5-2024

Benchmarking of Matrix Multiplication Acceleration Methods

Ashish Tondwalkar

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 artifiial 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 approahces of hardware acceleration.

Contents

List of Figures

List of Tables

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 testbench 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:

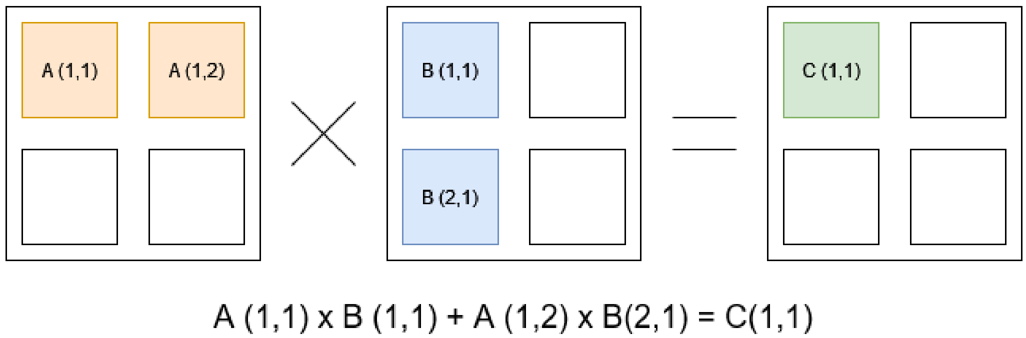

Figure 2.1: Blocked Matrix Multiplication

- 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

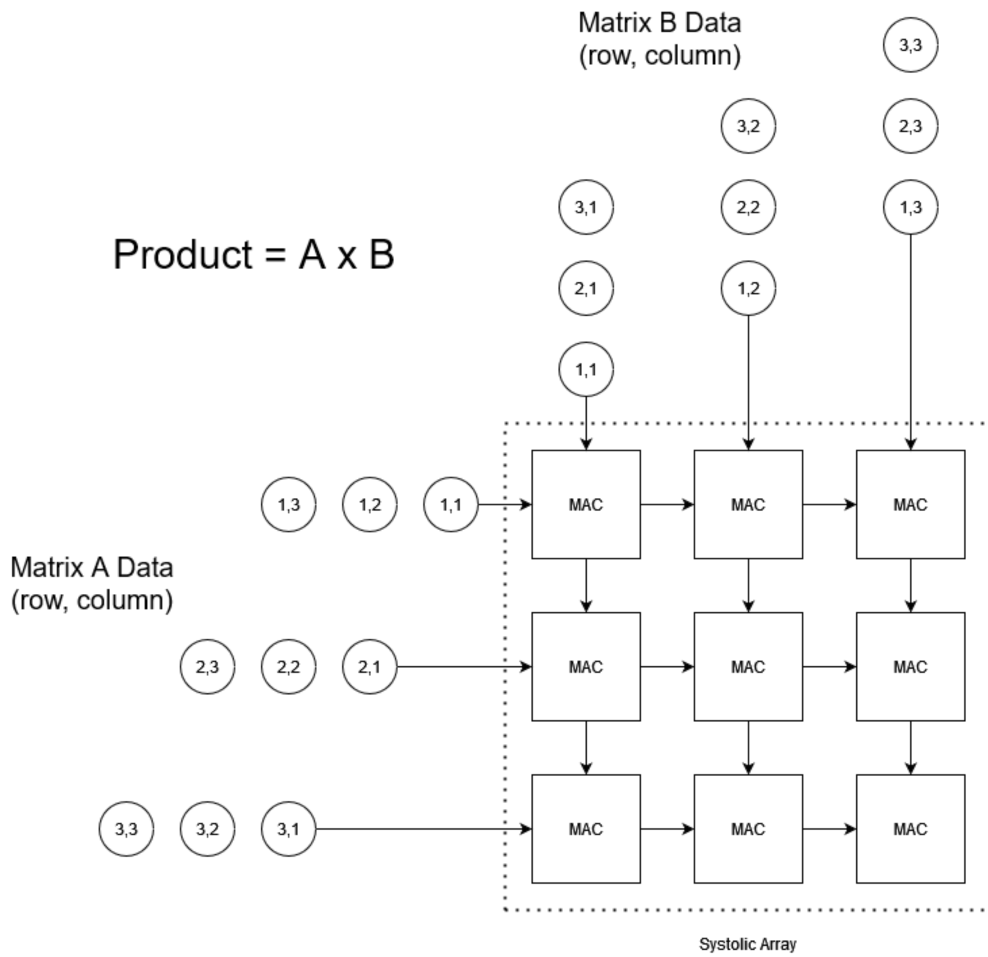

Figure 2.2: Operation of the Systolic Array

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:

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

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:

$$X = [\begin{matrix} A & B \\ C & D \end{matrix}], Y = [\begin{matrix} E & F \\ G & H \end{matrix}] \tag{2.2}$$

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:

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

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.

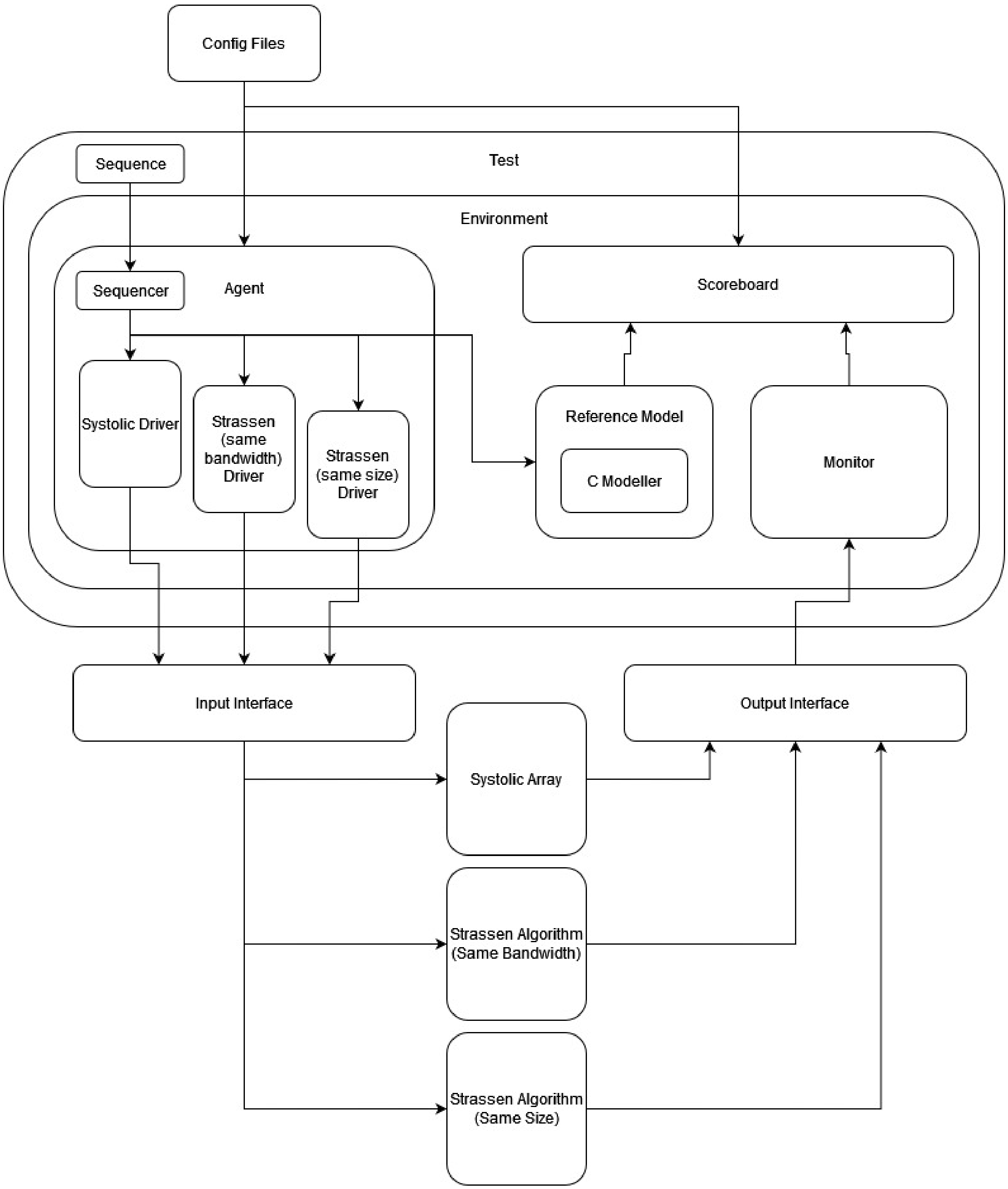

Figure 3.1: UVM Hierarchy

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 generated 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 multipliers 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.

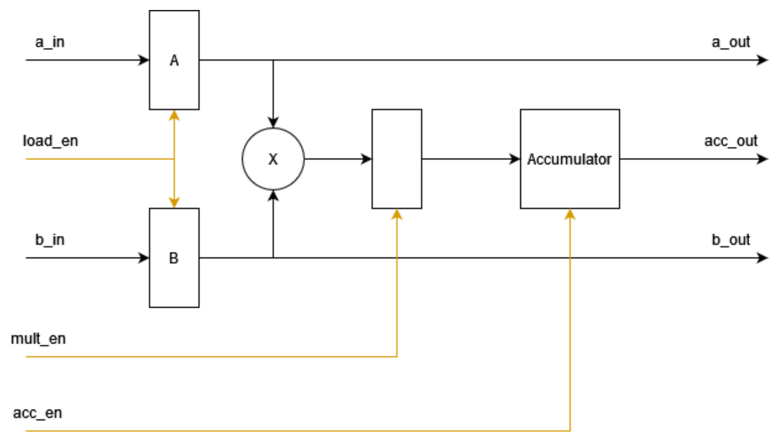

Figure 4.1: MAC Unit Diagram

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

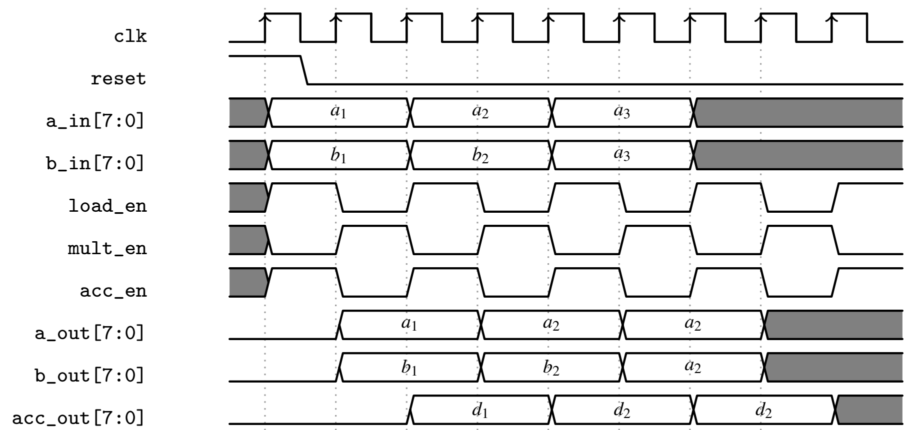

Figure 4.2: MAC Unit Timing Diagram

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

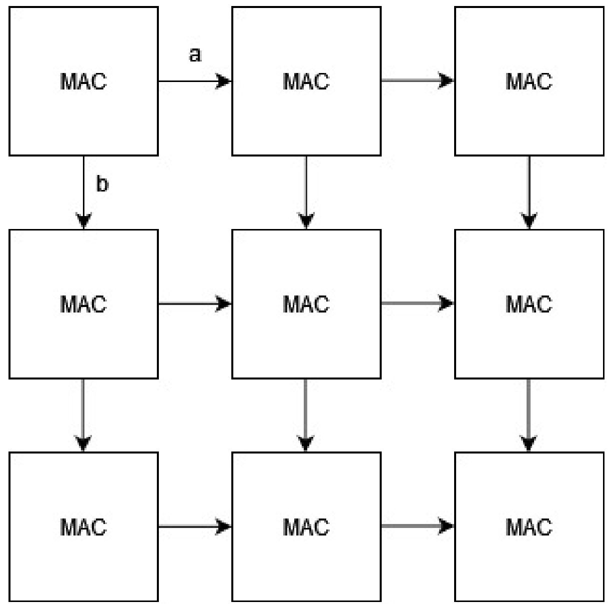

Figure 4.3: MAC Array

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

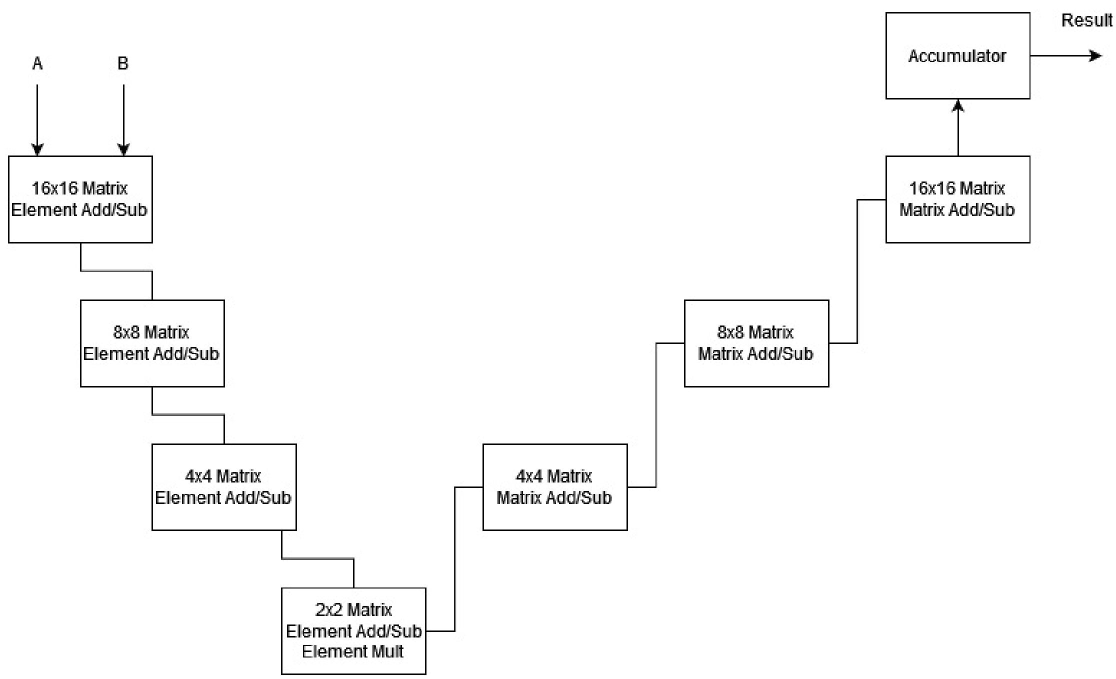

Figure 5.1: Recursive Algorithm

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 2×2 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:

$$A = [\begin{matrix} a & b \\ c & d \end{matrix}], B = [\begin{matrix} e & f \\ g & h \end{matrix}] \tag{5.1}$$

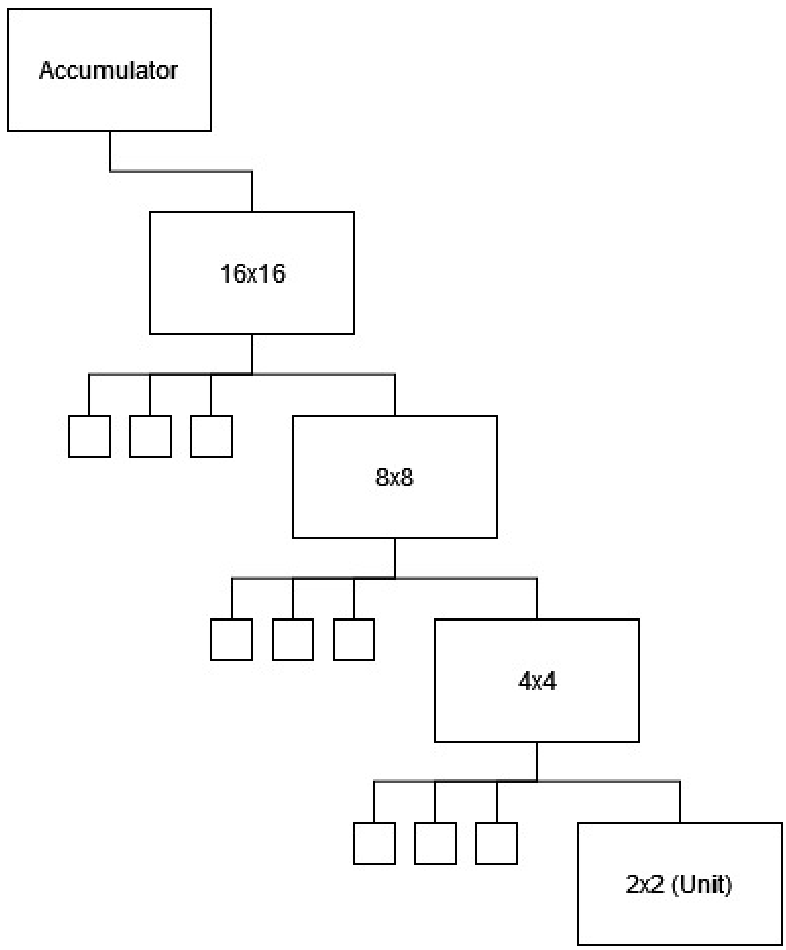

Figure 5.2: Recursive Algorithm

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.

