Midterm Report

Name: Akhil Reddy Gujju

Objective:

The main goal of this project is to design a Implement the basic multiply-and-accumulate (MAC) unit and to implement a 4x4 systolic array suggested on the next slides as a synchronous system controlled by a single global clock "clk" along with an asynchronous reset "rst" (toy version of Google's Tensor Processing Unit i.e. TPU).

Introduction:

This project consists of two parts:

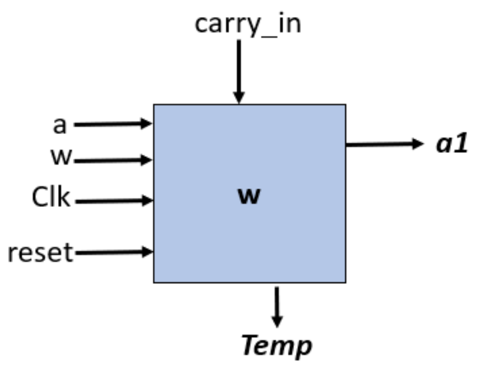
1) Mac unit design
2) 4X4 Systolic array TPU design

Where the mac unit is the basic cell in which the multiplication of the given two inputs 'a' and 'w' is done, the result is then propagated and gets added with a carry_in. Below is a brief description of the mac unit.

Mac unit:

Fig.1 depicts the actual mac unit's internal structure, where there are input ports and output ports:

Input ports:	output ports:
a (8bits)	a1(8bits)
w(8bits)	temp (24bits)
clock (1bit)	
reset(1bit)	
carry_in (24bits)	



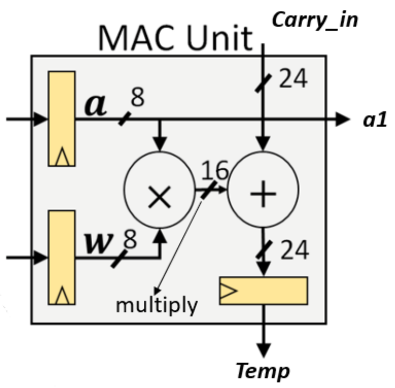


Fig.1 Depicts the Mac unit ports and internal structure of the mac

A register of 24 bits is introduced inside the design of the Mac unit, where it stores the value of multiplication and addition value of carry_in.

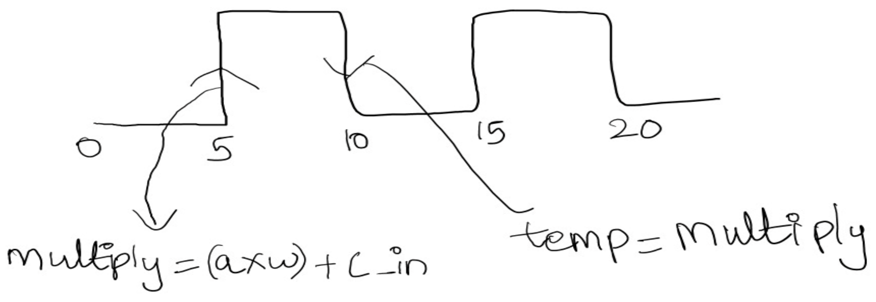


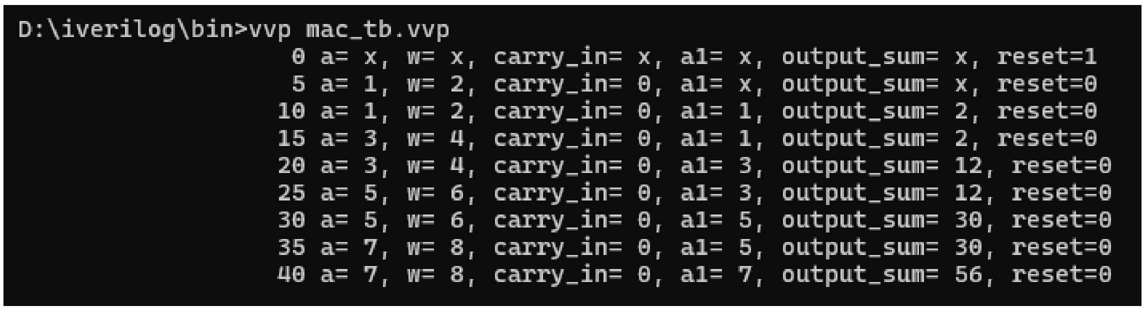
Fig.2 Depicts the functioning of the Mac with the clock

Fig.2 explains how the Mac produces the resultant product with carry-in, as we can see that,

1)For Posedge of the clock the multiplication and addition of carry-in is done (multiply is filled with value)

2) For Negedge of the clock the value of the computation done in posedge is copied in the variable Temp.

The results for the given question which are ran in iverilog are in the below Fig.3 and Fig.4:



```
D:\iverilog\bin>vvp mac_tb.vvp
                    0 a= x, w= x, carry_in= x, a1= x, output_sum= x, reset=1
                    5 a= 1, w= 2, carry_in= 0, a1= x, output_sum= x, reset=0
                   10 a= 1, w= 2, carry_in= 0, a1= 1, output_sum= 2, reset=0
                   15 a= 3, w= 4, carry_in= 0, a1= 1, output_sum= 2, reset=0
                   20 a= 3, w= 4, carry_in= 0, a1= 3, output_sum= 12, reset=0
                   25 a= 5, w= 6, carry_in= 0, a1= 3, output_sum= 12, reset=0
                   30 a= 5, w= 6, carry_in= 0, a1= 5, output_sum= 30, reset=0
                   35 a= 7, w= 8, carry_in= 0, a1= 5, output_sum= 30, reset=0
                   40 a= 7, w= 8, carry_in= 0, a1= 7, output_sum= 56, reset=0
```

