Digital ASIC Design : Midterm Exam

Name: Skanda Krishnan Balasubramanian

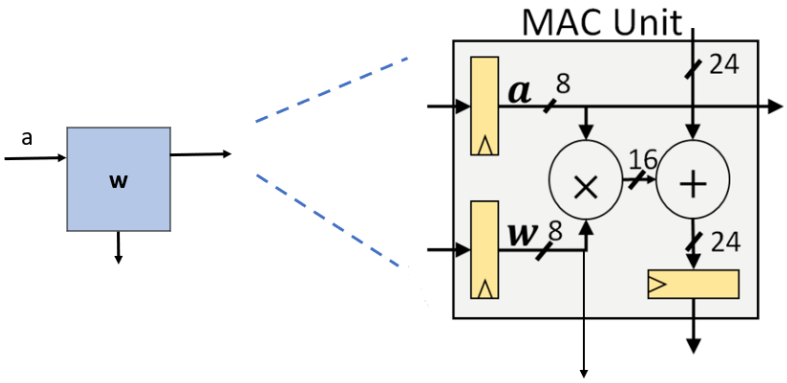
Designing a Simplified version of Google's Tensor Processing Unit.

Abstract

The project described below is a simplified version of Googles Tensor Processing Unit. The goal of this project is to design this unit using sub units such as the multiply and accumulate, quantization and activation units. After which iverilog and Cadence Virtuoso is used to functionally verify the design.

Question 1 : Designing the MAC Unit

The block diagram of the MAC unit is given below:



There are mainly 2 input Data and 3 output Data: The input data is the Ain and Win and the output is the Aout, Wout and the Pout. The Aout is passed to the next MAC in the same row while Wout and Pout are passed to the next MAC in the same column. Additional data valid signal are also given to maintain the context of the incoming and outgoing data. Pin and Pout are 24 bits wide while Ain,Aout,Win and Wout are 8 bits wide

```
always @(posedge clk or posedge rst) begin
        if(rst) begin
                aout <= 8'd0;
                dv_aout <= 1'b0;
        end
        else if(dv_ain)begin
                aout <= ain;
                dv_aout <= 1'b1;
        end
        else begin
                aout <= 8'd0;
                dv_aout <= 1'b0;
        end
end
```

flipflop defining aout

```
always @(posedge clk or posedge rst) begin
        if(rst)begin
                wout <= 8'd0;
                dv_wout <= 1'b0;
        end
        else if(init_win)begin
                wout <= win;
                dv_wout <= dv_win;
        end
```

flipflop defining wout

```
always @(posedge clk or posedge rst)begin
//      if(dv_aout && dv_wout && dv_pin)begin
        if(dv_aout && dv_wout)begin
        pout <= {8'd0,prod} + pin;
        dv_pout <= 1'b1;
        end
        else begin
                pout <= 24'd0;
                dv_pout <= 1'b0;
        end
end
```