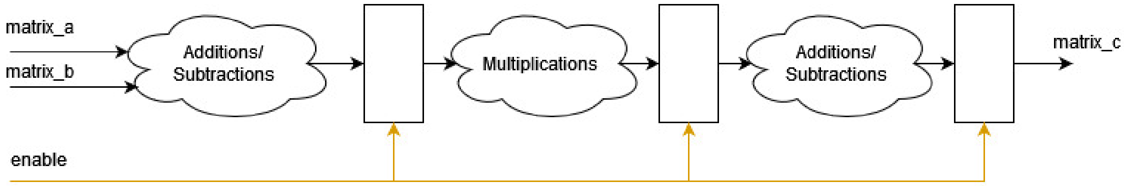

Figure 5.3: Strassen Unit Diagram

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	Ouptut	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.

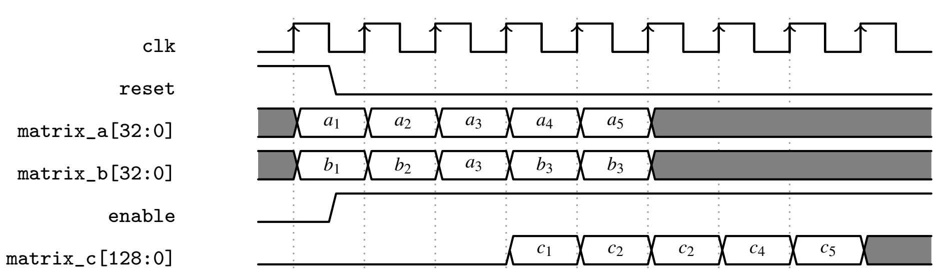

Figure 5.4: Strassen Unit Timing Diagram

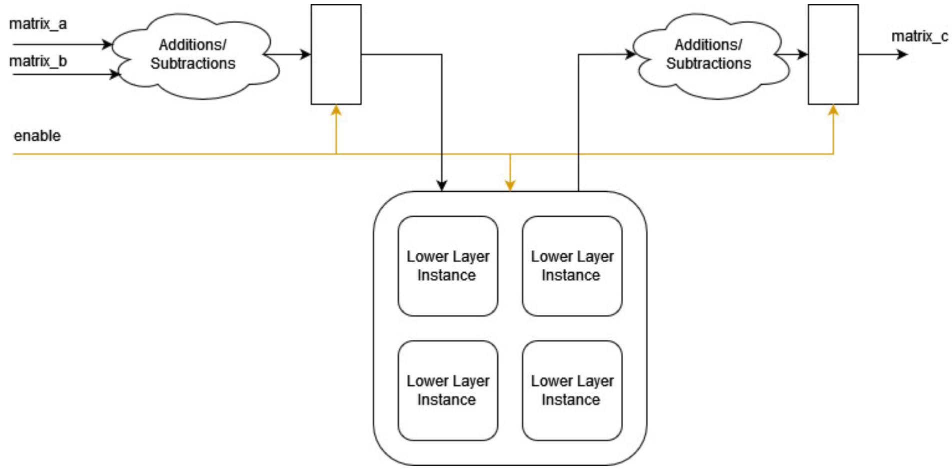

Figure 5.5: Strassen Layer Diagram

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	Ouptut	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: 4×4 Performance

	Clock Cycles		
Multiplications	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

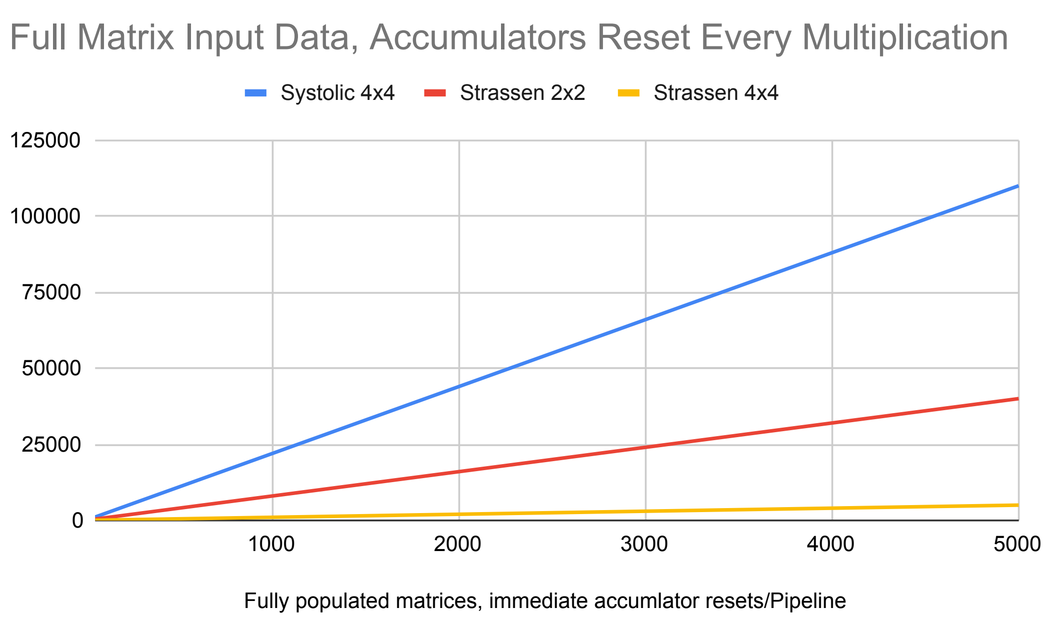

Figure 6.1: Plot of Performance Data for 4×4

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

	Clock Cycles		
Multiplications	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

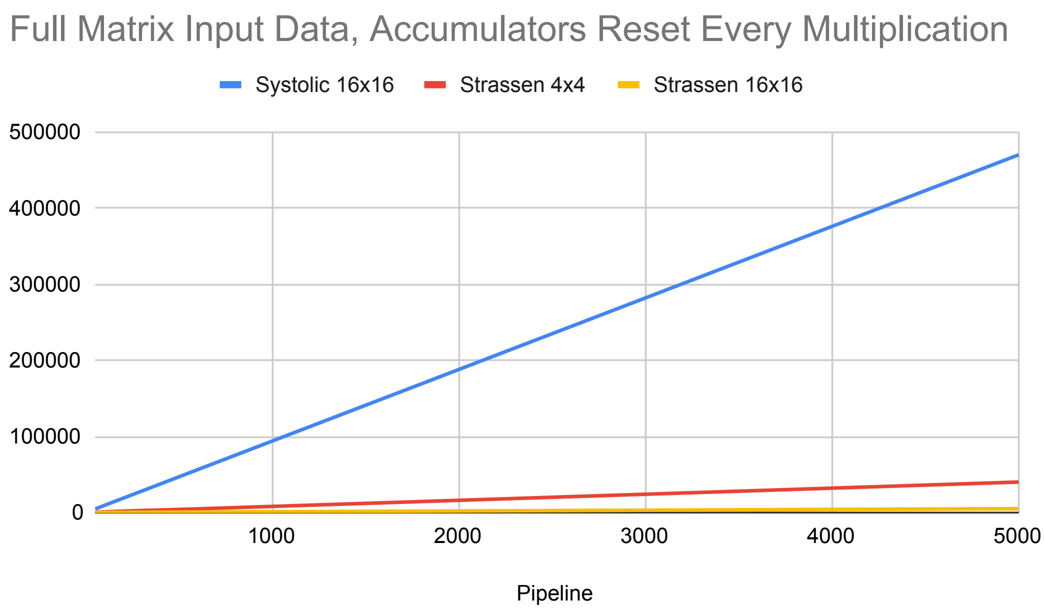

Figure 6.2: Plot of Performance Data for 16×16

6.1.3 64×64

The following results showcase performance for these matrices:

- 64×64 Systolic Array
- 8×8 Strassen Algorithm for matched bandwidth
- 64×64 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.3

Table 6.3: 64×64 Performance

	Clock Cycles		
Multiplications	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

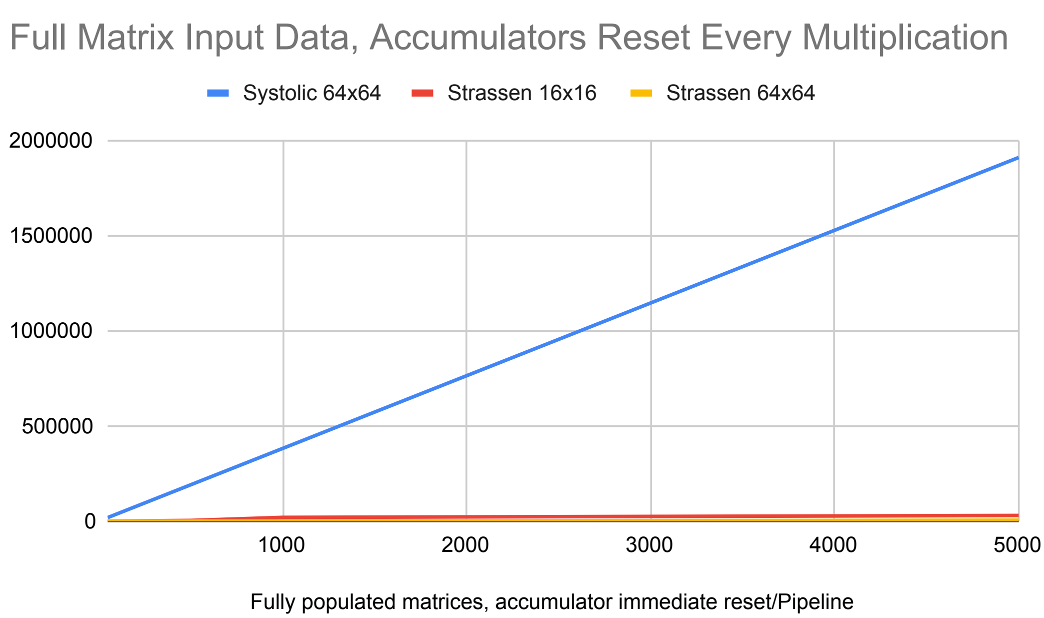

Figure 6.3: Plot of Performance Data for 64×64

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: 4×4 Performance

	Clock Cycles		
Multiplications	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

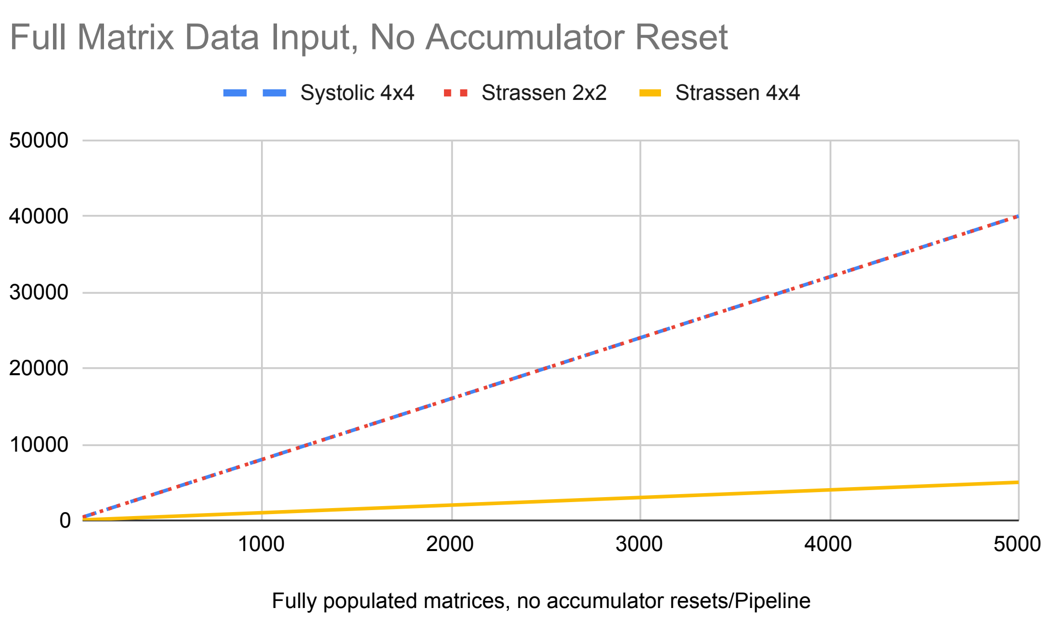

Figure 6.4: Plot of Performance Data for 4×4

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

	Clock Cycles		
Multiplications	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

