Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN4020: Advanced Digital Systems
GROUP 4

SoC Project Report

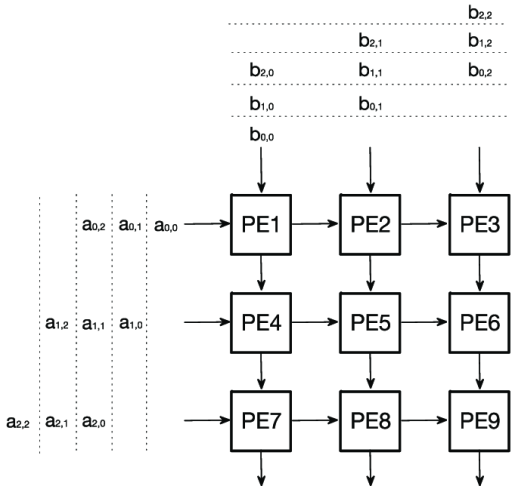
NAME	INDEX NUMBER	CONTRIBUTION
G.S.M.U.K. Samarakoon	**190543B**	Memory Access handling overseeing
G.B.N.M. Silva	**190592X**	Data Extraction method implementation, Data Feeding mechanism to systolic arrays, Scaling process of the matrix multiplication
H.M.P. Siriwardana	**190595J**	Implementing the systolic arrays, BRAM reading and writing part implementation, Parallel computation for increasing the throughput
C.H.W. Wijegunawardana	**190696U**	PS side programming, DMA configuration
D.R.R.T. Wijesuriya	**190712T**	Research and documentation

Overview

In this project we implemented a design which can be used to undergo matrix multiplication of square matrices. These matrices can be in the size of either 4x4 or 8x8 or 16x16. On top of that we have integrated a parallel path to the design which will allow us to process two sets of matrix pairs.

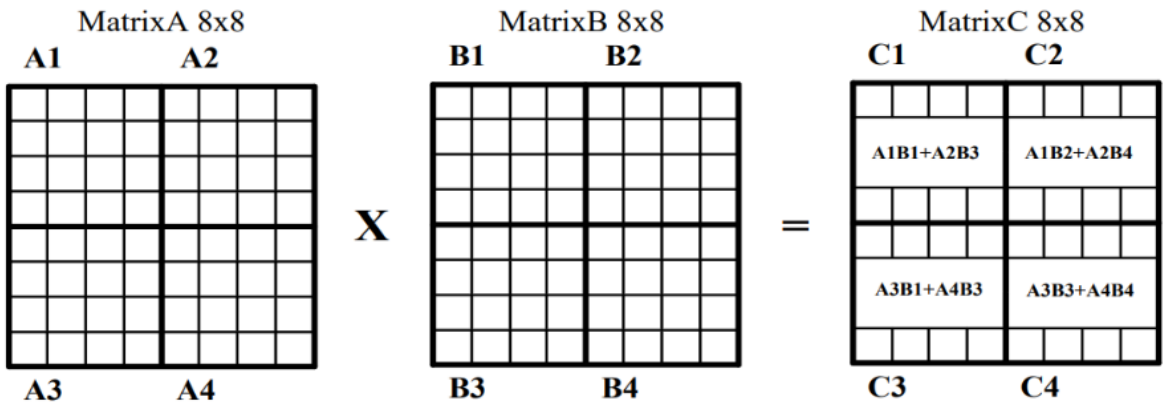
Systolic Array - Introduction

An n-dimensional structural pipeline with synchronous communication between the PEs is known as a systolic array. Hence, a systolic array utilizes the algorithm's parallelism and pipelining at the same time.Numerous identical simple processors, also known as processing elements (PEs), are grouped in a systolic array in a structured manner, much like in a linear or two-dimensional array. Every processing element has a limited amount of private storage and is connected to the other PEs.

