (c + d)e$

4. $P_4 = d(g - e)$

5. $P_5 = (a + d)(e + h)$

6. $P_6 = (b - d)(g + h)$

7. $P_7 = (a - c)(e + f)$

The elements of the resulting matrix C can then be computed using these intermediate products:

1. $C_{11} = P_5 + P_4 - P_2 + P_6$

2. $C_{12} = P_1 + P_2$

3. $C_{21} = P_3 + P_4$

4. $C_{22} = P_1 + P_5 - P_3 - P_7$

These element level operations are performed in a pipelined fashion, which can be seen in Figure 5.3.

The pin level details of the Strassen unit's signals are outlined in Table 5.1. The data width is determined by the hierarchy and is discussed in section 5.3.1.

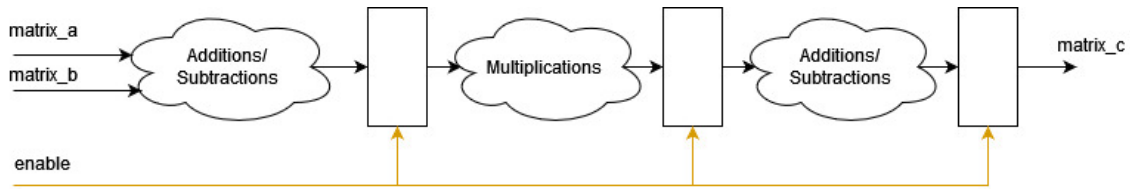


Figure 5.3: Strassen Unit Diagram

Table 5.1: Wishbone signal details

Port	Width	Direction	Description
clk	1	Input	Clock
reset	1	Input	Active high synchronous reset
enable	1	Input	Enables pipeline
matrix_a	4*Datawidth	Input	Flattened Input Matrix A
natrix_b	4*Datawidth	Input	Flattened Input Matrix B
matrix_c	128	Ouput	Flattened Matrix Result

5.2.1 Pipelining Methodology

This pipelining performs only one arithmetic operation each clock cycle in an effort to increase maximum clock frequency. While it could be possible to perform the element level additions/subtractions and multiplications in the same cycle, it would likely become the critical path of the design. The operation of the pipeline is simple, where each register is updated at each clock cycle, such that each stage in the pipeline contains data for a separate matrix multiplication. This can be seen with Figure 5.4.

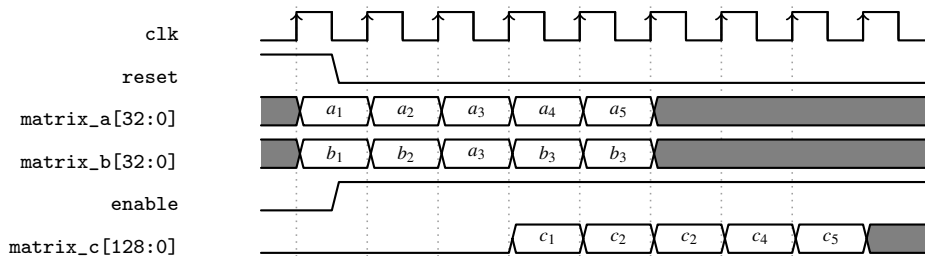


Figure 5.4: Strassen Unit Timing Diagram

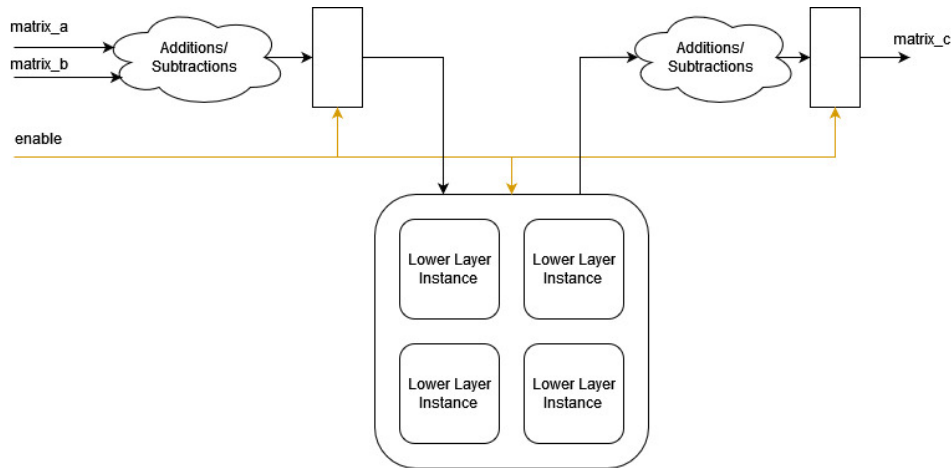


Figure 5.5: Strassen Layer Diagram

Table 5.2: Strassen Layer signal details

Port	Width	Direction	Description
clk	1	Input	Clock
reset	1	Input	Active high synchronous reset
enable	1	Input	Enables pipeline
matrix_a	8*Size	Input	Flattened Input Matrix A
matrix_b	8*Size	Input	Flattened Input Matrix B
matrix_c	32*Size ²	Oupput	Flattened Matrix Result

5.3 Strassen Layers

To build a matrix multiplier using the Strassen algorithm, a hierarchy using the Strassen unit has to be formed. These upper layers divide their input matrices into quadrants and send them to lower layers. A layer can be seen in Figure 5.5

The pin level details of a Strassen layer's signals are outlined in Table 5.2.

5.3.1 Data Precision

The recursive nature of the algorithm creates a precision concern due to the consecutive additions performed. With the assumption that every addition/subtraction operation requires

an additional bit to prevent overflow, each lower layer is instantiated with a larger data width than the preceding layer. This results in the Strassen Unit's data width to be dependent on the number of layers in the hierarchy.

5.3.2 Design Features

- The design is parameterized for data width
- Hierarchy is generated through a script
- Signed arithmetic is supported
- Flattened array data to the design's inputs
- The clock frequency can be easily programmed by software
- The design is completely synthesizable

Chapter 6

Results and Discussion

Provided are the results from simulations and synthesis. Performance from simulations are measured for cases of immediate accumulator reset, no accumulator reset, and variable matrix data dimensions. Synthesis data provided is for a 28 nanometer process. Trends are constructed from the data to make predictions for matrices of larger sizes.

6.1 Immediate Accumulator Reset

6.1.1 4×4

The following results showcase performance for these matrices:

- 4×4 Systolic Array
- 2×2 Strassen Algorithm for matched bandwidth
- 4×4 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.1

Table 6.1: 4x4 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	1100	417	57
100	2200	820	107
250	5500	2016	257
500	11000	4016	507
1000	22000	8016	1007
5000	110000	40016	5007

This data is representing with Fig 6.1

Full Matrix Input Data, Accumulators Reset Every Multiplication

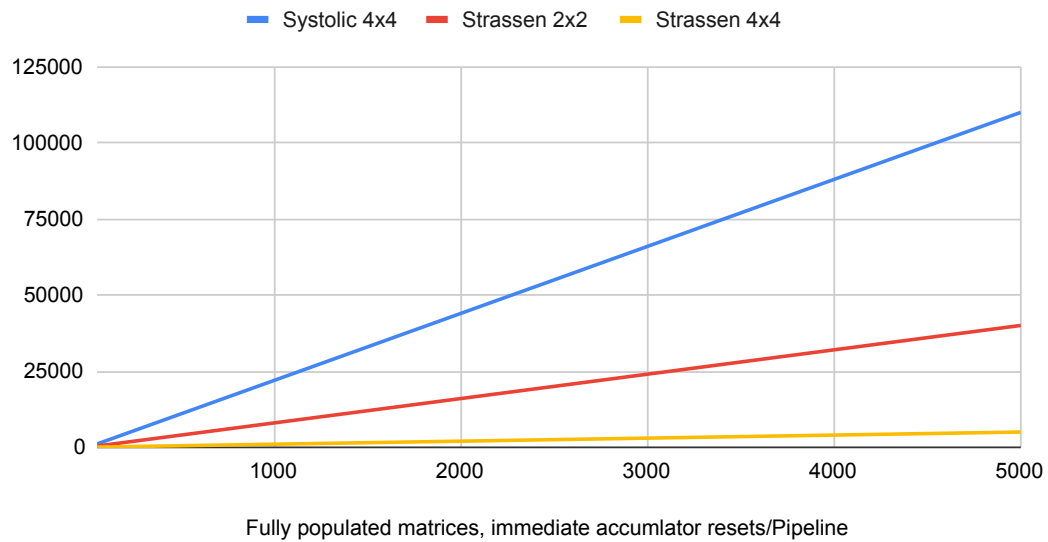


Figure 6.1: Plot of Performance Data for 4x4

6.1.2 16×16

The following results showcase performance for these matrices:

- 16×16 Systolic Array
- 4×4 Strassen Algorithm for matched bandwidth
- 16×16 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.2

Table 6.2: 16×16 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	4700	425	61
100	9400	825	111
250	23500	2250	351
500	47000	4025	511
1000	94000	8025	1011
5000	470000	40025	5011

This data is representing with Fig 6.2

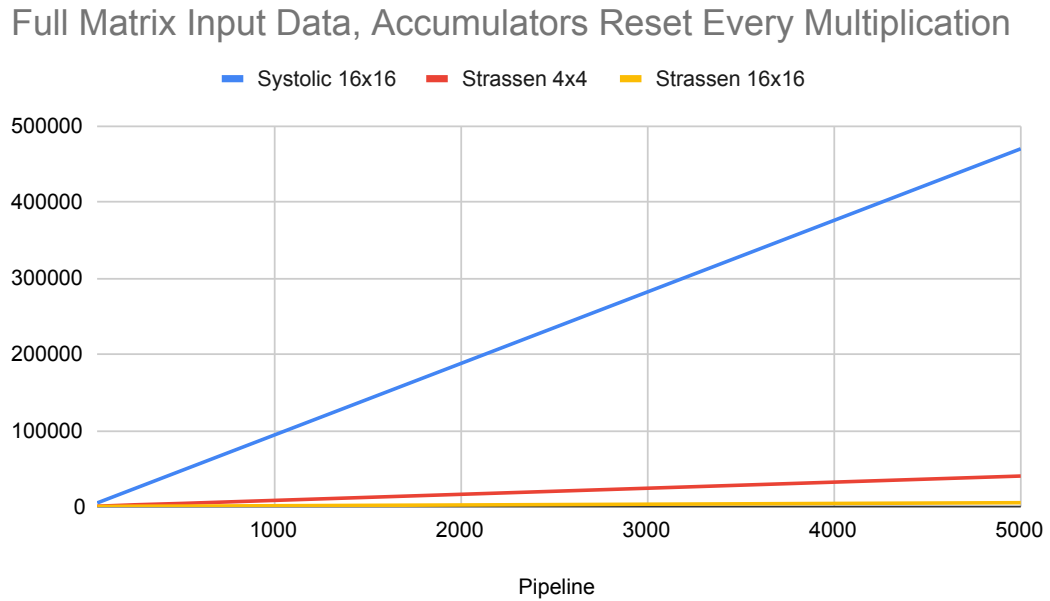


Figure 6.2: Plot of Performance Data for 16x16

6.1.3 64x64

The following results showcase performance for these matrices:

- 64x64 Systolic Array
- 8x8 Strassen Algorithm for matched bandwidth
- 64x64 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.3

Table 6.3: 64×64 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	19100	433	65
100	38200	833	115
250	95500	2033	265
500	191000	4033	515
1000	382000	8033	1015
5000	1910000	40033	5015

This data is representing with Fig 6.3

Full Matrix Input Data, Accumulators Reset Every Multiplication

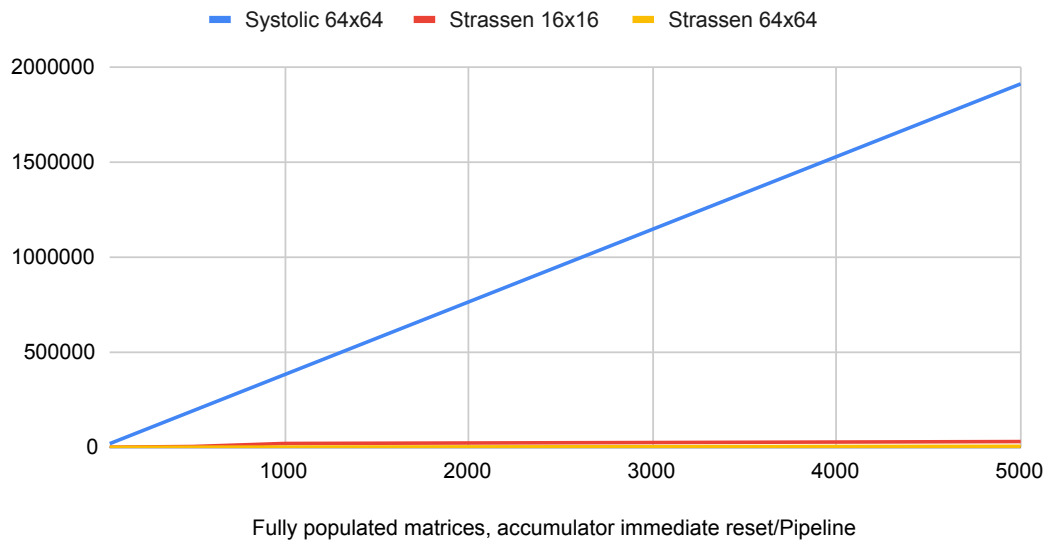


Figure 6.3: Plot of Performance Data for 64×64

6.1.4 Trends

The trend in performance of the multipliers can be observed with Table 6.3. The adjusted data is taken from the normalized results from the 5000 multiplication count simulations.

Table 6.4: Synthesis results

	Normalized Clock Cycles		
Size	Systolic Array	Strassen (bandwidth)	Strassen (size)
4	22	8.0032	1.0014
16	94	8.005	1.0022
64	382	8.0066	1.003

6.2 No Accumulator Reset

6.2.1 4×4

The following results showcase performance for these matrices:

- 4×4 Systolic Array
- 2×2 Strassen Algorithm for matched bandwidth
- 4×4 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.5

Table 6.5: 4x4 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	422	417	57
100	822	820	107
250	2022	2016	257
500	4022	4016	507
1000	8022	8016	1007
5000	40022	40016	5007

This data is representing with Fig 6.4

Full Matrix Data Input, No Accumulator Reset

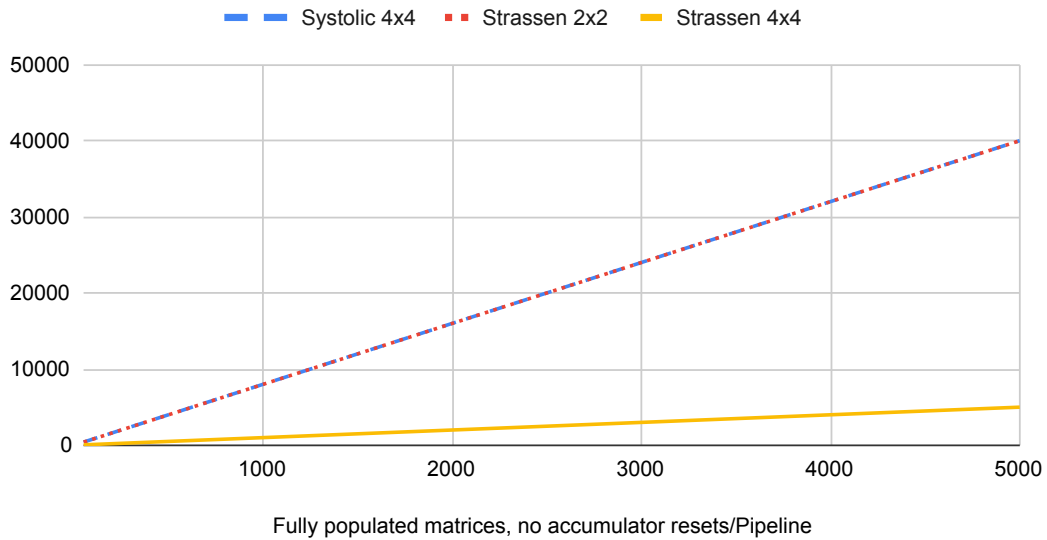


Figure 6.4: Plot of Performance Data for 4x4

6.2.2 16×16

The following results showcase performance for these matrices:

- 16×16 Systolic Array
- 4×4 Strassen Algorithm for matched bandwidth
- 16×16 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.6

Table 6.6: 16×16 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	1982	425	61
100	3582	825	111
250	8382	2250	351
500	16382	4025	511
1000	32382	8025	1011
5000	160382	40025	5011