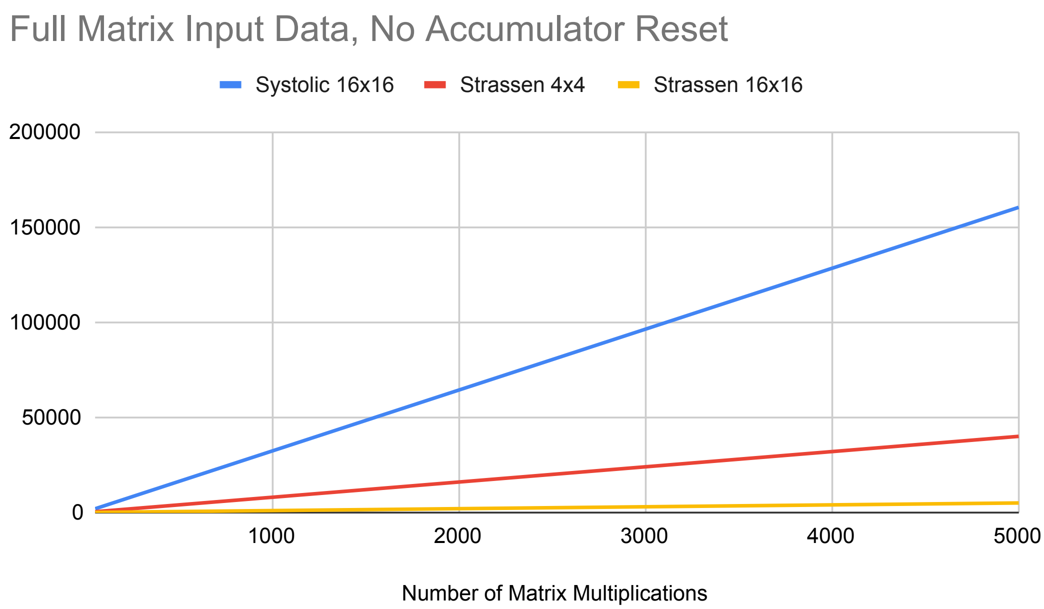

Figure 6.5: Plot of Performance Data for 16×16

6.2.3 64×64

The following results showcase performance for these matrices:

- 64×64 Systolic Array
- 8×8 Strassen Algorithm for matched bandwidth
- 64×64 Strassen Algorithm for matched size

The clock cycle count is seen from Table 6.7

Table 6.7: 64×64 Performance

	Clock Cycles		
Multiplications	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

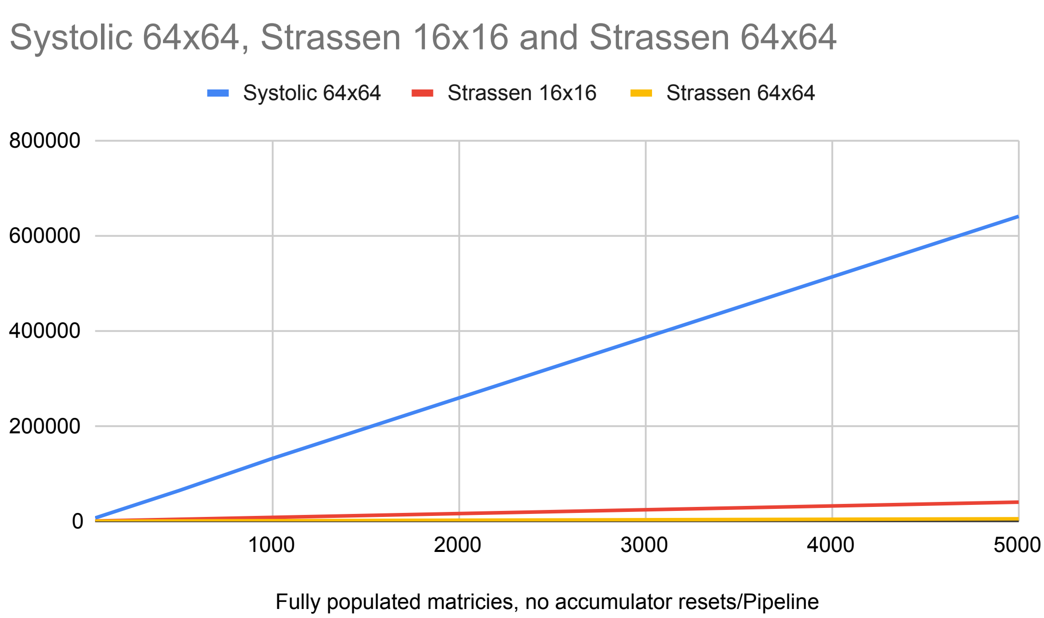

Figure 6.6: Plot of Performance Data for 64×64

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 Phyiscal 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

	Design Compiler		IC Compiler	
Size	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

	Design Compiler		IC Compiler	
Size	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.

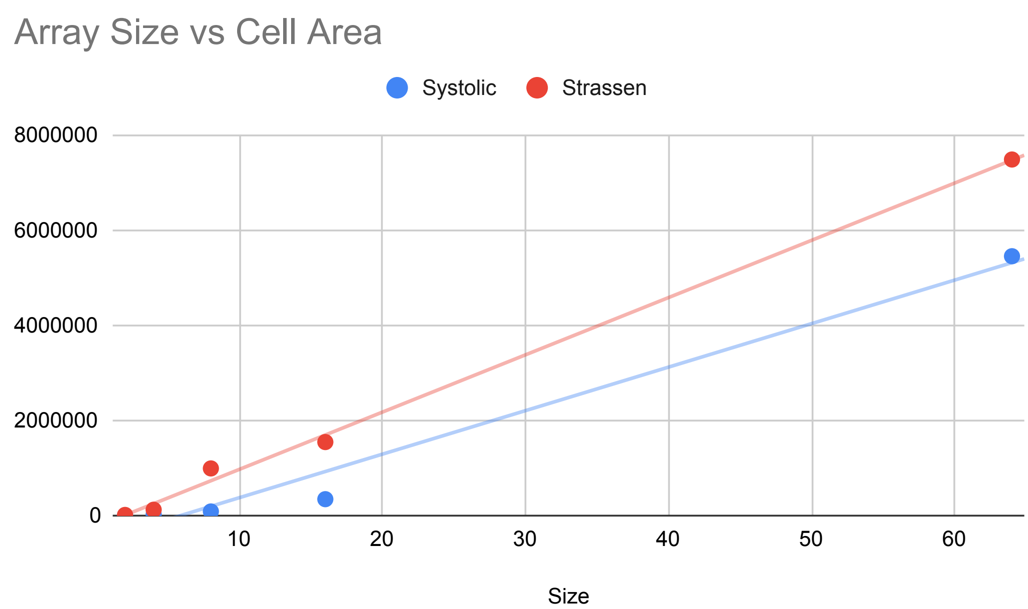

Figure 6.7: Plot of Size vs Cell Area for Pre-scan

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

	Systolic		Strassen	
Size	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.

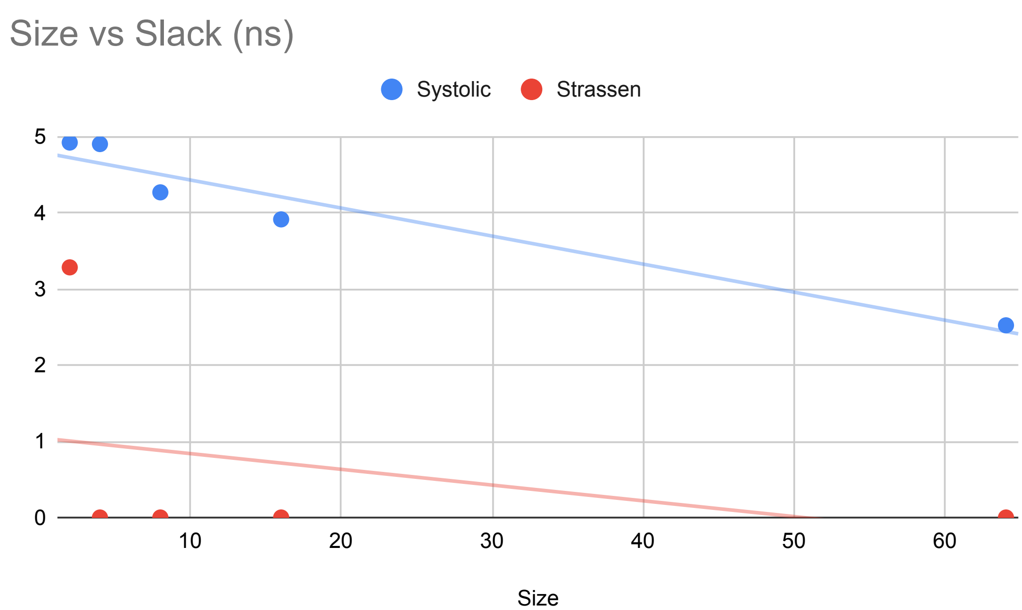

Figure 6.8: Plot of Size vs Cell Area for Pre-scan

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.

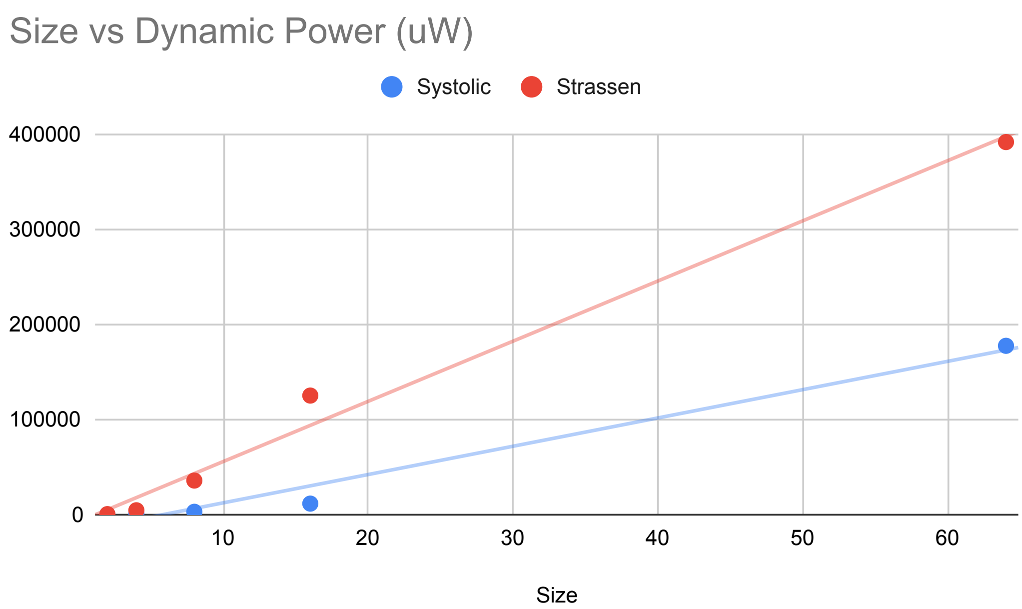

Figure 6.9: Plot of Size vs Dynamic Power

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

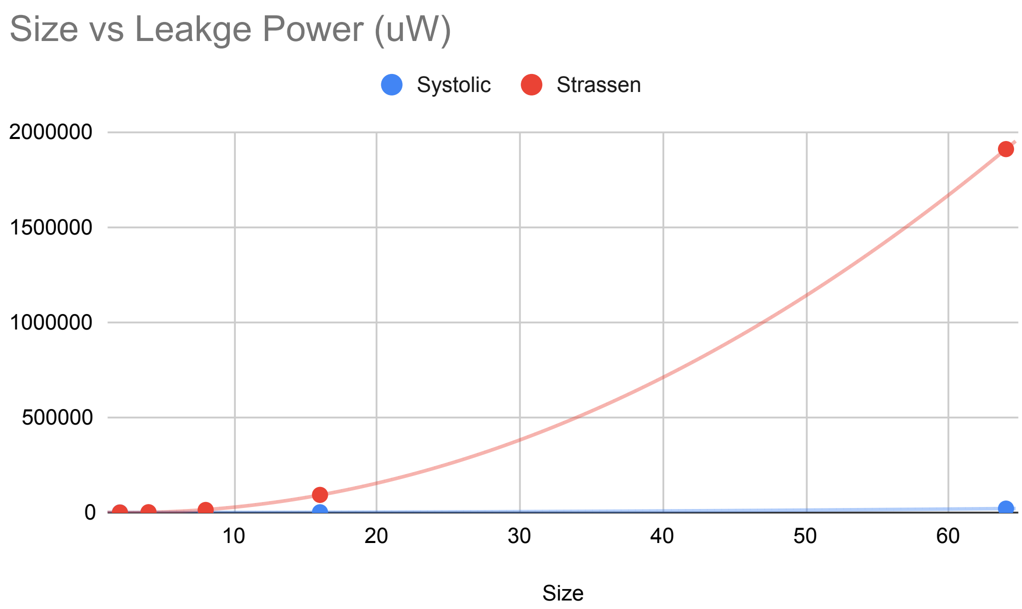

Figure 6.10: Plot of Size vs Leakage Power

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

	Systolic		Strassen	
Size	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.

‘

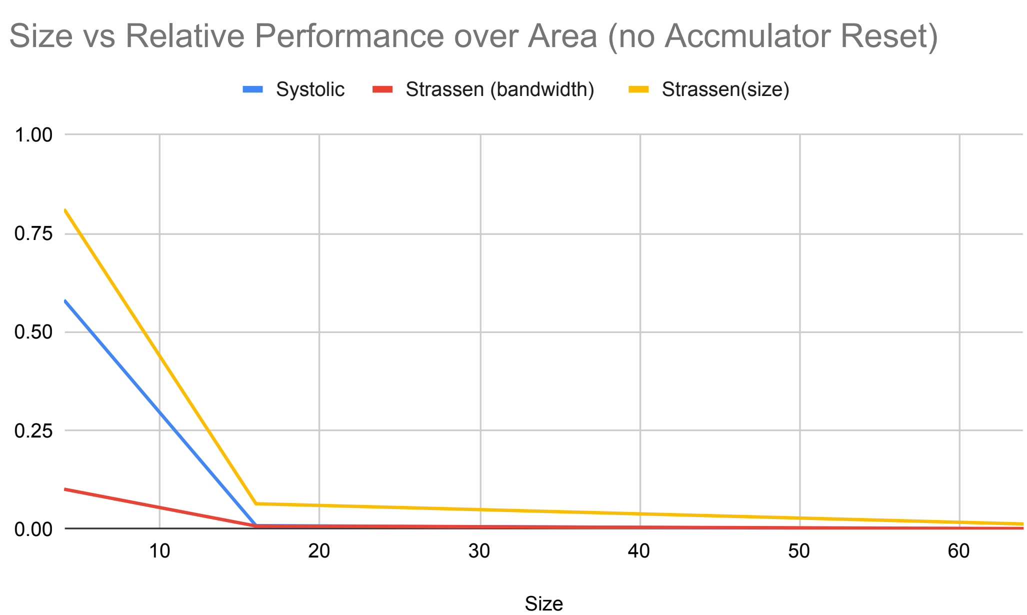

Figure 6.11: Plot of Size vs Performance over Area

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 that 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 and 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

```
module mac(
    clk,
    reset,
    a_in,
    b_in,
    mult_en,
    acc_en,
    load_en,
    acc_out,
    a_out,
    b_out
    );

```

```
parameter DATAWIDTH = 8;
  parameter ACCWIDTH = 32;

  input [DATAWIDTH-1:0] a_in, b_in;
  input clk, mult_en, acc_en, load_en, reset;
  output reg signed [ACCWIDTH-1:0] acc_out;
  output reg signed [DATAWIDTH-1:0] a_out, b_out;

  reg signed [(DATAWIDTH*2)-1:0] mult;

  always @ (posedge clk)
  begin
      if(reset)
      begin
          a_out <= 0;
          b_out <= 0;
          acc_out <= 0;
          mult <= 0;
      end
      else
      begin
          a_out <= load_en ? a_in : a_out;
          b_out <= load_en ? b_in : b_out;
          mult <= mult_en ? a_out*b_out : mult;
          acc_out <= acc_en ? mult + acc_out : acc_out;
```

```
            end
        end

endmodule
```

