Multiply and Accumulate Module based on Tensor Processing Unit processor

Raghavendra Chathurajupalli

1 Project Overview

The objective of this project is to implement a digital ASIC design capable of performing matrix multiplication using a systolic array architecture. This architecture is inspired by Google's Tensor Processing Unit (TPU) but simplified for educational purposes.

2 Design Specification

1. Multiply-and-Accumulate (MAC) Unit-

Objective: Implement a basic MAC unit by employing Systolic Array Unit, Quantization unit and Activation Unit.

Input: 8 bit Unsigned Multiplier Operand A (Input Feature), 8 bit Unsigned Multiplier Operand B (Weight), Accumulator Input (24 bit partial sum)

Output: Accumulator Output (Updated 24 bit Partial Sum)

Purpose: A MAC unit typically forms the fundamental computational element within each cell of the Systolic Array, performing the multiplication of two numbers and the accumulation of the product into an accumulator. The MAC units are integral to executing the primary arithmetic operations within the array.

2. Systolic Array Implementation-

Objective: Implement a 4x4 systolic array for matrix multiplication.

Inputs: Receives $8-bit$ unsigned integers from both weight and input feature memories.

Process: Adds the $16-bit$ product of inputs to the incoming 24-bit partial sum to generate a $24-bit$ output partial sum.

Control: A synchronous system controlled by a single global clock (clk) and an asynchronous reset (rst).

Purpose: This is the core of the matrix multiplication process, designed as a $4x4$ array to facilitate parallel computation. It efficiently computes the products and partial sums required for matrix multiplication. The Systolic Array architecture is specifically optimized for operations that can benefit from the structured data flow, such as matrix multiplication, making it a critical component of many high-performance computing systems, including TPUs.

3. Quantization Unit-

Objective: Quantize the 24-bit vertical partial sums down to $8-bits$.

Specification: Implement and integrate the quantization unit into the systolic array.

Purpose: This unit reduces the bit-width of the output partial sums from the Systolic Array (24-bit) down to $8-bit$. Quantization is a critical process in digital signal processing and neural network inference, where it helps manage the trade-off between precision and computational/storage efficiency. The Quantization Unit operates on the data produced by the Systolic Array but is a separate functional block that processes its outputs.

4. Activation Unit-

Objective: Process the output of the quantization unit through an activation function.

Parameter: Activation threshold, $\tau = 10$.

Specification: The output will subsequently be sent back to the feature memory.

Purpose: Following quantization, the Activation Unit applies a non-linear function (defined by a threshold $\tau = 10$ in this project) to the quantized values. This is a common practice in neural networks to introduce non-linearity into the model, enabling it to learn complex patterns. Like the Quantization Unit, the Activation Unit acts on the output of the previous stage but is distinct in its functionality.

5. Test Bench and Demonstration-

Objective: Write a test bench to clock the system at 100 MHz.

Specification: Demonstrate the multiplication of two matrices as shown in the project slides.

Tools: The setup must be tested and demonstrated using both a general simulation environment and Cadence XCelium.

6.Technical Requirements-

Global Clock (clk): System must be designed to operate synchronously under a global clock signal.

Asynchronous Reset (rst): System must include an asynchronous reset mechanism to initialize the state.

7.Input/Output Specifications-

Input Feature and Weight Memories: $8-bit$ unsigned integers.

Output Partial Sum: $24-bit$, leading to an $8-bit$ quantized output.

3 Behavioral Description

The images depict a simplified view of how matrix multiplication might be implemented in a hardware accelerator, specifically designed for neural network computations, with a focus on 8-bit Multiply-Accumulate (MAC) operations. The MAC operations are fundamental to the dot product calculations in matrix multiplication, which are central to many neural network operations, particularly in convolutional neural networks (CNNs).

The first image outlines a block diagram of the MAC units arranged to perform matrix multiplication with weight memory and feature memory, where the activation and quantization functions are applied. The second image shows a detailed view of a single MAC unit.

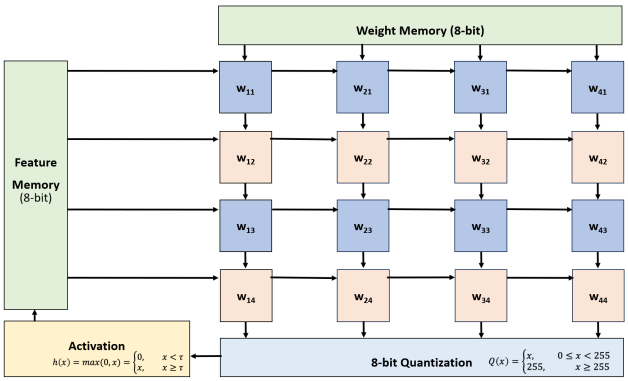

Figure 1: Complete Block Diagram of Multiply and Accumulate System

Here's a step-by-step explanation of how a 44 matrix multiplication would occur:

Initialization: Both the feature matrix and the weight matrix are stored in their respective 8-bit memories.