This data is representing with Fig 6.5

Full Matrix Input Data, No Accumulator Reset

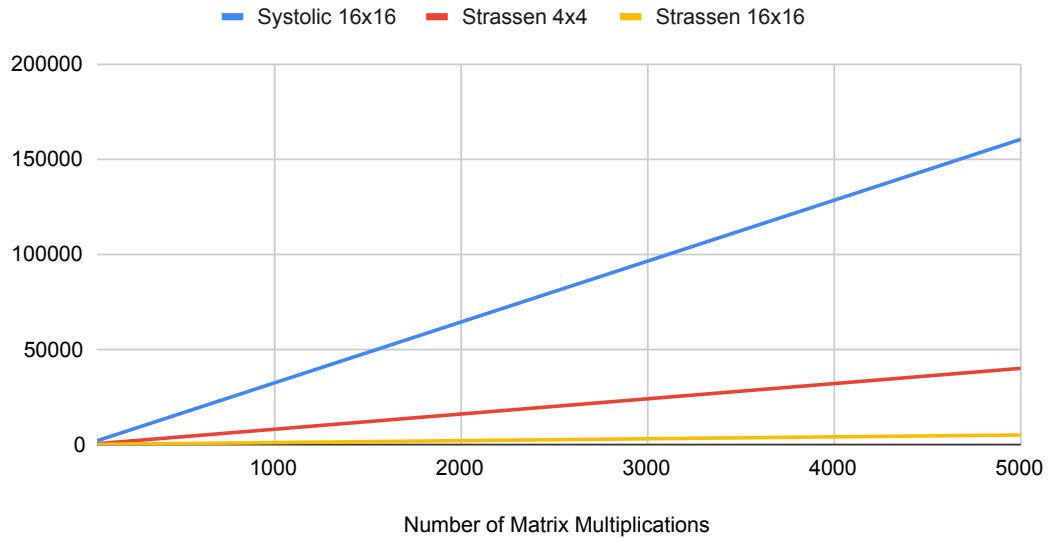


Figure 6.5: Plot of Performance Data for 16x16

6.2.3 64x64

The following results showcase performance for these matrices:

- 64x64 Systolic Array
- 8x8 Strassen Algorithm for matched bandwidth
- 64x64 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.7

Table 6.7: 64×64 Performance

Multiplications	Clock Cycles		
	Systolic Array	Strassen (bandwidth)	Strassen (size)
50	6782	433	65
100	13182	833	115
250	32382	2033	265
500	64382	4033	515
1000	320382	8033	1015
5000	640382	40033	5015

This data is representing with Fig 6.6

Systolic 64x64, Strassen 16x16 and Strassen 64x64

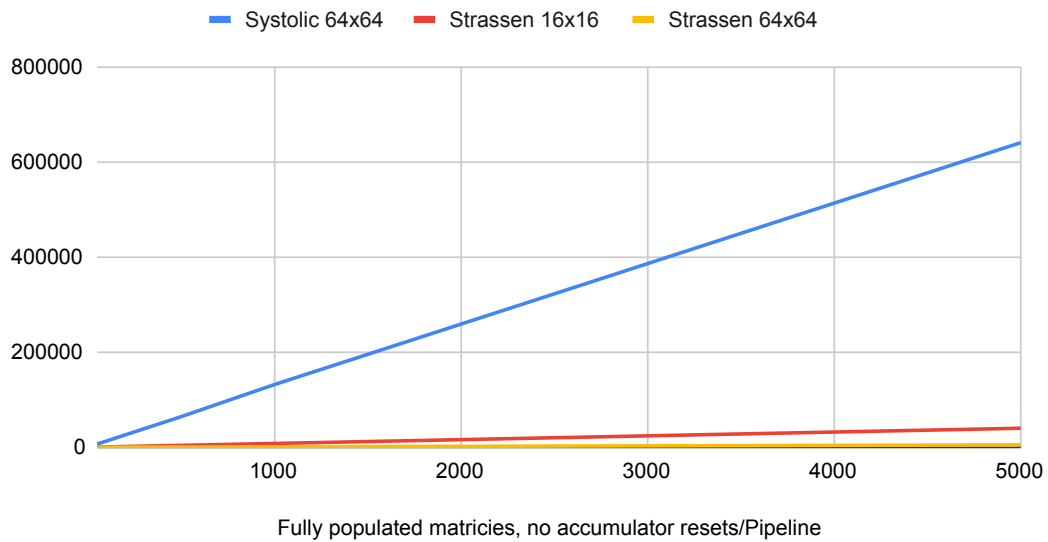


Figure 6.6: Plot of Performance Data for 64×64

6.2.4 Trends

The trend in performance of the multipliers can be observed with Table 6.7. The adjusted data is taken from the normalized results from the 5000 multiplication count simulations.

Table 6.8: Synthesis results

	Normalized Clock Cycles		
Size	Systolic Array	Strassen (bandwidth)	Strassen (size)
4	8.0044	8.0032	1.0014
16	32.0764	8.005	1.0022
64	128.0764	8.0066	1.003

6.3 Variable Matrix Input Dimensions

Performance is collected for the 64×64 multipliers in scenarios where the input matrix is variable seen in Table 6.9. Each matrix dimensions is run for 5000 multiplications with no resets.

Table 6.9: Variable Matrix Input Dimension

	Clock Cycles		
Dimension	Systolic Array	Strassen (bandwidth)	Strassen (size)
4	40022	40033	5015
16	160382	40033	5015
64	640382	40033	5015

6.4 Physical Characteristics

It is important to understand performance in the context of physical characteristics to better conclude trade-offs.

6.4.1 Cell Area

The pre-scan cell area characteristics for the systolic multiplier can be seen with Table 6.10

Table 6.10: Systolic Cell Area Pre-scan

Size	Cell Area (μm^2)	Cells
2	5360.151	1718
4	21508.200	6922
8	86124.827	27770
16	344487.873	110913
64	5452455.720	1727545

The pre-scan cell area characteristics for the Strassen multiplier can be seen with Table 6.11

Table 6.11: Strassen Cell Area Pre-scan

Size	Cell Area (μm^2)	Cells
2	13404.825	4071
4	123069.232	37277
8	991015.214	292547
16	1545123.450	514354
64	7485490.560	2030723

The post-scan cell area characteristics for the systolic multiplier using the Synopsis Design Compiler and IC Compiler synthesis tools can be seen with Table 6.12

Table 6.12: Systolic Cell Area Post-scan

Size	Design Compiler		IC Compiler	
	Cell Area (μm^2)	Cells	Cell Area (μm^2)	Cells
2	5938.583	1705	6842.318	126
4	23779.490	6835	27128.092	338
8	94892.031	27165	109503.531	1702
16	381316.382	109490	457785.259	12565
64	6098800.737	1755227	457785.259	9900

The post-scan cell area characteristics for the Strassen multiplier using the Synopsis Design Compiler and IC Compiler synthesis tools can be seen with Table 6.13. Multiplier sizes from 8, 16, and 64 were unable to be synthesized using the IC Compiler, so some data is not available

Table 6.13: Strassen Cell Area Post-scan

Size	Design Compiler		IC Compiler	
	Cell Area (μm^2)	Cells	Cell Area (μm^2)	Cells
2	14785.080	3922	14125.831	3663
4	135051.357	36404	153409.448	3685
8	1087093.068	290825	N/A	N/A
16	1545123.450	514354	N/A	N/A
64	7485490.560	2030723	N/A	N/a

A trend for cell area from the Design Compiler data can be observed with the plot in Fig 6.7. The plot uses the Pre-scan data to better represent the area of the multiplier without the included scan cells.

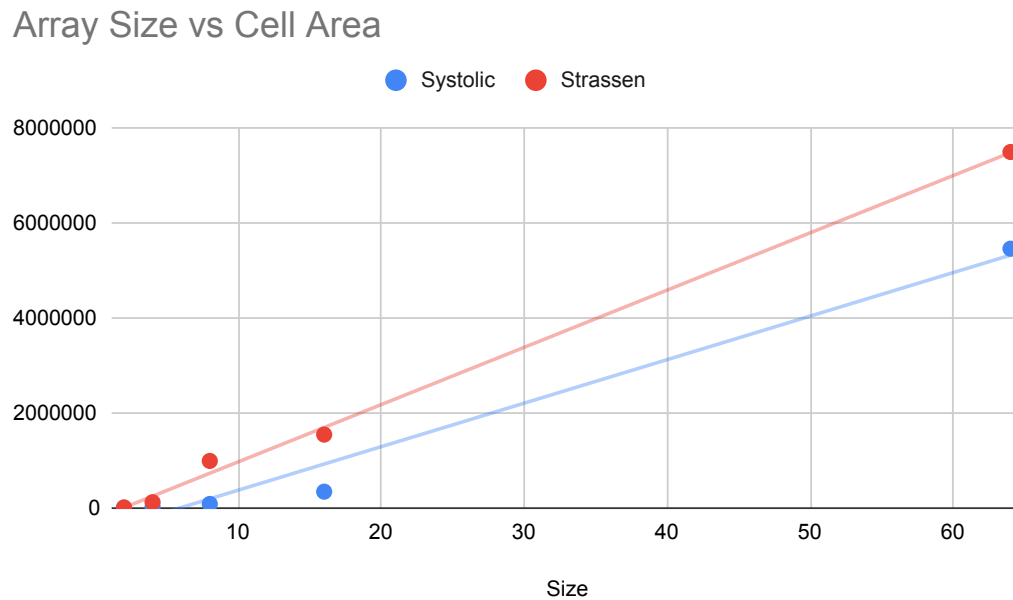


Figure 6.7: Plot of Size vs Cell Area for Pre-scan

6.4.2 Timing

The slack and prop delay of the multipliers using Synopsis Design Compiler can be seen with Table 6.14

Table 6.14: Timing

Size	Systolic		Strassen	
	Slack (ns)	Delay (ns)	Slack (ns)	Delay (ns)
2	4.923	4.946	3.284	6.594
4	4.903	4.983	0.004	11.392
8	4.269	6.672	0.002	13.84
16	3.914	7.735	0.001	15.12
64	2.523	12.454	0.001	16.451

A trend for slack can be observed with the plot in Fig 6.8.

Size vs Slack (ns)

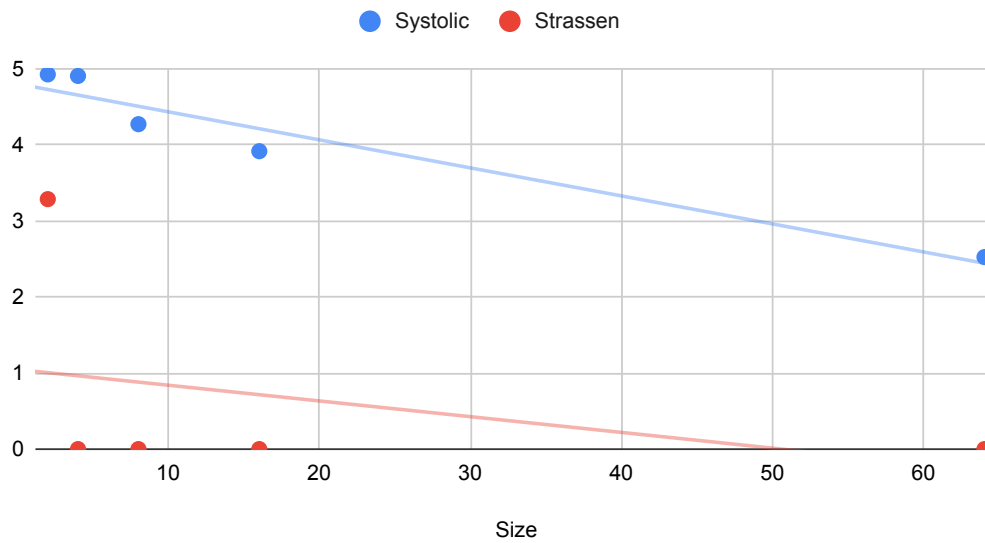


Figure 6.8: Plot of Size vs Cell Area for Pre-scan

The slack and prop delay of the multipliers using Synopsis IC Compiler can be seen with

Table 6.15. Multiplier sizes from 8, 16, and 64 were unable to be synthesized using the IC Compiler, so some data is not available.

Table 6.15: Timing

	Systolic		Strassen	
Size	Slack (ns)	Delay (ns)	Slack (ns)	Delay (ns)
2	5.6571	4.2603	4.8188	5.1556
4	5.2863	4.6577	3.9701	6.0577
8	5.1966	4.7034	N/A	N/A
16	5.1249	4.9814	N/A	N/A
64	5.0613	5.2131	N/A	N/A

6.4.3 Power

Dynamic and leakage power of the multipliers using Design Compiler can be seen with Table 6.16

Table 6.16: Design Compiler Power

	Systolic		Strassen	
Size	Dynamic (uW)	Leakage (uW)	Dynamic (uW)	Leakage (uW)
2	199.537	20.255	530.61	51.598
4	773.2	80.95	4575.8	1552.9
8	3000	323.663	35825.7	13406.5
16	11577.8	1291.3	125153.13	92540
64	177469.2	20527	391638.2	1910000

A trend for dynamic power can be observed with the plot in Fig 6.9.

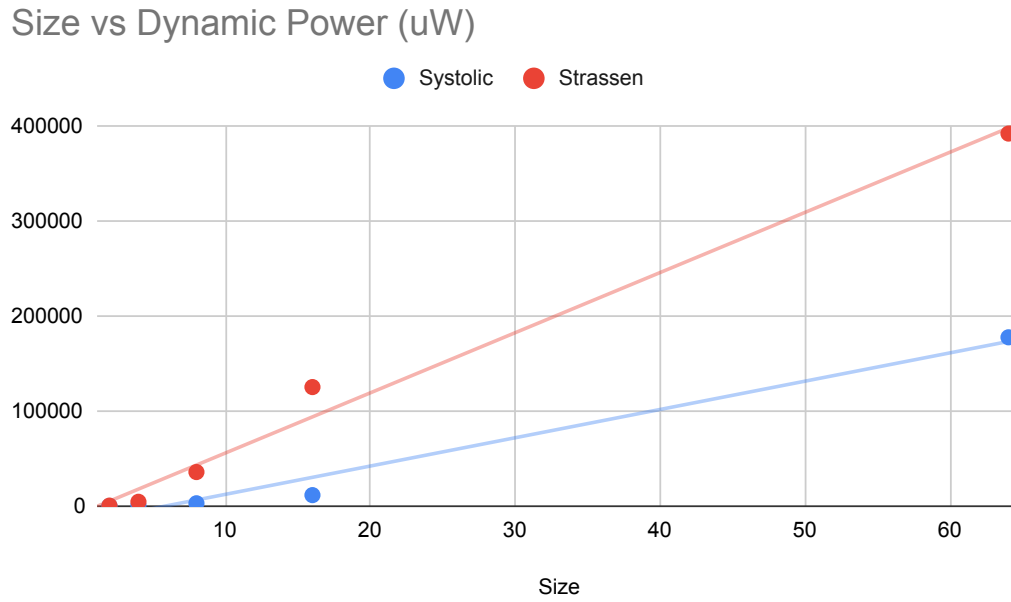


Figure 6.9: Plot of Size vs Dynamic Power

A trend for leakage power can be observed with the plot in Fig 6.10.

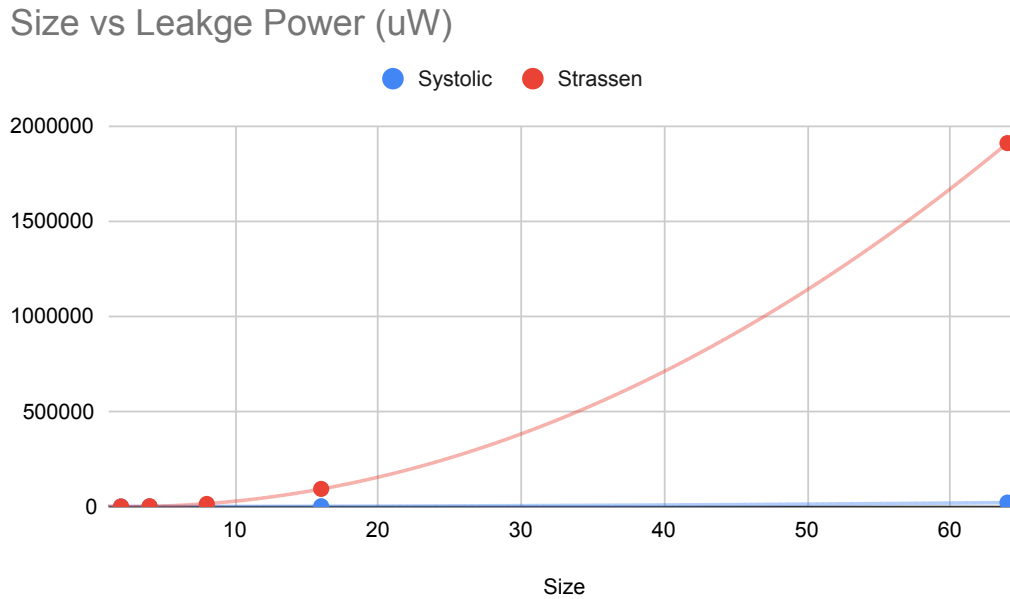


Figure 6.10: Plot of Size vs Leakage Power

Dynamic and leakage power of the multipliers using IC Compiler can be seen with Table 6.17. The tool was unable to complete for some sizes, so the data is omitted.

Table 6.17: IC Compiler Power

Size	Systolic		Strassen	
	Dynamic (μW)	Leakage (μW)	Dynamic (μW)	Leakage (μW)
2	190.62	89.231	483.347	284.85
4	1084.10	603.038	11391.500	4614.60
8	8335.60	1843.000	N/A	N/A

6.5 Area over Relative Performance

The most important result is the relationship between area and performance. This can be seen with Table 6.18 which shows the cell area over the relative performance of each multiplier for the no accumulator reset simulations.

Table 6.18: Performance over Area

Size	Systolic	Strassen (bandwidth)	Strassen (size)
4	0.5808542221	0.1015282355	0.8114147934
16	0.009049830422	0.008084915403	0.06457767691
64	0.0001431985897	0.001668520707	0.01331922023

This can be visualized using plot in Fig 6.11.