I.2 Systolic Array RTL Generation Script

```
use Getopt::Long;

#getting the user configuration
$command = GetOptions ( "help" => \$help,
            "param=s" => \$param,
            "width=i" => \$width,
            "dimension=i" => \$dimension,
            "outfile=s" => \$outfile );

#check for help
if($help > 0){
    Help();
    die;
}

# making sure either params file or other configs is specified
if((not defined $param) && ((not defined $width) or (not
    defined $dimension) or (not defined $outfile))) {
    ReturnError("Missing params");
}
elsif((defined $param) && ((defined $width) or (defined
    $dimension) or (defined $outfile))) {
```

```
    ReturnError("Command parameters shall either be a param file OR other input configs");
}

# parse the file into the variables or just parse the outfile
if( $param ne "" ) {
    ParseParams();
}

# check if outfile specified correctly
if($outfile =~ m/^(.+).v/) { $outfile = $1; }
else {
    ReturnError("Outfile not speicifed properly <outfile>.v");
}

$datawidth = $width - 1;

Debugger();

GenerateRTL();

# print error and die
sub ReturnError {
    my ($ret) = @_;
    print "\n==Error==\n$ret\n==Error==\n\n";
```

```
    Help();
    die;
}

# subroutine for debugging perl
sub Debugger {
    print "Generating files with: width=$width, dimension=$dimension, outfile=$outfile\n";
}

# parse param file
sub ParseParams
{
    open($fh, "<", $param) or die("==Error== File $param not found");
    while(my $line = <$fh>) {
        if($line =~ m/\s*width\s*=\s*(\d+)\s*;\s*/) { $width = $1 }
        elsif($line =~ m/\s*stages\s*=\s*(\d+)\s*;\s*/) { $dimension = $1; }
        elsif($line =~ m/\s*outfile\s*=\s*(.+)\s*;\s*/) { $outfile = $1; }
    }
    close $fh;
}
```

```
sub GenerateRTL {
    # write the first part of the module
    $dimensionminus1 = $dimension - 1;

    open $fh, ">", $outfile.".v";
    $message = <<"EOF";
//===========================================================
//        RTL Generated File
//        Author: Ashish Tondwalkar
//---------------------------------------------------------
//        Description: Systolic Array of MAC Units
//---------------------------------------------------------
//        Datawidth = $width
//        Dimension = $dimension
//===========================================================

module $outfile (
    clk,
    reset,
    scan_in0,
    scan_en,
    test_mode,
    scan_out0,
    a_in,
```

```
    b_in,
    mult_en,
    acc_en,
    load_en,
    d_out
    );

    parameter DATAWIDTH = $width;
    parameter MATRIXSIZE = $dimension;
    parameter ACCWIDTH = 32;

    input clk;
    input reset;

    input
        scan_in0,                   // test scan mode data input
        scan_en,                    // test scan mode enable
        test_mode;                  // test mode select

    output
        scan_out0;                  // test scan mode data output

    input mult_en;
```

```
    input acc_en;
    input load_en;
    input [DATAWIDTH*MATRIXSIZE-1:0] a_in;
    input [DATAWIDTH*MATRIXSIZE-1:0] b_in;
    output [ACCWIDTH*MATRIXSIZE*MATRIXSIZE-1:0] d_out;

    wire [DATAWIDTH-1:0] a [MATRIXSIZE:0][MATRIXSIZE:0]; // internal array row connections
    wire [DATAWIDTH-1:0] b [MATRIXSIZE:0][MATRIXSIZE:0]; // internal array col connections

EOF
    print $fh $message;

    @a = (0..$dimensionminus1);

    # generate the stage register declarations
    print $fh "\n    //row bounday\n";
    foreach my $k (@a) {
        $calc = $k*$width;
        print $fh "    assign a[$k][0] = a_in[$calc +: DATAWIDTH]\;\n";
    }
    print $fh "\n    //col bounday\n";
    foreach my $k (@a) {
```

```
        $calc = $k*$width;
        print $fh "     assign b[0][$k] = b_in[$calc +: DATAWIDTH]\;\n";
    }

    my $nextrow;
    my $nextcol;

    for my $row (@a) {
        for my $col (@a) {
            $nextrow = $row + 1;
            $nextcol = $col + 1;
            $calc = ($row*$dimension + $col)*32;
            $message = <<"EOF";

    mac element_$row\_$col (
        .clk(clk),
        .reset(reset),
        .a_in(a[$row][$col]),
        .b_in(b[$row][$col]),
        .mult_en(mult_en),
        .acc_en(acc_en),
        .load_en(load_en),
        .a_out(a[$row][$nextcol]),
        .b_out(b[$nextrow][$col]),
```

```
            .acc_out(d_out[$calc +: ACCWIDTH])
            );
EOF
                print $fh $message;

            }
      }

      print $fh "endmodule\n";

}


#subroutine for helping user
sub Help {
      my $help = <<"EOF";
--param=param config file;

--width=datawidth
      width must be within range [1, 64]

--dimension=dimensions of the systolic array
      stages must be within range [2, 128]

--outfile=name of generated rtl and tb files
```

```

If a params file is specified, no other configuration inputs should be specified.

EOF
    print $help
}


=pod

=head1 NAME

B<ast4798.pl> - generate multistage pipelined shift register

=head1 SYNOPSIS

perl ast4798.pl [ARGUMENT(s)] ...

=head1 DESCRIPTION

Generates RTL and TB files in verilog for a multistage pipelilned shift register in verilog, given configuration parameters. TB tests reset and active function of RTL

```

```
ARGUMENTS(s) specified in the command line...

B<--help> display this help and exit

B<--param> specify a text file that defines the configuration
    arguments. If a file is specified, no other configuration
    arguments should be specified.

B<--width> configuration argument that defines the datawidth

B<--dimension> configuration argument that defines the array
    dimensions of the systolic array

B<--reset> configuration argument that defines the reset value
     for each stage register. Can be in decimal or hex.

B<--outfile> configuration argument that describes the
    nomenclature of the generate files. Must be specified with
    <module_name>.v

=head1 FILES

B<<module_name>.v> generated RTL

B<<module_name>_test.v> generated TB
```

```

=head1 EXAMPLES

 perl ast4798.pl -param my_params.txt

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

=head1 AUTHOR

 Ashish Tondwalkar

=cut
```

I.3 Shift Register

```
module shift_reg(
  clk,
  reset,
  shift,
  din,
  dout
  );

  parameter SIZE = 4;
  parameter DATAWITDTH = 8;

  input clk;
  input reset;
  input shift;
  input [DATAWITDTH-1:0] din;
  output wire [DATAWITDTH-1:0] dout;

  reg [DATAWITDTH-1:0] mem [SIZE];

  integer q;

  always @ (posedge clk)
  begin
```

```
      if(reset)
      begin
        for(q = 0; q < SIZE; q = q + 1)
          mem[q] <= 'b0;
      end
      else
      begin
        mem[0] <= (shift) ? din : mem[0];
        for(q = 1; q < SIZE; q = q + 1)
          mem[q] <= (shift) ? mem[q-1] : mem[q];
      end
    end

  assign dout = mem[SIZE-1];

endmodule
```

I.4 Systolic Array Alignment Module RTL Generation Script

```
use Getopt::Long;

#getting the user configuration
$command = GetOptions ( "help" => \$help,
                  "width=i" => \$width,
                  "size=i" => \$size );

#check for help
if($help > 0){
      Help();
      die;
}

# check for valid width
if($width < 2 ) {
      ReturnError("Width out of bounds");
}

# check for valid stage count
if($size < 2) {
      ReturnError("Stages out of bounds");
}

```

```
GenerateRTL();

# print error and die
sub ReturnError {
    my ($ret) = @_;
    print "\n==Error==\n$ret\n==Error==\n\n";
    Help();
    die;
}


sub GenerateRTL {

    $dinw = $width*$size;

    # write the first part of the module
    open $fh, ">", "data_alignment.v";
    $message = <<"EOF";
//=============================================================
//        RTL Generated File
//        Author: Ashish Tondwalkar
//-----------------------------------------------------
//        Description: Data aligner for systolic array
//-----------------------------------------------------
//        Datawidth = $width
```

```
//          Matrix Size = $size
//===========================================================

module data_alignment(
       clk,
       reset,
       shift,
       a_in,
       b_in,
       a_out,
       b_out
       );

       input clk;
       input reset;
       input shift;
       input [$dinw-1:0] a_in;
       input [$dinw-1:0] b_in;
       output reg [$dinw-1:0] a_out;
       output reg [$dinw-1:0] b_out;

       assign a_out[0 +: $width] = a_in[0 +: $width];
       assign b_out[0 +: $width] = b_in[0 +: $width];

EOF
```

```
    print $fh $message;

    # array for generation loops
    @a = (1..$size-1);

    # generate the stage register declarations
    foreach my $i (@a)
    {

    $idx = $i*$width;

        $message = <<"EOF";
    shift_reg #(.DATAWIDTH(8), .SIZE($i)) del_a_$i (.clk(clk), .reset(reset), .shift(shift), .din(a_in[$idx +: $width]), .dout(a_out[$idx +: $width]));
    shift_reg #(.DATAWIDTH(8), .SIZE($i)) del_b_$i (.clk(clk), .reset(reset), .shift(shift), .din(b_in[$idx +: $width]), .dout(b_out[$idx +: $width]));

EOF

     print $fh $message;

    }
    $message = <<"EOF";
```

```

endmodule

EOF
    print $fh $message;

    close $fh;

}

#subroutine for helping user
sub Help {
    my $help = <<"EOF";
--param=param config file;

--width=word size/data width
    width must be within range [1, 64]

--stages=number of stages in pipeline
    stages must be within range [2, 128]

--reset=reset value for registers in binary or hex

--outfile=name of generated rtl and tb files

```

```
If a params file is specified, no other configuration inputs should be specified.

EOF
    print $help
}


=pod

=head1 NAME

B<ast4798.pl> - generate data alignment rtl

=head1 SYNOPSIS

perl data_alignment.pl [ARGUMENT(s)] ...

=head1 DESCRIPTION

Generates RTL for a data aligner required for systolic execution

ARGUMENTS(s) specified in the command line...

```

```
B<--help> display this help and exit

B<--width> configuration argument that defines the datawidth

B<--sizes> configuration argument that defines the systolic
    array size

=head1 FILES

B<data_alignment.v> generated RTL

=head1 EXAMPLES

 perl data_alignment.pl --width 8 --size 4

=head1 AUTHOR

 Ashish Tondwalkar

=cut
```

I.5 Strassen Layer RTL Generation Script

```
use Getopt::Long;

#getting the user configuration
$command = GetOptions ( "help" => \$help,
                    #"param=s" => \$param,
                    "size=i" => \$size,
                    "width=i" => \$width
                    #"reset=s" => \$reset,
                    #"outfile=s" => \$outfile
                     );

#check for help
if($help > 0){
       Help();
       die;
}

# making sure either params file or other configs is specified
if(not defined $size) {
       ReturnError("Missing parameter");
}

# check for valid range
```

```
if($size < 2) {
    ReturnError("Size has to be at least 2");
}
GenerateFiles();
# print error and die
sub ReturnError {
    my ($ret) = @_;
    print "\n==Error==\n$ret\n==Error==\n\n";
    Help();
    die;
}
sub GenerateFiles {
    # write the first part of the module
    open $fh, ">", "strassen_dim$size.v";
    $half = $size/2;
    $mab = $width * $size * $size;
    $aab = 32 * $size * $size;
    $qwidth = $width + 2;
    $qm = $qwidth * $half * $half;
    $mm = 32 * $half * $half;
```

```

    $message = <<"EOF";

module strassen_dim$size (
    reset,
    clk,
    scan_in0,
    scan_en,
    test_mode,
    scan_out0,
    enable,
    matrix_a,
    matrix_b,
    matrix_c
);

    input
        reset,                      // system reset
        clk;                        // system clock

    input
        scan_in0,                   // test scan mode data input
        scan_en,                    // test scan mode enable
        test_mode;                  // test mode select
```

```

    output
        scan_out0;                  // test scan mode data output

    parameter DATAWIDTH = $width; //data bus width
    parameter QWIDTH = DATAWIDTH + 2;
    parameter ACCWIDTH = 32;

    localparam DIM = $size;
    localparam SIZE = DIM * DIM;

    input enable;
    input [$mab-1:0] matrix_a;
    input [$mab-1:0] matrix_b;
    output reg [$aab-1:0] matrix_c;

wire signed [32-1:0] matrix_c_nxt [1:0][1:0][$half-1:0][$half-1:0];

reg signed [$qwidth-1:0] q_int [13:0][$half-1:0][$half-1:0];
wire signed [$qwidth-1:0] q_int_nxt [13:0][$half-1:0][$half-1:0];

reg [$mm-1:0] matrix_m [6:0];
```

```

wire signed [$width-1:0] a_arr [$size-1:0][$size-1:0];
wire signed [$width-1:0] b_arr [$size-1:0][$size-1:0];


wire [$qm-1:0] q_int_flat [13:0];

wire signed [32-1:0] matrix_m_unflat [6:0][$half-1:0][$half-1:0];

wire [$aab-1:0] matrix_c_nxt_flat;


EOF
    print $fh $message;


    @a = (0..$size-1);
    @b = (0..$size-1);
    foreach my $i (@a) {
        foreach my $j (@b) {
            $calc = $i*$size*$width + $j*$width;
            $message = <<"EOF";
assign a_arr[$i][$j] = matrix_a[$calc +: $width];
assign b_arr[$i][$j] = matrix_b[$calc +: $width];
```

```
EOF
        print $fh $message;

        }
    }


    $message = <<"EOF";

integer k, l, m;
always \@ (posedge clk)
begin
    if(reset)
    begin
        //matrix_c <= 0;
        for(k = 0; k < 14; k = k + 1)
            for(l = 0; l < $half; l = l + 1)
                for(m = 0; m < $half; m = m + 1)
                    q_int[k][l][m] <= 0;

        matrix_c <= 0;
    end
    else
    begin
        for(k = 0; k < 14; k = k + 1)
```

```
                for(l = 0; l < $half; l = l + 1)
                    for(m = 0; m < $half; m = m + 1)
                        q_int[k][l][m] <= enable ? q_int_nxt[k][l][m] : q_int[k][l][m];

            matrix_c <= enable ? matrix_c_nxt_flat : matrix_c;
        end
    end


EOF
    print $fh $message;

    @a = (0..($size/2)-1);
    @b = (0..($size/2)-1);

    #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(0), .ya(0), .xb(1), .yb(1), .q(q_int_nxt[0]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 0*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 0*($size/2) + $i;
            $yb = 0*($size/2) + $j;
            $message = <<"EOF";
```

```
    assign q_int_nxt[0][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb][$yb];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(0), .ya(0), .xb(1), .yb(1), .q(q_int_nxt[1]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 0*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 1*($size/2) + $i;
            $yb = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[1][$i][$j] = b_arr[$xa][$ya] + b_arr[$xb][$yb];
EOF
        print $fh $message;

        }
    }