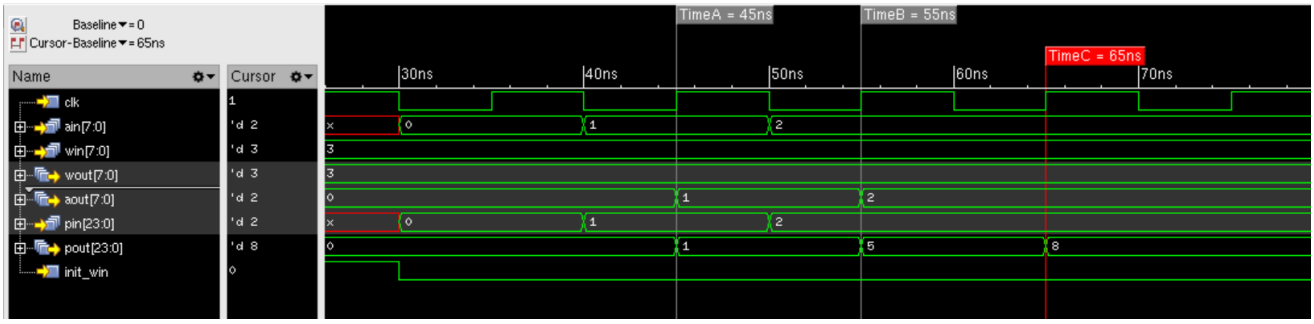
flipflop defining pout

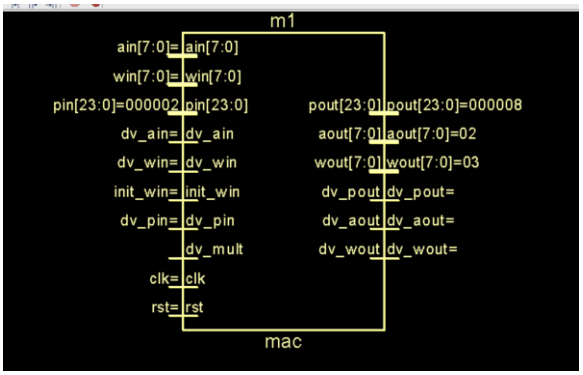
Iverilog testing:

```
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ iverilog tb_mac.v mac.v
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ vvp a.out
                1000
 win =                  3
 ain =                  2
 pin =                  2
 pout = (w*a)+p =       8

VCD info: dumpfile wave_mac.vcd opened for output.
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$
```

pout = (3*2) + 2 = 8

Cadence Waveform:





Question 2: Quantization Unit

The quantization unit is defined as an upper bounding unit where in the products are upper bounded from 24 bits to 8 bits. The way we do this if the 24 bit product is greater than 255 then the output of the quantization unit is 255. If it is less than 255 then the output will be the value itself. The quantization unit described below is handling 4 24 bit inputs and giving 4 8-bit outputs after quantization. No clock is used and computation is done combinatorially. Additional data valid pins are used to maintain the context and validity of data.

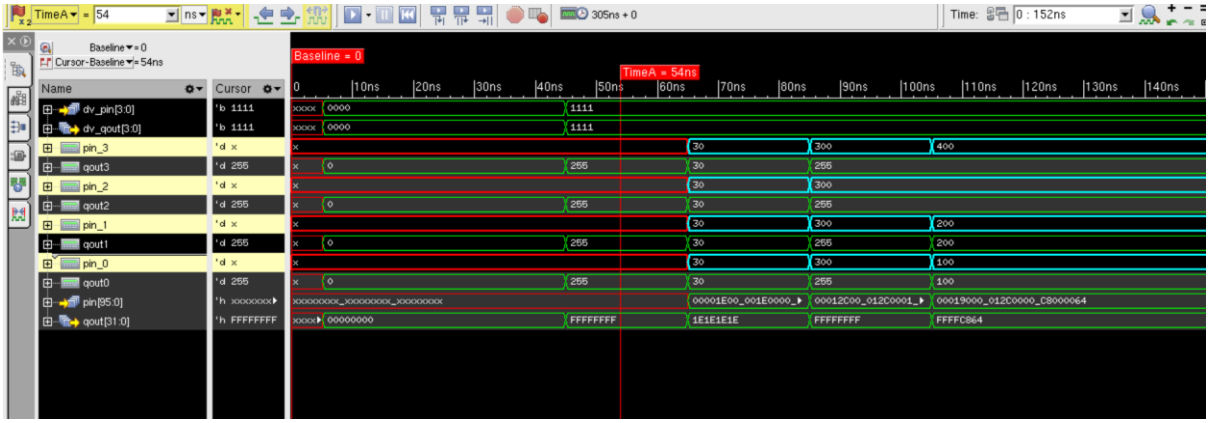
```
genvar i;
generate
        for(i =0;i<4;i=i+1) begin
                always @(*) begin
                        if (rst) begin
                                qout[(i*8)+7:(i*8)] <= 8'd0;
                                dv_qout[i] <= 1'b0;
                        end
                        else if(dv_pin[i]) begin
                                if(pin[(i*24)+23:(i*24)] <=24'd255)
                                        qout[(i*8)+7:(i*8)] <= pin[(i*24)+7:(i*24)];
                                else
                                        qout[(i*8)+7:(i*8)] <= 8'd255;
                                dv_qout[i] <= dv_pin[i];
                        end
                        else begin
                                qout[(i*8)+7:(i*8)] <= 8'd0;
                                dv_qout[i] <= 1'b0;
                        end
                end
        end
```