Size vs Relative Performance over Area (no Accumulator Reset)

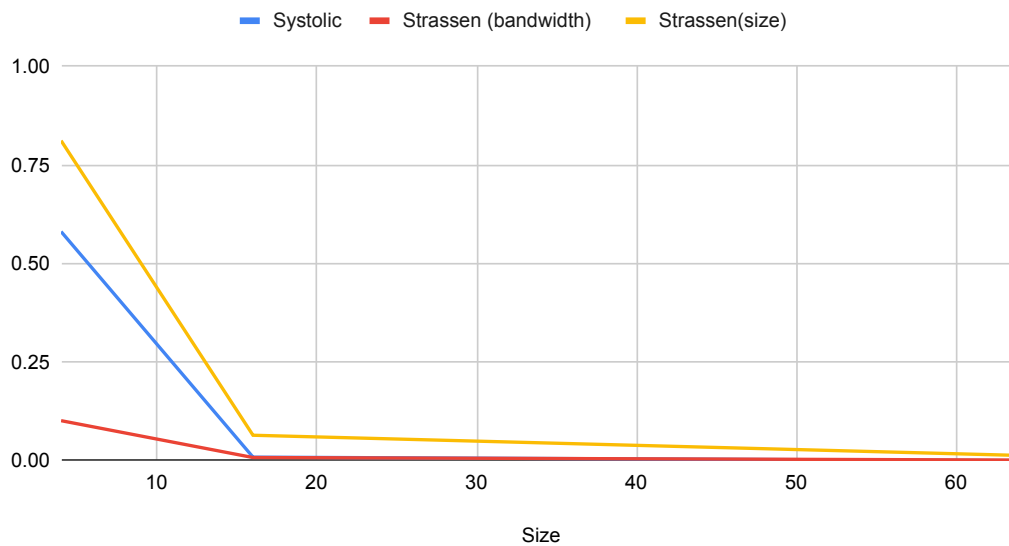


Figure 6.11: Plot of Size vs Performance over Area

6.6 Observations

- The Strassen algorithm generally performs better than the systolic array in both matched bandwidth and array size, requiring less clock cycles to complete.
- The systolic array is similar to the Strassen algorithm with matched bandwidth only for a dimension size of 4 for the ideal case of no accumulator resets.
- The systolic array's performance is dependent on the accumulator resets and the size of input matrices.
- The Strassen algorithm of equal size requires noticeably more area than the systolic array
- The Strassen algorithm of equal bandwidth requires more area than the systolic array
- To this paper's specific design, the Strassen algorithm has almost no slack for a given hierarchical layer. This may create an issue with bottle-necking the master clock frequency.
- The systolic array has more slack than the Strassen algorithm, allowing the possibility of increasing the system's clock frequency. This opens the possibility for improved performance.
- The Strassen algorithm has an increasingly higher propagation delay, which will be a major concern for large matrix sizes.
- The Strassen algorithm and the systolic area have very similar performance to area ratios, which may be different given the systolic array can push higher clock frequencies.
- The systolic array has much lower power consumption than the Strassen algorithm, which makes it the biggest defining difference between the two methods.

Chapter 7

Conclusion

The two matrix multiplication approaches, systolic execution and Strassen algorithm, were successfully implemented and verified. The designs were simulated over several different sizes and had several metrics recorded. The design were fully synthesizable and their physical characteristics were also recorded and analyzed.

The Strassen algorithm had significantly better performance than the systolic array in nearly all scenarios, but also had significantly larger cell area. By comparing the relative performance to area for different sizes of each multiplier, it became clear that the performance advantage of the algorithm was made insignificant due to the extremely high area. It is also important to note that the systolic array had much higher slack and lower propagation delay, allowing it the ability to use a higher clock frequency to promote better performance. Additionally, the systolic array had much lower power consumption.

In many AI accelerators, where the size of the die or multiplier is massive, it is likely that the Strassen algorithm is not a suitable design. A very large multiplier of size 256 or 512 would produce significant heat, which is a disadvantage of the Strassen algorithm. For applications where the processing must be done on the edge or in smaller devices with area constraints, the

systolic would likely be favorable for its smaller area. In many applications, it is important to push the clock frequency, where the systolic array has an advantage for its smaller area and routing thus lower propagation delay and increased slack. Although the Strassen algorithm has very attractive performance, other characteristics and complications would likely deter an engineer from incorporating it into a chip.

7.1 Future Work

Analysis of the Strassen algorithm against the systolic area is not comprehensive. This paper analyzes sizes up to 64 by 64, which is likely not the ideal size for AI applications. Sizes of 256 and larger must be studied to give better evidence to claims for using one method over the other. Additionally, it would be important to benchmark the designs with their highest possible clock frequencies to produce more meaningful results.

It is also important to study the Winograd and the Karstadt-Schwartz algorithms which share the same asymptotic nature of the Strassen algorithm with a reduced number of arithmetic operations [25]. This would significantly decrease the total area, power, and propagation delays of the design. There is a likelihood that these algorithms could have an advantage over the systolic array with a lower area.

Bibliography

- [1] M. Rizwan, E. Jung, Y. Park, J. Choi, and Y. Kim, “Optimization of Matrix-Matrix Multiplication Algorithm for Matrix-Panel Multiplication on Intel KNL,” in *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*, 2022, pp. 1–7.
- [2] F. Ishiguro, T. Katagiri, S. Ohshima, and T. Nagai, “Performance Evaluation of Accurate Matrix-Matrix Multiplication on GPU Using Sparse Matrix Multiplications,” in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, 2020, pp. 178–184.
- [3] X. Liao, S. Li, W. Yu, and Y. Lu, “Parallel matrix multiplication algorithms in Supercomputing,” in *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 2021, pp. 1–4.
- [4] M. Shanmugakumar, V. S. M. Srinivasavarma, and S. Noor Mahammad, “Energy Efficient Hardware Architecture for Matrix Multiplication,” in *2020 IEEE 4th Conference on Information and Communication Technology (CICT)*, 2020, pp. 1–6.
- [5] J. Wang, Z. Hao, and H. Li, “A High-Performance Multi-array Accelerator for Large-Scale Floating-Point Matrix Multiplication,” in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, 2019, pp. 1048–1052.

- [6] “IEEE Standard for Universal Verification Methodology Language Reference Manual,” *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, 2020.
- [7] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey,” *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.
- [8] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, “Fast Deep Neural Network Training on Distributed Systems and Cloud TPUs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2449–2462, 2019.
- [9] A. Elnaggar, M. Heinzinger, C. Dallago, G. Rehawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, M. Steinegger, D. Bhowmik, and B. Rost, “ProtTrans: Toward Understanding the Language of Life Through Self-Supervised Learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 7112–7127, 2022.
- [10] Z. Yao, K. Huang, H. Shen, and Z. Ming, “Deep Neural Network Acceleration With Sparse Prediction Layers,” *IEEE Access*, vol. 8, pp. 6839–6848, 2020.
- [11] J. Lee, S. Kang, J. Lee, D. Shin, D. Han, and H.-J. Yoo, “The Hardware and Algorithm Co-Design for Energy-Efficient DNN Processor on Edge/Mobile Devices,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 10, pp. 3458–3470, 2020.
- [12] H. Wu, S. Huang, Y. Zeng, S. Ying, and H. Lu, “A Lightweight Deep Neural Network for Wideband Power Amplifier Behavioral Modeling,” in *2021 7th International Conference on Computer and Communications (ICCC)*, 2021, pp. 1294–1298.
- [13] O. N. Tay and P. D. Noakes, “A neural network processor incorporating multiple on-chip cache memories,” in *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, 1991, pp. 2222–2227 vol.3.

-
- [14] T. Gaurav, A. Bhatt, and R. Parekh, "Design and Implementation of low power RISC V ISA based coprocessor design for Matrix multiplication," in *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2021, pp. 189–195.
- [15] X. Liao, S. Li, W. Yu, and Y. Lu, "Parallel matrix multiplication algorithms in Supercomputing," in *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 2021, pp. 1–4.
- [16] R. Xu, S. Ma, Y. Wang, Y. Guo, D. Li, and Y. Qiao, "Heterogeneous Systolic Array Architecture for Compact CNNs Hardware Accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2860–2871, 2022.
- [17] N. A. S. Alwan, "A fully pipelined systolic array for sinusoidal sequence generation," *IEEE Transactions on Computers*, vol. 55, no. 5, pp. 636–639, 2006.
- [18] S. Choi, S. Park, J. Park, J. Kim, G. Koo, S. Hong, M. K. Yoon, and Y. Oh, "SAVector: Vectored Systolic Arrays," *IEEE Access*, vol. 12, pp. 44 446–44 461, 2024.
- [19] S. Ortega-Cisneros, "Design and Implementation of an NoC-Based Convolution Architecture With GEMM and Systolic Arrays," *IEEE Embedded Systems Letters*, vol. 16, no. 1, pp. 49–52, 2024.
- [20] C. Misra, S. Bhattacharya, and S. K. Ghosh, "Stark: Fast and Scalable Strassen's Matrix Multiplication Using Apache Spark," *IEEE Transactions on Big Data*, vol. 8, no. 3, pp. 699–710, 2022.
- [21] P. J. Guerra and M. J. MunozBouzo, "On a Theorem of Strassen for Vector-Valued Measures," *Quarterly Journal of Mathematics*, vol. 53, no. 3, pp. 285–293, 2002.

-
- [22] S. Oh and B.-R. Moon, “Automatic Reproduction of a Genius Algorithm: Strassen’s Algorithm Revisited by Genetic Search,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 246–251, 2010.
- [23] “IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language,” *IEEE STD 1800-2009*, pp. 1–1285, 2009.
- [24] K. Salah, “A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities,” in *2014 9th International Design and Test Symposium (IDT)*, 2014, pp. 94–99.
- [25] D. Wu, X. Fan, W. Cao, and L. Wang, “SWM: A High-Performance Sparse-Winograd Matrix Multiplication CNN Accelerator,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 5, pp. 936–949, 2021.

Appendix I

Source Code

I.1 Multiply Accumulate Unit

```
1 module mac(  
2     clk ,  
3     reset ,  
4     a_in ,  
5     b_in ,  
6     mult_en ,  
7     acc_en ,  
8     load_en ,  
9     acc_out ,  
10    a_out ,  
11    b_out  
12    );  
13
```

```
14  parameter DATAWIDTH = 8;
15  parameter ACCWIDTH = 32;
16
17  input [DATAWIDTH-1:0] a_in , b_in;
18  input clk , mult_en , acc_en , load_en , reset;
19  output reg signed [ACCWIDTH-1:0] acc_out;
20  output reg signed [DATAWIDTH-1:0] a_out , b_out;
21
22  reg signed [(DATAWIDTH*2)-1:0] mult;
23
24  always @ (posedge clk)
25  begin
26      if(reset)
27      begin
28          a_out <= 0;
29          b_out <= 0;
30          acc_out <= 0;
31          mult <= 0;
32      end
33      else
34      begin
35          a_out <= load_en ? a_in : a_out;
36          b_out <= load_en ? b_in : b_out;
37          mult <= mult_en ? a_out*b_out : mult;
38          acc_out <= acc_en ? mult + acc_out : acc_out;
```

39 end

40 end

41

42 endmodule

I.2 Systolic Array RTL Generation Script

```
1 use Getopt::Long;
2
3 #getting the user configuration
4 $command = GetOptions ( "help" => \$help ,
5     "param=s" => \$param ,
6     "width=i" => \$width ,
7     "dimension=i" => \$dimension ,
8     "outfile=s" => \$outfile );
9
10 #check for help
11 if($help > 0){
12     Help();
13     die;
14 }
15
16 # making sure either params file or other configs is specified
17 if((not defined $param) && ((not defined $width) or (not
18     defined $dimension) or (not defined $outfile))) {
19     ReturnError("Missing params");
20 }
21
22 elseif((defined $param) && ((defined $width) or (defined
23     $dimension) or (defined $outfile))) {
```

```
21     ReturnError("Command parameters shall either be a param
        file OR other input configs");
22 }
23
24 # parse the file into the variables or just parse the outfile
25 if( $param ne "" ) {
26     ParseParams();
27 }
28
29 # check if outfile specified correctly
30 if($outfile =~ m/^(.+).v/) { $outfile = $1; }
31 else {
32     ReturnError("Outfile not speicifed properly <outfile >.v");
33 }
34
35 $datawidth = $width - 1;
36
37 Debugger();
38
39 GenerateRTL();
40
41 # print error and die
42 sub ReturnError {
43     my ($ret) = @_ ;
44     print "\n==Error==\n$ret\n==Error==\n\n";
```

```
45     Help ();
46     die ;
47 }
48
49 # subroutine for debugging perl
50 sub Debugger {
51     print "Generating files with: width=$width , dimension=
        $dimension , outfile=$outfile\n";
52 }
53
54 # parse param file
55 sub ParseParams
56 {
57     open($fh , "<" , $param) or die("==Error== File $param not
        found");
58     while(my $line = <$fh>) {
59         if($line =~ m/\s*width\s*=\s*(\d+)\s*;\s*/) { $width =
            $1 }
60         elsif($line =~ m/\s*stages\s*=\s*(\d+)\s*;\s*/) {
            $dimension = $1; }
61         elsif($line =~ m/\s*outfile\s*=\s*(.+)\s*;\s*/) {
            $outfile = $1; }
62     }
63     close $fh;
64 }
```

```
65
66 sub GenerateRTL {
67     # write the first part of the module
68     $dimensionminus1 = $dimension - 1;
69
70     open $fh, ">", $outfile.".v";
71     $message = <<"EOF";
72     //=====
73     //     RTL Generated File
74     //     Author: Ashish Tondwalkar
75     //-----
76     //     Description: Systolic Array of MAC Units
77     //-----
78     //     Datawidth = $width
79     //     Dimension = $dimension
80     //=====
81
82 module $outfile (
83     clk ,
84     reset ,
85     scan_in0 ,
86     scan_en ,
87     test_mode ,
88     scan_out0 ,
89     a_in ,
```



```
90     b_in ,
91     mult_en ,
92     acc_en ,
93     load_en ,
94     d_out
95 );
96
97     parameter DATAWIDTH = $width;
98     parameter MATRIXSIZE = $dimension;
99     parameter ACCWIDTH = 32;
100
101     input clk;
102     input reset;
103
104     input
105         scan_in0 ,           // test scan mode data
106         scan_en ,           // test scan mode enable
107         test_mode;         // test mode select
108
109     output
110         scan_out0;         // test scan mode data
111
112     input mult_en;
```

```
113     input acc_en;
114     input load_en;
115     input [DATAWIDTH*MATRIXSIZE-1:0] a_in;
116     input [DATAWIDTH*MATRIXSIZE-1:0] b_in;
117     output [ACCWIDTH*MATRIXSIZE*MATRIXSIZE-1:0] d_out;
118
119     wire [DATAWIDTH-1:0] a [MATRIXSIZE:0][MATRIXSIZE:0]; //
           internal array row connections
120     wire [DATAWIDTH-1:0] b [MATRIXSIZE:0][MATRIXSIZE:0]; //
           internal array col connections
121
122 EOF
123     print $fh $message;
124
125     @a = (0.. $dimensionminus1);
126
127     # generate the stage register declarations
128     print $fh "\n //row bounday\n";
129     foreach my $k (@a) {
130         $calc = $k*$width;
131         print $fh " assign a[$k][0] = a_in[$calc +:
           DATAWIDTH]\;\n";
132     }
133     print $fh "\n // col bounday\n";
134     foreach my $k (@a) {
```

```
135         $calc = $k*$width;
136         print $fh "    assign b[0][$k] = b_in[$calc +:
            DATAWIDTH]\;\n";
137     }
138
139     my $nextrow;
140     my $nextcol;
141
142     for my $row (@a) {
143         for my $col (@a) {
144             $nextrow = $row + 1;
145             $nextcol = $col + 1;
146             $calc = ($row*$dimension + $col)*32;
147             $message = <<"EOF";
148
149     mac element_$row\_ $col (
150         .clk(clk),
151         .reset(reset),
152         .a_in(a[$row][$col]),
153         .b_in(b[$row][$col]),
154         .mult_en(mult_en),
155         .acc_en(acc_en),
156         .load_en(load_en),
157         .a_out(a[$row][$nextcol]),
158         .b_out(b[$nextrow][$col]),
```

```
159         .acc_out(d_out[ $calc +: ACCWIDTH])
160     );
161 EOF
162         print $fh $message;
163
164     }
165 }
166
167     print $fh "endmodule\n";
168
169 }
170
171
172 #subroutine for helping user
173 sub Help {
174     my $help = <<"EOF";
175     --param=param config file;
176
177     --width=datawidth
178         width must be within range [1, 64]
179
180     --dimension=dimensions of the systolic array
181         stages must be within range [2, 128]
182
183     --outfile=name of generated rtl and tb files
```

```
184
185 If a params file is specified , no other configuration inputs
    should be specified.
186
187 EOF
188     print $help
189 }
190
191
192 =pod
193
194 =head1 NAME
195
196 B<ast4798.pl> - generate multistage pipelined shift register
197
198 =head1 SYNOPSIS
199
200 perl ast4798.pl [ARGUMENT(s)] ...
201
202 =head1 DESCRIPTION
203
204 Generates RTL and TB files in verilog for a multistage
    pipelined shift register in verilog , given configuration
    parameters. TB tests reset and active function of RTL
205
```