```

```
    #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(1), .ya(0),
        .xb(1), .yb(1), .q(q_int_nxt[2]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 1*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 1*($size/2) + $i;
            $yb = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[2][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
EOF
        print $fh $message;

        }
    }

    #direct_assign_m(.a(b_arr), .xa(0), .ya(0), .q(q_int_nxt
        [3]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $message = <<"EOF";
    assign q_int_nxt[3][$i][$j] = b_arr[$i][$j];
EOF
        print $fh $message;
```

```

        }
    }

    #direct_assign_m(.a(a_arr), .xa(0), .ya(0), .q(q_int_nxt[4]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $message = <<"EOF";
    assign q_int_nxt[4][$i][$j] = a_arr[$i][$j];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(b_arr), .b(b_arr), .op(1), .xa(0), .ya(1), .xb(1), .yb(1), .q(q_int_nxt[5]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 0*($size/2) + $i;
            $ya = 1*($size/2) + $j;
            $xb = 1*($size/2) + $i;
            $yb = 1*($size/2) + $j;
            $message = <<"EOF";
```

```
    assign q_int_nxt[5][$i][$j] = b_arr[$xa][$ya] - b_arr[$xb][$yb];
EOF
        print $fh $message;

        }
    }

    #direct_assign_m(.a(a_arr), .xa(1), .ya(1), .q(q_int_nxt[6]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $x = 1*($size/2) + $i;
            $y = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[6][$i][$j] = a_arr[$x][$y];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(b_arr), .b(b_arr), .op(1), .xa(1), .ya(0), .xb(0), .yb(0), .q(q_int_nxt[7]));
    foreach my $i (@a) {
```

```
        foreach my $j (@b) {
            $xa = 1*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 0*($size/2) + $i;
            $yb = 0*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[7][$i][$j] = b_arr[$xa][$ya] - b_arr[$xb][$yb];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(a_arr), .b(a_arr), .op(0), .xa(0), .ya(0), .xb(0), .yb(1), .q(q_int_nxt[8]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 0*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 0*($size/2) + $i;
            $yb = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[8][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb][$yb];
```

```
EOF
        print $fh $message;

        }
    }

    #direct_assign_m(.a(b_arr), .xa(1), .ya(1), .q(q_int_nxt
        [9]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $x = 1*($size/2) + $i;
            $y = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[9][$i][$j] = b_arr[$x][$y];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(a_arr), .b(a_arr), .op(1), .xa(1), .ya(0),
        .xb(0), .yb(0), .q(q_int_nxt[10]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 1*($size/2) + $i;
```

```
            $ya = 0*($size/2) + $j;
            $xb = 0*($size/2) + $i;
            $yb = 0*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[10][$i][$j] = a_arr[$xa][$ya] - a_arr[$xb][$yb];
EOF
        print $fh $message;

        }
    }

    #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(0), .ya(0), .xb(0), .yb(1), .q(q_int_nxt[11]));
    foreach my $i (@a) {
        foreach my $j (@b) {
            $xa = 0*($size/2) + $i;
            $ya = 0*($size/2) + $j;
            $xb = 0*($size/2) + $i;
            $yb = 1*($size/2) + $j;
            $message = <<"EOF";
    assign q_int_nxt[11][$i][$j] = b_arr[$xa][$ya] + b_arr[$xb][$yb];
EOF
        print $fh $message;
```

```
            }
        }

        #add_sub_m (.a(a_arr), .b(a_arr), .op(1), .xa(0), .ya(1),
            .xb(1), .yb(1), .q(q_int_nxt[12]));
        foreach my $i (@a) {
            foreach my $j (@b) {
                $xa = 0*($size/2) + $i;
                $ya = 1*($size/2) + $j;
                $xb = 1*($size/2) + $i;
                $yb = 1*($size/2) + $j;
                $message = <<"EOF";
        assign q_int_nxt[12][$i][$j] = a_arr[$xa][$ya] - a_arr[$xb
            ][$yb];
EOF
            print $fh $message;

            }
        }

        #add_sub_m (.a(b_arr), .b(b_arr), .op(0), .xa(1), .ya(0),
            .xb(1), .yb(1), .q(q_int_nxt[13]));
        foreach my $i (@a) {
            foreach my $j (@b) {
```

```
                $xa = 1*($size/2) + $i;
                $ya = 0*($size/2) + $j;
                $xb = 1*($size/2) + $i;
                $yb = 1*($size/2) + $j;
                $message = <<"EOF";
    assign q_int_nxt[13][$i][$j] = a_arr[$xa][$ya] + a_arr[$xb
        ][$yb];
EOF
            print $fh $message;

            }
    }


    @a = (0..6);
    @b = (0..($size/2)-1);
    @u = (0..($size/2)-1);

    foreach my $i (@a) {
        foreach my $j (@b) {
            foreach my $k (@u) {
            $calc = $j*($size/2)*32 + $k*32;
            $message = <<"EOF";
assign matrix_m_unflat[$i][$j][$k] = matrix_m[$i][$calc +:
    32];
```

```
EOF
        print $fh $message;

            }
        }
    }


    @th = (0..13);



    foreach my $t (@th) {


        print $fh "assign q_int_flat[$t] = {\n";

        $modsize = ($size/2)-1;

        @r = (0..$modsize);
        @c = (0..$modsize);

        @r = reverse @r;
        @c = reverse @c;

```

```
        for my $i (@r) {
            for my $j (@c) {
                if($i == 0 && $j == 0) {
                    $message = <<"EOF";
                                        q_int[$t][$i][$j]
EOF
                }
                else {
                    $message = <<"EOF";
                                        q_int[$t][$i][$j],
EOF
                }


                print $fh $message;
            }
        }

        print $fh "                         };\n";
    }




    @a = (0..($size/2)-1);
```

```
    @b = (0..($size/2)-1);
    foreach my $i (@a) {
        foreach my $j (@b) {
            $message = <<"EOF";
assign matrix_c_nxt[0][0][$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][$j];
assign matrix_c_nxt[0][1][$i][$j] = matrix_m_unflat[2][$i][$j]
    + matrix_m_unflat[4][$i][$j];
assign matrix_c_nxt[1][0][$i][$j] = matrix_m_unflat[1][$i][$j]
    + matrix_m_unflat[3][$i][$j];
assign matrix_c_nxt[1][1][$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];
EOF
        print $fh $message;

        }
    }

    @a = (0..1);
    @b = (0..1);
    @c = (0..($size/2)-1);
    @d = (0..($size/2)-1);
    foreach my $i (@a) {
```

```
          foreach my $j (@b) {
               foreach my $u (@c) {
                    foreach my $w (@d) {
                         $calc = $i*$size*($size/2)*32 + ($j*($size
                              /2))*32 + ($u*$size)*32 + $w*32;
                         $message = <<"EOF";
assign matrix_c_nxt_flat[$calc +: 32] = matrix_c_nxt[$i][$j][
    $u][$w];
EOF
          print $fh $message;
                    }
               }
          }
     }

     $message = <<"EOF";


strassen_dim$half #(.DATAWIDTH(QWIDTH)) M1 (
     .clk(clk),
     .reset(reset),
     .enable(enable),
     .matrix_a(q_int_flat[0]),
     .matrix_b(q_int_flat[1]),
     .matrix_c(matrix_m[0])
```

```
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M2 (
        .clk(clk),
        .reset(reset),
        .enable(enable),
        .matrix_a(q_int_flat[2]),
        .matrix_b(q_int_flat[3]),
        .matrix_c(matrix_m[1])
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M3 (
        .clk(clk),
        .reset(reset),
        .enable(enable),
        .matrix_a(q_int_flat[4]),
        .matrix_b(q_int_flat[5]),
        .matrix_c(matrix_m[2])
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M4 (
        .clk(clk),
        .reset(reset),
        .enable(enable),
        .matrix_a(q_int_flat[6]),
```

```
    .matrix_b(q_int_flat[7]),
    .matrix_c(matrix_m[3])
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M5 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .matrix_a(q_int_flat[8]),
    .matrix_b(q_int_flat[9]),
    .matrix_c(matrix_m[4])
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M6 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .matrix_a(q_int_flat[10]),
    .matrix_b(q_int_flat[11]),
    .matrix_c(matrix_m[5])
);

strassen_dim$half #(.DATAWIDTH(QWIDTH)) M7 (
    .clk(clk),
    .reset(reset),
```

```
        .enable(enable),
        .matrix_a(q_int_flat[12]),
        .matrix_b(q_int_flat[13]),
        .matrix_c(matrix_m[6])
);

endmodule // strassen_dim$size
EOF

        print $fh $message;

        close $fh;

}

#subroutine for helping user
sub Help {
        my $help = <<"EOF";
--size=multiplier dimension
EOF
        print $help
}


=pod
```

```

=head1 NAME

B<strassen_layer_gen.pl> - generate Verilog file for a
    strassen hierachy layer

=head1 SYNOPSIS

 perl strassen_layer_gen.pl [ARGUMENT(s)] ...

=head1 DESCRIPTION

ARGUMENTS(s) specified in the command line...

B<--help> display this help and exit

B<--size> the dimensions of the matrix multiplier

=head1 FILES

B<q_int_flat.txt> q_int_flat concat code

=head1 EXAMPLES

 perl strassen_layer_gen.pl --size 64
```

```

=head1 AUTHOR

Ashish Tondwalkar

=cut
```

I.6 Test Config File Generation Script

```
use Getopt::Long;

#getting the user configuration
$command = GetOptions ( "help" => \$help,
                #"param=s" => \$param,
                "runs=i" => \$runs,
                "size=i" => \$size,
                "seed=i" => \$seed,
                "full=i" => \$full,
                "racr=i" => \$racr
                #"reset=s" => \$reset,
                #"outfile=s" => \$outfile
                 );

#check for help
if($help > 0){
    Help();
    die;
}

# making sure either params file or other configs is specified
if(not defined $runs or not defined $size) {
    ReturnError("Missing parameter");
```

```
}

# check for valid range
if($runs < 1) {
    ReturnError("Runs has to be at least 1");
}

if($size < 2) {
    ReturnError("Size has to be at least 2");
}

GenerateFiles();

system("cp test_pipeline.txt test_pipeline_str.txt");
system("cp test_pipeline.txt test_pipeline_strfull.txt");

# print error and die
sub ReturnError {
    my ($ret) = @_;
    print "\n==Error==\n$ret\n==Error==\n\n";
    Help();
    die;
}

sub GenerateFiles {
```

```
    # write the first part of the module
    open $fh, ">", "test_pipeline.txt";
    open $fh2, ">", "test_config.txt";

    $data = 0;

    while($runs > 0) {

    #pipe
        if(not defined $racr) {
            $data = $runs;
            $runs = 0;
        }
        else {
            do {
                $data = 1#int(rand($runs + 1));
            } while($data == 0);

            if($data > $runs) {
                $data = $runs;
                $runs = 0;
            }
            else {
                $runs = $runs - $data;
            }
```

```
        }

        $message = <<"EOF";
$data
EOF
        print $fh $message;

    #config

    if(defined $full) {
        $dima = $size;
        $dimshared = $size;
        $dimb = $size;
    }
    else {
        $randval = $size + 1 - 2;
        $dima = int(rand($randval)) + 2;
        $dimshared = int(rand($randval)) + 2;
        $dimb = int(rand($randval)) + 2;
    }

        $message = <<"EOF";
$dima
$dimshared
$dimb
```

```
EOF
            print $fh2 $message;

      }

      close $fh;
      close $fh2;

}

#subroutine for helping user
sub Help {
      my $help = <<"EOF";
--runs=number of multiplications

--size=multiplier dimension

--seed=random generation seed
EOF
      print $help
}


=pod

```

```
=head1 NAME

B<test_gen.pl> - generate text files for test configurations

=head1 SYNOPSIS

 perl test_gen.pl [ARGUMENT(s)] ...

=head1 DESCRIPTION

 Generates RTL and TB files in verilog for a multistage pipelilned shift register in verilog, given configuration parameters. TB tests reset and active function of RTL

ARGUMENTS(s) specified in the command line...

B<--help> display this help and exit

B<--runs> amount of multiplications

B<--size> the dimensions of the matrix multiplier

B<--seed> random generation seed

=head1 FILES
```

```

B<test_pipeline.txt> multiplications before a reset

B<test_config.txt> matrix multiplication dimensions

=head1 EXAMPLES

 perl test_gen.pl --runs 500 --size 16

 perl test_gen.pl --runs 500 --size 16 --full 1

 perl test_gen.pl --runs 500 --size 16 --full 1 --racr 1

=head1 AUTHOR

 Ashish Tondwalkar

=cut
```

I.7 Strassen Accumulator Top File

```


module multiplier_strassen (
	reset,
	clk,
	scan_in0,
	scan_en,
	test_mode,
	scan_out0,
	enable,
	matrix_a,
	matrix_b,
	matrix_c
);


	input
		reset,                      // system reset
		clk;                        // system clock

	input
		scan_in0,                   // test scan mode data
		  input
```

```
        scan_en,                  // test scan mode enable
        test_mode;                // test mode select

    output
        scan_out0;                // test scan mode data output

    parameter DATAWIDTH = 8; //data bus width
    parameter ACCWIDTH = 32;

    localparam DIM = 4;
    localparam SIZE = DIM * DIM;

    input enable;
    input [(DATAWIDTH*SIZE)-1:0] matrix_a;
    input [(DATAWIDTH*SIZE)-1:0] matrix_b;
    output reg [(ACCWIDTH*SIZE)-1:0] matrix_c;

    wire [(ACCWIDTH*SIZE)-1:0] matrix_c_nxt;
    reg signed [ACCWIDTH-1:0] matrix_c_unflat [DIM-1:0][DIM-1:0];

    wire [(ACCWIDTH*SIZE)-1:0] raw_out;
    wire signed [ACCWIDTH-1:0] raw_arr [DIM-1:0][DIM-1:0];

```

```
    strassen_dim4 #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH(ACCWIDTH)
        ) str (
        .clk(clk),
        .reset(reset),
        .enable(enable),
        .matrix_a(matrix_a),
        .matrix_b(matrix_b),
        .matrix_c(raw_out)
    );


    genvar x, y, w, u;
    generate
    for (x = 0; x < DIM; x = x + 1)
    begin
        for (y = 0; y < DIM; y = y + 1)
        begin
            assign raw_arr[x][y] = raw_out[(x*DIM)*ACCWIDTH +
                y*ACCWIDTH+: ACCWIDTH];
            assign matrix_c_unflat[x][y] = matrix_c[(x*DIM)*
                ACCWIDTH + y*ACCWIDTH +: ACCWIDTH];
            assign matrix_c_nxt[(x*DIM)*ACCWIDTH + y*ACCWIDTH
                +: ACCWIDTH] = raw_arr[x][y] + matrix_c_unflat[
                x][y];
```

```
        end
    end

    integer k, l, m;
    always @ (posedge clk)
    begin
        if(reset)
            matrix_c <= 0;
        else
            matrix_c <= enable ? matrix_c_nxt : matrix_c;
    end


endgenerate



endmodule
```

I.8 Systolic Array Driver

```
typedef virtual input_if input_vif;

import "DPI-C" context function void resetAccumulator();

class driver_systolic extends uvm_driver #(packet_in);

    int index;

    uvm_analysis_port #(packet_in) item_collected_port_sys;

    `uvm_component_utils(driver_systolic)
    input_vif vif;

    packet_in modded;

    int i, j;

    bit letrun;

    int fd;
    int config_fp;
    string config_line;
    int config_data_a, config_data_shared, config_data_b;
```

```
    string line;
    int products;

    event sample_cg;
    event config_cg;

    covergroup input_cov @ (sample_cg);
        option.per_instance = 1;

        matrix_a : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
            [1536 : 1791],
            [1792 : 2047]
            };
        }
        matrix_b : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
```

```
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
            [1536 : 1791],
            [1792 : 2047]
            };
        }
    endgroup

    covergroup config_cov @ (config_cg);
        option.per_instance = 1;

        config_row : coverpoint config_data_a {
            bins range[] = {[2 : MATRIXSIZE]};
        }
        config_shared : coverpoint config_data_shared {
            bins range[] = {[2 : MATRIXSIZE]};
        }
        config_col : coverpoint config_data_b {
            bins range[] = {[2 : MATRIXSIZE]};
        }
    endgroup