MAC Operations: Each MAC unit receives one element from the feature vector (activation) and one weight from the weight matrix. In the detailed view, the MAC unit multiplies an 8-bit activation ('a') with an 8-bit weight ('w'), yielding a $16-bit$ product.

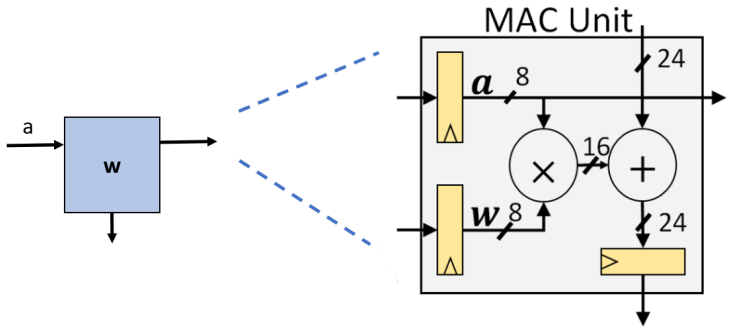

Figure 2: MAC Unit

Accumulation: The $16-bit$ products from the MAC units are accumulated (summed) with a previously stored value in an accumulator, which is at least 24 bits to prevent overflow.

Activation Function: After accumulation, the result may pass through an activation function $h(x) = max(0, x)$ (shown in the first image as ReLU or Rectified Linear Unit). This operation introduces non-linearity, which is crucial for the neural network's ability to learn complex patterns. If the accumulated value is below a threshold τ , it is set to zero.

Quantization: The final step involves quantizing the result back to an 8-bit value to maintain consistency with the input bit-width. If the value is within the 8-bit range (0-255), it's kept as is; if it's larger, it's clamped to 255.

Processing Algorithm: The process for a 4×4 matrix multiplication would involve performing these steps for each element of the output matrix. For instance, to compute the first element of the result matrix:

1. 16 8-bit numbers are loaded into weight memory and feature memory units through 8-bit wide dedicated ports. Once finished, a startSignal is initiated to kickstart matrix multiplication.

2. Weight matrix is loaded into the systolic array in 4 cycles through the north lines i.e MACUnit $C11\_0, C12\_1, C13\_2, C14\_3$.
3. Then feature memory is loaded through west lines $C11\_0, C21\_4, C31\_8, C41\_12$ sequentially per each cycle.

	Weight memory							
	C11_0		C21_1		C13_2		C14_3	
1	C11_0	W11	C12_1	W12	C13_2	W13	C14_3	W14
2	C21_4	W21	C22_5	W22	C23_6	W23	C24_7	W24
3	C31_8	W31	C32_9	W32	C33_10	W33	C34_11	W34
4	C41_12	W41	C42_13	W42	C43_14	W43	C44_15	W44
CLK Cycle	Block	Value	Block	Value	Block	Value	Block	Value

Figure 3: Weight Elements Loading

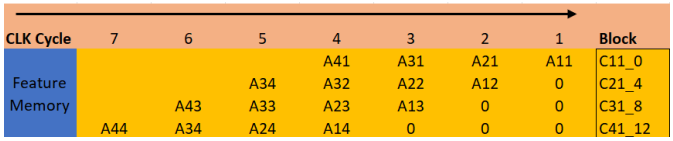

Figure 4: Feature Elements Loading

4. At first cycle the product of sum and feature is calculated. This product is propagated along south and is accumulated by the receiving blocks.

5. At subsequent cycles, the weight and the newly received feature element is multiplied and added to the previously accumulated sum. This sum is propagated again to south. This process continues until the last feature memory element is loaded i.e A44.

6. The result matrix is passed through Quantization unit, which sets the value to 255 if it is greater than 255 else the number is unchanged.

7. The Quantized result is passed through Activation Unit where if the number greater than the threshold of 10 is allowed to pass else it is set to 0.

8. This result is stored in feature memory and the output port displays the result from feature memory.

4 RTL Description :

ASIC Block Diagram at RTL level:

Code Hierarchy:

Memory Loader Unit: This unit is responsible to take inputs from 8 bit ports port_A, port_W and load it into Weight_Memory and Feature_Memory. Algorithm for this module is:

1. Create ports of 8 bit wide, control signals for clock, reset, write enable for feature memory and weight memory in the port list.
2. Create memory unit of size 16 with each element supporting 8 bits i.e 16KB for Weight memory.
3. Create memory unit of size 32 with each element supporting 8 bits i.e 32KB for Feature memory. Only first 16 spaces are used for storing inputs and the rest is used for output data storage.
4. Set the Write_enable_A to 1 and Write_enable_W to 0. With clock sensitivity, load the memory from port_A in Feature memory. Once completed set the Write_enable_A to 0 and Write_enabl_W to 1.
5. With clock sensitivity, load the memory from port_W in Feature memory.
6. Once completed set the Write_enable_W to 0.