206 ARGUMENTS(s) specified in the command line ...
207
208 B<--help> display this help and exit
209
210 B<--param> specify a text file that defines the configuration
arguments. If a file is specified, no other configuration
arguments should be specified.
211
212 B<--width> configuration argument that defines the datawidth
213
214 B<--dimension> configuration argument that defines the array
dimensions of the systolic array
215
216 B<--reset> configuration argument that defines the reset value
for each stage register. Can be in decimal or hex.
217
218 B<--outfile> configuration argument that describes the
nomenclature of the generate files. Must be specified with
<module_name>.v
219
220 =head1 FILES
221
222 B<<module_name>.v> generated RTL
223
224 B<<module_name>_test.v> generated TB

225

226 =head1 EXAMPLES

227

228 perl ast4798.pl -param my_params.txt

229

230 perl ast4798.pl -width 16 -stages 32 -outfile
ast_systolic_array.v

231

232 =head1 AUTHOR

233

234 Ashish Tondwalkar

235

236 =cut

I.3 Shift Register

```
1 module shift_reg (
2     clk ,
3     reset ,
4     shift ,
5     din ,
6     dout
7 );
8
9     parameter SIZE = 4;
10    parameter DATAWIDTH = 8;
11
12    input clk;
13    input reset;
14    input shift;
15    input [DATAWIDTH-1:0] din;
16    output wire [DATAWIDTH-1:0] dout;
17
18    reg [DATAWIDTH-1:0] mem [SIZE];
19
20    integer q;
21
22    always @ (posedge clk)
23    begin
```



```
24     if (reset)
25     begin
26         for (q = 0; q < SIZE; q = q + 1)
27             mem[q] <= 'b0;
28     end
29     else
30     begin
31         mem[0] <= (shift) ? din : mem[0];
32         for (q = 1; q < SIZE; q = q + 1)
33             mem[q] <= (shift) ? mem[q-1] : mem[q];
34     end
35 end
36
37 assign dout = mem[SIZE-1];
38
39 endmodule
```

I.4 Systolic Array Alignment Module RTL Generation Script

```
1 use Getopt::Long;
2
3 #getting the user configuration
4 $command = GetOptions ( "help" => \$help ,
5                         "width=i" => \$width ,
6                         "size=i" => \$size );
7
8 #check for help
9 if($help > 0){
10     Help();
11     die;
12 }
13
14 # check for valid width
15 if($width < 2 ) {
16     ReturnError("Width out of bounds");
17 }
18
19 # check for valid stage count
20 if($size < 2) {
21     ReturnError("Stages out of bounds");
22 }
23
```

```
24 GenerateRTL ();
25
26 # print error and die
27 sub ReturnError {
28     my ($ret) = @_ ;
29     print "\n==Error==\n$ret\n==Error==\n\n";
30     Help ();
31     die ;
32 }
33
34
35 sub GenerateRTL {
36
37     $dinw = $width*$size ;
38
39     # write the first part of the module
40     open $fh, ">", "data_alignment.v";
41     $message = <<"EOF";
42 //=====
43 //     RTL Generated File
44 //     Author: Ashish Tondwalkar
45 //-----
46 //     Description: Data aligner for systolic array
47 //-----
48 //     Datewidth = $width
```

```
49 //      Matrix Size = $size
50 //=====
51
52 module data_alignment(
53     clk ,
54     reset ,
55     shift ,
56     a_in ,
57     b_in ,
58     a_out ,
59     b_out
60 );
61
62     input clk ;
63     input reset ;
64     input shift ;
65     input [$dinw-1:0] a_in ;
66     input [$dinw-1:0] b_in ;
67     output reg [$dinw-1:0] a_out ;
68     output reg [$dinw-1:0] b_out ;
69
70     assign a_out[0 +: $width] = a_in[0 +: $width] ;
71     assign b_out[0 +: $width] = b_in[0 +: $width] ;
72
73 EOF
```

```
74     print $fh $message;
75
76     # array for generation loops
77     @a = (1.. $size -1);
78
79     # generate the stage register declarations
80     foreach my $i (@a)
81     {
82
83         $idx = $i*$width;
84
85         $message = <<"EOF";
86         shift_reg #(.DATAWIDTH(8), .SIZE($i)) del_a_$i (. clk(clk),
            . reset(reset), . shift(shift), . din(a_in[$idx +: $width
            ]), . dout(a_out[$idx +: $width]));
87         shift_reg #(.DATAWIDTH(8), .SIZE($i)) del_b_$i (. clk(clk),
            . reset(reset), . shift(shift), . din(b_in[$idx +: $width
            ]), . dout(b_out[$idx +: $width]));
88
89 EOF
90
91     print $fh $message;
92
93     }
94     $message = <<"EOF";
```

```
95
96 endmodule
97
98 EOF
99     print $fh $message;
100
101     close $fh;
102
103 }
104
105 #subroutine for helping user
106 sub Help {
107     my $help = <<"EOF";
108     --param=param config file;
109
110     --width=word size/data width
111     width must be within range [1, 64]
112
113     --stages=number of stages in pipeline
114     stages must be within range [2, 128]
115
116     --reset=reset value for registers in binary or hex
117
118     --outfile=name of generated rtl and tb files
119
```

```
120 If a params file is specified , no other configuration inputs
    should be specified.
121
122 EOF
123     print $help
124 }
125
126
127 =pod
128
129 =head1 NAME
130
131 B<ast4798.pl> - generate data alignment rtl
132
133 =head1 SYNOPSIS
134
135 perl data_alignment.pl [ARGUMENT(s)] ...
136
137 =head1 DESCRIPTION
138
139 Generates RTL for a data aligner required for systolic
    execution
140
141 ARGUMENTS(s) specified in the command line...
142
```

```
143 B<--help> display this help and exit
144
145 B<--width> configuration argument that defines the datawidth
146
147 B<--sizes> configuration argument that defines the systolic
    array size
148
149 =head1 FILES
150
151 B<data_alignment.v> generated RTL
152
153 =head1 EXAMPLES
154
155 perl data_alignment.pl --width 8 --size 4
156
157 =head1 AUTHOR
158
159 Ashish Tondwalkar
160
161 =cut
```

I.5 Strassen Layer RTL Generation Script

```
1 use Getopt::Long;
2
3 #getting the user configuration
4 $command = GetOptions ( "help" => \$help ,
5     "param=s" => \$param ,
6     "size=i" => \$size ,
7     "width=i" => \$width
8     "reset=s" => \$reset ,
9     "outfile=s" => \$outfile
10    );
11
12 #check for help
13 if($help > 0){
14     Help();
15     die;
16 }
17
18 # making sure either params file or other configs is specified
19 if(not defined $size) {
20     ReturnError("Missing parameter");
21 }
22
23 # check for valid range
```

```
24 if($size < 2) {
25     ReturnError("Size has to be at least 2");
26 }
27
28 GenerateFiles();
29
30
31 # print error and die
32 sub ReturnError {
33     my ($ret) = @_ ;
34     print "\n==Error==\n$ret\n==Error==\n\n";
35     Help();
36     die;
37 }
38
39 sub GenerateFiles {
40     # write the first part of the module
41     open $fh, ">", "strassen_dim$size.v";
42
43     $half = $size/2;
44     $mab = $width * $size * $size;
45     $aab = 32 * $size * $size;
46     $qwidth = $width + 2;
47     $qm = $qwidth * $half * $half;
48     $mm = 32 * $half * $half;
```

```
49
50     $message = <<"EOF" ;
51
52 module strassen_dim$size (
53     reset ,
54     clk ,
55     scan_in0 ,
56     scan_en ,
57     test_mode ,
58     scan_out0 ,
59     enable ,
60     matrix_a ,
61     matrix_b ,
62     matrix_c
63 );
64
65 input
66     reset ,                // system reset
67     clk ;                  // system clock
68
69 input
70     scan_in0 ,            // test scan mode data
71     scan_en ,            // test scan mode enable
72     test_mode ;          // test mode select
```

```
73
74     output
75         scan_out0;                // test scan mode data
76         output
77     parameter DATAWIDTH = $width; // data bus width
78     parameter QWIDTH = DATAWIDTH + 2;
79     parameter ACCWIDTH = 32;
80
81     localparam DIM = $size;
82     localparam SIZE = DIM * DIM;
83
84     input enable;
85     input [$mab-1:0] matrix_a;
86     input [$mab-1:0] matrix_b;
87     output reg [$aab-1:0] matrix_c;
88
89     wire signed [32-1:0] matrix_c_nxt [1:0][1:0][$half-1:0][$half
90         -1:0];
91     reg signed [$qwidth-1:0] q_int [13:0][$half-1:0][$half-1:0];
92     wire signed [$qwidth-1:0] q_int_nxt [13:0][$half-1:0][$half
93         -1:0];
94     reg [$mm-1:0] matrix_m [6:0];
```

```
95
96 wire signed [$width-1:0] a_arr [$size-1:0][$size-1:0];
97 wire signed [$width-1:0] b_arr [$size-1:0][$size-1:0];
98
99
100 wire [$qm-1:0] q_int_flat [13:0];
101
102 wire signed [32-1:0] matrix_m_unflat [6:0][$half-1:0][$half
    -1:0];
103
104 wire [$aab-1:0] matrix_c_nxt_flat;
105
106
107 EOF
108     print $fh $message;
109
110
111     @a = (0..$size-1);
112     @b = (0..$size-1);
113     foreach my $i (@a) {
114         foreach my $j (@b) {
115             $calc = $i*$size*$width + $j*$width;
116             $message = <<"EOF";
117 assign a_arr[$i][$j] = matrix_a[$calc +: $width];
118 assign b_arr[$i][$j] = matrix_b[$calc +: $width];
```

```
119 EOF
120     print $fh $message;
121
122     }
123 }
124
125
126     $message = <<"EOF";
127
128 integer k, l, m;
129 always \@ (posedge clk)
130 begin
131     if(reset)
132     begin
133         // matrix_c <= 0;
134         for(k = 0; k < 14; k = k + 1)
135             for(l = 0; l < $half; l = l + 1)
136                 for(m = 0; m < $half; m = m + 1)
137                     q_int[k][l][m] <= 0;
138
139         matrix_c <= 0;
140     end
141     else
142     begin
143         for(k = 0; k < 14; k = k + 1)
```

```
144         for (l = 0; l < $half; l = l + 1)
145             for (m = 0; m < $half; m = m + 1)
146                 q_int[k][l][m] <= enable ? q_int_nxt[k][l
                    ][m] : q_int[k][l][m];
147
148     matrix_c <= enable ? matrix_c_nxt_flat : matrix_c;
149     end
150 end
151
152
153 EOF
154     print $fh $message;
155
156     @a = (0..($size/2)-1);
157     @b = (0..($size/2)-1);
158
159     #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(0), .ya(0),
        .xb(1), .yb(1), .q(q_int_nxt[0]));
160     foreach my $i (@a) {
161         foreach my $j (@b) {
162             $xa = 0*($size/2) + $i;
163             $ya = 0*($size/2) + $j;
164             $xb = 0*($size/2) + $i;
165             $yb = 0*($size/2) + $j;
166             $message = <<"EOF";
```

```
167     assign q_int_nxt[0][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
168 EOF
169     print $fh $message;
170
171     }
172 }
173
174 #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(0), .ya(0),
        .xb(1), .yb(1), .q(q_int_nxt[1]));
175 foreach my $i (@a) {
176     foreach my $j (@b) {
177         $xa = 0*($size/2) + $i;
178         $ya = 0*($size/2) + $j;
179         $xb = 1*($size/2) + $i;
180         $yb = 1*($size/2) + $j;
181         $message = <<"EOF";
182     assign q_int_nxt[1][$i][$j] = b_arr[$xa][$ya] + b_arr[$xb
        ][$yb];
183 EOF
184     print $fh $message;
185
186     }
187 }
188
```



```
189     #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(1), .ya(0),
        .xb(1), .yb(1), .q(q_int_nxt[2]));
190     foreach my $i (@a) {
191         foreach my $j (@b) {
192             $xa = 1*($size/2) + $i;
193             $ya = 0*($size/2) + $j;
194             $xb = 1*($size/2) + $i;
195             $yb = 1*($size/2) + $j;
196             $message = <<"EOF";
197     assign q_int_nxt[2][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
198 EOF
199     print $fh $message;
200
201     }
202 }
203
204 #direct_assign_m (.a(b_arr), .xa(0), .ya(0), .q(q_int_nxt
        [3]));
205 foreach my $i (@a) {
206     foreach my $j (@b) {
207         $message = <<"EOF";
208     assign q_int_nxt[3][$i][$j] = b_arr[$i][$j];
209 EOF
210     print $fh $message;
```

```
211
212     }
213 }
214
215 #direct_assign_m (.a(a_arr), .xa(0), .ya(0), .q(q_int_nxt
    [4]));
216 foreach my $i (@a) {
217     foreach my $j (@b) {
218         $message = <<"EOF";
219         assign q_int_nxt[4][$i][$j] = a_arr[$i][$j];
220 EOF
221         print $fh $message;
222
223     }
224 }
225
226 #add_sub_m (.a(b_arr), .b(b_arr), .op(1), .xa(0), .ya(1),
    .xb(1), .yb(1), .q(q_int_nxt[5]));
227 foreach my $i (@a) {
228     foreach my $j (@b) {
229         $xa = 0*($size/2) + $i;
230         $ya = 1*($size/2) + $j;
231         $xb = 1*($size/2) + $i;
232         $yb = 1*($size/2) + $j;
233         $message = <<"EOF";
```

```
234     assign q_int_nxt[5][$i][$j] = b_arr[$xa][$ya] - b_arr[$xb
        ][$yb];
235 EOF
236     print $fh $message;
237
238     }
239 }
240
241 #direct_assign_m (.a(a_arr), .xa(1), .ya(1), .q(q_int_nxt
        [6]));
242 foreach my $i (@a) {
243     foreach my $j (@b) {
244         $x = 1*($size/2) + $i;
245         $y = 1*($size/2) + $j;
246         $message = <<"EOF";
247     assign q_int_nxt[6][$i][$j] = a_arr[$x][$y];
248 EOF
249     print $fh $message;
250
251     }
252 }
253
254 #add_sub_m (.a(b_arr), .b(b_arr), .op(1), .xa(1), .ya(0),
        .xb(0), .yb(0), .q(q_int_nxt[7]));
255 foreach my $i (@a) {
```

```
256     foreach my $j (@b) {
257         $xa = 1*($size/2) + $i;
258         $ya = 0*($size/2) + $j;
259         $xb = 0*($size/2) + $i;
260         $yb = 0*($size/2) + $j;
261         $message = <<"EOF";
262     assign q_int_nxt[7][$i][$j] = b_arr[$xa][$ya] - b_arr[$xb
        ][$yb];
263 EOF
264     print $fh $message;
265
266     }
267 }
268
269 #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(0), .ya(0),
        .xb(0), .yb(1), .q(q_int_nxt[8]));
270 foreach my $i (@a) {
271     foreach my $j (@b) {
272         $xa = 0*($size/2) + $i;
273         $ya = 0*($size/2) + $j;
274         $xb = 0*($size/2) + $i;
275         $yb = 1*($size/2) + $j;
276         $message = <<"EOF";
277     assign q_int_nxt[8][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
```

```
278 EOF
279     print $fh $message;
280
281     }
282 }
283
284 #direct_assign_m (.a(b_arr), .xa(1), .ya(1), .q(q_int_nxt
    [9]));
285 foreach my $i (@a) {
286     foreach my $j (@b) {
287         $x = 1*($size/2) + $i;
288         $y = 1*($size/2) + $j;
289         $message = <<"EOF";
290         assign q_int_nxt[9][$i][$j] = b_arr[$x][$y];
291 EOF
292     print $fh $message;
293
294     }
295 }
296
297 #add_sub_m (.a(a_arr), .b(a_arr), .op(1), .xa(1), .ya(0),
    .xb(0), .yb(0), .q(q_int_nxt[10]));
298 foreach my $i (@a) {
299     foreach my $j (@b) {
300         $xa = 1*($size/2) + $i;
```

```
301         $ya = 0*($size/2) + $j;
302         $xb = 0*($size/2) + $i;
303         $yb = 0*($size/2) + $j;
304         $message = <<"EOF";
305     assign q_int_nxt[10][$i][$j] = a_arr[$xa][$ya] - a_arr[$xb
        ][$yb];
306 EOF
307     print $fh $message;
308
309 }
310 }
311
312 #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(0), .ya(0),
        .xb(0), .yb(1), .q(q_int_nxt[11]));
313 foreach my $i (@a) {
314     foreach my $j (@b) {
315         $xa = 0*($size/2) + $i;
316         $ya = 0*($size/2) + $j;
317         $xb = 0*($size/2) + $i;
318         $yb = 1*($size/2) + $j;
319         $message = <<"EOF";
320     assign q_int_nxt[11][$i][$j] = b_arr[$xa][$ya] + b_arr[$xb
        ][$yb];
321 EOF
322     print $fh $message;
```

```
323
324     }
325 }
326
327 #add_sub_m (.a(a_arr), .b(a_arr), .op(1), .xa(0), .ya(1),
           .xb(1), .yb(1), .q(q_int_nxt[12]));
328 foreach my $i (@a) {
329     foreach my $j (@b) {
330         $xa = 0*($size/2) + $i;
331         $ya = 1*($size/2) + $j;
332         $xb = 1*($size/2) + $i;
333         $yb = 1*($size/2) + $j;
334         $message = <<"EOF";
335         assign q_int_nxt[12][$i][$j] = a_arr[$xa][$ya] - a_arr[$xb
           ][$yb];
336 EOF
337         print $fh $message;
338
339     }
340 }
341
342 #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(1), .ya(0),
           .xb(1), .yb(1), .q(q_int_nxt[13]));
343 foreach my $i (@a) {
344     foreach my $j (@b) {
```

```
345         $xa = 1*($size/2) + $i;
346         $ya = 0*($size/2) + $j;
347         $xb = 1*($size/2) + $i;
348         $yb = 1*($size/2) + $j;
349         $message = <<"EOF";
350     assign q_int_nxt[13][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
351 EOF
352     print $fh $message;
353
354     }
355 }
356
357
358 @a = (0..6);
359 @b = (0..($size/2)-1);
360 @u = (0..($size/2)-1);
361
362 foreach my $i (@a) {
363     foreach my $j (@b) {
364         foreach my $k (@u) {
365             $calc = $j*($size/2)*32 + $k*32;
366             $message = <<"EOF";
367 assign matrix_m_unflat[$i][$j][$k] = matrix_m[$i][$calc +:
        32];
```



```
368 EOF
369     print $fh $message;
370
371     }
372 }
373 }
374
375
376 @th = (0..13);
377
378
379
380 foreach my $t (@th) {
381
382
383     print $fh "assign q_int_flat[$t] = {\n";
384
385     $modsize = ($size/2)-1;
386
387     @r = (0..$modsize);
388     @c = (0..$modsize);
389
390     @r = reverse @r;
391     @c = reverse @c;
392
```

```
393     for my $i (@r) {
394         for my $j (@c) {
395             if($i == 0 && $j == 0) {
396                 $message = <<"EOF";
397                                     q_int[$t][$i][$j]
398 EOF
399             }
400             else {
401                 $message = <<"EOF";
402                                     q_int[$t][$i][$j],
403 EOF
404             }
405
406
407             print $fh $message;
408         }
409     }
410
411     print $fh "                                     };\n";
412 }
413
414
415
416
417 @a = (0..($size/2)-1);
```

```
418     @b = (0..($size/2)-1);
419     foreach my $i (@a) {
420         foreach my $j (@b) {
421             $message = <<"EOF";
422 assign matrix_c_nxt[0][0][$i][$j] = matrix_m_unflat[0][$i][$j]
         + matrix_m_unflat[3][$i][$j] - matrix_m_unflat[4][$i][$j]
         + matrix_m_unflat[6][$i][$j];
423 assign matrix_c_nxt[0][1][$i][$j] = matrix_m_unflat[2][$i][$j]
         + matrix_m_unflat[4][$i][$j];
424 assign matrix_c_nxt[1][0][$i][$j] = matrix_m_unflat[1][$i][$j]
         + matrix_m_unflat[3][$i][$j];
425 assign matrix_c_nxt[1][1][$i][$j] = matrix_m_unflat[0][$i][$j]
         - matrix_m_unflat[1][$i][$j] + matrix_m_unflat[2][$i][$j]
         + matrix_m_unflat[5][$i][$j];
426 EOF
427     print $fh $message;
428
429     }
430 }
431
432 @a = (0..1);
433 @b = (0..1);
434 @c = (0..($size/2)-1);
435 @d = (0..($size/2)-1);
436 foreach my $i (@a) {
```

```
437     foreach my $j (@b) {
438         foreach my $u (@c) {
439             foreach my $w (@d) {
440                 $calc = $i*$size*($size/2)*32 + ($j*($size
441                     /2))*32 + ($u*$size)*32 + $w*32;
442                 $message = <<"EOF";
443                 assign matrix_c_nxt_flat[$calc +: 32] = matrix_c_nxt[$i][$j][
444                     $u][$w];
445             EOF
446                 print $fh $message;
447             }
448         }
449     }
450     $message = <<"EOF";
451
452
453 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M1 (
454     . clk (clk) ,
455     . reset (reset) ,
456     . enable (enable) ,
457     . matrix_a (q_int_flat [0]) ,
458     . matrix_b (q_int_flat [1]) ,
459     . matrix_c (matrix_m [0])
```

```
460 );
461
462 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M2 (
463     . clk ( clk ) ,
464     . reset ( reset ) ,
465     . enable ( enable ) ,
466     . matrix_a ( q_int_flat [ 2 ] ) ,
467     . matrix_b ( q_int_flat [ 3 ] ) ,
468     . matrix_c ( matrix_m [ 1 ] )
469 );
470
471 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M3 (
472     . clk ( clk ) ,
473     . reset ( reset ) ,
474     . enable ( enable ) ,
475     . matrix_a ( q_int_flat [ 4 ] ) ,
476     . matrix_b ( q_int_flat [ 5 ] ) ,
477     . matrix_c ( matrix_m [ 2 ] )
478 );
479
480 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M4 (
481     . clk ( clk ) ,
482     . reset ( reset ) ,
483     . enable ( enable ) ,
484     . matrix_a ( q_int_flat [ 6 ] ) ,
```

```
485     .matrix_b ( q_int_flat [7] ) ,
486     .matrix_c ( matrix_m [3] )
487 );
488
489 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M5 (
490     .clk ( clk ) ,
491     .reset ( reset ) ,
492     .enable ( enable ) ,
493     .matrix_a ( q_int_flat [8] ) ,
494     .matrix_b ( q_int_flat [9] ) ,
495     .matrix_c ( matrix_m [4] )
496 );
497
498 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M6 (
499     .clk ( clk ) ,
500     .reset ( reset ) ,
501     .enable ( enable ) ,
502     .matrix_a ( q_int_flat [10] ) ,
503     .matrix_b ( q_int_flat [11] ) ,
504     .matrix_c ( matrix_m [5] )
505 );
506
507 strassen_dim$half #(.DATAWIDTH(QWIDTH)) M7 (
508     .clk ( clk ) ,
509     .reset ( reset ) ,
```

```
510     .enable(enable),
511     .matrix_a(q_int_flat[12]),
512     .matrix_b(q_int_flat[13]),
513     .matrix_c(matrix_m[6])
514 );
515
516 endmodule // strassen_dim$size
517 EOF
518
519     print $fh $message;
520
521     close $fh;
522
523 }
524
525 #subroutine for helping user
526 sub Help {
527     my $help = <<"EOF";
528     --size=multiplier dimension
529 EOF
530     print $help
531 }
532
533
534 =pod
```

```
535
536 =head1 NAME
537
538 B<strassen_layer_gen.pl> - generate Verilog file for a
    strassen hierachy layer
539
540 =head1 SYNOPSIS
541
542 perl strassen_layer_gen.pl [ARGUMENT(s)] ...
543
544 =head1 DESCRIPTION
545
546 ARGUMENTS(s) specified in the command line ...
547
548 B<--help> display this help and exit
549
550 B<--size> the dimensions of the matrix multiplier
551
552 =head1 FILES
553
554 B<q_int_flat.txt> q_int_flat concat code
555
556 =head1 EXAMPLES
557
558 perl strassen_layer_gen.pl --size 64
```