Fig .3 Output result of mac unit

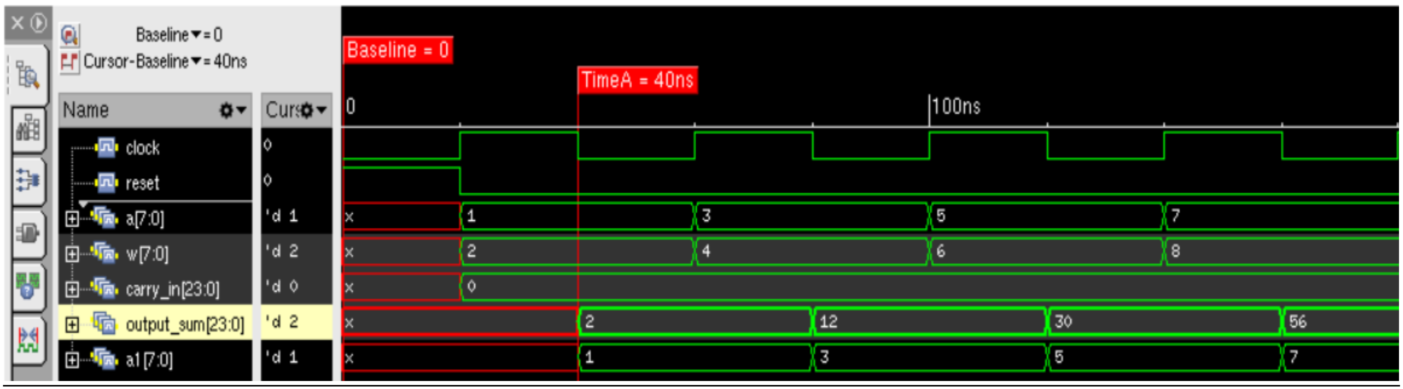


Fig.4 Output of mac unit waveform in Cadence Xcelium

Note:

The output results are updated in temp (output_sum) register where the values are stored at the “NEGEDGE” of the clock.

Systolic Array:

The Systolic array consists of 5 units:

1) 16 Mac units
2) Weight memory
3) Feature memory
4) Quantization unit
5) Activation Unit

16 Mac units:

The systolic array is implemented by using the mac units where there are connected to one another, the below diagram shows the design of systolic array in Fig.4,

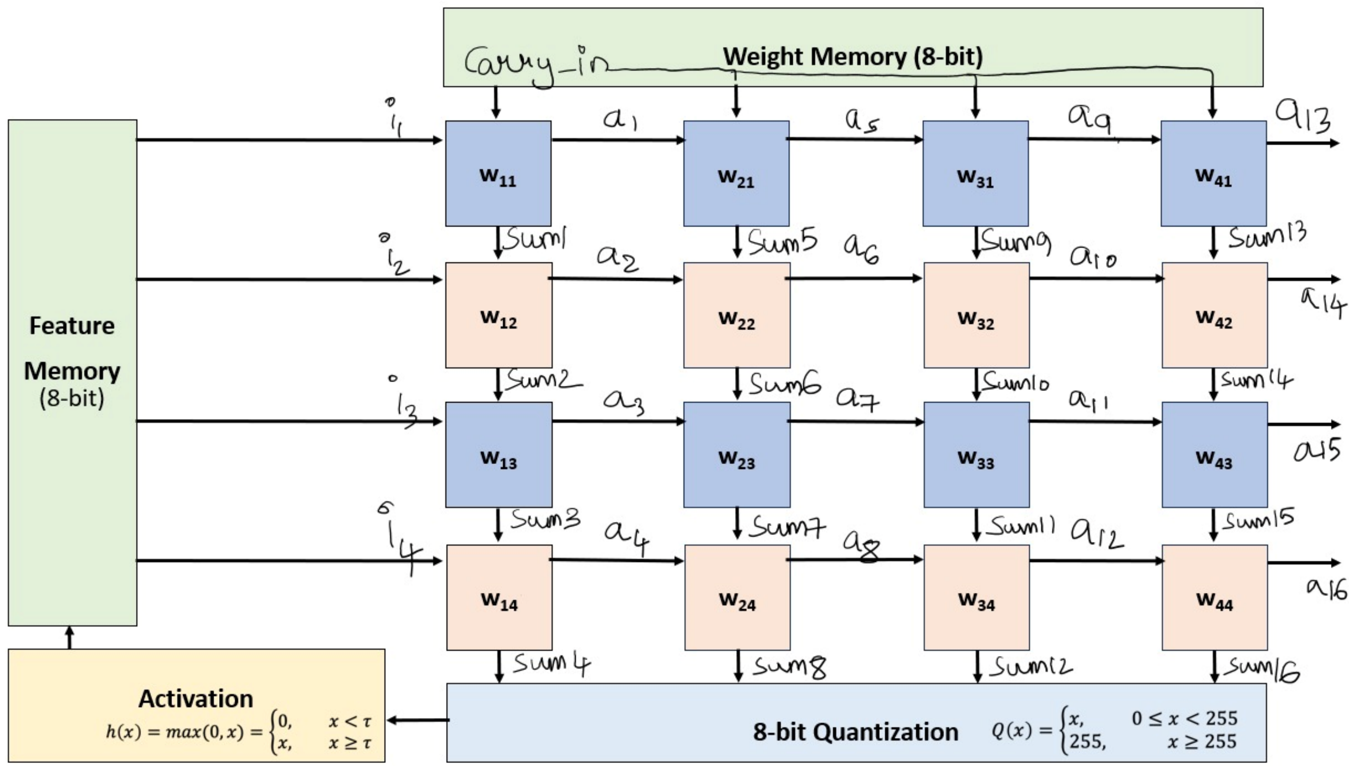


Fig 6 Design of the Systolic Array Unit

The above diagram shows the design of the systolic array units where there are:

Inputs i1, i2, i3, i4 are 4 elements of the A matrix coming clock by clock basis,

Output partial sums are sum1, sum2, sum3, …..sum16.