Iverilog Test :

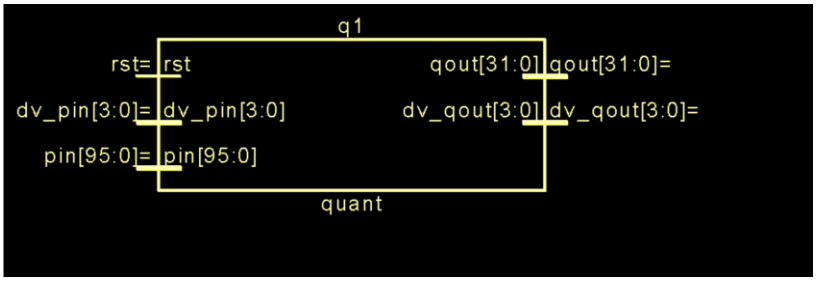
```
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ iverilog tb_quant.v quant.v
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ vvp a.out
VCD info: dumpfile wave_quant.vcd opened for output.
                  65
pin0 =       30, qout0 =  30
pin1 =       30, qout1 =  30
pin2 =       30, qout2 =  30
pin3 =       30, qout3 =  30
                  85
pin0 =      300, qout0 = 255
pin1 =      300, qout1 = 255
pin2 =      300, qout2 = 255
pin3 =      300, qout3 = 255
                 105
pin0 =      100, qout0 = 100
pin1 =      200, qout1 = 200
pin2 =      300, qout2 = 255
pin3 =      400, qout3 = 255
```

anything above 255 is locked to 255.

Cadence Test and Waveform



Blue traces are the inputs and the green traces are the outputs.

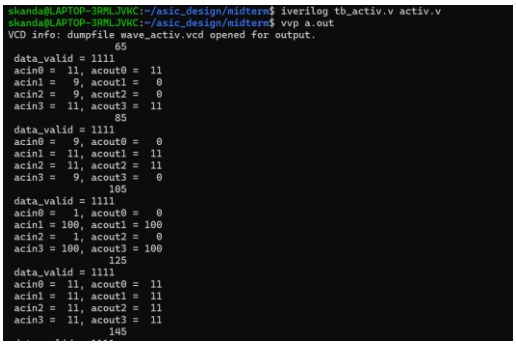


Question 2: Activation Unit

The quantization unit is defined as a lower bounding unit where in the quantized outputs are lower bounded decimal value 10. That is, if the input data is less than 10 then the output of the activation unit would be 0. If the input is greater than 10 then the output would be the same as the input. The activation unit described below is handling 4 8 bit inputs and slotting them into 127 bit bus according to the data valid. An additional state machine is inside the activation unit to slot the output products into the correct address of the feature memory.

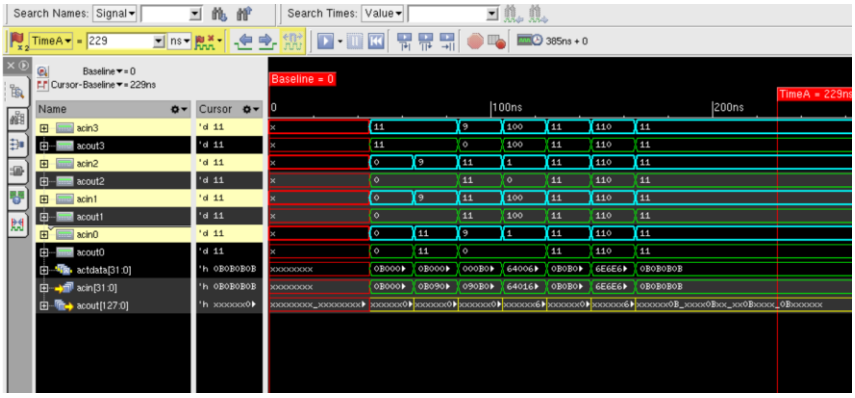
```
genvar l;
generate
        for(l =0;l <4;l=l+1)begin
                always @(*)begin
                        if(acin [(l*8)+7:l*8] > 8'd10)
                                actdata [(l*8)+7:l*8] <= acin [(l*8)+7:l*8];
                        else

                                actdata [(l*8)+7:l*8] <= 8'd0;
                end
        end
endgenerate
```

Iverilog Test :



anything below 10 is locked to 0.