```

```
    function new(string name = "driver_systolic",
        uvm_component parent = null);
        super.new(name, parent);
        item_collected_port_sys = new ("
            item_collected_port_sys", this);
        modded = packet_in::type_id::create("modded", this);
        input_cov = new();
        config_cov = new();
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        void'(uvm_resource_db#(input_vif)::read_by_name(.scope
            ("ifs"), .name("input_vif"), .val(vif)));
    endfunction

    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        //phase.raise_objection(this);
        //fork
        //      reset_signals();
        letrun = 0;
        fork
            get_and_drive(phase);
            run_perf_cnt();
```

```
            run_enables();
            //run_validity();
        join

        //join
        //phase.drop_objection(this);
    endtask

    virtual protected task reset_signals();
        //wait (vif.rst === 1);
        vif.rst <= 1;
        //forever begin
        vif.valid <= '0;
        vif.load <= '0;
        vif.mult <= '0;
        vif.acc <= 0;
        vif.matrix_a_data <= 0;
        vif.matrix_b_data <= 0;
        vif.en_perf <= 0;

        resetAccumulator();

        #22;
        @(posedge vif.clk);
        vif.rst <= 0;
```

```
        //end
    endtask

    virtual protected task fetch_fdata();
        $fgets(line, fd);
        products = line.atoi();

        $fgets(config_line, config_fp);
        config_data_a = config_line.atoi();

        $fgets(config_line, config_fp);
        config_data_shared = config_line.atoi();

        $fgets(config_line, config_fp);
        config_data_b = config_line.atoi();

        $display("%t | Systolic: products:%0d dima:%0d
           dimshared:%0d dimb:%0d", $time, products,
           config_data_a, config_data_shared, config_data_b);
    endtask

    virtual protected task get_and_drive(uvm_phase phase);

        fd = $fopen("test_pipeline.txt","r");
        config_fp = $fopen("test_config.txt","r");
```

```

        forever
        begin
            reset_signals();

            fetch_fdata();

            if(products)
            begin
                //-> config_cg;
                repeat(products)
                begin

                    vif.ready <= 1;
                    //vif.valid <= 0;
                    seq_item_port.get(req);

                    for(i = 0; i < MATRIXSIZE; i++)
                    begin
                        for(j = 0; j < MATRIXSIZE; j++)
                        begin
                            if(i < config_data_a && j < config_data_shared)
                                modded.data_a[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH]
```

```
                    = req.data_a[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH];
                else
                    modded.data_a[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH] = 0;

                if(i < config_data_shared && j < config_data_b)
                    modded.data_b[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH] = req.data_b[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH];
                else
                    modded.data_b[(i*MATRIXSIZE + j)*DATAWIDTH +: DATAWIDTH] = 0;
            end
        end
        item_collected_port_sys.write(modded);
        letrun = 1;
        drive_transfer(modded);
        vif.matrix_a_data <= 0;
```

```
                    vif.matrix_b_data <= 0;
                    //vif.valid <= 1;
                    //@(posedge vif.clk) ;
                end

                repeat(config_data_shared+config_data_a+
                    config_data_b -1)
                begin
                    @(posedge vif.clk) ;
                    @(posedge vif.clk) ;
                end


                vif.en_perf = 0;

                vif.valid <= 1;
                @(posedge vif.clk) ;
                vif.valid <= 0;
            end
            else
            begin
                $fclose(fd);
                $fclose(config_fp);
                //@(posedge vif.rst);
```

```
                $display("%t | Finished Systolic Multiplications", $time);
                reset_signals();
                letrun = 0;

                break;
            end
        end

        reset_signals();
    endtask

    virtual protected task drive_transfer(packet_in tr);
        index = 0;

        //repeat(config_data_a+config_data_shared+config_data_b-1)
        repeat(MATRIXSIZE)
        begin
            if(index < MATRIXSIZE)
            begin
                vif.matrix_a_data <= tr.data_a[index*(DATAWIDTH*MATRIXSIZE) +: DATAWIDTH*MATRIXSIZE];
```

```
                vif.matrix_b_data <= tr.data_b[index*(
                    DATAWIDTH*MATRIXSIZE) +: DATAWIDTH*
                    MATRIXSIZE];
            end
            else
            begin
                vif.matrix_a_data <= 0;
                vif.matrix_b_data <= 0;
            end
            index = index + 1;
            @(posedge vif.clk) ;
            @(posedge vif.clk) ;
            //vif.valid <= 0;
            //-> sample_cg;
        end
    endtask

    virtual protected task run_enables();
        @(posedge letrun)
        vif.en_perf = 1;
        forever
        begin
            vif.load <= 1;
            vif.mult <= 0;
            vif.acc <= 1;
```

```
            @(posedge vif.clk) ;
             vif.load <= 0;
             vif.mult <= 1;
             vif.acc <= 0;
            @(posedge vif.clk) ;
        end
    endtask

    virtual protected task run_perf_cnt();
        forever
        begin
            @(posedge vif.clk)
                if(vif.en_perf)
                    vif.perf_cnt = vif.perf_cnt + 1;
        end
    endtask

endclass
```

I.9 Strassen Matched Bandwidth Driver

```
import "DPI-C" context function void resetAccumulatorStrassen
    ();

class driver_strassen extends uvm_driver #(packet_in);

    int index;

    uvm_analysis_port #(packet_in) item_collected_port_str;

    `uvm_component_utils(driver_strassen)
    input_vif vif;

    int i, j;

    int pipe_cnt;

    int fd;
    string line;
    int products;

    int products_modded;

    event sample_cg;
```

```

    covergroup input_cov @ (sample_cg);
        option.per_instance = 1;

        matrix_a : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
            [1536 : 1791],
            [1792 : 2047]
            };
        }
        matrix_b : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
            [1536 : 1791],
```

```
            [1792 : 2047]
            };
        }
    endgroup

    function new(string name = "driver_strassen",
      uvm_component parent = null);
        super.new(name, parent);
        item_collected_port_str = new ("
          item_collected_port_str", this);
        input_cov = new();
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        void '(uvm_resource_db#(input_vif)::read_by_name(.scope
          ("ifs"), .name("input_vif"), .val(vif)));
    endfunction

    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        //phase.raise_objection(this);
        //fork
        //      reset_signals();
        fork
```

```
                get_and_drive(phase);
                run_perf_cnt();
            join

            //join
            //phase.drop_objection(this);
        endtask

        virtual protected task reset_signals();
            //wait (vif.rst === 1);
            vif.str_rst <= 1;
            //forever begin
            //vif.valid <= '0;
            vif.str_matrix_a <= 0;
            vif.str_matrix_b <= 0;
            vif.str_valid <= 0;
            vif.str_en_perf <= 0;

            resetAccumulatorStrassen();

            pipe_cnt <= 0;
            //@(posedge vif.clk) ;
            //vif.str_valid <= 0;
            #22;
            @(posedge vif.clk);
```

```
        vif.str_rst <= 0;
        //end
    endtask

    virtual protected task fetch_fdata();
        $fgets(line, fd);
        products = line.atoi();

        $display("%t | Strassen: products:%0d", $time,
           products);
    endtask

    virtual protected task get_and_drive(uvm_phase phase);

        fd = $fopen("test_pipeline_str.txt","r");
        //config_fp = $fopen("../test_config.txt","r");


        reset_signals();
        forever
        begin
            //vif.str_valid <= 0;
            fetch_fdata();

            //products = 100;
```

```
        //vif.str_en_perf = 1;


        if(products)
        begin
            repeat(4)
            begin
                vif.str_en_perf = 1;
                products_modded = products * 2;
                fork
                    begin
                        repeat(products_modded)
                        begin
                            seq_item_port.get(req);

                            item_collected_port_str.write(
                                req);
                            drive_transfer(req);

                            @(posedge vif.clk) ;
                        end
                    end

                    forever
```

```
                begin
                    //$display("%t | Strassen Pipe Counter!", $time);

                    //$display("%t | Strassen Pipe Counter: %d", $time, pipe_cnt);

                    pipe_cnt++;

                    if(pipe_cnt == 7)
                        vif.str_valid <= 1;

                    if(pipe_cnt == products_modded + 7)
                        break;

                    @(posedge vif.clk) ;
                end
            join

            reset_signals();
        end
    end
    else
    begin
```

```
                $fclose(fd);
                //@(posedge vif.rst);
                $display("%t | Finished Strassen
                   Multiplications", $time);
                break;
            end

            reset_signals();
        end

        reset_signals();
    endtask

    virtual protected task drive_transfer(packet_in tr);

        //vif.str_en_perf = 1;

        vif.str_matrix_a <= tr.str_data_a;
        vif.str_matrix_b <= tr.str_data_b;
        vif.str_en <= 1;

        //$display("\nStrassen input data\n%X\n%X", tr.
           str_data_a, tr.str_data_b);

        //vif.str_en_perf = 0;
```

```
    endtask

    virtual protected task run_perf_cnt();
    forever
    begin
    @(posedge vif.clk)
        if(vif.str_en_perf)
            vif.str_perf_cnt = vif.str_perf_cnt + 1;
    end
    endtask

endclass
```

I.10 Strassen Matched Size Driver

```
import "DPI-C" context function void
    resetAccumulatorStrassenFull();

class driver_strassen_full extends uvm_driver #(packet_in);

    int index;

    uvm_analysis_port #(packet_in) item_collected_port_strfull
        ;

    `uvm_component_utils(driver_strassen_full)
    input_vif vif;

    int i, j;

    int pipe_cnt;

    int fd;
    string line;
    int products;

    int products_modded;

```

```
    event sample_cg;

    covergroup input_cov @ (sample_cg);
        option.per_instance = 1;

        matrix_a : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
            [1536 : 1791],
            [1792 : 2047]
            };
        }
        matrix_b : coverpoint vif.matrix_a_data {
            bins range[] = {
            [0 : 255],
            [256 : 511],
            [512 : 767],
            [768 : 1023],
            [1024 : 1279],
            [1280 : 1535],
```

```
                [1536 : 1791],
                [1792 : 2047]
                };
            }
        endgroup

        function new(string name = "driver_strassen_full",
            uvm_component parent = null);
            super.new(name, parent);
            item_collected_port_strfull = new ("
                item_collected_port_strfull", this);
            input_cov = new();
        endfunction

        virtual function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            void'(uvm_resource_db#(input_vif)::read_by_name(.scope
                ("ifs"), .name("input_vif"), .val(vif)));
        endfunction

        virtual task run_phase(uvm_phase phase);
            super.run_phase(phase);

            fork
                get_and_drive(phase);
```

```
            run_perf_cnt();
        join

    endtask

    virtual protected task reset_signals();
        vif.strfull_rst <= 1;
        vif.strfull_matrix_a <= 0;
        vif.strfull_matrix_b <= 0;
        vif.strfull_valid <= 0;
        vif.strfull_en_perf <= 0;

        resetAccumulatorStrassenFull();

        pipe_cnt <= 0;
        #22;
        @(posedge vif.clk);
        vif.strfull_rst <= 0;
    endtask

    virtual protected task fetch_fdata();
        $fgets(line, fd);
        products = line.atoi();

```

```
        $display("%t | Strassen_Full: products:%0d", $time,
            products);
    endtask

    virtual protected task get_and_drive(uvm_phase phase);

        fd = $fopen("test_pipeline_strfull.txt","r");

        reset_signals();
        forever
        begin
            fetch_fdata();

            if(products)
            begin
                vif.strfull_en_perf = 1;
                fork
                    begin
                        repeat(products)
                        begin
                            seq_item_port.get(req);

                            item_collected_port_strfull.write(
                                req);
                            drive_transfer(req);
```

```

                    @(posedge vif.clk) ;
                end
            end

            forever
            begin
                //$display("%t | Strassen Pipe Counter
                    !", $time);

                //$display("%t | Strassen Pipe Counter
                    : %d", $time, pipe_cnt);

                pipe_cnt++;

                if(pipe_cnt == 11)
                    vif.strfull_valid <= 1;

                if(pipe_cnt == products + 11)
                    break;

                @(posedge vif.clk) ;
            end
        join
    end
```

```
            else
            begin
                $fclose(fd);
                //@(posedge vif.rst);
                $display("%t | Finished Strassen_Full
                    Multiplications", $time);
                break;
            end

            reset_signals();
        end

        reset_signals();
    endtask

    virtual protected task drive_transfer(packet_in tr);

        //vif.str_en_perf = 1;

        vif.strfull_matrix_a <= tr.strfull_data_a;
        vif.strfull_matrix_b <= tr.strfull_data_b;
        vif.strfull_en <= 1;

        //$display("\nStrassen input data\n%X\n%X", tr.
            str_data_a, tr.str_data_b);
```

```

            //vif.str_en_perf = 0;
        endtask

        virtual protected task run_perf_cnt();
        forever
        begin
        @(posedge vif.clk)
            if(vif.strfull_en_perf)
                vif.strfull_perf_cnt = vif.strfull_perf_cnt + 1;
        end
        endtask

endclass
```

I.11 Sequencer

```
class sequencer extends uvm_sequencer #(packet_in);
    `uvm_component_utils(sequencer)

    function new (string name = "sequencer", uvm_component
       parent = null);
        super.new(name, parent);
    endfunction
endclass: sequencer
```

I.12 Sequence

```
class sequence_in extends uvm_sequence #(packet_in);
    `uvm_object_utils(sequence_in)

    function new(string name="sequence_in");
        super.new(name);
    endfunction: new

    task body;
        packet_in tx;

        forever begin
            tx = packet_in::type_id::create("tx");
            start_item(tx);
            assert(tx.randomize());
            finish_item(tx);
        end
    endtask: body
endclass: sequence_in
```

I.13 Agent

```
class agent extends uvm_agent;
    sequencer sqr;
    driver_systolic drvsys;
    driver_strassen drvstr;
    driver_strassen_full drvstrfull;

    uvm_analysis_port #(packet_in) item_collected_port_sys;
    uvm_analysis_port #(packet_in) item_collected_port_str;
    uvm_analysis_port #(packet_in) item_collected_port_strfull;

    `uvm_component_utils(agent)

    function new(string name = "agent", uvm_component parent = null);
        super.new(name, parent);
        item_collected_port_sys = new("item_collected_port_sys", this);
        item_collected_port_str = new("item_collected_port_str", this);
        item_collected_port_strfull = new("item_collected_port_strfull", this);
    endfunction
```

```

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        sqr = sequencer::type_id::create("sqr", this);
        drvsys = driver_systolic::type_id::create("drvsys", this);
        drvstr = driver_strassen::type_id::create("drvstr", this);
        drvstrfull = driver_strassen_full::type_id::create("drvstrfull", this);
    endfunction

    virtual function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        drvsys.seq_item_port.connect(sqr.seq_item_export);
        drvsys.item_collected_port_sys.connect(item_collected_port_sys);

        drvstr.seq_item_port.connect(sqr.seq_item_export);
        drvstr.item_collected_port_str.connect(item_collected_port_str);

        drvstrfull.seq_item_port.connect(sqr.seq_item_export);
        drvstrfull.item_collected_port_strfull.connect(item_collected_port_strfull);
```

```
    endfunction
endclass: agent
```