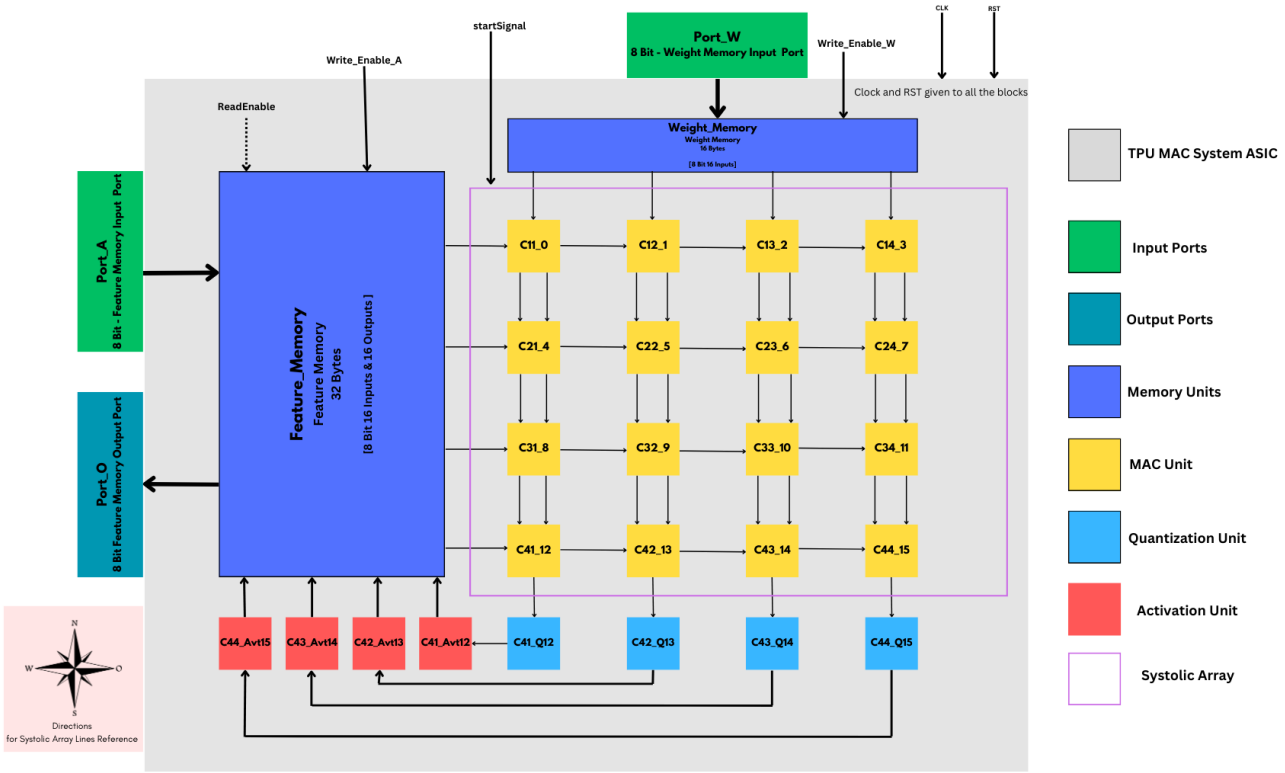

Figure 5: Block Diagram

```
PS C:\iverilog\DE\Loader> iverilog ./memory_loader_tb.v ./memory_loader.v
PS C:\iverilog\DE\Loader> vvp ./a.out
Time:                    0       Input Value Through port_A:  0
Time:      360000000000000       Input Value Through port_A:  1
Time:      380000000000000       Input Value Through port_A:  2
Time:      400000000000000       Input Value Through port_A:  3
Time:      420000000000000       Input Value Through port_A:  4
Time:      440000000000000       Input Value Through port_A:  1
Time:      460000000000000       Input Value Through port_A:  2
Time:      480000000000000       Input Value Through port_A:  3
Time:      500000000000000       Input Value Through port_A:  4
Time:      520000000000000       Input Value Through port_A:  1
Time:      540000000000000       Input Value Through port_A:  2
Time:      560000000000000       Input Value Through port_A:  3
Time:      580000000000000       Input Value Through port_A:  4
Time:      600000000000000       Input Value Through port_A:  1
Time:      620000000000000       Input Value Through port_A:  2
Time:      640000000000000       Input Value Through port_A:  3
Time:      660000000000000       Input Value Through port_A:  4
Feature Memory:
[0] =   1
[1] =   2
[2] =   3
[3] =   4
[4] =   1
[5] =   2
[6] =   3
[7] =   4
[8] =   1
[9] =   2
[10] =   3
[11] =   4
[12] =   1
[13] =   2
[14] =   3
[15] =   4
The feature Matrix is:
```