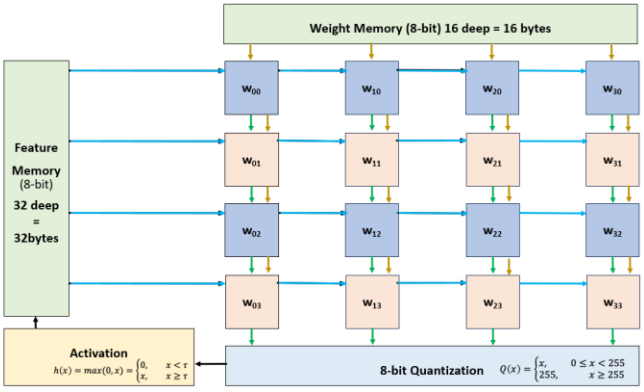
Cadence Test and Waveform



Blue Traces are in inputs while the green traces are outputs

Question 2: Systolic Array

In this section we are going to define a 4x4 Systolic array using the MAC unit described in question 1. A 4x4 systolic array contains 16 MAC units connected as shown below. The blue lines represent the data from the A matrix. Yellow lines represent the data from the Weight matrix and the Green line represent the Product output of each MAC. As shown below the Weight and Products are connected to the next MAC in the same column while the A data lines are connected to the next MAC in the same row.



The systolic array is first initialized with the weigh matrix using 4 cycles and then the A matrix is passed on as shown below. The product output of each MAC in the Last row will contain the required output of the matrix multiplication. The validity of the output product is maintained by a data valid signal that travels parallel to it. The whole process takes about 14 clock cycles. This includes the weight loading, A loading and product output flushing. It is described as shown below:

	Clk #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Ports	Weight Loading (4 cycles)														
i/p	mac_wdata[0]	W03	W02	W01	W00											
i/p	mac_wdata[5]	W13	W12	W11	W10											
i/p	mac_wdata[10]	W23	W22	W21	W20											
i/p	mac_wdata[15]	W33	W32	W31	W30											
						A Matrix Loading (7 cycles)										
i/p	mac_adata[0]					A00	A01	A02	A03							
i/p	mac_adata[1]						A10	A11	A12	A13						
i/p	mac_adata[2]							A20	A21	A22	A23					
i/p	mac_adata[3]								A30	A31	A32	A33				
										Output Matrix flushing (7 Cycles)						
o/p	mac_pdata[16]									P00	P01	P02	P03			
o/p	mac_pdata[17]										P10	P11	P12	P13		
o/p	mac_pdata[18]											P20	P21	P22	P23	
o/p	mac_pdata[19]												P30	P31	P32	P33