559

560 =head1 AUTHOR

561

562 Ashish Tondwalkar

563

564 =cut

I.6 Test Config File Generation Script

```
1 use Getopt::Long;
2
3 #getting the user configuration
4 $command = GetOptions ( "help" => \$help ,
5     "param=s" => \$param ,
6     "runs=i" => \$runs ,
7     "size=i" => \$size ,
8     "seed=i" => \$seed ,
9     "full=i" => \$full ,
10    "racr=i" => \$racr
11    "reset=s" => \$reset ,
12    "outfile=s" => \$outfile
13    );
14
15 #check for help
16 if($help > 0){
17     Help();
18     die;
19 }
20
21 # making sure either params file or other configs is specified
22 if(not defined $runs or not defined $size) {
23     ReturnError("Missing parameter");
```

```
24 }
25
26 # check for valid range
27 if($runs < 1) {
28     ReturnError("Runs has to be at least 1");
29 }
30
31 if($size < 2) {
32     ReturnError("Size has to be at least 2");
33 }
34
35 GenerateFiles();
36
37 system("cp test_pipeline.txt test_pipeline_str.txt");
38 system("cp test_pipeline.txt test_pipeline_strfull.txt");
39
40 # print error and die
41 sub ReturnError {
42     my ($ret) = @_;
43     print "\n==Error==\n$ret\n==Error==\n\n";
44     Help();
45     die;
46 }
47
48 sub GenerateFiles {
```

```
49     # write the first part of the module
50     open $fh, ">", "test_pipeline.txt";
51     open $fh2, ">", "test_config.txt";
52
53     $data = 0;
54
55     while($runs > 0) {
56
57         #pipe
58         if(not defined $racr) {
59             $data = $runs;
60             $runs = 0;
61         }
62         else {
63             do {
64                 $data = 1#int(rand($runs + 1));
65             } while($data == 0);
66
67             if($data > $runs) {
68                 $data = $runs;
69                 $runs = 0;
70             }
71             else {
72                 $runs = $runs - $data;
73             }
```

```
74     }
75
76     $message = <<"EOF";
77 $data
78 EOF
79     print $fh $message;
80
81     #config
82
83     if(defined $full) {
84         $dima = $size;
85         $dimshared = $size;
86         $dimb = $size;
87     }
88     else {
89         $randval = $size + 1 - 2;
90         $dima = int(rand($randval)) + 2;
91         $dimshared = int(rand($randval)) + 2;
92         $dimb = int(rand($randval)) + 2;
93     }
94
95     $message = <<"EOF";
96 $dima
97 $dimshared
98 $dimb
```

```
99 EOF
100         print $fh2 $message;
101
102     }
103
104     close $fh;
105     close $fh2;
106
107 }
108
109 #subroutine for helping user
110 sub Help {
111     my $help = <<"EOF";
112     --runs=number of multiplications
113
114     --size=multiplier dimension
115
116     --seed=random generation seed
117 EOF
118     print $help
119 }
120
121
122 =pod
123
```

```
124 =head1 NAME
125
126 B<test_gen.pl> - generate text files for test configurations
127
128 =head1 SYNOPSIS
129
130 perl test_gen.pl [ARGUMENT(s)] ...
131
132 =head1 DESCRIPTION
133
134 Generates RTL and TB files in verilog for a multistage
      pipelined shift register in verilog, given configuration
      parameters. TB tests reset and active function of RTL
135
136 ARGUMENTS(s) specified in the command line...
137
138 B<--help> display this help and exit
139
140 B<--runs> amount of multiplications
141
142 B<--size> the dimensions of the matrix multiplier
143
144 B<--seed> random generation seed
145
146 =head1 FILES
```

```
147
148 B<test_pipeline.txt> multiplications before a reset
149
150 B<test_config.txt> matrix multiplication dimensions
151
152 =head1 EXAMPLES
153
154 perl test_gen.pl --runs 500 --size 16
155
156 perl test_gen.pl --runs 500 --size 16 --full 1
157
158 perl test_gen.pl --runs 500 --size 16 --full 1 --racr 1
159
160 =head1 AUTHOR
161
162 Ashish Tondwalkar
163
164 =cut
```

I.7 Strassen Accumulator Top File

```
1
2
3 module multiplier_strassen (
4     reset ,
5     clk ,
6     scan_in0 ,
7     scan_en ,
8     test_mode ,
9     scan_out0 ,
10    enable ,
11    matrix_a ,
12    matrix_b ,
13    matrix_c
14 );
15
16
17    input
18        reset ,                // system reset
19        clk ;                  // system clock
20
21    input
22        scan_in0 ,            // test scan mode data
23        input
```

```
23     scan_en ,                // test scan mode enable
24     test_mode ;             // test mode select
25
26     output
27     scan_out0 ;            // test scan mode data
28                             output
29     parameter DATAWIDTH = 8; // data bus width
30     parameter ACCWIDTH = 32;
31
32     localparam DIM = 4;
33     localparam SIZE = DIM * DIM;
34
35     input enable ;
36     input [(DATAWIDTH*SIZE) - 1:0] matrix_a ;
37     input [(DATAWIDTH*SIZE) - 1:0] matrix_b ;
38     output reg [(ACCWIDTH*SIZE) - 1:0] matrix_c ;
39
40     wire [(ACCWIDTH*SIZE) - 1:0] matrix_c_nxt ;
41     reg signed [ACCWIDTH-1:0] matrix_c_unflat [DIM-1:0][DIM
42     - 1:0];
43
44     wire [(ACCWIDTH*SIZE) - 1:0] raw_out ;
45     wire signed [ACCWIDTH-1:0] raw_arr [DIM-1:0][DIM-1:0];
```

```
46
47     strassen_dim4 #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH(ACCWIDTH)
48         ) str (
49         .clk(clk),
50         .reset(reset),
51         .enable(enable),
52         .matrix_a(matrix_a),
53         .matrix_b(matrix_b),
54         .matrix_c(raw_out)
55     );
56
57     genvar x, y, w, u;
58     generate
59     for (x = 0; x < DIM; x = x + 1)
60     begin
61         for (y = 0; y < DIM; y = y + 1)
62         begin
63             assign raw_arr[x][y] = raw_out[(x*DIM)*ACCWIDTH +
64                 y*ACCWIDTH+: ACCWIDTH];
65             assign matrix_c_unflat[x][y] = matrix_c[(x*DIM)*
66                 ACCWIDTH + y*ACCWIDTH +: ACCWIDTH];
67             assign matrix_c_nxt[(x*DIM)*ACCWIDTH + y*ACCWIDTH
68                 +: ACCWIDTH] = raw_arr[x][y] + matrix_c_unflat[
69                 x][y];
```

```
66         end
67     end
68
69     integer k, l, m;
70     always @ (posedge clk)
71     begin
72         if(reset)
73             matrix_c <= 0;
74         else
75             matrix_c <= enable ? matrix_c_nxt : matrix_c;
76     end
77
78
79 endgenerate
80
81
82
83 endmodule
```

I.8 Systolic Array Driver

```
1 typedef virtual input_if input_vif;
2
3 import "DPI-C" context function void resetAccumulator();
4
5 class driver_systolic extends uvm_driver #(packet_in);
6
7     int index;
8
9     uvm_analysis_port #(packet_in) item_collected_port_sys;
10
11     `uvm_component_utils(driver_systolic)
12     input_vif vif;
13
14     packet_in modded;
15
16     int i, j;
17
18     bit letrun;
19
20     int fd;
21     int config_fp;
22     string config_line;
23     int config_data_a, config_data_shared, config_data_b;
```

```
24     string line ;
25     int products ;
26
27     event sample_cg ;
28     event config_cg ;
29
30     covergroup input_cov @ (sample_cg) ;
31         option .per_instance = 1 ;
32
33         matrix_a : coverpoint vif.matrix_a_data {
34             bins range [] = {
35                 [0 : 255],
36                 [256 : 511],
37                 [512 : 767],
38                 [768 : 1023],
39                 [1024 : 1279],
40                 [1280 : 1535],
41                 [1536 : 1791],
42                 [1792 : 2047]
43             };
44     }
45     matrix_b : coverpoint vif.matrix_a_data {
46         bins range [] = {
47             [0 : 255],
48             [256 : 511],
```

```
49         [512 : 767],
50         [768 : 1023],
51         [1024 : 1279],
52         [1280 : 1535],
53         [1536 : 1791],
54         [1792 : 2047]
55     };
56 }
57 endgroup
58
59 covergroup config_cov @ (config_cg);
60     option.per_instance = 1;
61
62     config_row : coverpoint config_data_a {
63         bins range[] = {[2 : MATRIXSIZE]};
64     }
65     config_shared : coverpoint config_data_shared {
66         bins range[] = {[2 : MATRIXSIZE]};
67     }
68     config_col : coverpoint config_data_b {
69         bins range[] = {[2 : MATRIXSIZE]};
70     }
71 endgroup
72
```

```
73     function new(string name = "driver_systolic",
74                 uvm_component parent = null);
75         super.new(name, parent);
76         item_collected_port_sys = new ("
77             item_collected_port_sys", this);
78         modded = packet_in::type_id::create("modded", this);
79         input_cov = new();
80         config_cov = new();
81     endfunction
82
83     virtual function void build_phase(uvm_phase phase);
84         super.build_phase(phase);
85         void '(uvm_resource_db#(input_vif)::read_by_name(.scope
86             ("ifs"), .name("input_vif"), .val(vif)));
87     endfunction
88
89     virtual task run_phase(uvm_phase phase);
90         super.run_phase(phase);
91         // phase.raise_objection(this);
92         // fork
93         //     reset_signals();
94         letrun = 0;
95     fork
96         get_and_drive(phase);
97     fork
98         run_perf_cnt();
```



```
95         run_enables ();
96         // run_validity ();
97     join
98
99     // join
100    // phase.drop_objection(this);
101 endtask
102
103 virtual protected task reset_signals ();
104     // wait (vif.rst === 1);
105     vif.rst <= 1;
106     // forever begin
107     vif.valid <= '0;
108     vif.load <= '0;
109     vif.mult <= '0;
110     vif.acc <= 0;
111     vif.matrix_a_data <= 0;
112     vif.matrix_b_data <= 0;
113     vif.en_perf <= 0;
114
115     resetAccumulator ();
116
117     #22;
118     @(posedge vif.clk);
119     vif.rst <= 0;
```

```
120         // end
121     endtask
122
123     virtual protected task fetch_fdata();
124         $fgets(line, fd);
125         products = line.atoi();
126
127         $fgets(config_line, config_fp);
128         config_data_a = config_line.atoi();
129
130         $fgets(config_line, config_fp);
131         config_data_shared = config_line.atoi();
132
133         $fgets(config_line, config_fp);
134         config_data_b = config_line.atoi();
135
136         $display("%t | Systolic: products:%0d dima:%0d
137                 dimshared:%0d dimb:%0d", $time, products,
138                 config_data_a, config_data_shared, config_data_b);
139     endtask
140
141     virtual protected task get_and_drive(uvm_phase phase);
142         fd = $fopen("test_pipeline.txt", "r");
143         config_fp = $fopen("test_config.txt", "r");
```



```

= req . data_a [( i * MATRIXSIZE
+ j ) * DATAWIDTH +: DATAWIDTH
];
166     else
167         modded . data_a [( i * MATRIXSIZE +
+ j ) * DATAWIDTH +: DATAWIDTH]
= 0;
168
169     if ( i < config_data_shared && j <
+ config_data_b )
170         modded . data_b [( i * MATRIXSIZE +
+ j ) * DATAWIDTH +: DATAWIDTH]
= req . data_b [( i * MATRIXSIZE
+ j ) * DATAWIDTH +: DATAWIDTH
];
171     else
172         modded . data_b [( i * MATRIXSIZE +
+ j ) * DATAWIDTH +: DATAWIDTH]
= 0;
173     end
174 end
175 item_collected_port_sys . write ( modded );
176 letrun = 1;
177 drive_transfer ( modded );
178 vif . matrix_a_data <= 0;
```

```
179         vif.matrix_b_data <= 0;
180         // vif.valid <= 1;
181         //@(posedge vif.clk) ;
182     end
183
184     repeat( config_data_shared+config_data_a+
185            config_data_b -1)
186     begin
187         @(posedge vif.clk) ;
188         @(posedge vif.clk) ;
189     end
190
191     vif.en_perf = 0;
192
193     vif.valid <= 1;
194     @(posedge vif.clk) ;
195     vif.valid <= 0;
196 end
197 else
198 begin
199     $fclose( fd );
200     $fclose( config_fp );
201     //@(posedge vif.rst);
```

```
202         $display ("%t | Finished Systolic
                Multiplications", $time);
203         reset_signals ();
204         letrun = 0;
205
206         break;
207     end
208 end
209
210     reset_signals ();
211 endtask
212
213 virtual protected task drive_transfer(packet_in tr);
214     index = 0;
215
216     // repeat (config_data_a+config_data_shared+
                config_data_b-1)
217     repeat (MATRIXSIZE)
218     begin
219         if (index < MATRIXSIZE)
220         begin
221             vif.matrix_a_data <= tr.data_a[index*(
                DATAWIDTH*MATRIXSIZE) +: DATAWIDTH*
                MATRIXSIZE];
```

```
222         vif.matrix_b_data <= tr.data_b[index*(
           DATAWIDTH*MATRIXSIZE) +: DATAWIDTH*
           MATRIXSIZE];
223     end
224     else
225     begin
226         vif.matrix_a_data <= 0;
227         vif.matrix_b_data <= 0;
228     end
229     index = index + 1;
230     @(posedge vif.clk) ;
231     @(posedge vif.clk) ;
232     // vif.valid <= 0;
233     // -> sample_cg;
234     end
235 endtask
236
237 virtual protected task run_enables();
238     @(posedge letrun)
239     vif.en_perf = 1;
240     forever
241     begin
242         vif.load <= 1;
243         vif.mult <= 0;
244         vif.acc <= 1;
```

```
245         @(posedge vif.clk) ;
246         vif.load <= 0;
247         vif.mult <= 1;
248         vif.acc <= 0;
249         @(posedge vif.clk) ;
250     end
251 endtask
252
253 virtual protected task run_perf_cnt();
254     forever
255     begin
256         @(posedge vif.clk)
257         if(vif.en_perf)
258             vif.perf_cnt = vif.perf_cnt + 1;
259     end
260 endtask
261
262 endclass
```