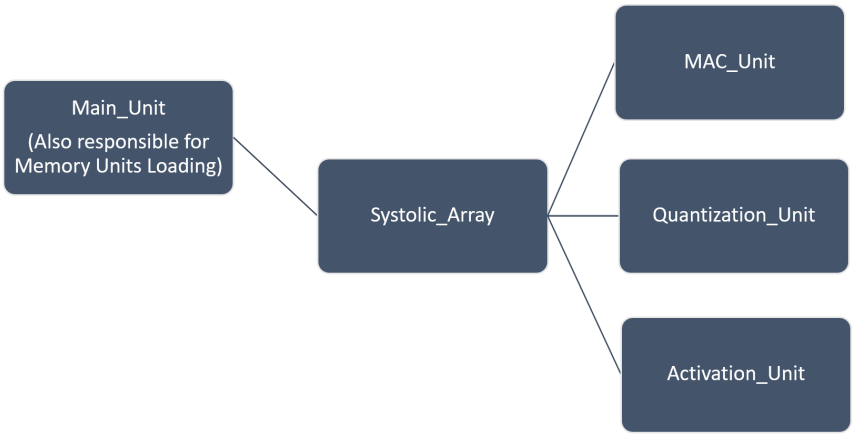

Figure 6: Code Hierarchy

```
Time:                    0       Input Value Through port_W:  0
Time:       40000000000000       Input Value Through port_W:  4
Time:       60000000000000       Input Value Through port_W:  0
Time:       80000000000000       Input Value Through port_W:  2
Time:      100000000000000       Input Value Through port_W:  1
Time:      120000000000000       Input Value Through port_W:  4
Time:      140000000000000       Input Value Through port_W:  3
Time:      160000000000000       Input Value Through port_W:  2
Time:      180000000000000       Input Value Through port_W:  0
Time:      200000000000000       Input Value Through port_W:  4
Time:      220000000000000       Input Value Through port_W:  3
Time:      240000000000000       Input Value Through port_W:  0
Time:      260000000000000       Input Value Through port_W:  1
Time:      280000000000000       Input Value Through port_W:  4
Time:      300000000000000       Input Value Through port_W:  3
Time:      320000000000000       Input Value Through port_W:  2
Time:      340000000000000       Input Value Through port_W:  1

Weight Memory:
[0] =   4
[1] =   0
[2] =   2
[3] =   1
[4] =   4
[5] =   3
[6] =   2
[7] =   0
[8] =   4
[9] =   3
[10] =   0
[11] =   1
[12] =   4
[13] =   3
[14] =   2
[15] =   1
The Weight Matrix is:
   4   0   2   1
   4   3   2   0
   4   3   0   1
   4   3   2   1
```

Figure 8: Execution output of Memory Loading Unit for Weight Memory

Systolic Array with Mac Unit :Systolic array is responsible for performing matrix multiplication. The algorithm for the process is mentioned in behavioral.

Result: The result is passed through ports Activation_result12, Activation_result13, Activation_result14, Activation_result15 to main module along with the count (counter). In the main module, based on the count value, the output is written into output section of Feature_Memory. The count based memory writing code snippet is given below:

```
always@(*) begin
    if(startSignal && cnt <=3)begin
        Wen <= 1;
        inp_north0 <= Weight_Memory[12-(cnt*4)];
        inp_north1 <= Weight_Memory[13-(cnt*4)];
        inp_north2 <= Weight_Memory[14-(cnt*4)];
        inp_north3 <= Weight_Memory[15-(cnt*4)];
    end
    else
        Wen<= 0;
end
always @(*)begin
    if(startSignal && cnt >3 && cnt <=7)begin
        inp_west0 <= Feature_Memory[(cnt-4)*4];
    end
end


always @(*)begin
    if(startSignal && cnt >4 && cnt <=8)begin
        inp_west1 <= Feature_Memory[(cnt-5)*4 +1];
    end
end

always @(*)begin
    if(startSignal && cnt >5 && cnt <=9)begin
        inp_west2 <= Feature_Memory[(cnt-6)*4+2];
    end
end

always @(*)begin
    if(startSignal && cnt >6 && cnt <=10)begin
        inp_west3 <= Feature_Memory[(cnt-7)*4+3];
    end
end
```