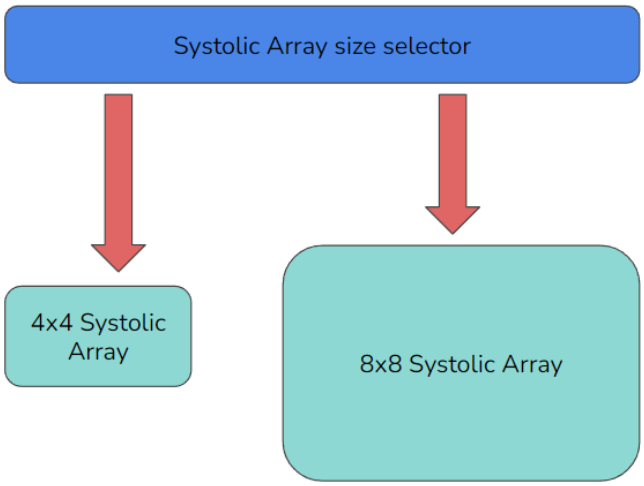


Adapting a Single Systolic Array for Variable Matrix Sizes

As we discussed in the previous part we may have to implement a NxN systolic array to compute the multiplication of the two NxN matrices. We have gone through this brute force approach. We first implemented a 4x4 systolic array and then went to a 16x16 systolic array. But the 16x16 systolic array implementation was restricted due the limitations of the resources. Due to that reason we had to implement a 8x8 systolic array.
Then we were looking for a method to scale the matrix sizes. Then we went through a research paper and came up with a method to scale the process as depicted in the diagram.



As it shows we calculated the quarters of the resulting matrix and sent it to the “Result BRAM”. Then when it comes to the scaling process, we could have used an 8x8 systolic array to compute a 4x4 matrix multiplication. But it may consume some additional time for padding and extraction processes. So we decided to use a combination of both 4x4 systolic array and 8x8 systolic array implementation. Based on the matrix size we can pick the systolic array implementation that we want to carry on as follows.



Block diagram of the design

Following diagram depicts the combination of our design in the both PS and PL side. In the PL side, there is the systolic array implementation. Then there are two modules called, “Data Extractor A” and “Data Extractor B”. Those two modules are responsible for arranging the data values to be fed into the systolic array. These two separate operations have to be done synchronously and it is monitored by the “Data Feeder” module.

In our project, the Zynq processor system plays a crucial role by establishing a seamless connection to the Programmable Logic (PL) side through the integration of

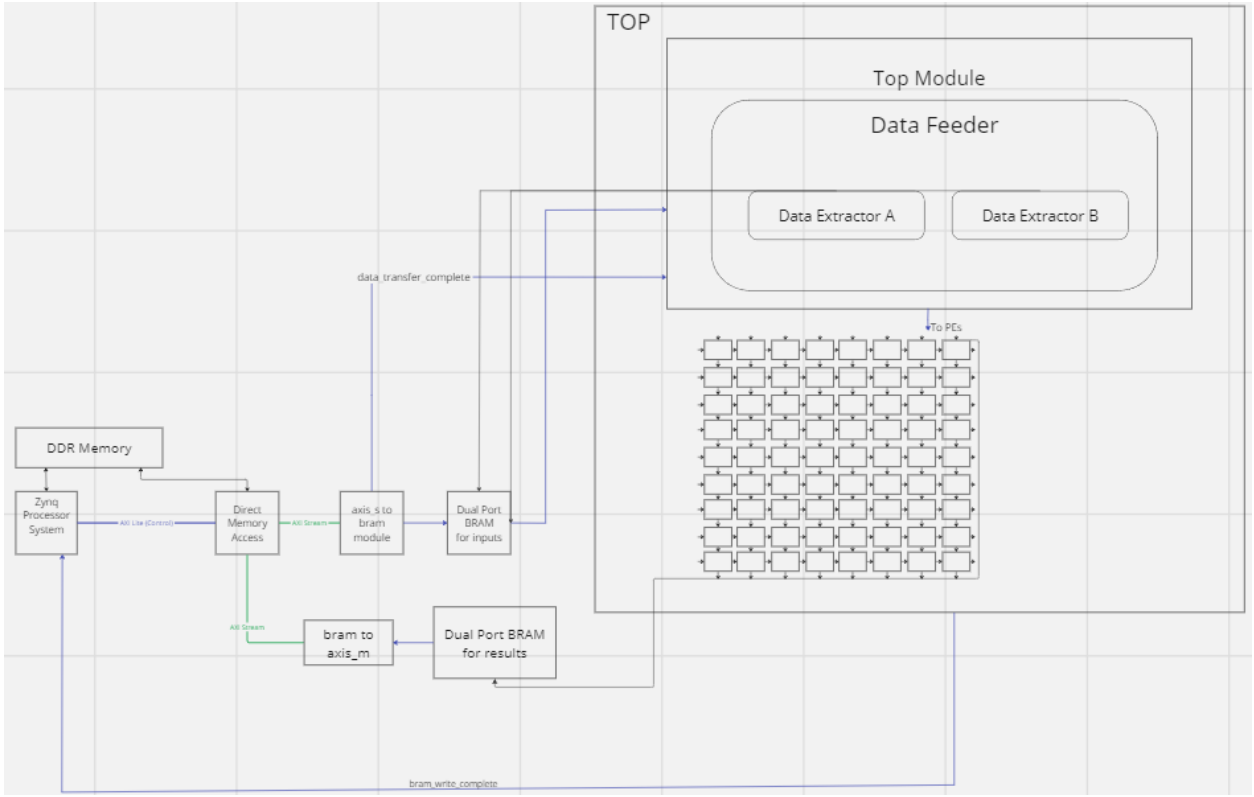
Direct Memory Access (DMA) IP. This strategic integration facilitates the mapping of the Double Data Rate (DDR) memory located in the Processing System (PS) side into an Advanced eXtensible Interface (AXI) stream.