I.9 Strassen Matched Bandwidth Driver

```
1 import "DPI-C" context function void resetAccumulatorStrassen
    ();
2
3 class driver_strassen extends uvm_driver #(packet_in);
4
5     int index;
6
7     uvm_analysis_port #(packet_in) item_collected_port_str;
8
9     `uvm_component_utils(driver_strassen)
10    input_vif vif;
11
12    int i, j;
13
14    int pipe_cnt;
15
16    int fd;
17    string line;
18    int products;
19
20    int products_modded;
21
22    event sample_cg;
```

```
23
24     covergroup input_cov @ (sample_cg);
25         option.per_instance = 1;
26
27         matrix_a : coverpoint vif.matrix_a_data {
28             bins range[] = {
29                 [0 : 255],
30                 [256 : 511],
31                 [512 : 767],
32                 [768 : 1023],
33                 [1024 : 1279],
34                 [1280 : 1535],
35                 [1536 : 1791],
36                 [1792 : 2047]
37             };
38     }
39     matrix_b : coverpoint vif.matrix_a_data {
40         bins range[] = {
41             [0 : 255],
42             [256 : 511],
43             [512 : 767],
44             [768 : 1023],
45             [1024 : 1279],
46             [1280 : 1535],
47             [1536 : 1791],
```

```
48         [1792 : 2047]
49     };
50 }
51 endgroup
52
53 function new(string name = "driver_strassen",
54             uvm_component parent = null);
55     super.new(name, parent);
56     item_collected_port_str = new ("
57         item_collected_port_str", this);
58     input_cov = new();
59 endfunction
60
61 virtual function void build_phase(uvm_phase phase);
62     super.build_phase(phase);
63     void '(uvm_resource_db #(input_vif) :: read_by_name (.scope
64         ("ifs"), .name("input_vif"), .val(vif)));
65 endfunction
66
67 virtual task run_phase(uvm_phase phase);
68     super.run_phase(phase);
69     // phase.raise_objection(this);
70     // fork
71     //     reset_signals();
72     fork
```

```
70         get_and_drive(phase);
71         run_perf_cnt();
72     join
73
74     // join
75     // phase.drop_objection(this);
76 endtask
77
78 virtual protected task reset_signals();
79     // wait (vif.rst === 1);
80     vif.str_rst <= 1;
81     // forever begin
82     // vif.valid <= '0;
83     vif.str_matrix_a <= 0;
84     vif.str_matrix_b <= 0;
85     vif.str_valid <= 0;
86     vif.str_en_perf <= 0;
87
88     resetAccumulatorStrassen();
89
90     pipe_cnt <= 0;
91     //@(posedge vif.clk) ;
92     // vif.str_valid <= 0;
93     #22;
94     @(posedge vif.clk);
```

```
95         vif.str_rst <= 0;
96         //end
97     endtask
98
99     virtual protected task fetch_fdata();
100         $fgets(line, fd);
101         products = line.atoi();
102
103         $display("%t | Strassen: products:%0d", $time,
104                 products);
105     endtask
106
107     virtual protected task get_and_drive(uvm_phase phase);
108
109         fd = $fopen("test_pipeline_str.txt", "r");
110         // config_fp = $fopen("../test_config.txt", "r");
111
112         reset_signals();
113         forever
114         begin
115             // vif.str_valid <= 0;
116             fetch_fdata();
117
118             // products = 100;
```

```
119
120     // vif.str_en_perf = 1;
121
122
123     if (products)
124     begin
125         repeat (4)
126         begin
127             vif.str_en_perf = 1;
128             products_modded = products * 2;
129             fork
130                 begin
131                     repeat (products_modded)
132                     begin
133                         seq_item_port.get(req);
134
135                         item_collected_port_str.write(
136                             req);
137
138                         drive_transfer(req);
139
140                         @(posedge vif.clk) ;
141                     end
142                 end
143             end
144         end
145     end
146     forever
```

```
143         begin
144             // $display("%t | Strassen Pipe
145                 Counter!", $time);
146
147             // $display("%t | Strassen Pipe
148                 Counter: %d", $time, pipe_cnt);
149
150             pipe_cnt++;
151
152             if (pipe_cnt == 7)
153                 vif.str_valid <= 1;
154
155             if (pipe_cnt == products_modded +
156                 7)
157                 break;
158
159             @(posedge vif.clk) ;
160         end
161     join
162     reset_signals ();
163 end
164 else
165 begin
```

```
165         $fclose (fd);
166         //@(posedge vif.rst);
167         $display ("%t | Finished Strassen
                Multiplications", $time);
168         break;
169     end
170
171     reset_signals ();
172 end
173
174     reset_signals ();
175 endtask
176
177 virtual protected task drive_transfer(packet_in tr);
178
179     // vif.str_en_perf = 1;
180
181     vif.str_matrix_a <= tr.str_data_a;
182     vif.str_matrix_b <= tr.str_data_b;
183     vif.str_en <= 1;
184
185     // $display ("\nStrassen input data\n%X\n%X", tr.
            str_data_a , tr.str_data_b);
186
187     // vif.str_en_perf = 0;
```

```
188     endtask
189
190     virtual protected task run_perf_cnt();
191     forever
192     begin
193     @(posedge vif.clk)
194         if(vif.str_en_perf)
195             vif.str_perf_cnt = vif.str_perf_cnt + 1;
196     end
197     endtask
198
199 endclass
```

I.10 Strassen Matched Size Driver

```
1 import "DPI-C" context function void
    resetAccumulatorStrassenFull();
2
3 class driver_strassen_full extends uvm_driver #(packet_in);
4
5     int index;
6
7     uvm_analysis_port #(packet_in) item_collected_port_strfull
        ;
8
9     `uvm_component_utils(driver_strassen_full)
10    input_vif vif;
11
12    int i, j;
13
14    int pipe_cnt;
15
16    int fd;
17    string line;
18    int products;
19
20    int products_modded;
21
```

```
22     event sample_cg;
23
24     covergroup input_cov @ (sample_cg);
25         option.per_instance = 1;
26
27         matrix_a : coverpoint vif.matrix_a_data {
28             bins range[] = {
29                 [0 : 255],
30                 [256 : 511],
31                 [512 : 767],
32                 [768 : 1023],
33                 [1024 : 1279],
34                 [1280 : 1535],
35                 [1536 : 1791],
36                 [1792 : 2047]
37             };
38     }
39     matrix_b : coverpoint vif.matrix_a_data {
40         bins range[] = {
41             [0 : 255],
42             [256 : 511],
43             [512 : 767],
44             [768 : 1023],
45             [1024 : 1279],
46             [1280 : 1535],
```

```
47         [1536 : 1791],
48         [1792 : 2047]
49     };
50 }
51 endgroup
52
53 function new(string name = "driver_strassen_full",
54             uvm_component parent = null);
55     super.new(name, parent);
56     item_collected_port_strfull = new ("
57         item_collected_port_strfull", this);
58     input_cov = new();
59 endfunction
60
61 virtual function void build_phase(uvm_phase phase);
62     super.build_phase(phase);
63     void '(uvm_resource_db#(input_vif)::read_by_name(.scope
64         ("ifs"), .name("input_vif"), .val(vif)));
65 endfunction
66
67 virtual task run_phase(uvm_phase phase);
68     super.run_phase(phase);
69
70     fork
71         get_and_drive(phase);
```

```
69         run_perf_cnt ();
70     join
71
72     endtask
73
74     virtual protected task reset_signals ();
75         vif.strfull_rst <= 1;
76         vif.strfull_matrix_a <= 0;
77         vif.strfull_matrix_b <= 0;
78         vif.strfull_valid <= 0;
79         vif.strfull_en_perf <= 0;
80
81         resetAccumulatorStrassenFull ();
82
83         pipe_cnt <= 0;
84         #22;
85         @(posedge vif.clk);
86         vif.strfull_rst <= 0;
87     endtask
88
89     virtual protected task fetch_fdata ();
90         $fgets(line , fd);
91         products = line.atoi ();
92
```

```
93     $display ("%t | Strassen_Full: products:%0d", $time ,
           products);
94 endtask
95
96 virtual protected task get_and_drive(uvm_phase phase);
97
98     fd = $fopen("test_pipeline_strfull.txt","r");
99
100    reset_signals();
101    forever
102    begin
103        fetch_fdata();
104
105        if(products)
106        begin
107            vif.strfull_en_perf = 1;
108            fork
109                begin
110                    repeat(products)
111                    begin
112                        seq_item_port.get(req);
113
114                        item_collected_port_strfull.write(
                            req);
115                        drive_transfer(req);
```

```
116
117             @(posedge vif.clk) ;
118         end
119     end
120
121     forever
122     begin
123         // $display("%t | Strassen Pipe Counter
124             !", $time);
125
126         // $display("%t | Strassen Pipe Counter
127             : %d", $time, pipe_cnt);
128
129         pipe_cnt++;
130
131         if(pipe_cnt == 11)
132             vif.strfull_valid <= 1;
133
134         if(pipe_cnt == products + 11)
135             break;
136
137         @(posedge vif.clk) ;
138     end
139
140     join
141 end
```

```
139         else
140         begin
141             $fclose (fd);
142             //@(posedge vif.rst);
143             $display ("%t | Finished Strassen_Full
                    Multiplications", $time);
144             break;
145         end
146
147         reset_signals ();
148     end
149
150     reset_signals ();
151 endtask
152
153 virtual protected task drive_transfer(packet_in tr);
154
155     // vif.str_en_perf = 1;
156
157     vif.strfull_matrix_a <= tr.strfull_data_a;
158     vif.strfull_matrix_b <= tr.strfull_data_b;
159     vif.strfull_en <= 1;
160
161     // $display ("\nStrassen input data\n%X\n%X", tr.
            str_data_a , tr.str_data_b);
```



```
162
163     // vif.str_en_perf = 0;
164     endtask
165
166     virtual protected task run_perf_cnt();
167     forever
168     begin
169     @(posedge vif.clk)
170         if(vif.strfull_en_perf)
171             vif.strfull_perf_cnt = vif.strfull_perf_cnt + 1;
172     end
173     endtask
174
175 endclass
```

I.11 Sequencer

```
1 class sequencer extends uvm_sequencer #(packet_in);
2     'uvm_component_utils(sequencer)
3
4     function new (string name = "sequencer", uvm_component
5         parent = null);
6         super.new(name, parent);
7     endfunction
8 endclass: sequencer
```

I.12 Sequence

```
1 class sequence_in extends uvm_sequence #(packet_in);
2     'uvm_object_utils(sequence_in)
3
4     function new(string name="sequence_in");
5         super.new(name);
6     endfunction: new
7
8     task body;
9         packet_in tx;
10
11         forever begin
12             tx = packet_in::type_id::create("tx");
13             start_item(tx);
14             assert(tx.randomize());
15             finish_item(tx);
16         end
17     endtask: body
18 endclass: sequence_in
```

I.13 Agent

```
1 class agent extends uvm_agent;
2     sequencer sqr;
3     driver_systolic drvsys;
4     driver_strassen drvstr;
5     driver_strassen_full drvstrfull;
6
7     uvm_analysis_port #(packet_in) item_collected_port_sys;
8     uvm_analysis_port #(packet_in) item_collected_port_str;
9     uvm_analysis_port #(packet_in) item_collected_port_strfull
10
11     'uvm_component_utils(agent)
12
13     function new(string name = "agent", uvm_component parent =
14         null);
15         super.new(name, parent);
16         item_collected_port_sys = new("item_collected_port_sys
17             ", this);
18         item_collected_port_str = new("item_collected_port_str
19             ", this);
20         item_collected_port_strfull = new("
21             item_collected_port_strfull", this);
22     endfunction
```

```
19
20     virtual function void build_phase(uvm_phase phase);
21         super.build_phase(phase);
22         sqr = sequencer::type_id::create("sqr", this);
23         drvsys = driver_systolic::type_id::create("drvsys",
24             this);
25         drvstr = driver_strassen::type_id::create("drvstr",
26             this);
27         drvstrfull = driver_strassen_full::type_id::create("
28             drvstrfull", this);
29     endfunction
30
31     virtual function void connect_phase(uvm_phase phase);
32         super.connect_phase(phase);
33         drvsys.seq_item_port.connect(sqr.seq_item_export);
34         drvsys.item_collected_port_sys.connect(
35             item_collected_port_sys);
36
37         drvstr.seq_item_port.connect(sqr.seq_item_export);
38         drvstr.item_collected_port_str.connect(
39             item_collected_port_str);
40
41         drvstrfull.seq_item_port.connect(sqr.seq_item_export);
42         drvstrfull.item_collected_port_strfull.connect(
43             item_collected_port_strfull);
```

38 `endfunction`

39 `endclass: agent`

I.14 Input Interface

```
1
2 interface input_if (input clk);
3
4     import array_pkg::*;
5     arraydim_t matrix_a_data;
6     arraydim_t matrix_b_data;
7
8     logic rst;
9
10    logic start;
11
12    logic load;
13    logic mult;
14    logic acc;
15    logic valid;
16    logic ready;
17
18    logic str_rst;
19    logic str_en;
20    logic str_valid;
21    strindim4_t str_matrix_a;
22    strindim4_t str_matrix_b;
23
```

```
24
25     logic strfull_rst;
26     logic strfull_en;
27     logic strfull_valid;
28     matrix_t strfull_matrix_a;
29     matrix_t strfull_matrix_b;
30
31
32     logic en_perf;
33     int perf_cnt;
34
35     logic str_en_perf;
36     int str_perf_cnt;
37
38     logic strfull_en_perf;
39     int strfull_perf_cnt;
40
41     logic test_mode;
42     logic scan_in0;
43     logic scan_in1;
44     logic scan_en;
45
46 endinterface : input_if
```

