Designing and verification of Systolic Array Multiplication (SAM) Unit

A project report submitted by

Avinash Choudhary [EE24E017]
Dadichiluka Shiva Lalith [EE24M089]

Under the guidance of
PURUSHOTHAMAN RAMAKRISHNAN
And
GOPALAKRISHNAN SRINIVASAN

In the partial fulfillment of
CAD for VLSI
subject

ACKNOWLEDGEMENTS

We want to sincerely thank our colleagues from Indian Institute of technology,Madras for their helpful criticism and cooperative attitude during our conversations. Their advice and assistance greatly influenced how we understood and implemented the Systolic Array Multiplication (SAM) Unit design.

We also like to thank the teaching assistants for helping us with the cocotb framework and giving resources, which made our verification process much easier.

Finally, we recognize that this project requires teamwork. By cooperating, we were able to leverage our own talents and successfully execute the SAM unit.

Avinash Choudhary [EE24E017]

Dadichiluka Shiva Lalith [EE24M089]

Table of contents:

Objective:

Matrix multiplication has become very common in use from basic mathematical solutions, signal processing to advanced Machine learning algorithms all use matrix multiplication. But the generic process of multiplying matrices is time and power consuming. A Systolic Array Multiplier (SAM) block is a module which is in the structure of a mesh which computes the individual product at each processing element block and propagates input to the next processing element. Each processing element having MAC unit with below functionality

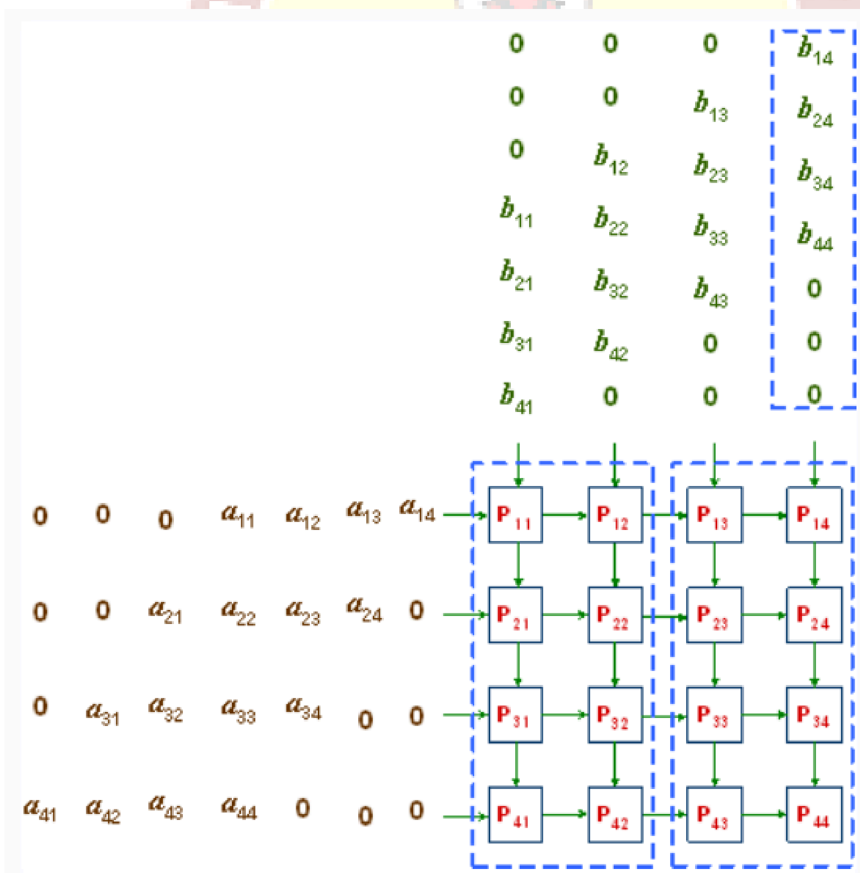
MAC: Result=Result + (Input_1 * Input_2)

We have implemented a SAM block which can handle both Integer and floating point representation type inputs. Inputs and outputs are both in binary format. Since it can handle different types of input formats, its usability range will be increased. The design of SAM block has been implemented in Bluespec Verilog. The design (DUT) is verified using the COroutine based COsimulation TestBench environment (COCOTB) and python.

Introduction:

SAM Module:

- The very basic building block of Systolic Array Multiplier comes from the MAC unit implemented earlier.
- In this design, we have initiated an array of 4x4 size (since given matrix size is 4), containing the interface of previously designs MAC unit.
- Then we need to realign the input matrices so that the input into the array flows in a diagonal fashion similar to that shown in below figure.



- After we align our inputs, one by one every clock cycle we send the inputs as shown in above figure.
- Once we assign inputs for one cycle, we need to propagate the inputs as inputs from top matrix flows down into next mac element, and the inputs from left matrix flows right into next mac element.
- Finally when all inputs are sent either row wise for top or column wise for left, we then get our output in the each mac element as the final result.

Design Architecture:

In the design architecture implemented, the input matrices, say a,b, are stored in the fashion shown in above figure. So a being an array of 4x7 with registers of Bit#(16) type, similarly b being an array of 7x4 with registers of Bit#(16) type. An array of mac blocks are initiated in mac_matrix. a_prop and b_prop are the array of size 4x4 with Bit#(16) type registers which stores the input of mac_matrix. Then the aligned_out stores the output value.

The rule matrix_prop stores the propagated value of inputs for mac_matrix from the input matrices a and b. For each cycle, the a_prop is shifted by one column right, and the b_prop is shifted by one row down.

Then in rule rl_instantiate, all the input values are assigned to mac_matrix from a_prop and b_prop.

The rule rl_assign_output stores the value of output from the mac_matrix into aligned_out.

But since all rules run parallelly, we cant allow rule rl_instantiate to fire until the matrix_prop is already fired.

To solve this, we have put one Boolean type variable compute, which when true, then only the rl_instantiate will fire. And the value of compute toggles every any of both rules fires.

The method get_in is to take input values.

Verification:

Verification is done using COCOTB and python. Each of input is given one by one for both input matrices.