The propagated inputs are a1, a2, a3,…..a16, which go to the other Mac units which go on a clock basis.

The actual design overview in cadence Xcelium is shown below :

In the left side the Red block indicates the feature memory and the orange block indicates the weight memory of the design.

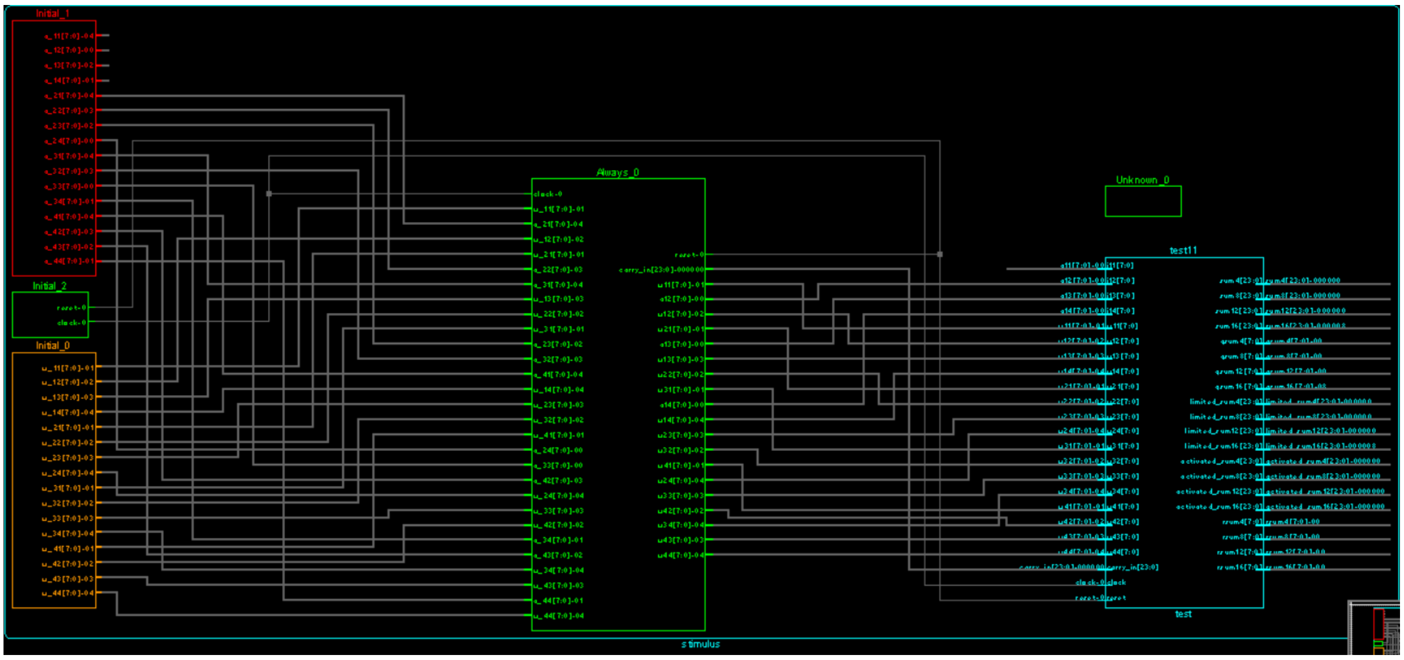


Fig 7 Overview of the design in Cadence Xcelium

Weight memory:

The weight memory is a 8 bits registers of 16 elements starting from w11, w12, w13, w14, …..w44.

```
 reg[7:0]  w_11,w_12,w_13, w_14,w_21,w_22,w_23, w_24,w_31,w_32,w_33, w_34,w_41, w_42,w_43,w_44;
          w_11<=1;w_12<=2;w_13<=3; w_14<=4;
          w_21<=1;w_22<=2;w_23<=3; w_24<=4;  // Weight memory
          w_31<=1;w_32<=2;w_33<=3; w_34<=4;
          w_41<=1; w_42<=2;w_43<=3;w_44<=4;
```

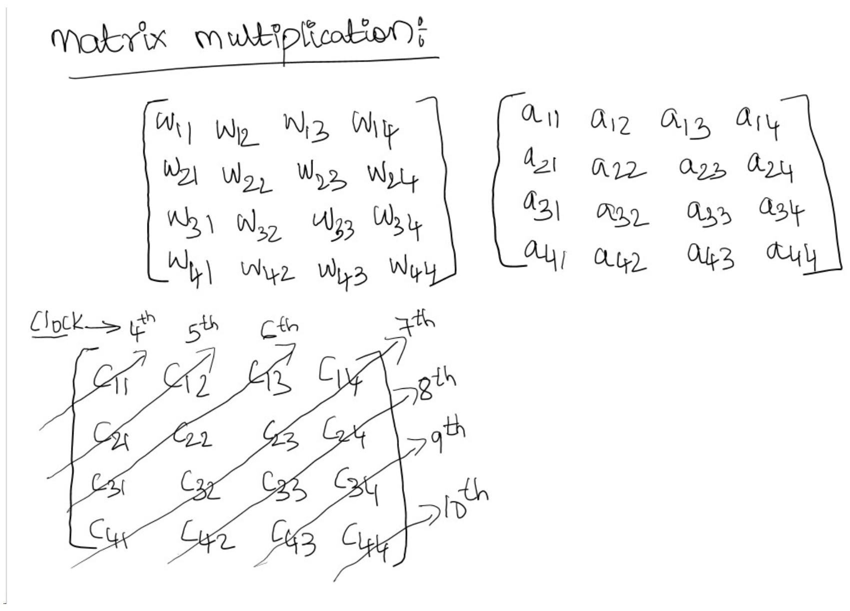
Feature memory:

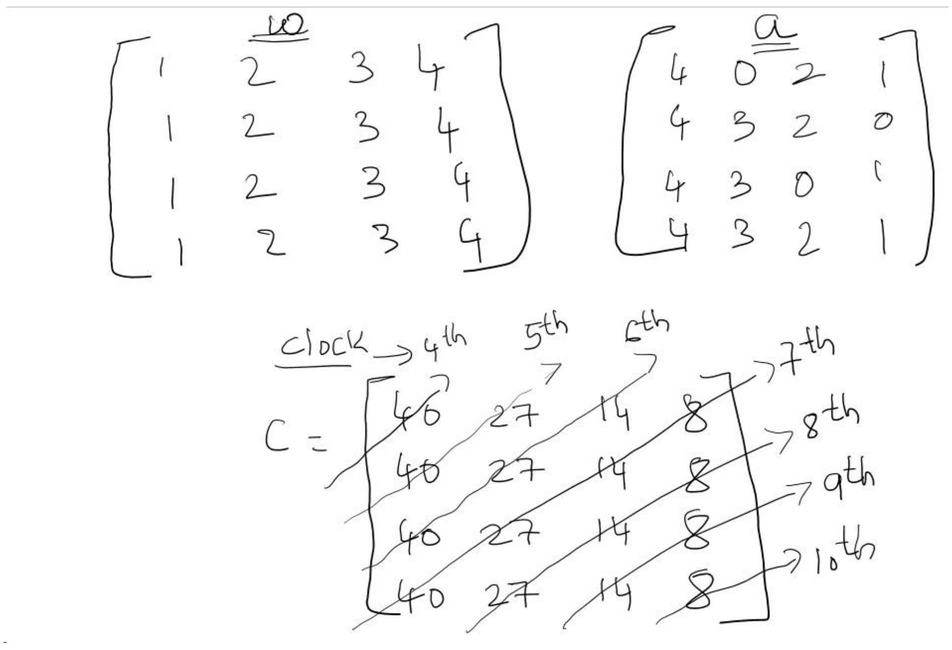
The feature memory is a 8 bits registers of 16 elements starting from a11, a12, a13, a14,….a44.

```
reg[7:0] a_11,a_12,a_13, a_14, a_21,a_22,a_23, a_24, a_31,a_32,a_33, a_34, a_41, a_42,a_43,a_44;

         a_11<=4;a_12<=0;a_13<=2; a_14<=1;
         a_21<=4;a_22<=3;a_23<=2; a_24<=0;// feature memory
         a_31<=4;a_32<=3;a_33<=0; a_34<=1;
         a_41<=4; a_42<=3;a_43<=2;a_44<=1;
```

Matrix multiplication:

Here the matrix multiplication is shown below with the A and W matrix being multiplied and the output matrix is C, which is filled on a clock basis. The below is the multiplication of the values and the output.





The output matrix gets filled by 10 clock cycles where the first element is filled for the 4th clock C11,

In 5th clock the next two element are filled C12 and C21, from the above diagram we can see how the elements are getting filled on clock basis.

Below is the table of the Computations of the 10 clock cycles and matrix filled are shown in detail with expressions:

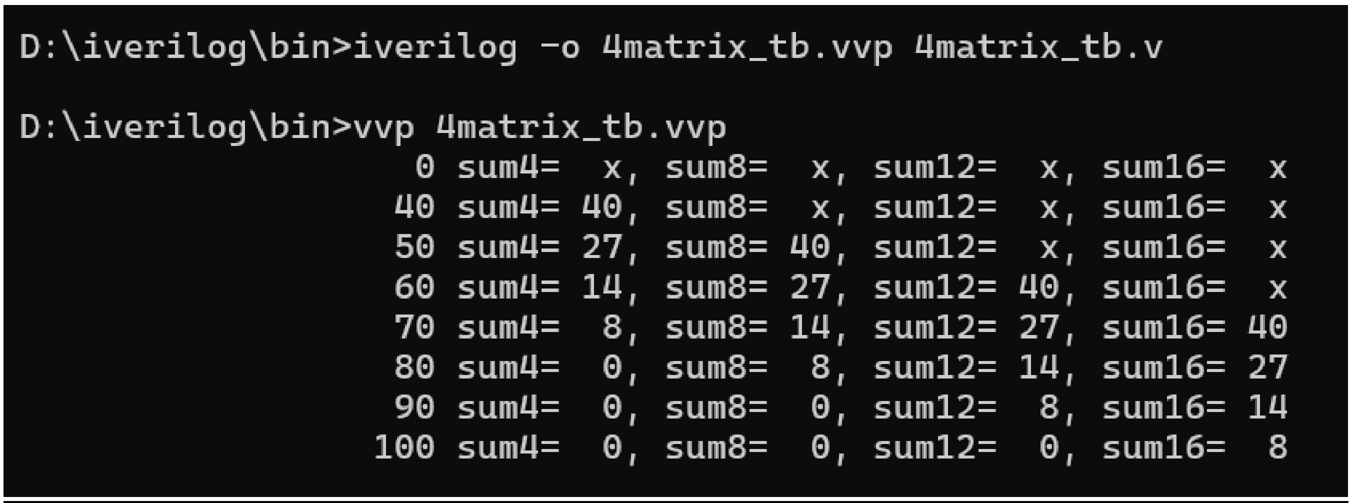
Clock	columns	Computation Equations	Matrix Elements
1	column1	(a11 *w11)	x
	column2	0	x
	column3	0	x
	column4	0	x
2	column1	(a11 w11) + (a21 w12)	x
	column2	(a11 w21)	x
	column3	0	x
	column4	0	x
3	column1	(a11 w11) + (a21 w12) + (a31 w13)	x
	column2	(a11 w21) + (a12 w22)	x
	column3	(a11 w31)	x
	column4	0	x
4	column1	(a11 w11) + (a21 w12) + (a31 w13) + (a41 w14)	c11
	column2	(a11 w21) + (a12 w22) + (a13 w23)	x
	column3	(a11 w31) + (a21 w32)	x
	column4	(a11 w41)	x
5	column1	(a11 w21) + (a21 w22) + (a31 w23) + (a41 w24)	c12
	column2	(a12 w11) + (a22 w12) + (a32 w13) + (a42 w14)	c21
	column3	(a11 w31) + (a21 w32) + (a31 w33)	x
	column4	(a11 w41) + (a21 w42)	x
6	column1	(a11 w31)+ (a21 w32) + (a31 w33) + (a41 w34)	c13
	column2	(a12 w21)+ (a22 w22) + (a32 w23) + (a42 w24)	c22
	column3	(a13 w11)+ (a23 w12) + (a33 w13) + (a43 w14)	c31
	column4	(a11 w41)+ (a21 w42) + (a31 w43)	x
7	column1	(a11 w41)+ (a21 w42) + (a31 w43) + (a41 w44)	c14
	column2	(a12 w31)+ (a22 w32) + (a32 w33) + (a42 w34)	c23
	column3	(a13 w21)+ (a23 w22) + (a33 w23) + (a43 w24)	c32
	column4	(a14 w11)+ (a24 w12) + (a34 w13) + (a44 w14)	c41
8	column1	0	x
	column2	(a12 w41)+ (a22 w42) + (a32 w43) + (a42 w44)	c24
	column3	(a13 w31)+ (a23 w32) + (a33 w33) + (a43 w34)	c33
	column4	(a14 w21)+ (a24 w22) + (a34 w23) + (a44 w24)	c42
9	column1	0	x
	column2	0	x
	column3	(a13 w41)+ (a23 w42) + (a33 w43) + (a43 w44)	c34
	column4	(a14 w31)+ (a24 w32) + (a34 w33) + (a44 w34)	c43
10	column1	0	x
	column2	0	x
	column3	0	x
	column4	(a14 w41)+ (a24 w42) + (a34 w43) + (a44 w44)	c44

Results:

Here the output matrix is shown on different clock cycles starting from 40th sec which is the 4th cycle,

And completing the matrix in column wise at 100th sec which the 10th cycle.

Here the outputs sum4 is row1, sum8 is row2, sum12 is row3, sum16 is row4 of the output matrix.



```
D:\iverilog\bin>iverilog -o 4matrix_tb.vvp 4matrix_tb.v

D:\iverilog\bin>vvp 4matrix_tb.vvp
                   0 sum4=  x, sum8=  x, sum12=  x, sum16=  x
                  40 sum4= 40, sum8=  x, sum12=  x, sum16=  x
                  50 sum4= 27, sum8= 40, sum12=  x, sum16=  x
                  60 sum4= 14, sum8= 27, sum12= 40, sum16=  x
                  70 sum4=  8, sum8= 14, sum12= 27, sum16= 40
                  80 sum4=  0, sum8=  8, sum12= 14, sum16= 27
                  90 sum4=  0, sum8=  0, sum12=  8, sum16= 14
                 100 sum4=  0, sum8=  0, sum12=  0, sum16=  8
```

Quantization unit:

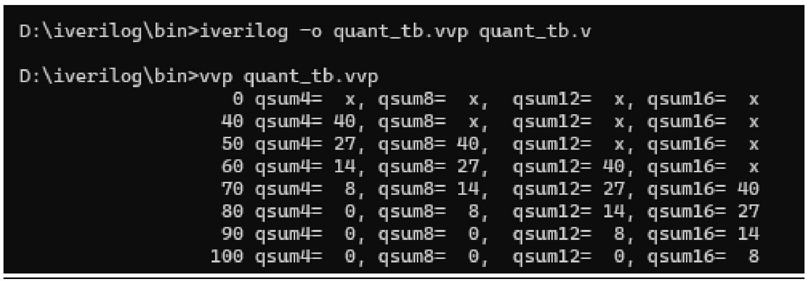
The quantization unit of registers where receives the elements from the output matrix of 24bits each then, it compares if the element is greater than 255 if yes it sets the value as 255 which 8bits (11111111) in binary form.

The set value is then truncated from 24-bit to 8-bit.