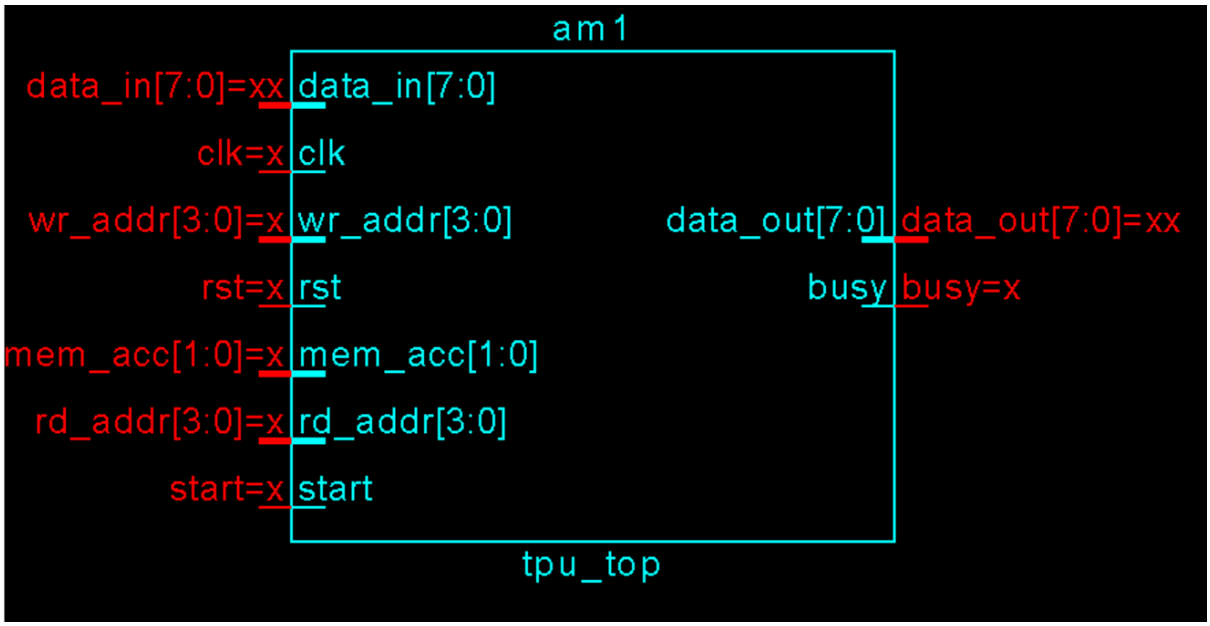
The output of each column would have the below calculations within them:

	Column 1 - Output	Column 2 - Output	Column 3 - Output	Column 4 - Output
Clk #4	W00*A00 + W01*A10 +W02*A20 + W03*A30			
Clk #5	W00*A01 + W01*A11 +W02*A21 + W03*A31	W10*A00 + W11*A10 +W12*A20 + W13*A30		
Clk #6	W00*A02 + W01*A12 +W02*A22 + W03*A32	W10*A01 + W11*A11 +W12*A21 + W13*A31	W20*A00 + W21*A10 +W22*A20 + W23*A30	
Clk #7	W00*A03 + W01*A13 +W02*A23 + W03*A33	W10*A02 + W11*A12 +W12*A22 + W13*A32	W20*A01 + W21*A11 +W22*A21 + W23*A31	W30*A00 + W31*A10 +W32*A20 + W33*A30
Clk #8		W10*A03 + W11*A13 +W13*A23 + W13*A33	W20*A02 + W21*A12 +W22*A22 + W23*A32	W30*A01 + W31*A11 +W32*A21 + W33*A31
Clk #9			W20*A03 + W21*A13 +W22*A23 + W23*A33	W30*A02 + W31*A12 +W32*A22 + W33*A32
Clk #10				W30*A03 + W31*A13 +W32*A23 + W33*A33

```
genvar m,j;
generate
        for(m=0;m<4;m=m+1)begin
                for(j=0;j<4;j=j+1)begin
                  mac mc(
                          .pout(mac_pdata[(j*5)+m+1]),
                          .dv_pout(mac_pdv[(j*5)+m+1]),
                          .aout(mac_adata[(m*5)+j+1]),
                          .dv_aout(mac_adv[(m*5)+j+1]),
                          .wout(mac_wdata[(j*5)+m+1]),
                          .dv_wout(mac_wdv[(j*5)+m+1]),
                          .ain(mac_adata[(m*5)+j]),
                          .dv_ain(mac_adv[(m*5)+j]),
                          .win(mac_wdata[(j*5)+m]),
                          .dv_win(mac_wdv[(j*5)+m]),
                          .init_win(init_win),
                          .pin(mac_pdata[(j*5)+m]),
                          .dv_pin(mac_pdv[(j*5)+m]),
                          .dv_mult(dv_mult),
                          .clk(clk),
                          .rst(rst)
                  );
                end
//              assign mac_pdata [m*5]= 24'd0;
```

The systolic array is defined by this generate statement. The Connections across the rows (A matrix) are handled by the variable “j” while the connections across the column (weight and product) are handled by the variable “m”.