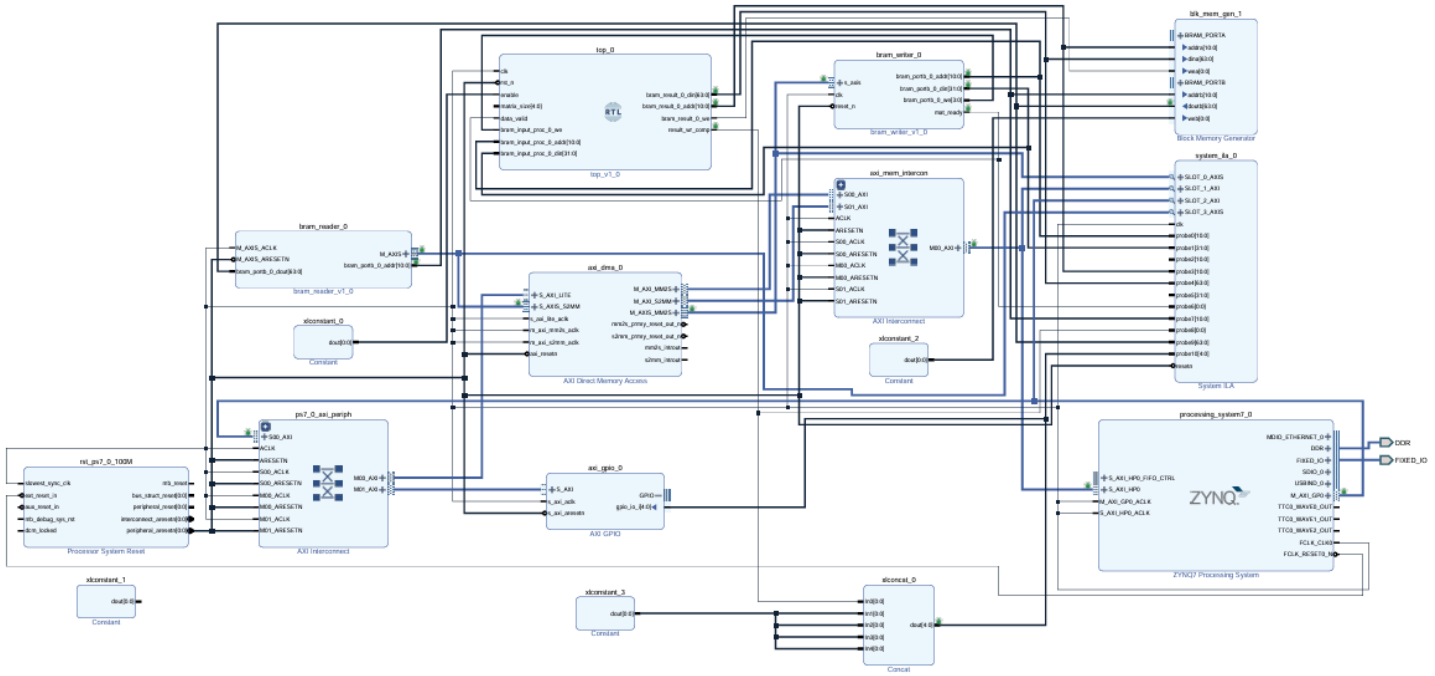
The PS takes charge of transmitting data by effectively leveraging the AXI DMA driver functions, streamlining the flow of information between the processor and the programmable logic. To facilitate the interpretation of the data streams entering and leaving the DMA, two custom Intellectual Properties (IPs) have been developed: "bram_writer" and "bram_reader."