I.14 Input Interface

```
interface input_if (input clk);

    import array_pkg::*;
    arraydim_t matrix_a_data;
    arraydim_t matrix_b_data;

    logic rst;

    logic start;

    logic load;
    logic mult;
    logic acc;
    logic valid;
    logic ready;

    logic str_rst;
    logic str_en;
    logic str_valid;
    strindim4_t str_matrix_a;
    strindim4_t str_matrix_b;

```

```

    logic strfull_rst;
    logic strfull_en;
    logic strfull_valid;
    matrix_t strfull_matrix_a;
    matrix_t strfull_matrix_b;


    logic en_perf;
    int perf_cnt;

    logic str_en_perf;
    int str_perf_cnt;

    logic strfull_en_perf;
    int strfull_perf_cnt;

    logic test_mode;
    logic scan_in0;
    logic scan_in1;
    logic scan_en;

endinterface : input_if
```

I.15 Output Interface

```
interface output_if (input clk);

    import array_pkg::*;
    arrayout_t matrix_out;
    stroutdim4_t str_out;
    arrayout_t strfull_out;

    logic scan_out0, scan_out1;
    logic valid, ready;

endinterface : output_if
```

I.16 Monitor

```
typedef virtual output_if output_vif;

class monitor_out extends uvm_monitor;
    `uvm_component_utils(monitor_out)

    output_vif vif;
    input_vif vif_in;
    event begin_record, end_record;
    packet_out tr_sys;
    packet_out tr_str;
    packet_out tr_strfull;
    uvm_analysis_port #(packet_out) item_collected_port_sys;
    uvm_analysis_port #(packet_out) item_collected_port_str;
    uvm_analysis_port #(packet_out) item_collected_port_strfull;

    function new(string name, uvm_component parent);
        super.new(name, parent);
        item_collected_port_sys = new ("item_collected_port_sys", this);
        item_collected_port_str = new ("item_collected_port_str", this);
```

```
        item_collected_port_strfull = new ("
            item_collected_port_strfull", this);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        void'(uvm_resource_db#(output_vif)::read_by_name(.
            scope("ifs"), .name("output_vif"), .val(vif)));
        void'(uvm_resource_db#(input_vif)::read_by_name(.scope
            ("ifs"), .name("input_vif"), .val(vif_in)));
        tr_sys = packet_out::type_id::create("tr_sys");
        tr_str = packet_out::type_id::create("tr_str");
        tr_strfull = packet_out::type_id::create("tr_strfull")
            ;
    endfunction

    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        fork
            collect_transactions(phase);
        join
    endtask

    virtual task collect_transactions(uvm_phase phase);

```

```

        wait(vif_in.rst === 1);
        @(negedge vif_in.rst);

        fork
            begin
                forever begin
                //  vif_in.start <= 1;
                    @(posedge vif_in.valid);
                //    vif_in.start = 0;
                    tr_sys.dout = vif.matrix_out;
                //    $display("Matrix Data from RTL @ %t: %X", $time, tr.dout);
                    item_collected_port_sys.write(tr_sys);
                    @(posedge vif_in.clk) ;
                    @(posedge vif_in.clk) ;
                end
            end
            begin
                forever begin
                    //wait(vif_in.str_valid == 1);
                    @(negedge vif.clk);
                    if(vif_in.str_valid == 1)
                    begin
                        tr_str.str_out = vif.str_out;
```

```
                    //$display("%t | Sending Strassen Packet Dut", $time);
                    item_collected_port_str.write(tr_str);
                end
            end
        end
        begin
            forever begin
                //wait(vif_in.str_valid == 1);
                @(negedge vif.clk);
                if(vif_in.strfull_valid == 1)
                begin
                    tr_strfull.strfull_out = vif.strfull_out;
                    //$display("%t | Sending Strassen Packet Dut", $time);
                    item_collected_port_strfull.write(tr_strfull);
                end
            end
        end
    join


endtask
```

```
endclass
```

I.17 Reference Model

```
import array_pkg::*;

//import "DPI-C" context function void runModeller(matrix_t data_a, matrix_t data_b, arrayout_t data_r);
import "DPI-C" context function void runModeller(matrix_t data_a, matrix_t data_b);
import "DPI-C" context function void runStrassen(strindim4_t data_a, strindim4_t data_b);
import "DPI-C" context function void runStrassenFull(matrix_t data_a, matrix_t data_b);
import "DPI-C" context function int getElement(int index);
import "DPI-C" context function int getElementStrassen(int index);
import "DPI-C" context function int getElementStrassenFull(int index);
//https://methi1999.github.io/2020/07/08/dpi.html

class refmod extends uvm_component;
    `uvm_component_utils(refmod)

    input_vif  vif;

    packet_in tr_in_sys;
```

```
    packet_in tr_in_str;
    packet_in tr_in_strfull;
    packet_out tr_out_sys;
    packet_out tr_out_str[];
    packet_out tr_out_strfull[];
    uvm_get_port #(packet_in) in_sys;
    uvm_get_port #(packet_in) in_str;
    uvm_get_port #(packet_in) in_strfull;
    uvm_analysis_port #(packet_out) out_sys;
    uvm_analysis_port #(packet_out) out_str;
    uvm_analysis_port #(packet_out) out_strfull;

    int i, j, k, q;

    function new(string name = "refmod", uvm_component parent)
        ;
        super.new(name, parent);
        in_sys = new("in_sys", this);
        in_str = new("in_str", this);
        in_strfull = new("in_strfull", this);
        out_sys = new("out_sys", this);
        out_str = new("out_str", this);
        out_strfull = new("out_strfull", this);
    endfunction

```

```
    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        tr_out_sys = packet_out::type_id::create("tr_out_sys", this);
        tr_out_str = new[20];
        foreach(tr_out_str[ii]) tr_out_str[ii] = packet_out::type_id::create("tr_out_str");
        tr_out_strfull = new[20];
        foreach(tr_out_strfull[iii]) tr_out_strfull[iii] = packet_out::type_id::create("tr_out_strfull");
        //tr_out_str = packet_out::type_id::create("tr_out_str", this);
        void'(uvm_resource_db#(input_vif)::read_by_name(.scope("ifs"), .name("input_vif"), .val(vif)));
        k = 0;
        q = 0;
    endfunction: build_phase

    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);

        fork
            forever begin
                fork
                forever begin
```

```
          in_sys.get(tr_in_sys);
          //$display("Data passed to model @ %t", $time);
          runModeller(tr_in_sys.data_a, tr_in_sys.data_b);
          //$display("Data to model @ %t: %X, %X", $time, tr_in_sys.data_a, tr_in_sys.data_b);
      end

      begin
          @ (posedge vif.valid) ;

          for(i = 0; i < MATRIXSIZE; i = i + 1)
          begin
              for(j = 0; j < MATRIXSIZE; j = j + 1)
              begin
                  //$display("%X", getElement(i*MATRIXSIZE + j));
                  tr_out_sys.dout[(i*MATRIXSIZE + j)*ACCWIDTH +: ACCWIDTH] = getElement(i*MATRIXSIZE + j);
              end
          end
```

```
                //$display("Data from model @ %t: %X",
                    $time, tr_out_sys.dout);
                out_sys.write(tr_out_sys);
            end
            join

        end
        forever begin
            in_str.get(tr_in_str);
            //$display("Data passed to model @ %t", $time)
                ;
            //$display("Data to model @ %t: %X, %X", $time
                , tr_in_str.str_data_a, tr_in_str.
                str_data_b);
            runStrassen(tr_in_str.str_data_a, tr_in_str.
                str_data_b);

            for(i = 0; i < STRDIM; i = i + 1)
            begin
                for(j = 0; j < STRDIM; j = j + 1)
                begin
                    //$display("%X", getElementStrassen(i
                        *4 + j));
                    tr_out_str[k].str_out[(i*STRDIM + j)*
                        ACCWIDTH +: ACCWIDTH] =
```

```
                                    getElementStrassen(i*STRDIM + j);
                                end
                            end
                            //$display("%t | Sending Strassen Packet Ref,
                                %d", $time, k);
                            out_str.write(tr_out_str[k]);

                            if(k == 19)
                                k = 0;
                            else
                                k = k + 1;

                        end
                        forever begin
                            in_strfull.get(tr_in_strfull);
                            //$display("Data passed to model @ %t", $time)
                                ;
                            //$display("Data to model @ %t: %X, %X", $time
                                , tr_in_str.str_data_a, tr_in_str.
                                str_data_b);
                            runStrassenFull(tr_in_strfull.strfull_data_a,
                                tr_in_strfull.strfull_data_b);

                            for(i = 0; i < MATRIXSIZE; i = i + 1)
                            begin
```

```
                    for(j = 0; j < MATRIXSIZE; j = j + 1)
                    begin
                        //$display("%X", getElementStrassen(i
                            *4 + j));
                        tr_out_strfull[q].strfull_out[(i*
                            MATRIXSIZE + j)*ACCWIDTH +:
                            ACCWIDTH] = getElementStrassenFull(
                            i*MATRIXSIZE + j);
                    end
                end
                //$display("%t | Sending Strassen Packet Ref,
                    %d", $time, k);
                out_strfull.write(tr_out_strfull[q]);

                if(q == 19)
                    q = 0;
                else
                    q = q + 1;

            end
        join

    endtask: run_phase
endclass: refmod
```

I.18 C Modeller

```
#include <stdlib.h>
#include "svdpi.h"
#include <math.h>
#include <stdio.h>


#define DATAWIDTH 8
#define ACCWIDTH 32

#define MATRIXSIZE 16
#define STRDIM 4

signed char matrix_tmp[MATRIXSIZE][MATRIXSIZE];
static signed int matrix_r[MATRIXSIZE*MATRIXSIZE];
static signed int matrix_s[STRDIM*STRDIM];
static signed int matrix_sf[MATRIXSIZE*MATRIXSIZE];

void transpose(signed char* A)
{
    int i, j;
    for (i = 0; i < MATRIXSIZE; i++)
        for (j = 0; j < MATRIXSIZE; j++)
            matrix_tmp[i][j] = A[j*MATRIXSIZE+i];
```

```
    for (i = 0; i < MATRIXSIZE; i++)
        for (j = 0; j < MATRIXSIZE; j++)
            A[i*MATRIXSIZE + j] = matrix_tmp[i][j];
}


void printInputs(signed char* a, signed char* b) {
    printf("Matrix A\n");
    for (int j = 0; j < MATRIXSIZE; j++)
    {
        for (int k = 0; k < MATRIXSIZE; k++)
        {
            printf("%02X ", a[j*MATRIXSIZE + k]);
        }
        printf("\n");
    }

    printf("Matrix B\n");
    for (int j = 0; j < MATRIXSIZE; j++)
    {
        for (int k = 0; k < MATRIXSIZE; k++)
        {
            printf("%02X ", b[j*MATRIXSIZE + k]);
        }
```

```
            printf("\n");
        }
}

void printProducts(signed int* r) {
        printf("Product Matrix\n");
        for (int j = 0; j < MATRIXSIZE; j++)
        {
            for (int k = 0; k < MATRIXSIZE; k++)
            {
                printf("%08X ", r[j*MATRIXSIZE + k]);
            }
            printf("\n");
        }
}

extern "C" void runModeller(signed char* matrix_a, signed char
    * matrix_b) {

        int suma;
        int index;

        //printInputs(matrix_a, matrix_b);

        transpose(matrix_a) ;
```

```

    for (int j = 0; j < MATRIXSIZE; j++)
    {
        for (int k = 0; k < MATRIXSIZE; k++)
        {
            index = j*MATRIXSIZE + k;

            suma = matrix_r[index];
            for (int l = 0; l < MATRIXSIZE; l++)
                suma += matrix_a[j*MATRIXSIZE + l]*matrix_b[l*MATRIXSIZE + k];

            matrix_r[index] = suma;
        }

    }

    //printProducts(matrix_r);
}

extern "C" void runStrassen(signed char* matrix_a, signed char* matrix_b) {

    int suma;
    int index;
```

```


    //printf("Matrix A\n");
    //for (int j = 0; j < STRDIM; j++)
    //{
    //      for (int k = 0; k < STRDIM; k++)
    //      {
    //              printf("%02X ", matrix_a[j*STRDIM + k]);
    //      }
    //      printf("\n");
    //}
    //printf("Matrix B\n");
    //for (int j = 0; j < STRDIM; j++)
    //{
    //      for (int k = 0; k < STRDIM; k++)
    //      {
    //              printf("%02X ", matrix_b[j*STRDIM + k]);
    //      }
    //      printf("\n");
    //}

    for (int j = 0; j < STRDIM; j++)
    {
        for (int k = 0; k < STRDIM; k++)
        {
```

```
                index = j*STRDIM + k;

                suma = matrix_s[index];
                for (int l = 0; l < STRDIM; l++)
                    suma += matrix_a[j*STRDIM + l]*matrix_b[l*
                        STRDIM + k];

                matrix_s[index] = suma;
            }

        }

        //printf("Strassen product\n");
        //for (int j = 0; j < STRDIM; j++)
        //{
        //      for (int k = 0; k < STRDIM; k++)
        //      {
        //            printf("%08X ", matrix_s[j*STRDIM + k]);
        //      }
        //      printf("\n");
        //}

}
```

```
extern "C" void runStrassenFull(signed char* matrix_a, signed
    char* matrix_b) {

    int suma;
    int index;


    for (int j = 0; j < MATRIXSIZE; j++)
    {
        for (int k = 0; k < MATRIXSIZE; k++)
        {
            index = j*MATRIXSIZE + k;

            suma = matrix_sf[index];
            for (int l = 0; l < MATRIXSIZE; l++)
                suma += matrix_a[j*MATRIXSIZE + l]*matrix_b[l*
                    MATRIXSIZE + k];

            matrix_sf[index] = suma;
        }

    }
}
```

```
extern "C" unsigned int getElement(int index) {
    return matrix_r[index];
}

extern "C" unsigned int getElementStrassen(int index) {
    return matrix_s[index];
}

extern "C" unsigned int getElementStrassenFull(int index) {
    return matrix_sf[index];
}

extern "C" void resetAccumulator() {
    for (int j = 0; j < MATRIXSIZE; j++)
        for (int k = 0; k < MATRIXSIZE; k++)
            matrix_r[j*MATRIXSIZE + k] = 0;
}

extern "C" void resetAccumulatorStrassen() {
    for (int j = 0; j < STRDIM; j++)
        for (int k = 0; k < STRDIM; k++)
            matrix_s[j*STRDIM + k] = 0;
}

```

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

I.19 Packet In

```
class packet_in extends uvm_sequence_item;
    //`uvm_object_utils(packet_in)

    rand matrix_t data_a;
    rand matrix_t data_b;

    rand strindim4_t str_data_a;
    rand strindim4_t str_data_b;

    rand matrix_t strfull_data_a;
    rand matrix_t strfull_data_b;

    `uvm_object_utils_begin(packet_in)
        `uvm_field_int(data_a, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(data_b, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(str_data_a, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(str_data_b, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(strfull_data_a, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(strfull_data_b, UVM_ALL_ON|UVM_HEX)
    `uvm_object_utils_end

    function new(string name="packet_in");
        super.new(name);
```

```
    endfunction: new
endclass: packet_in
```