Below is the logic implementation of the quantization unit:

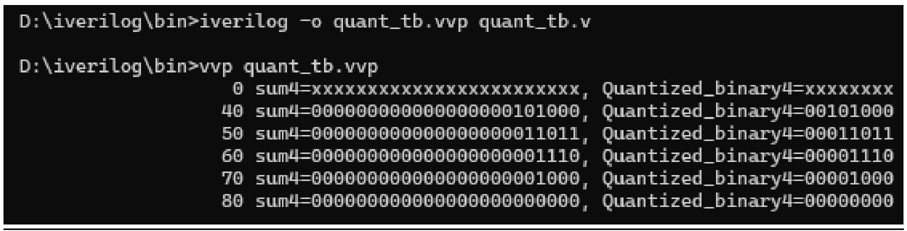
```
// quantization unit

assign limited_sum4 = (sum4 > 255) ? 24'd255 : sum4;
assign limited_sum8 = (sum8 > 255) ? 24'd255 : sum8;
assign limited_sum12 = (sum12 > 255) ? 24'd255 : sum12;
assign limited_sum16 = (sum16 > 255) ? 24'd255 : sum16;


 // Quantize the limited values

 assign qsum4 = limited_sum4[7:0];
 assign qsum8 = limited_sum8[7:0];
 assign qsum12 = limited_sum12[7:0];
 assign qsum16 = limited_sum16[7:0];
```

Result:



```
D:\iverilog\bin>iverilog -o quant_tb.vvp quant_tb.v

D:\iverilog\bin>vvp quant_tb.vvp
                   0 qsum4=  x, qsum8=  x,  qsum12=  x, qsum16=  x
                  40 qsum4= 40, qsum8=  x,  qsum12=  x, qsum16=  x
                  50 qsum4= 27, qsum8= 40,  qsum12=  x, qsum16=  x
                  60 qsum4= 14, qsum8= 27,  qsum12= 40, qsum16=  x
                  70 qsum4=  8, qsum8= 14,  qsum12= 27, qsum16= 40
                  80 qsum4=  0, qsum8=  8,  qsum12= 14, qsum16= 27
                  90 qsum4=  0, qsum8=  0,  qsum12=  8, qsum16= 14
                 100 qsum4=  0, qsum8=  0,  qsum12=  0, qsum16=  8
```

Truncation from 24bit to 8bit of qsum4 which 40 in decimal, below shows the binary form of it:



```
D:\iverilog\bin>iverilog -o quant_tb.vvp quant_tb.v

D:\iverilog\bin>vvp quant_tb.vvp
                   0 sum4=xxxxxxxxxxxxxxxxxxxxxxxx, Quantized_binary4=xxxxxxxx
                  40 sum4=000000000000000000101000, Quantized_binary4=00101000
                  50 sum4=000000000000000000011011, Quantized_binary4=00011011
                  60 sum4=000000000000000000001110, Quantized_binary4=00001110
                  70 sum4=000000000000000000001000, Quantized_binary4=00001000
                  80 sum4=000000000000000000000000, Quantized_binary4=00000000
```

Activation Unit:

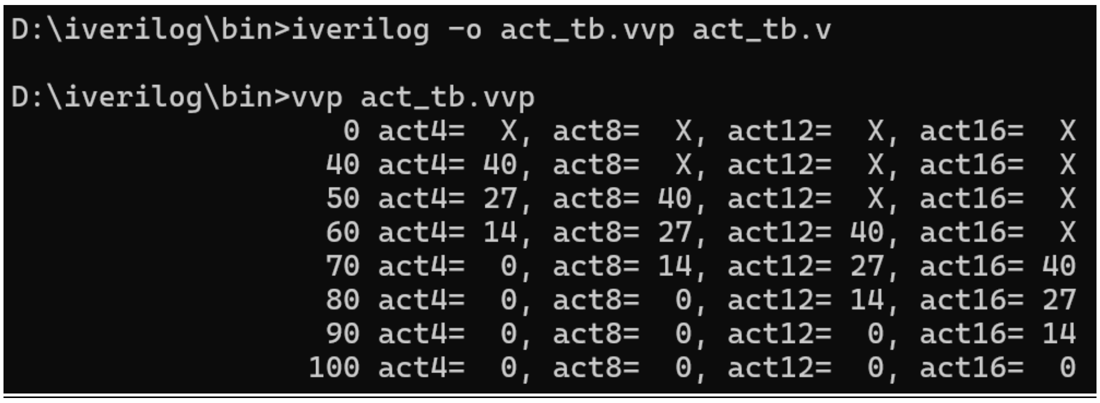
The activation unit is an 8-bit registers where it receives the values from the quantization unit, from here there is a value called Tau, which is used to compare if the value is then 10, the value will change to 0. If the element is greater than 10 it remains the same value.

Below is the logic implementation of the activation :

```
// **Activation Unit**
wire lt_tau4, lt_tau8, lt_tau12, lt_tau16; // signals for comparison with tau (assuming tau = 10)
wire [23:0] activated_sum4, activated_sum8, activated_sum12, activated_sum16;

assign lt_tau4 = qsum4 < 10;
assign activated_sum4 = lt_tau4 ? 0 : qsum4;

assign lt_tau8 = qsum8 < 10;
assign activated_sum8 = lt_tau8 ? 0 : qsum8;

assign lt_tau12 = qsum12 < 10;
assign activated_sum12 = lt_tau12 ? 0 :qsum12;

assign lt_tau16 = qsum16 < 10;
assign activated_sum16 = lt_tau16 ? 0 : qsum16;

// **Update final outputs with activated values**
assign rsum4 = activated_sum4;
assign rsum8 = activated_sum8;
assign rsum12 = activated_sum12;
assign rsum16 = activated_sum16;
```

Result:

8 vanishes because it is less than 10.