Custom IPs: "bram_writer" and "bram_reader"

These custom IPs serve as essential components in our system, functioning to interface with block RAM. The "bram_writer" IP is responsible for efficiently writing data to the block RAM, ensuring a seamless and reliable process for storing information. On the other hand, the "bram_reader" IP is designed to read data from the block RAM, completing the cycle by enabling the retrieval of stored information.

In summary, the integration of the Zynq processor system with the PL side through DMA IP, combined with the specialized functionalities of "bram_writer" and "bram_reader," creates a robust and efficient data flow mechanism within our project architecture. This well-coordinated system ensures optimal utilization of resources and enhances the overall performance of our embedded system.





Increasing Throughput

Design improved adding two systolic array subsystems to get operations done parallel manner. The instructions for the 2 systolic arrays are given by the first 32 bit word transferred to the PL side from PS

It is decoded as,
0-4 bits - Matrix size for systolic array 1
5-9 bits - Matrix size for systolic array 2
10 - 11 bits - enable parallel (2 bits used for further expansion)
Other bits - not used yet in the design

The inputs for the systolic array array are taken from the two ports of the input BRAM after the data is written by the PS side.

Time Evaluations

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.512 ns	Worst Hold Slack (WHS):	0.028 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	51459	Total Number of Endpoints:	51443	Total Number of Endpoints:	21721

All user specified timing constraints are met.

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	1.512 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	51459

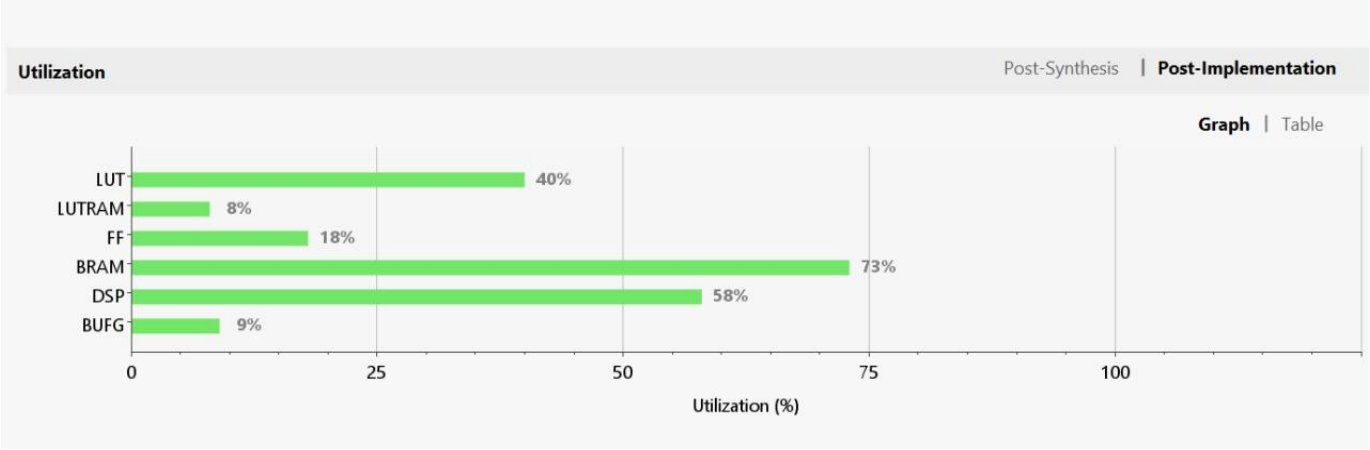
Implemented Timing Report

Power — **Summary** | On-Chip

Total On-Chip Power:	**1.822 W**
Junction Temperature:	**46.0 °C**
Thermal Margin:	39.0 °C (3.2 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Implemented Power Report

Utilization report



Verification steps taken

We have implemented testbenches for the modules that we have implemented. There are some waveform diagrams of those testbenches.

Module level verification

Verification was initially done at module level.The RTL was developed and testbenches were written to test its functionality.

PL side combined verification

After individual verification the module were combined from input BRAM to results BRAM with all the modules in between. Input BRAM was set to known values by a coe file and the results and functionality was tested with testbenches.

PS side verification

The DMA implementation was tested initially by integrating with an inverter, which inverts every bit of a 32-bit number, sent as input from the PS through the DMA, and sends the result back to the PS. A custom IP was created using this inverter module, block design was created and successfully verified the implementation of the data interface between the PS and the PL. Then the data read and write modules designed to transfer data from the AXI Stream interface to BRAMs of the PL side were combined to check the accuracy of data transfer.

Full combined Verification

The PL and PS side which were already tested was combined by using common BRAMs. The design was tested afterwards checking results and functionality using ILA modules.