I.20 Packet Out

```
import array_pkg::*;

class packet_out extends uvm_sequence_item;
    //`uvm_object_utils(packet_out)

    arrayout_t dout;
    stroutdim4_t str_out;
    arrayout_t strfull_out;
    //refout_t refout;

    `uvm_object_utils_begin(packet_out)
        `uvm_field_int(dout, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(str_out, UVM_ALL_ON|UVM_HEX)
        `uvm_field_int(strfull_out, UVM_ALL_ON|UVM_HEX)
        //`uvm_field_int(dout_flag, UVM_ALL_ON|UVM_HEX)
        //`uvm_field_int(digit_clk, UVM_ALL_ON|UVM_HEX)
    `uvm_object_utils_end

    function new(string name="packet_out");
        super.new(name);
    endfunction: new
endclass: packet_out
```

I.21 Environment

```
class env extends uvm_env;
    agent       mst;
    refmod      rfm;
    agent_out   slv;
    comparator  comp;
    uvm_tlm_analysis_fifo #(packet_in) to_refmod_sys;
    uvm_tlm_analysis_fifo #(packet_out) from_refmod_sys;
    uvm_tlm_analysis_fifo #(packet_out) from_dut_sys;

    uvm_tlm_analysis_fifo #(packet_in) to_refmod_str;
    uvm_tlm_analysis_fifo #(packet_out) from_refmod_str;
    uvm_tlm_analysis_fifo #(packet_out) from_dut_str;

    uvm_tlm_analysis_fifo #(packet_in) to_refmod_strfull;
    uvm_tlm_analysis_fifo #(packet_out) from_refmod_strfull;
    uvm_tlm_analysis_fifo #(packet_out) from_dut_strfull;

    `uvm_component_utils(env)

    function new(string name, uvm_component parent = null);
        super.new(name, parent);
        to_refmod_sys = new("to_refmod_sys", this);
        from_refmod_sys = new("from_refmod_sys", this);
```

```
        from_dut_sys = new("from_dut_sys", this);

        to_refmod_str = new("to_refmod_str", this);
        from_refmod_str = new("from_refmod_str", this);
        from_dut_str = new("from_dut_str", this);

        to_refmod_strfull = new("to_refmod_strfull", this);
        from_refmod_strfull = new("from_refmod_strfull", this);
        from_dut_strfull = new("from_dut_strfull", this);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        mst = agent::type_id::create("mst", this);
        slv = agent_out::type_id::create("slv", this);
        rfm = refmod::type_id::create("rfm", this);
        comp = comparator::type_id::create("comp", this);
    endfunction

    virtual function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        // Connect MST to FIFO
        mst.item_collected_port_sys.connect(to_refmod_sys.analysis_export);
```

```
        mst.item_collected_port_str.connect(to_refmod_str.
            analysis_export);
        mst.item_collected_port_strfull.connect(
            to_refmod_strfull.analysis_export);
        // Connect FIFO to REFMOD
        rfm.in_sys.connect(to_refmod_sys.get_export);
        rfm.in_str.connect(to_refmod_str.get_export);
        rfm.in_strfull.connect(to_refmod_strfull.get_export);

        //Connect scoreboard
        rfm.out_sys.connect(from_refmod_sys.analysis_export);
        rfm.out_str.connect(from_refmod_str.analysis_export);
        rfm.out_strfull.connect(from_refmod_strfull.
            analysis_export);
        slv.item_collected_port_sys.connect(from_dut_sys.
            analysis_export);
        slv.item_collected_port_str.connect(from_dut_str.
            analysis_export);
        slv.item_collected_port_strfull.connect(
            from_dut_strfull.analysis_export);

        comp.from_refmod_sys.connect(from_refmod_sys.
            get_export);
        comp.from_refmod_str.connect(from_refmod_str.
            get_export);
```

```
        comp.from_refmod_strfull.connect(from_refmod_strfull.
            get_export);

        comp.from_dut_sys.connect(from_dut_sys.get_export);
        comp.from_dut_str.connect(from_dut_str.get_export);
        comp.from_dut_strfull.connect(from_dut_strfull.
            get_export);
    endfunction

    virtual function void end_of_elaboration_phase(uvm_phase
        phase);
        super.end_of_elaboration_phase(phase);
    endfunction

    virtual function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info(get_type_name(), $sformatf("Reporting
            matched %0d", comp.m_matches), UVM_NONE)
        if (comp.m_mismatches) begin
            `uvm_error(get_type_name(), $sformatf("Saw %0d
                mismatched samples", comp.m_mismatches))
        end
    endfunction
endclass
```

I.22 Comparator

```
//class comparator #(type T = packet_out) extends
    uvm_scoreboard;
class comparator extends uvm_scoreboard;
  //typedef comparator #(T) this_type;
  `uvm_component_param_utils(comparator)

  const static string type_name = "comparator";

  input_vif  vif;
  output_vif  vif_out;

  uvm_get_port #(packet_out) from_refmod_sys;
  uvm_get_port #(packet_out) from_dut_sys;

  uvm_get_port #(packet_out) from_refmod_str;
  uvm_get_port #(packet_out) from_dut_str;

  uvm_get_port #(packet_out) from_refmod_strfull;
  uvm_get_port #(packet_out) from_dut_strfull;

  packet_out tr_refmod_sys;
  packet_out tr_dut_sys;

```

```
  packet_out tr_refmod_str;
  packet_out tr_dut_str;

  packet_out tr_refmod_strfull;
  packet_out tr_dut_strfull;

  int m_matches, m_mismatches;
  int str_matches, str_mismatches;
  int strfull_matches, strfull_mismatches;
  event compared, end_of_simulation;

  int sum, i, j;

  int fd;
  string line;
  int length;
  int sys_length;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    void'(uvm_resource_db#(input_vif)::read_by_name
              (.scope("ifs"), .name("input_vif"), .val(vif)));
    void'(uvm_resource_db#(output_vif)::read_by_name
              (.scope("ifs"), .name("output_vif"), .val(vif_out
                 )));
```

```
    from_refmod_sys = new("from_refmod_sys", this);
    from_dut_sys = new("from_dut_sys", this);
    from_refmod_str = new("from_refmod_str", this);
    from_dut_str = new("from_dut_str", this);
    from_refmod_strfull = new("from_refmod_strfull", this);
    from_dut_strfull = new("from_dut_strfull", this);
  endfunction

  virtual function string get_type_name();
    return type_name;
  endfunction


  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    fd = $fopen("test_pipeline.txt","r");
    length = 0;

    forever begin
      $fgets(line, fd);
      if($feof(fd))
        break;
      length = length + line.atoi();
    end

```

```
    $display("Performing %0d multiplications", length);

    $fclose(fd);
  endfunction

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);

    //length = 20;
    m_matches = 0;
    m_mismatches = 0;

    str_matches = 0;
    str_mismatches = 0;


    //$display("%t | Running Comparator", $time);

    fork
      begin
        //$display("%t | Entering Systolic Comparison", $time)
          ;
        if(NORESET)
          sys_length = 1;
        else
```

```
          sys_length = length;
        repeat(sys_length)
        begin
          from_refmod_sys.get(tr_refmod_sys);
          from_dut_sys.get(tr_dut_sys);
          //$display("%t | Recieved Systolic Packets", $time);

          //$display("%0t\t", $time);
          //$display("\tREF: %X\n\tDUT: %X", tr_refmod.dout, tr_dut.dout);

          if(check_identical())
            m_matches++;
          else
          begin
            m_mismatches++;
            $display("%t | MISMATCH SYSTOLIC\n\tREF: %X\n\tDUT: %X", $time, tr_refmod_sys.dout, tr_dut_sys.dout);
            //phase.drop_objection(this);
          end
          //$display("%t | m total : %d", $time, m_matches + m_mismatches);

        end
```

```
      end

      begin
        //$display("%t | Entering Strassen Comparison", $time)
           ;
        repeat(length*8)
        begin
          from_dut_str.get(tr_dut_str);
          from_refmod_str.get(tr_refmod_str);
          //$display("%t | Recieved Strassen Packets", $time);
          //$display("%0t\t", $time);
          //$display ("\tREF: %X\n\tDUT: %X", tr_refmod_str.
             str_out, tr_dut_str.str_out);

            //$display ("REF");
            //for (int j = 0; j < 4; j++)
            //begin
            //   $display("%X ", tr_refmod_str.str_out[(j*4)*
               ACCWIDTH +: ACCWIDTH*4]);
            //end
            //$display ("DUT");
            //for (int j = 0; j < 4; j++)
            //begin
            //   $display("%X ", tr_dut_str.str_out[(j*4)*
               ACCWIDTH +: ACCWIDTH*4]);
```

```
            //end

          if(check_identical_str())
            str_matches++;
          else
          begin
            str_mismatches++;
            $display("%t | MISMATCH STRASSEN\n\tREF: %X\n\tDUT
               : %X", $time, tr_refmod_str.str_out, tr_dut_str
               .str_out);

            //phase.drop_objection(this);
          end
          //$display("%t | str total : %d", $time, str_matches
              + str_mismatches);

        end
      end

      begin
        //$display("%t | Entering Strassen Comparison", $time)
            ;
        repeat(length)
        begin
          from_dut_strfull.get(tr_dut_strfull);
```

```
          from_refmod_strfull.get(tr_refmod_strfull);

          if(check_identical_str_full())
            strfull_matches++;
          else
          begin
            strfull_mismatches++;
            $display("%t | MISMATCH STRASSEN FULL\n\tREF: %X\n\tDUT: %X", $time, tr_refmod_strfull.strfull_out, tr_dut_strfull.strfull_out);

            //phase.drop_objection(this);
          end
          //$display("%t | str_full total : %d", $time, strfull_matches + strfull_mismatches);

        end
      end
    join

    phase.drop_objection(this);
  endtask

  function int check_identical();
    return ~|(tr_refmod_sys.dout ^ tr_dut_sys.dout);
```

```
  endfunction : check_identical

  function int check_identical_str();
    return ~|(tr_refmod_str.str_out ^ tr_dut_str.str_out);
  endfunction : check_identical_str

  function int check_identical_str_full();
    return ~|(tr_refmod_strfull.strfull_out ^ tr_dut_strfull.
       strfull_out);
  endfunction : check_identical_str_full


  virtual function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    $display("Program End");
    $display("====== Results ======");
    $display("Systolic Matches : [%0d] vs Mismatches : [%0d]",
        m_matches, m_mismatches);
    $display("Strassen Matches : [%0d] vs Mismatches : [%0d]",
        str_matches, str_mismatches);
    $display("Strassen_Full Matches : [%0d] vs Mismatches :
       [%0d]", strfull_matches, strfull_mismatches);
    $display("Systolic Performance Counter : [%0d] cycles",
       vif.perf_cnt);
```

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

endclass
```

I.23 Top

```
`include "uvm_macros.svh"
`include "array_pkg.sv"
`include "test_pkg.sv"
`include "input_if.sv"
`include "output_if.sv"
`include "assertions.sv"

//Top
module test;

  import uvm_pkg::*;
  import array_pkg::*;
  import test_pkg::*;

  logic clk;

  initial begin
    clk = 0;
    //rst = 1;
    //#22 rst = 0;

  end

```

```
  always #5 clk = !clk;

  input_if in(clk);
  output_if out(clk);

 matrix_multiplier #(
  .DATAWIDTH(DATAWIDTH),
  .ACCWIDTH(ACCWIDTH),
  .MATRIXSIZE(MATRIXSIZE))
  top
  (
  .clk(clk),
  .rst(in.rst),
  .scan_in0(in.scan_in0),
  .scan_en(in.scan_en),
  .test_mode(in.test_mode),

  .mult(in.mult),
  .load(in.load),
  .acc(in.acc),
  .matrix_a_data(in.matrix_a_data),
  .matrix_b_data(in.matrix_b_data),
  .matrix_out(out.matrix_out),

  .str_rst(in.str_rst),
```

```
    .str_en(in.str_en),
    .str_matrix_a(in.str_matrix_a),
    .str_matrix_b(in.str_matrix_b),
    .str_out(out.str_out),

    .strfull_rst(in.strfull_rst),
    .strfull_en(in.strfull_en),
    .strfull_matrix_a(in.strfull_matrix_a),
    .strfull_matrix_b(in.strfull_matrix_b),
    .strfull_out(out.strfull_out)

    );


/*
    arraydim_t a_aligned;
    arraydim_t b_aligned;

    data_alignment #(.DATAWIDTH(DATAWIDTH), .MATRIXSIZE(
        MATRIXSIZE)) aligner (
        .clk(clk),
        .reset(in.rst),
        .shift(in.mult),
        .a_in(in.matrix_a_data),
        .b_in(in.matrix_b_data),
```

```
    .a_out(a_aligned),
    .b_out(b_aligned)
    );

  multiplier_systolic_array #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH(ACCWIDTH), .MATRIXSIZE(MATRIXSIZE)) sys (
    .clk(clk),
    .reset(in.rst),
    .a_in(a_aligned),
    .b_in(b_aligned),
    .load_en(in.load),
    .mult_en(in.mult),
    .acc_en(in.acc),
    .d_out(out.matrix_out),

    .scan_in0(in.scan_in0),
    .scan_en(in.scan_en),
    .test_mode(in.test_mode)
    //.scan_out0(out.scan_out0)
    );

  multiplier_strassen #(.DATAWIDTH(DATAWIDTH), .ACCWIDTH(ACCWIDTH)) str (
    .clk(clk),
    .reset(in.str_rst),
```

```
    .enable(in.str_en),
    .matrix_a(in.str_matrix_a),
    .matrix_b(in.str_matrix_b),
    .matrix_c(out.str_out),

    .scan_in0(in.scan_in0),
    .scan_en(in.scan_en),
    .test_mode(in.test_mode)
    //.scan_out0(out.scan_out0)
    );

    multiplier_strassen_full #(.DATAWIDTH(DATAWIDTH), .
        ACCWIDTH(ACCWIDTH)) strfull (
    .clk(clk),
    .reset(in.strfull_rst),
    .enable(in.strfull_en),
    .matrix_a(in.strfull_matrix_a),
    .matrix_b(in.strfull_matrix_b),
    .matrix_c(out.strfull_out),

    .scan_in0(in.scan_in0),
    .scan_en(in.scan_en),
    .test_mode(in.test_mode)
    //.scan_out0(out.scan_out0)
    );
```

```

*/

//assertion_cov ass(out);

initial begin
  `ifdef INCA
     $recordvars();
  `endif
  `ifdef VCS
     $vcdpluson;
  `endif
  `ifdef QUESTA
     $wlfdumpvars();
     set_config_int("*", "recording_detail", 1);
  `endif

  `ifdef SDFSCAN
    $sdf_annotate("sdf/matrix_multiplier_tsmc18_scan.sdf",
       test.sys);
  `endif
  $set_coverage_db_name("matrix_multiplier");
  uvm_resource_db#(input_vif)::set(.scope("ifs"), .name("
     input_vif"), .val(in));
```

```
    uvm_resource_db#(output_vif)::set(.scope("ifs"), .name("
        output_vif"), .val(out));

    run_test("simple_test");
  end
endmodule
```

I.24 Test

```
class simple_test extends uvm_test;
  env env_h;
  sequence_in seq;

  `uvm_component_utils(simple_test)

  function new(string name, uvm_component parent = null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env_h = env::type_id::create("env_h", this);
    seq = sequence_in::type_id::create("seq", this);
  endfunction

  task run_phase(uvm_phase phase);
    seq.start(env_h.mst.sqr);
  endtask: run_phase

endclass
```