Figure 9: Code for Writing Output to Feature Memory

The MAC Unit within the systolic array has two south lines to propagate weight and accumulated sum respectively. It also has an output port to propagate eastern blocks. The two inputs are from north are for weight and accumulated sum and western side input for feature elements. The MAC unit is also clocked and has the logic below.

```
module MACUnit( output reg [23:0]AccumulatedSum, output [7:0]siglineEast, output [7:0] siglineSouth,  input [7:0]FM,input [7:0]WM, input [23:0] AddCarry, input WEn, input clk,input rst ); // Port Declarations

    reg [15:0]m;
    reg [7:0]siglineEast; reg[7:0] siglineSouth;
    //wire[23:0] macOutwire;




always @(posedge clk or posedge rst)
    begin

            AccumulatedSum <= (siglineEast*siglineSouth) + AddCarry; // Calculate product and previously received accumulated sum


    end
//////////////////////////////////////////////////
always @(posedge clk or posedge rst) // Logic to propagte values to East and south lines
    begin
        if(rst)
            begin
                siglineEast <= 8'b00000000;
                siglineSouth <= 8'b00000000;
            end
        else
            begin
                siglineEast<= FM;

                if( WEn==1 )
                siglineSouth <= WM;

                else siglineSouth<=siglineSouth;
            end
    end


endmodule;
//////////////////////////////////////////////////////////////////////////////////
```

Figure 10: MAC Unit Code

The resultant Output in the sequence mentioned in below tables:

result12	result13	result14	result15
C11	0	0	0
C21	C12	0	0
C31	C22	C13	0
C41	C32	C23	C14
0	C42	C33	C24
0	0	C43	C34
0	0	0	C44

Figure 11: Output Sequence from Systolic Array

Test result, for matrix elements with all 1s:

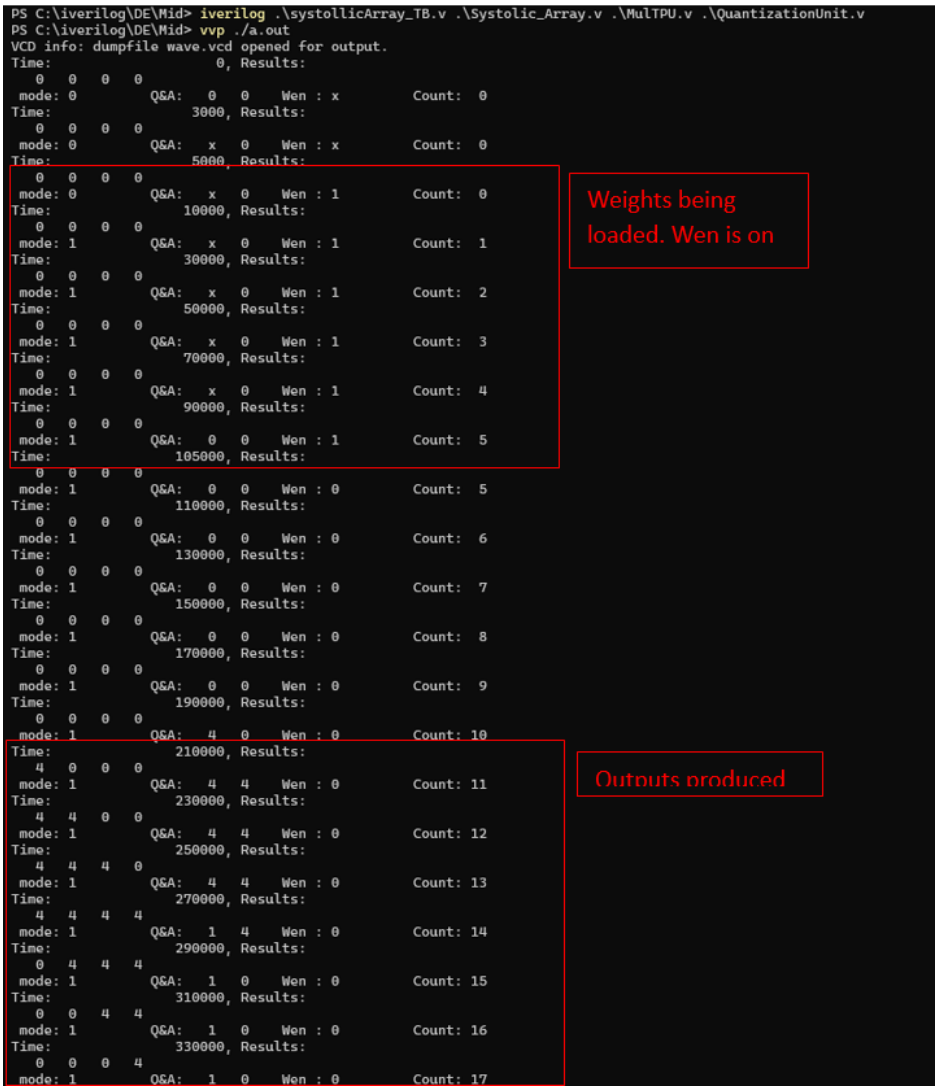

Figure 12: Terminal Output from Systolic Array

Quantization Unit: This unit is responsible for converting a 24 bit data into 8 bit data by removing the redundant bits. If output is greater than 255, the data is set to 255 else, the data is allowed to pass as it is. This module is also clocked and uses condition if-else statements.

```
PS C:\iverilog\DE\MidTerm_Raghavendra\Quantization Unit> iverilog ./QuantizationUnit_TB.v ./QuantizationUnit.v
PS C:\iverilog\DE\MidTerm_Raghavendra\Quantization Unit> vvp ./a.out
Time=                   0, Reset Applied, output_data=  x
Time=              100000, Reset Released
Time=              120000, input_data=     100, output_data=100
Time=              140000, input_data=     256, output_data=255
Time=              160000, input_data=    1024, output_data=255
Time=              180000, input_data=     200, output_data=200
Time=              190000, Reset Applied, output_data=  0
./QuantizationUnit_TB.v:53: $finish called at 200000 (1ps)
PS C:\iverilog\DE\MidTerm_Raghavendra\Quantization Unit>
```

Figure 13: Quantization Unit Result

Activation unit: This unit is responsible to remove values which are not above the threshold $\tau = 10$. This

module is also clocked and uses condition if-else statements.

```
PS C:\iverilog\DE\MidTerm_Raghavendra\Activation Unit> vvp ./a.out
 Inpu Data   0, data_out=  0
 Inpu Data  12, data_out= 12
 Inpu Data   8, data_out=  0
 Inpu Data 255, data_out=255
 Inpu Data  18, data_out= 18
./Activation_Unit_TB.v:48: $finish called at 220000 (1ps)
```

Figure 14: Activation Unit Result

5 Functional Verification and Testing

The main module is the top module for this design project. It is built on Memory Loader Unit. It's main functions are to receive inputs for matrix multiplication, invoke systolic array module to perform matrix multiplication and then to receive the output of the process and store it in feature memory.

The systolic array uses set of MAC_Unit modules to calculate the matrix multiplication and produces a 24 bit partial sum and this result is quantized to 8 bit with a maximum cap of 255. The quantized 8 bit result is sent through Activation Unit, which allows only values above 10 and if it is below 10, it sets the value to 0.

The provided Input Matrices are :

$$\begin{Bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{Bmatrix} * \begin{Bmatrix} 4 & 0 & 2 & 1 \\ 4 & 3 & 2 & 0 \\ 4 & 3 & 0 & 1 \\ 4 & 3 & 2 & 1 \end{Bmatrix}$$

Figure 15: Feature (L) and Weight (R) Matrices

P11	P12	P13	P14
P21	P22	P23	P24
P31	P32	P33	P34
P41	P42	P43	P44

Figure 16: Product Matrix

iverilog Simulation:
Test bench process to load inputs:

1. Start clock at 100 MHz
2. Disable reset
3. Disable startSignal
4. Enable write_enable_W (Write enable for Weight Memory) and give inputs to Weight Memory through port_W.

5. Once the above step is completed, disable write_enable_W and enable write_enable_A. Then load feature memory elements through port_A. Once complete disable write_enable_A.

6. The matrices are now loaded into the memory units. startSignal is then enabled to kick-start matrix multiplication.

Commands to run in iverilog: To compile: *iverilog* : .
mainTB.v.
main.v.
Systolic_Array.v.
MACUnit.v.
QuantizationUnit.v.
ActivationUnit.v
To run the executable: vvp ./a.out

Result:

Clock Cycles:	Column 1	Column 2	Column 3	Column 4
Cycle 4	P11: A11* W11+ A12*W21+A13*W31+A14*W41			
Cycle 5	P21: A21* W11+ A22*W21+A23*W31+A24*W41	P12: A11* W12+ A12*W22+A13*W32+A14*W42		
Cycle 6	P31: A31* W11+ A32*W21+A33*W31+A34*W41	P22: A21* W12+ A22*W22+A23*W32+A24*W42	P13: A11*W13+A12*W24+A13*W33+A14*W43	
Cycle 7	P41: A41* W11+ A42*W21+A43*W31+A44*W41	P32: A31* W12+ A32*W22+A33*W32+A34*W42	P23: A21*W13+A22*W24+A23*W33+A24*W43	P14: A11*W14+A12*W24+A13*W34+A14*W44
Cycle 8		P42: A41* W12+ A42*W22+A43*W32+A44*W42	P33: A31*W13+A32*W24+A33*W33+A34*W43	P24: A21*W14+A22*W24+A23*W34+A24*W44
Cycle 9			P43: A41*W13+A42*W24+A43*W33+A44*W43	P34: A31*W14+A32*W24+A33*W34+A34*W44
Cycle 10				P44: A41*W14+A42*W24+A43*W34+A44*W44