I.15 Output Interface

```
1 interface output_if (input clk);
2
3     import array_pkg::*;
4     arrayout_t matrix_out;
5     stroutdim4_t str_out;
6     arrayout_t strfull_out;
7
8     logic scan_out0 , scan_out1;
9     logic valid , ready;
10
11 endinterface : output_if
```

I.16 Monitor

```
1 typedef virtual output_if output_vif;
2
3 class monitor_out extends uvm_monitor;
4     'uvm_component_utils(monitor_out)
5
6     output_vif vif;
7     input_vif vif_in;
8     event begin_record, end_record;
9     packet_out tr_sys;
10    packet_out tr_str;
11    packet_out tr_strfull;
12    uvm_analysis_port #(packet_out) item_collected_port_sys;
13    uvm_analysis_port #(packet_out) item_collected_port_str;
14    uvm_analysis_port #(packet_out)
15        item_collected_port_strfull;
16
17    function new(string name, uvm_component parent);
18        super.new(name, parent);
19        item_collected_port_sys = new ("
20            item_collected_port_sys", this);
21        item_collected_port_str = new ("
22            item_collected_port_str", this);
```

```
20         item_collected_port_strfull = new ("
           item_collected_port_strfull", this);
21     endfunction
22
23     virtual function void build_phase(uvm_phase phase);
24         super.build_phase(phase);
25         void'(uvm_resource_db #(output_vif)::read_by_name(.
           scope("ifs"), .name("output_vif"), .val(vif)));
26         void'(uvm_resource_db #(input_vif)::read_by_name(.scope
           ("ifs"), .name("input_vif"), .val(vif_in)));
27         tr_sys = packet_out::type_id::create("tr_sys");
28         tr_str = packet_out::type_id::create("tr_str");
29         tr_strfull = packet_out::type_id::create("tr_strfull")
           ;
30     endfunction
31
32     virtual task run_phase(uvm_phase phase);
33         super.run_phase(phase);
34         fork
35             collect_transactions(phase);
36         join
37     endtask
38
39     virtual task collect_transactions(uvm_phase phase);
40
```

```
41
42     wait(vif_in.rst === 1);
43     @(negedge vif_in.rst);
44
45     fork
46         begin
47             forever begin
48                 // vif_in.start <= 1;
49                 @(posedge vif_in.valid);
50                 // vif_in.start = 0;
51                 tr_sys.dout = vif.matrix_out;
52                 // $display("Matrix Data from RTL @ %t: %X
53                 ", $time, tr.dout);
54                 item_collected_port_sys.write(tr_sys);
55                 @(posedge vif_in.clk) ;
56                 @(posedge vif_in.clk) ;
57             end
58         end
59         begin
60             forever begin
61                 //wait(vif_in.str_valid == 1);
62                 @(negedge vif.clk);
63                 if(vif_in.str_valid == 1)
64                     begin
65                         tr_str.str_out = vif.str_out;
```

```
65             // $display("%t | Sending Strassen
                Packet Dut", $time);
66             item_collected_port_str.write(tr_str);
67         end
68     end
69 end
70 begin
71     forever begin
72         // wait(vif_in.str_valid == 1);
73         @(negedge vif.clk);
74         if(vif_in.strfull_valid == 1)
75             begin
76                 tr_strfull.strfull_out = vif.
                    strfull_out;
77                 // $display("%t | Sending Strassen
                    Packet Dut", $time);
78                 item_collected_port_strfull.write(
                    tr_strfull);
79             end
80         end
81     end
82     join
83
84
85     endtask
```

86 `endclass`

I.17 Reference Model

```
1 import array_pkg::*;
2
3 //import "DPI-C" context function void runModeller(matrix_t
4     data_a, matrix_t data_b, arrayout_t data_r);
5 import "DPI-C" context function void runModeller(matrix_t
6     data_a, matrix_t data_b);
7 import "DPI-C" context function void runStrassen(strindim4_t
8     data_a, strindim4_t data_b);
9 import "DPI-C" context function void runStrassenFull(matrix_t
10    data_a, matrix_t data_b);
11
12 import "DPI-C" context function int getElement(int index);
13 import "DPI-C" context function int getElementStrassen(int
14    index);
15 import "DPI-C" context function int getElementStrassenFull(int
16    index);
17 // https://methi1999.github.io/2020/07/08/dpi.html
18
19 class refmod extends uvm_component;
20     'uvm_component_utils(refmod)
21
22     input_vif vif;
23
24     packet_in tr_in_sys;
```

```
18     packet_in tr_in_str;
19     packet_in tr_in_strfull;
20     packet_out tr_out_sys;
21     packet_out tr_out_str[];
22     packet_out tr_out_strfull[];
23     uvm_get_port #(packet_in) in_sys;
24     uvm_get_port #(packet_in) in_str;
25     uvm_get_port #(packet_in) in_strfull;
26     uvm_analysis_port #(packet_out) out_sys;
27     uvm_analysis_port #(packet_out) out_str;
28     uvm_analysis_port #(packet_out) out_strfull;
29
30     int i, j, k, q;
31
32     function new(string name = "refmod", uvm_component parent)
33         ;
34         super.new(name, parent);
35         in_sys = new("in_sys", this);
36         in_str = new("in_str", this);
37         in_strfull = new("in_strfull", this);
38         out_sys = new("out_sys", this);
39         out_str = new("out_str", this);
40         out_strfull = new("out_strfull", this);
41     endfunction
```



```
42     virtual function void build_phase(uvm_phase phase);
43         super.build_phase(phase);
44         tr_out_sys = packet_out::type_id::create("tr_out_sys",
45             this);
46         tr_out_str = new[20];
47         foreach(tr_out_str[ii]) tr_out_str[ii] = packet_out::
48             type_id::create("tr_out_str");
49         tr_out_strfull = new[20];
50         foreach(tr_out_strfull[iii]) tr_out_strfull[iii] =
51             packet_out::type_id::create("tr_out_strfull");
52         // tr_out_str = packet_out::type_id::create("tr_out_str",
53             this);
54     void'(uvm_resource_db#(input_vif)::read_by_name(.scope
55         ("ifs"), .name("input_vif"), .val(vif)));
56     k = 0;
57     q = 0;
58     endfunction: build_phase
59
60     virtual task run_phase(uvm_phase phase);
61         super.run_phase(phase);
62
63         fork
64             forever begin
65                 fork
66                     forever begin
```

```
62         in_sys.get(tr_in_sys);
63         // $display("Data passed to model @ %t",
64             $time);
65         runModeller(tr_in_sys.data_a, tr_in_sys.
66             data_b);
67
68         // $display("Data to model @ %t: %X, %X",
69             $time, tr_in_sys.data_a, tr_in_sys.
70             data_b);
71     end
72
73     begin
74         @ (posedge vif.valid) ;
75
76         for(i = 0; i < MATRIXSIZE; i = i + 1)
77             begin
78                 for(j = 0; j < MATRIXSIZE; j = j + 1)
79                     begin
80                         // $display("%X", getElement(i*
81                             MATRIXSIZE + j));
82
83                         tr_out_sys.dout[(i*MATRIXSIZE + j)
84                             *ACCWIDTH +: ACCWIDTH] =
85                             getElement(i*MATRIXSIZE + j);
86
87                     end
88                 end
89             end
90     end
```

```
79         // $display("Data from model @ %t: %X",
80             $time, tr_out_sys.dout);
81     out_sys.write(tr_out_sys);
82     end
83     join
84 end
85 forever begin
86     in_str.get(tr_in_str);
87     // $display("Data passed to model @ %t", $time)
88     ;
89     // $display("Data to model @ %t: %X, %X", $time
90         , tr_in_str.str_data_a, tr_in_str.
91         str_data_b);
92     runStrassen(tr_in_str.str_data_a, tr_in_str.
93         str_data_b);
94     for(i = 0; i < STRDIM; i = i + 1)
95     begin
96         for(j = 0; j < STRDIM; j = j + 1)
97         begin
98             // $display("%X", getElementStrassen(i
99                 *4 + j));
100             tr_out_str[k].str_out[(i*STRDIM + j)*
101                 ACCWIDTH +: ACCWIDTH] =
```

```

                                getElementStrassen(i*STRDIM + j);
97         end
98     end
99     // $display("%t | Sending Strassen Packet Ref,
        %d", $time, k);
100    out_str.write(tr_out_str[k]);
101
102    if(k == 19)
103        k = 0;
104    else
105        k = k + 1;
106
107    end
108    forever begin
109        in_strfull.get(tr_in_strfull);
110        // $display("Data passed to model @ %t", $time)
        ;
111        // $display("Data to model @ %t: %X, %X", $time
        , tr_in_str.str_data_a, tr_in_str.
        str_data_b);
112        runStrassenFull(tr_in_strfull.strfull_data_a,
        tr_in_strfull.strfull_data_b);
113
114        for(i = 0; i < MATRIXSIZE; i = i + 1)
115            begin
```

```
116         for(j = 0; j < MATRIXSIZE; j = j + 1)
117             begin
118                 // $display("%X", getElementStrassen(i
119                     // *4 + j));
120                 tr_out_strfull[q].strfull_out[(i*
121                     MATRIXSIZE + j)*ACCWIDTH +:
122                     ACCWIDTH] = getElementStrassenFull(
123                         i*MATRIXSIZE + j);
124             end
125         end
126         // $display("%t | Sending Strassen Packet Ref,
127             // %d", $time, k);
128         out_strfull.write(tr_out_strfull[q]);
129
130         if(q == 19)
131             q = 0;
132         else
133             q = q + 1;
134         end
135     join
136
137     endtask: run_phase
138 endclass: refmod
```

I.18 C Modeller

```
1 #include <stdlib.h>
2 #include "svdpi.h"
3 #include <math.h>
4 #include <stdio.h>
5
6
7 #define DATAWIDTH 8
8 #define ACCWIDTH 32
9
10 #define MATRIXSIZE 16
11 #define STRDIM 4
12
13 signed char matrix_tmp[MATRIXSIZE][MATRIXSIZE];
14 static signed int matrix_r[MATRIXSIZE*MATRIXSIZE];
15 static signed int matrix_s[STRDIM*STRDIM];
16 static signed int matrix_sf[MATRIXSIZE*MATRIXSIZE];
17
18 void transpose(signed char* A)
19 {
20     int i, j;
21     for (i = 0; i < MATRIXSIZE; i++)
22         for (j = 0; j < MATRIXSIZE; j++)
23             matrix_tmp[i][j] = A[j*MATRIXSIZE+i];
```

```
24
25     for (i = 0; i < MATRIXSIZE; i++)
26         for (j = 0; j < MATRIXSIZE; j++)
27             A[i*MATRIXSIZE + j] = matrix_tmp[i][j];
28 }
29
30
31 void printInputs(signed char* a, signed char* b) {
32     printf("Matrix A\n");
33     for (int j = 0; j < MATRIXSIZE; j++)
34     {
35         for (int k = 0; k < MATRIXSIZE; k++)
36         {
37             printf("%02X ", a[j*MATRIXSIZE + k]);
38         }
39         printf("\n");
40     }
41
42     printf("Matrix B\n");
43     for (int j = 0; j < MATRIXSIZE; j++)
44     {
45         for (int k = 0; k < MATRIXSIZE; k++)
46         {
47             printf("%02X ", b[j*MATRIXSIZE + k]);
48         }
```

```
49     printf("\n");
50 }
51 }
52
53 void printProducts(signed int* r) {
54     printf("Product Matrix\n");
55     for (int j = 0; j < MATRIXSIZE; j++)
56     {
57         for (int k = 0; k < MATRIXSIZE; k++)
58         {
59             printf("%08X ", r[j*MATRIXSIZE + k]);
60         }
61         printf("\n");
62     }
63 }
64
65 extern "C" void runModeller(signed char* matrix_a, signed char
    * matrix_b) {
66
67     int suma;
68     int index;
69
70     // printInputs(matrix_a, matrix_b);
71
72     transpose(matrix_a) ;
```



```
73
74     for (int j = 0; j < MATRIXSIZE; j++)
75     {
76         for (int k = 0; k < MATRIXSIZE; k++)
77         {
78             index = j*MATRIXSIZE + k;
79
80             suma = matrix_r[index];
81             for (int l = 0; l < MATRIXSIZE; l++)
82                 suma += matrix_a[j*MATRIXSIZE + l]*matrix_b[l*
                        MATRIXSIZE + k];
83
84             matrix_r[index] = suma;
85         }
86
87     }
88
89     // printProducts(matrix_r);
90 }
91
92 extern "C" void runStrassen(signed char* matrix_a, signed char
    * matrix_b) {
93
94     int suma;
95     int index;
```

```
96
97
98     // printf(" Matrix A\n");
99     // for (int j = 0; j < STRDIM; j++)
100     // {
101     //     for (int k = 0; k < STRDIM; k++)
102     //     {
103     //         printf("%02X ", matrix_a[j*STRDIM + k]);
104     //     }
105     //     printf("\n");
106     // }
107     // printf(" Matrix B\n");
108     // for (int j = 0; j < STRDIM; j++)
109     // {
110     //     for (int k = 0; k < STRDIM; k++)
111     //     {
112     //         printf("%02X ", matrix_b[j*STRDIM + k]);
113     //     }
114     //     printf("\n");
115     // }
116
117     for (int j = 0; j < STRDIM; j++)
118     {
119         for (int k = 0; k < STRDIM; k++)
120         {
```

```
121         index = j*STRDIM + k;
122
123         suma = matrix_s[index];
124         for (int l = 0; l < STRDIM; l++)
125             suma += matrix_a[j*STRDIM + l]*matrix_b[l*
                STRDIM + k];
126
127         matrix_s[index] = suma;
128     }
129
130 }
131
132 // printf(" Strassen product\n");
133 // for (int j = 0; j < STRDIM; j++)
134 // {
135 //     for (int k = 0; k < STRDIM; k++)
136 //     {
137 //         printf("%08X ", matrix_s[j*STRDIM + k]);
138 //     }
139 //     printf("\n");
140 // }
141
142 }
143
144
```

```
145 extern "C" void runStrassenFull(signed char* matrix_a, signed
    char* matrix_b) {
146
147     int suma;
148     int index;
149
150
151     for (int j = 0; j < MATRIXSIZE; j++)
152     {
153         for (int k = 0; k < MATRIXSIZE; k++)
154         {
155             index = j*MATRIXSIZE + k;
156
157             suma = matrix_sf[index];
158             for (int l = 0; l < MATRIXSIZE; l++)
159                 suma += matrix_a[j*MATRIXSIZE + l]*matrix_b[l*
                    MATRIXSIZE + k];
160
161             matrix_sf[index] = suma;
162         }
163
164     }
165 }
166
167
```

```
168
169 extern "C" unsigned int getElement(int index) {
170     return matrix_r[index];
171 }
172
173 extern "C" unsigned int getElementStrassen(int index) {
174     return matrix_s[index];
175 }
176
177 extern "C" unsigned int getElementStrassenFull(int index) {
178     return matrix_sf[index];
179 }
180
181 extern "C" void resetAccumulator() {
182     for (int j = 0; j < MATRIXSIZE; j++)
183         for (int k = 0; k < MATRIXSIZE; k++)
184             matrix_r[j*MATRIXSIZE + k] = 0;
185 }
186
187 extern "C" void resetAccumulatorStrassen() {
188     for (int j = 0; j < STRDIM; j++)
189         for (int k = 0; k < STRDIM; k++)
190             matrix_s[j*STRDIM + k] = 0;
191 }
192
```

```
193 extern "C" void resetAccumulatorStrassenFull () {
194     for (int j = 0; j < MATRIXSIZE; j++)
195         for (int k = 0; k < MATRIXSIZE; k++)
196             matrix_sf[j*MATRIXSIZE + k] = 0;
197 }
```

I.19 Packet In

```
1 class packet_in extends uvm_sequence_item;
2     // 'uvm_object_utils(packet_in)
3
4     rand matrix_t data_a;
5     rand matrix_t data_b;
6
7     rand strindim4_t str_data_a;
8     rand strindim4_t str_data_b;
9
10    rand matrix_t strfull_data_a;
11    rand matrix_t strfull_data_b;
12
13    'uvm_object_utils_begin(packet_in)
14        'uvm_field_int(data_a , UVM_ALL_ON|UVM_HEX)
15        'uvm_field_int(data_b , UVM_ALL_ON|UVM_HEX)
16        'uvm_field_int(str_data_a , UVM_ALL_ON|UVM_HEX)
17        'uvm_field_int(str_data_b , UVM_ALL_ON|UVM_HEX)
18        'uvm_field_int(strfull_data_a , UVM_ALL_ON|UVM_HEX)
19        'uvm_field_int(strfull_data_b , UVM_ALL_ON|UVM_HEX)
20    'uvm_object_utils_end
21
22    function new(string name="packet_in");
23        super.new(name);
```

24 `endfunction: new`

25 `endclass: packet_in`

I.20 Packet Out

```
1 import array_pkg::*;
2
3 class packet_out extends uvm_sequence_item;
4     // 'uvm_object_utils(packet_out)
5
6     arrayout_t dout;
7     stroutdim4_t str_out;
8     arrayout_t strfull_out;
9     // refout_t refout;
10
11     'uvm_object_utils_begin(packet_out)
12         'uvm_field_int(dout, UVM_ALL_ON|UVM_HEX)
13         'uvm_field_int(str_out, UVM_ALL_ON|UVM_HEX)
14         'uvm_field_int(strfull_out, UVM_ALL_ON|UVM_HEX)
15         // 'uvm_field_int(dout_flag, UVM_ALL_ON|UVM_HEX)
16         // 'uvm_field_int(digit_clk, UVM_ALL_ON|UVM_HEX)
17     'uvm_object_utils_end
18
19     function new(string name="packet_out");
20         super.new(name);
21     endfunction: new
22 endclass: packet_out
```