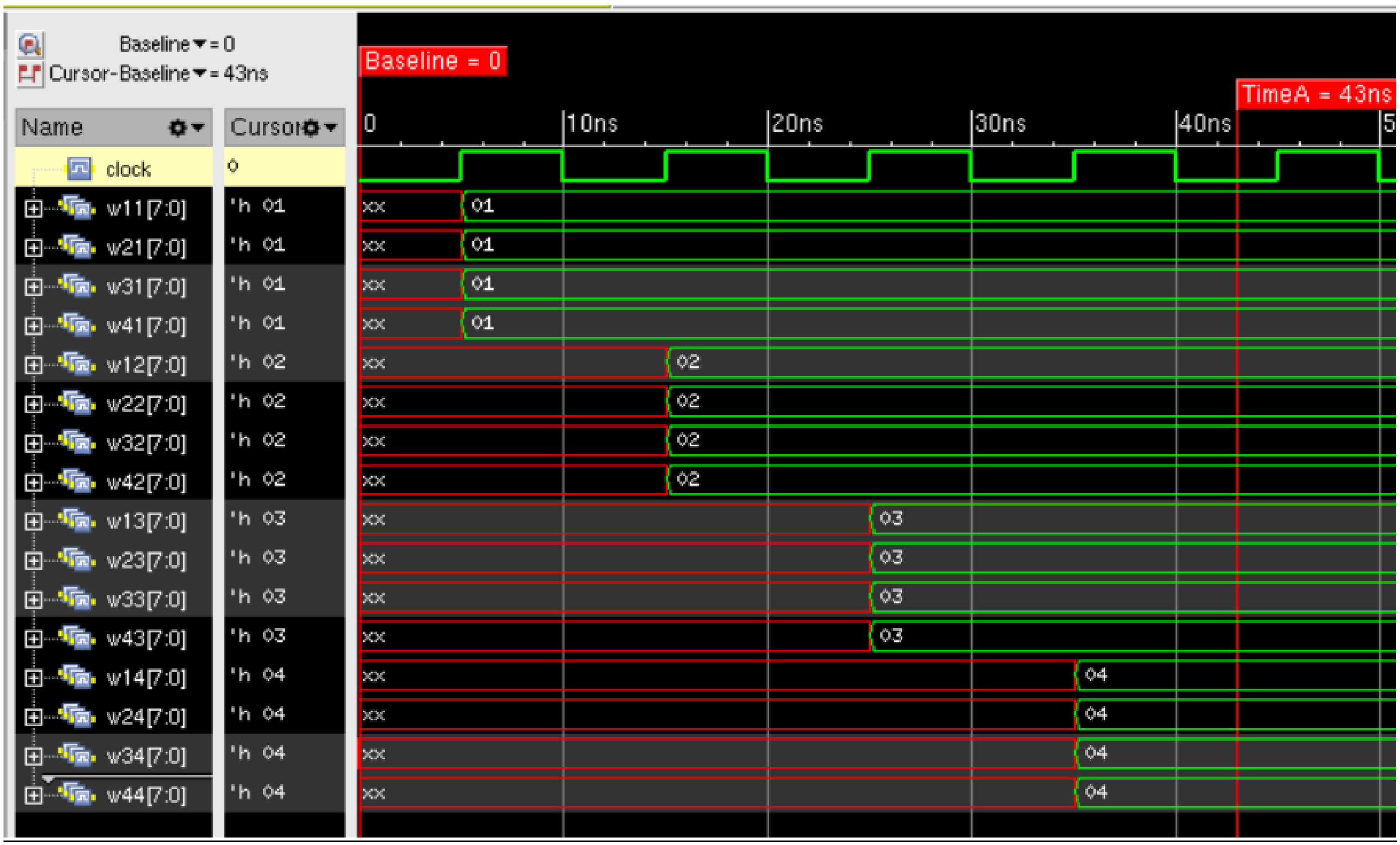


```
D:\iverilog\bin>iverilog -o act_tb.vvp act_tb.v

D:\iverilog\bin>vvp act_tb.vvp
                   0 act4=  X, act8=  X, act12=  X, act16=  X
                  40 act4= 40, act8=  X, act12=  X, act16=  X
                  50 act4= 27, act8= 40, act12=  X, act16=  X
                  60 act4= 14, act8= 27, act12= 40, act16=  X
                  70 act4=  0, act8= 14, act12= 27, act16= 40
                  80 act4=  0, act8=  0, act12= 14, act16= 27
                  90 act4=  0, act8=  0, act12=  0, act16= 14
                 100 act4=  0, act8=  0, act12=  0, act16=  0
```

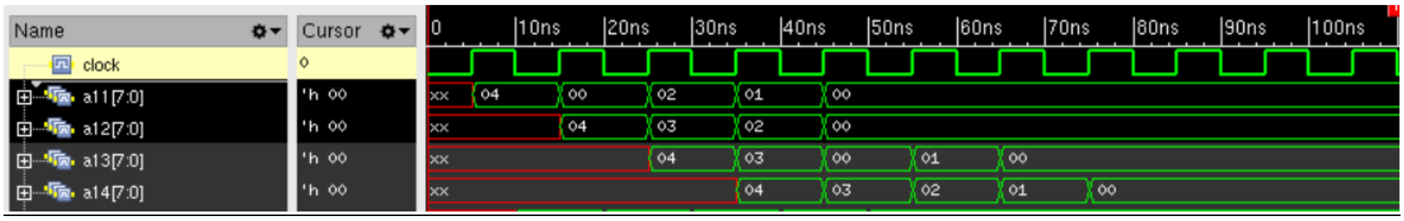
Cadence Xcelium:

Showing four column outputs across the 4 clock cycles of initialization:

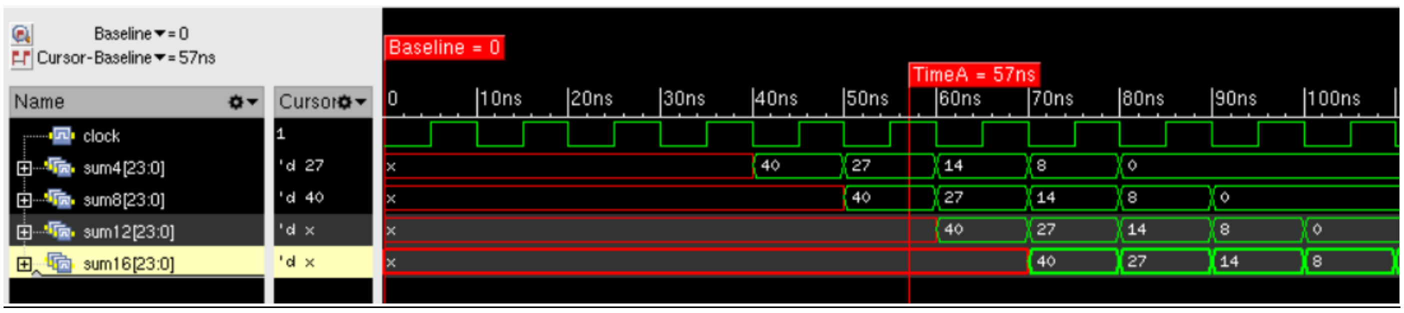
The initialization of weighted memory:



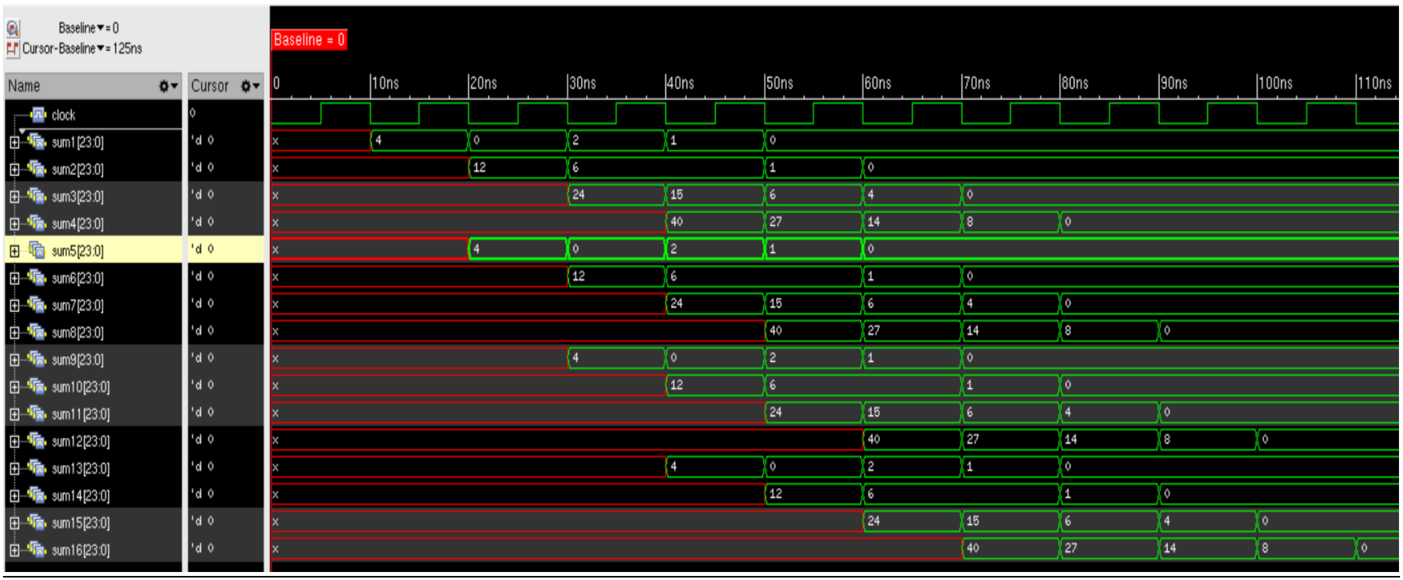
The initialization of Feature memory:



Showing four column outputs across the 10 clock cycles of computation:



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


Baseline ▾ = 0
Cursor-Baseline ▾ = 125ns
Baseline = 0
Name
Cursor
0 10ns 20ns 30ns 40ns 50ns 60ns 70ns 80ns 90ns 100ns 110ns
clock 0
sum1[23:0] 'd 0 x 4 0 2 1 0
sum2[23:0] 'd 0 x 12 6 1 0
sum3[23:0] 'd 0 x 24 15 6 4 0
sum4[23:0] 'd 0 x 40 27 14 8 0
sum5[23:0] 'd 0 x 4 0 2 1 0
sum6[23:0] 'd 0 x 12 6 1 0
sum7[23:0] 'd 0 x 24 15 6 4 0
sum8[23:0] 'd 0 x 40 27 14 8 0
sum9[23:0] 'd 0 x 4 0 2 1 0
sum10[23:0] 'd 0 x 12 6 1 0
sum11[23:0] 'd 0 x 24 15 6 4 0
sum12[23:0] 'd 0 x 40 27 14 8 0
sum13[23:0] 'd 0 x 4 0 2 1 0
sum14[23:0] 'd 0 x 12 6 1 0
sum15[23:0] 'd 0 x 24 15 6 4 0
sum16[23:0] 'd 0 x 40 27 14 8 0