Figure 17: Column Vs Clock Cycle Output

```
PS C:\iverilog\DE\Mid> iverilog .\mainTB.v .\memory_loader.v .\Systolic_Array.v .\MACUnit.v .\QuantizationUnit.v .\ActivationUnit.v
PS C:\iverilog\DE\Mid> vvp ./a.out
Time:                40000, startSignal:0

Feature Memory:
[0] =   1
[1] =   2
[2] =   3
[3] =   4
[4] =   1
[5] =   2
[6] =   3
[7] =   4
[8] =   1
[9] =   2
[10] =   3
[11] =   4
[12] =   1
[13] =   2
[14] =   3
[15] =   4

Weight Memory:
[0] =   4
[1] =   0
[2] =   2
[3] =   1
[4] =   4
[5] =   3
[6] =   2
[7] =   0
[8] =   4
[9] =   3
[10] =   0
[11] =   1
[12] =   4
[13] =   3
[14] =   2
[15] =   1
The feature Matrix Loaded into Feature Memory is:
   1   2   3   4
   1   2   3   4
   1   2   3   4
   1   2   3   4

The Weight Matrix loaded into Weight Memory is:
   4   0   2   1
   4   3   2   0
   4   3   0   1
   4   3   2   1

Time:                680000, startSignal:1

 Matrix Multiplication Output fetched from Feature Memory =
  40   27   14    0
  40   27   14    0
  40   27   14    0
  40   27   14    0

PS C:\iverilog\DE\Mid>
```

Figure 18: iverilog Simulation Result

Cadence Xcelium Timing Analysis Simulation Results:
1. Showing four column outputs across the 4 clock cycles of initialization:

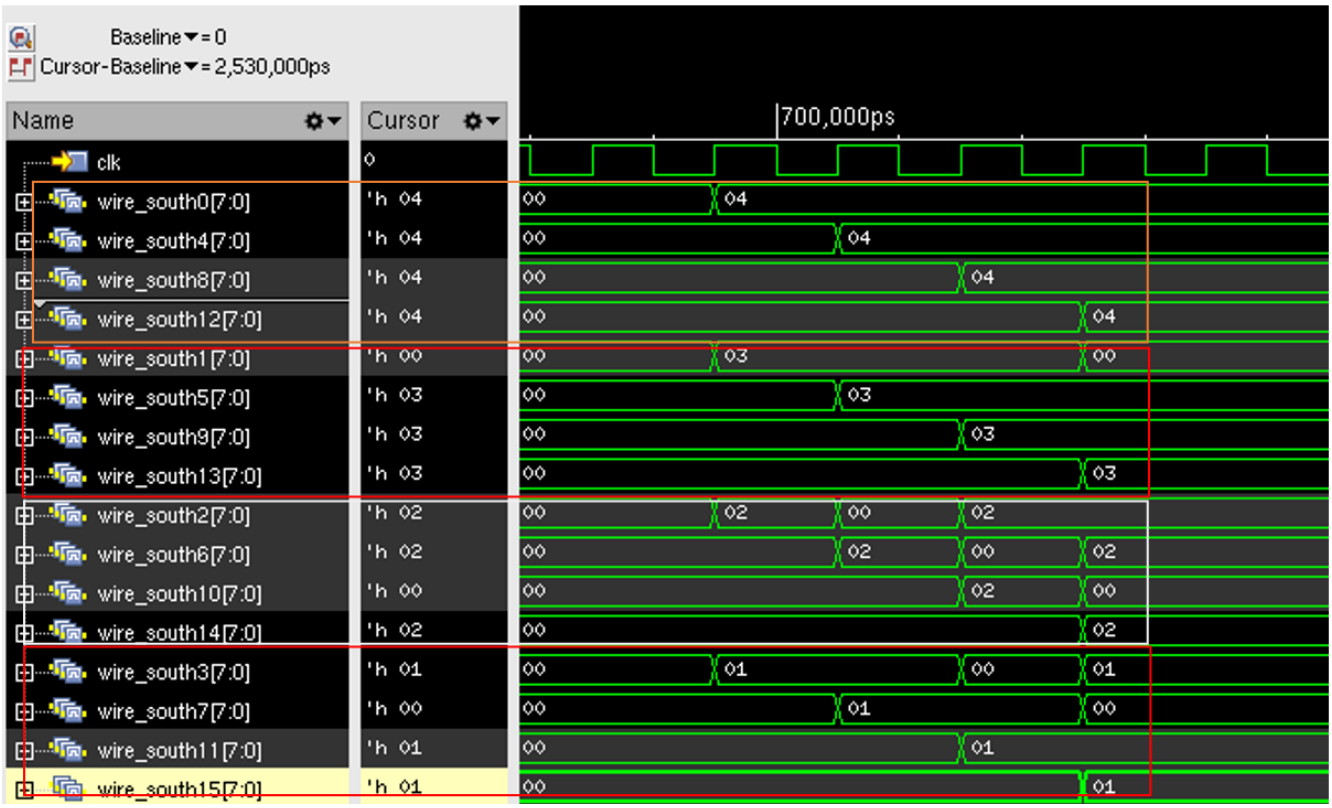

Figure 19: Weight Instatiation in 4 Cycles

2. Showing four column outputs across the 10 clock cycles of computation.

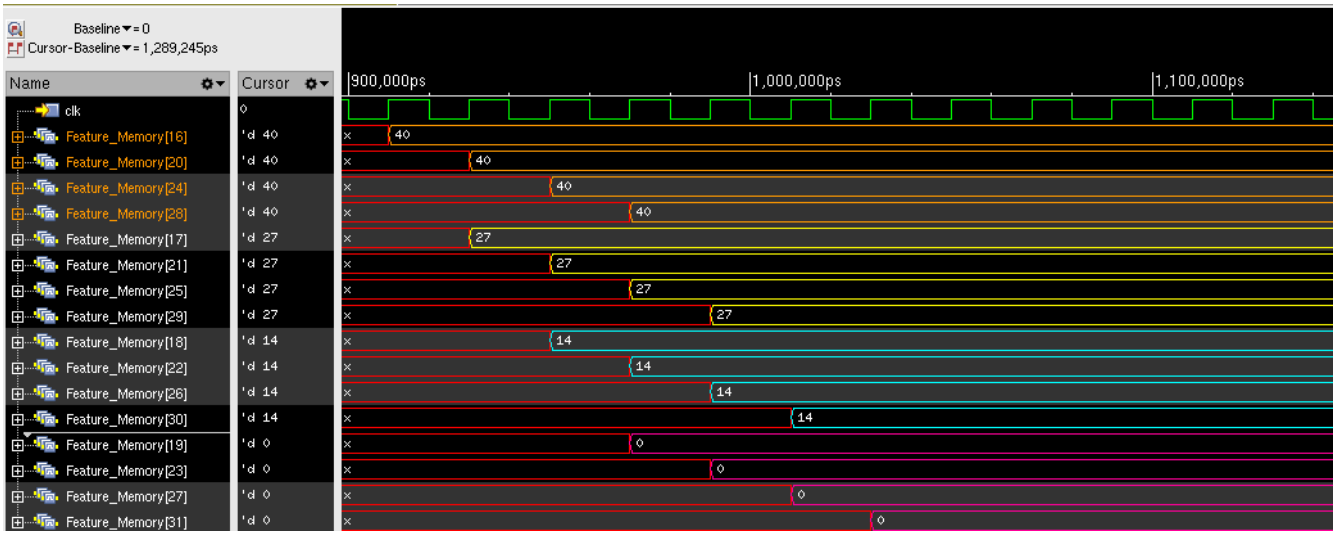

Figure 20: Column Wise 10 Cycle Waveform Orange: Column 1, Yellow: Column 2, Blue: Column 3, Pink: Column 4

3. Showing changes in memory (any 4 elements of the output matrix of your choice) that stores the output matrix, across the different clock cycles.

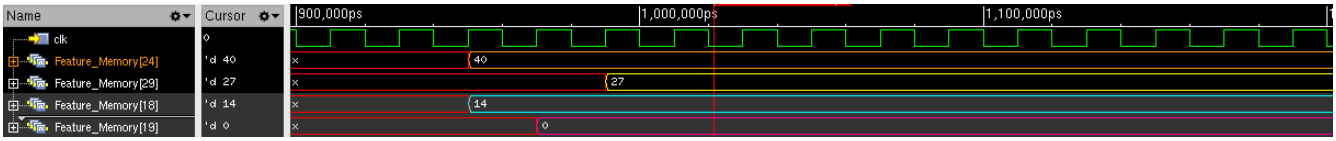

Figure 21: Value Changes in 4 elements of output matrix : P31, P42, P13, P14