I.21 Environment

```
1 class env extends uvm_env;
2     agent      mst;
3     refmod     rfm;
4     agent_out  slv;
5     comparator comp;
6     uvm_tlm_analysis_fifo #(packet_in) to_refmod_sys;
7     uvm_tlm_analysis_fifo #(packet_out) from_refmod_sys;
8     uvm_tlm_analysis_fifo #(packet_out) from_dut_sys;
9
10    uvm_tlm_analysis_fifo #(packet_in) to_refmod_str;
11    uvm_tlm_analysis_fifo #(packet_out) from_refmod_str;
12    uvm_tlm_analysis_fifo #(packet_out) from_dut_str;
13
14    uvm_tlm_analysis_fifo #(packet_in) to_refmod_strfull;
15    uvm_tlm_analysis_fifo #(packet_out) from_refmod_strfull;
16    uvm_tlm_analysis_fifo #(packet_out) from_dut_strfull;
17
18    `uvm_component_utils(env)
19
20    function new(string name, uvm_component parent = null);
21        super.new(name, parent);
22        to_refmod_sys = new("to_refmod_sys", this);
23        from_refmod_sys = new("from_refmod_sys", this);
```

```
24     from_dut_sys = new("from_dut_sys", this);
25
26     to_refmod_str = new("to_refmod_str", this);
27     from_refmod_str = new("from_refmod_str", this);
28     from_dut_str = new("from_dut_str", this);
29
30     to_refmod_strfull = new("to_refmod_strfull", this);
31     from_refmod_strfull = new("from_refmod_strfull", this)
32     ;
32     from_dut_strfull = new("from_dut_strfull", this);
33 endfunction
34
35 virtual function void build_phase(uvm_phase phase);
36     super.build_phase(phase);
37     mst = agent::type_id::create("mst", this);
38     slv = agent_out::type_id::create("slv", this);
39     rfm = refmod::type_id::create("rfm", this);
40     comp = comparator::type_id::create("comp", this);
41 endfunction
42
43 virtual function void connect_phase(uvm_phase phase);
44     super.connect_phase(phase);
45     // Connect MST to FIFO
46     mst.item_collected_port_sys.connect(to_refmod_sys.
47         analysis_export);
```

```
47     mst.item_collected_port_str.connect(to_refmod_str.  
        analysis_export);  
48     mst.item_collected_port_strfull.connect(  
        to_refmod_strfull.analysis_export);  
49     // Connect FIFO to REFMOD  
50     rfm.in_sys.connect(to_refmod_sys.get_export);  
51     rfm.in_str.connect(to_refmod_str.get_export);  
52     rfm.in_strfull.connect(to_refmod_strfull.get_export);  
53  
54     // Connect scoreboard  
55     rfm.out_sys.connect(from_refmod_sys.analysis_export);  
56     rfm.out_str.connect(from_refmod_str.analysis_export);  
57     rfm.out_strfull.connect(from_refmod_strfull.  
        analysis_export);  
58     slv.item_collected_port_sys.connect(from_dut_sys.  
        analysis_export);  
59     slv.item_collected_port_str.connect(from_dut_str.  
        analysis_export);  
60     slv.item_collected_port_strfull.connect(  
        from_dut_strfull.analysis_export);  
61  
62     comp.from_refmod_sys.connect(from_refmod_sys.  
        get_export);  
63     comp.from_refmod_str.connect(from_refmod_str.  
        get_export);
```

```
64         comp.from_refmod_strfull.connect(from_refmod_strfull.  
        get_export);  
65  
66         comp.from_dut_sys.connect(from_dut_sys.get_export);  
67         comp.from_dut_str.connect(from_dut_str.get_export);  
68         comp.from_dut_strfull.connect(from_dut_strfull.  
        get_export);  
69     endfunction  
70  
71     virtual function void end_of_elaboration_phase(uvm_phase  
        phase);  
72         super.end_of_elaboration_phase(phase);  
73     endfunction  
74  
75     virtual function void report_phase(uvm_phase phase);  
76         super.report_phase(phase);  
77         `uvm_info(get_type_name(), $sformatf("Reporting  
        matched %0d", comp.m_matches), UVM_NONE)  
78         if (comp.m_mismatches) begin  
79             `uvm_error(get_type_name(), $sformatf("Saw %0d  
        mismatched samples", comp.m_mismatches))  
80         end  
81     endfunction  
82 endclass
```

I.22 Comparator

```
1 //class comparator #(type T = packet_out) extends
    uvm_scoreboard;
2 class comparator extends uvm_scoreboard;
3 //typedef comparator #(T) this_type;
4 `uvm_component_param_utils(comparator)
5
6 const static string type_name = "comparator";
7
8 input_vif vif;
9 output_vif vif_out;
10
11 uvm_get_port #(packet_out) from_refmod_sys;
12 uvm_get_port #(packet_out) from_dut_sys;
13
14 uvm_get_port #(packet_out) from_refmod_str;
15 uvm_get_port #(packet_out) from_dut_str;
16
17 uvm_get_port #(packet_out) from_refmod_strfull;
18 uvm_get_port #(packet_out) from_dut_strfull;
19
20 packet_out tr_refmod_sys;
21 packet_out tr_dut_sys;
22
```

```
23  packet_out tr_refmod_str;
24  packet_out tr_dut_str;
25
26  packet_out tr_refmod_strfull;
27  packet_out tr_dut_strfull;
28
29  int m_matches, m_mismatches;
30  int str_matches, str_mismatches;
31  int strfull_matches, strfull_mismatches;
32  event compared, end_of_simulation;
33
34  int sum, i, j;
35
36  int fd;
37  string line;
38  int length;
39  int sys_length;
40
41  function new(string name, uvm_component parent);
42      super.new(name, parent);
43      void '(uvm_resource_db#(input_vif)::read_by_name
44          (.scope("ifs"), .name("input_vif"), .val(vif)));
45      void '(uvm_resource_db#(output_vif)::read_by_name
46          (.scope("ifs"), .name("output_vif"), .val(vif_out
47              )));
```

```
47     from_refmod_sys = new("from_refmod_sys", this);
48     from_dut_sys = new("from_dut_sys", this);
49     from_refmod_str = new("from_refmod_str", this);
50     from_dut_str = new("from_dut_str", this);
51     from_refmod_strfull = new("from_refmod_strfull", this);
52     from_dut_strfull = new("from_dut_strfull", this);
53     endfunction
54
55     virtual function string get_type_name();
56         return type_name;
57     endfunction
58
59
60     virtual function void build_phase(uvm_phase phase);
61         super.build_phase(phase);
62         fd = $fopen("test_pipeline.txt", "r");
63         length = 0;
64
65         forever begin
66             $fgets(line, fd);
67             if($feof(fd))
68                 break;
69             length = length + line.atoi();
70         end
71
```



```
72     $display("Performing %0d multiplications", length);
73
74     $fclose(fd);
75 endfunction
76
77 task run_phase(uvm_phase phase);
78     phase.raise_objection(this);
79
80     //length = 20;
81     m_matches = 0;
82     m_mismatches = 0;
83
84     str_matches = 0;
85     str_mismatches = 0;
86
87
88     //$display("%t | Running Comparator", $time);
89
90     fork
91         begin
92             //$display("%t | Entering Systolic Comparison", $time)
93             ;
94             if(NORESET)
95                 sys_length = 1;
96         else
```

```
96         sys_length = length ;
97     repeat (sys_length)
98     begin
99         from_refmod_sys.get(tr_refmod_sys);
100        from_dut_sys.get(tr_dut_sys);
101        // $display("%t | Recieved Systolic Packets", $time);
102
103        // $display("%0t\t", $time);
104        // $display("\tREF: %X\n\tDUT: %X", tr_refmod.dout,
105            tr_dut.dout);
106
107        if (check_identical())
108            m_matches++;
109        else
110            begin
111                m_mismatches++;
112                $display("%t | MISMATCH SYSTOLIC\n\tREF: %X\n\tDUT
113                    : %X", $time, tr_refmod_sys.dout, tr_dut_sys.
114                    dout);
115                // phase.drop_objection(this);
116            end
117        // $display("%t | m total : %d", $time, m_matches +
118            m_mismatches);
119    end
```

```
117     end
118
119     begin
120         // $display("%t | Entering Strassen Comparison", $time)
121         ;
122     repeat (length*8)
123     begin
124         from_dut_str.get(tr_dut_str);
125         from_refmod_str.get(tr_refmod_str);
126         // $display("%t | Recieved Strassen Packets", $time);
127         // $display("%0t\t", $time);
128         // $display("\tREF: %X\n\tDUT: %X", tr_refmod_str.
129             str_out, tr_dut_str.str_out);
130
131         // $display("REF");
132         // for (int j = 0; j < 4; j++)
133         // begin
134         //     $display("%X ", tr_refmod_str.str_out[(j*4)*
135             ACCWIDTH +: ACCWIDTH*4]);
136         // end
137         // $display("DUT");
138         // for (int j = 0; j < 4; j++)
139         // begin
140         //     $display("%X ", tr_dut_str.str_out[(j*4)*
141             ACCWIDTH +: ACCWIDTH*4]);
```

```
138         //end
139
140         if( check_identical_str() )
141             str_matches++;
142         else
143             begin
144                 str_mismatches++;
145                 $display("%t | MISMATCH STRASSEN\n\tREF: %X\n\tDUT
                    : %X", $time, tr_refmod_str.str_out, tr_dut_str
                    .str_out);
146
147                 //phase.drop_objection(this);
148             end
149             // $display("%t | str total : %d", $time, str_matches
                    + str_mismatches);
150
151         end
152     end
153
154     begin
155         // $display("%t | Entering Strassen Comparison", $time)
                    ;
156         repeat(length)
157             begin
158                 from_dut_strfull.get(tr_dut_strfull);
```

```
159         from_refmod_strfull.get(tr_refmod_strfull);
160
161         if(check_identical_str_full())
162             strfull_matches++;
163         else
164             begin
165                 strfull_mismatches++;
166                 $display("%t | MISMATCH STRASSEN FULL\n\tREF: %X\n
\tDUT: %X", $time, tr_refmod_strfull.
strfull_out, tr_dut_strfull.strfull_out);
167
168                 //phase.drop_objection(this);
169             end
170             // $display("%t | str_full total : %d", $time,
strfull_matches + strfull_mismatches);
171
172         end
173     end
174 join
175
176     phase.drop_objection(this);
177 endtask
178
179 function int check_identical();
180     return ~!(tr_refmod_sys.dout ^ tr_dut_sys.dout);
```

```
181  endfunction : check_identical
182
183  function int check_identical_str();
184      return ~!(tr_refmod_str.str_out ^ tr_dut_str.str_out);
185  endfunction : check_identical_str
186
187  function int check_identical_str_full();
188      return ~!(tr_refmod_strfull.strfull_out ^ tr_dut_strfull.
          strfull_out);
189  endfunction : check_identical_str_full
190
191
192  virtual function void report_phase(uvm_phase phase);
193      super.report_phase(phase);
194      $display("Program End");
195      $display("==== Results =====");
196      $display("Systolic Matches : [%0d] vs Mismatches : [%0d]",
          m_matches, m_mismatches);
197      $display("Strassen Matches : [%0d] vs Mismatches : [%0d]",
          str_matches, str_mismatches);
198      $display("Strassen_Full Matches : [%0d] vs Mismatches :
          [%0d]", strfull_matches, strfull_mismatches);
199      $display("Systolic Performance Counter : [%0d] cycles",
          vif.perf_cnt);
```

```
200     $display("Strassen Performance Counter : [%0d] cycles",
           vif.str_perf_cnt);
201     $display("Strassen_Full Performance Counter : [%0d] cycles
           ", vif.strfull_perf_cnt);
202 endfunction
203
204 endclass
```

I.23 Top

```
1  `include "uvm_macros.svh"
2  `include "array_pkg.sv"
3  `include "test_pkg.sv"
4  `include "input_if.sv"
5  `include "output_if.sv"
6  `include "assertions.sv"
7
8  //Top
9  module test;
10
11     import uvm_pkg::*;
12     import array_pkg::*;
13     import test_pkg::*;
14
15     logic clk;
16
17     initial begin
18         clk = 0;
19         //rst = 1;
20         // #22 rst = 0;
21
22     end
23
```



```
24  always #5 clk = !clk;
25
26  input_if in(clk);
27  output_if out(clk);
28
29  matrix_multiplier #(
30  .DATAWIDTH(DATAWIDTH) ,
31  .ACCWIDTH(ACCWIDTH) ,
32  .MATRIXSIZE(MATRIXSIZE) )
33  top
34  (
35  .clk(clk) ,
36  .rst(in.rst) ,
37  .scan_in0(in.scan_in0) ,
38  .scan_en(in.scan_en) ,
39  .test_mode(in.test_mode) ,
40
41  .mult(in.mult) ,
42  .load(in.load) ,
43  .acc(in.acc) ,
44  .matrix_a_data(in.matrix_a_data) ,
45  .matrix_b_data(in.matrix_b_data) ,
46  .matrix_out(out.matrix_out) ,
47
48  .str_rst(in.str_rst) ,
```

```
49     .str_en(in.str_en),
50     .str_matrix_a(in.str_matrix_a),
51     .str_matrix_b(in.str_matrix_b),
52     .str_out(out.str_out),
53
54     .strfull_rst(in.strfull_rst),
55     .strfull_en(in.strfull_en),
56     .strfull_matrix_a(in.strfull_matrix_a),
57     .strfull_matrix_b(in.strfull_matrix_b),
58     .strfull_out(out.strfull_out)
59
60 );
61
62
63 /*
64     arraydim_t a_aligned;
65     arraydim_t b_aligned;
66
67     data_alignment #(.DATAWIDTH(DATAWIDTH), .MATRIXSIZE(
68         MATRIXSIZE)) aligner (
69         .clk(clk),
70         .reset(in.rst),
71         .shift(in.mult),
72         .a_in(in.matrix_a_data),
73         .b_in(in.matrix_b_data),
```

```
73     .a_out(a_aligned),
74     .b_out(b_aligned)
75 );
76
77 multiplier_systolic_array #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH
    (ACCWIDTH), .MATRIXSIZE(MATRIXSIZE)) sys (
78     .clk(clk),
79     .reset(in.rst),
80     .a_in(a_aligned),
81     .b_in(b_aligned),
82     .load_en(in.load),
83     .mult_en(in.mult),
84     .acc_en(in.acc),
85     .d_out(out.matrix_out),
86
87     .scan_in0(in.scan_in0),
88     .scan_en(in.scan_en),
89     .test_mode(in.test_mode)
90     // .scan_out0(out.scan_out0)
91 );
92
93 multiplier_strassen #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH(
    ACCWIDTH)) str (
94     .clk(clk),
95     .reset(in.str_rst),
```

```
96     .enable(in.str_en),
97     .matrix_a(in.str_matrix_a),
98     .matrix_b(in.str_matrix_b),
99     .matrix_c(out.str_out),
100
101     .scan_in0(in.scan_in0),
102     .scan_en(in.scan_en),
103     .test_mode(in.test_mode)
104     // .scan_out0(out.scan_out0)
105 );
106
107 multiplier_strassen_full #(.DATAWIDTH(DATAWIDTH), .
108     ACCWIDTH(ACCWIDTH)) strfull (
109     .clk(clk),
110     .reset(in.strfull_rst),
111     .enable(in.strfull_en),
112     .matrix_a(in.strfull_matrix_a),
113     .matrix_b(in.strfull_matrix_b),
114     .matrix_c(out.strfull_out),
115
116     .scan_in0(in.scan_in0),
117     .scan_en(in.scan_en),
118     .test_mode(in.test_mode)
119     // .scan_out0(out.scan_out0)
120 );
```

```
120
121  */
122
123  //assertion_cov ass(out);
124
125  initial begin
126    `ifdef INCA
127      $recordvars ();
128    `endif
129    `ifdef VCS
130      $vcdpluson ;
131    `endif
132    `ifdef QUESTA
133      $wlfdumpvars ();
134      set_config_int ("*", "recording_detail", 1);
135    `endif
136
137    `ifdef SDFSCAN
138      $sdf_annotate ("sdf/matrix_multiplier_tsmc18_scan.sdf",
139                    test.sys);
139    `endif
140    $set_coverage_db_name ("matrix_multiplier");
141    uvm_resource_db#(input_vif)::set (.scope ("ifs"), .name ("
    input_vif"), .val (in));
```

```
142     uvm_resource_db #(output_vif) :: set (.scope (" ifs " ), .name ("
        output_vif" ), .val (out));
143
144     run_test (" simple_test ");
145     end
146 endmodule
```

I.24 Test

```
1 class simple_test extends uvm_test;
2   env env_h;
3   sequence_in seq;
4
5   `uvm_component_utils(simple_test)
6
7   function new(string name, uvm_component parent = null);
8     super.new(name, parent);
9   endfunction
10
11  virtual function void build_phase(uvm_phase phase);
12    super.build_phase(phase);
13    env_h = env::type_id::create("env_h", this);
14    seq = sequence_in::type_id::create("seq", this);
15  endfunction
16
17  task run_phase(uvm_phase phase);
18    seq.start(env_h.mst.sqr);
19  endtask: run_phase
20
21 endclass
```