Question 3 Testing the TPU using iverilog and cadence



The TPU consists of the systolic array, activation unit and the quantization unit. Additionally there is a 32 Byte RTL memory called feature memory and another 16 byte rtl memory called weight memory. The test bench would load the weight matrix and the A matrix using the data_in bus and the wr_addr bus. The test bench should make sure to use mem_acc = 2'b01 for accessing the A matrix and mem_acc = 2'b10 for accessing the weight matrix. After this the test bench is expected to give a 1'b1 for one clock cycle in the start input port of the tpu_top. This would tell the tpu_top that the 2 matrix's are loaded into the feature and weight memory and to start the multiplication process. When the tpu_top start the weight loading the busy signal will go high. It will only go down one the products are written into the upper 16 bytes of the feature memory. The busy signal going down is an indication to the testbench that the multiplication is complete and output matrix is ready to be read out. The rd_addr and data_out ports are used for this process. Further more the mem_acc = 2'b11 to access the feature memory.

Weight and A matrix in the test bench :

```
initial begin
        a[0]  = 8'd4; a[1]  = 8'd0; a[2]  = 8'd2; a[3]  = 8'd1;
        a[4]  = 8'd4; a[5]  = 8'd3; a[6]  = 8'd2; a[7]  = 8'd0;
        a[8]  = 8'd4; a[9]  = 8'd3; a[10] = 8'd0; a[11] = 8'd1;
        a[12] = 8'd4; a[13] = 8'd3; a[14] = 8'd2; a[15] = 8'd1;

        w[0]  = 8'd1; w[1]  = 8'd2; w[2]  = 8'd3; w[3]  = 8'd4;
        w[4]  = 8'd1; w[5]  = 8'd2; w[6]  = 8'd3; w[7]  = 8'd4;
        w[8]  = 8'd1; w[9]  = 8'd2; w[10] = 8'd3; w[11] = 8'd4;
        w[12] = 8'd1; w[13] = 8'd2; w[14] = 8'd3; w[15] = 8'd4;
end
```

W00	W01	W02	W03
W10	W11	W12	W13
W20	W21	W22	W23
W30	W31	W32	W33

=

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

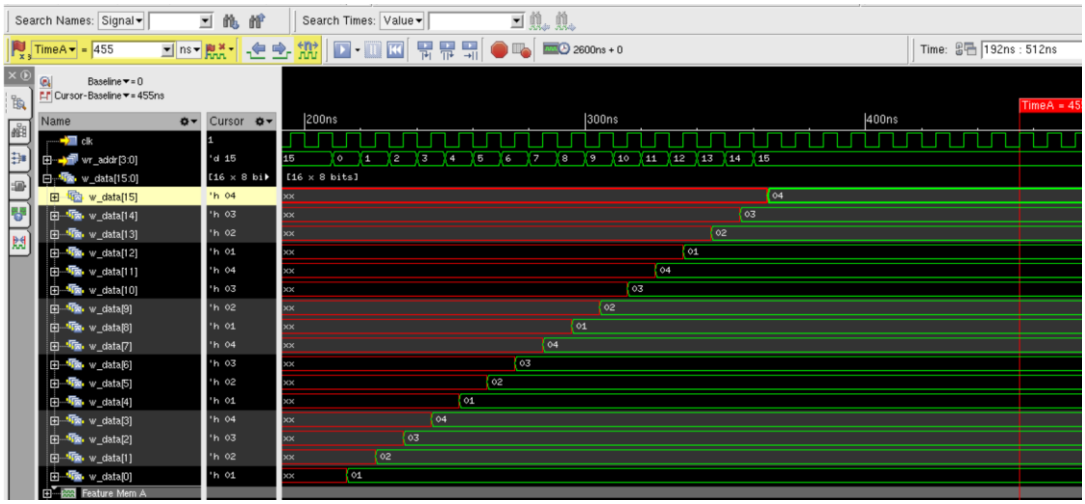
=

4	0	2	1
4	3	2	0
4	3	0	1
4	3	2	1

Test bench Writing Weight Matrix into the TPU

```
        wait(count == 4'd0)
        start_a = 1'b0;
        #20
        start_w = 1'b1;
        #20
```

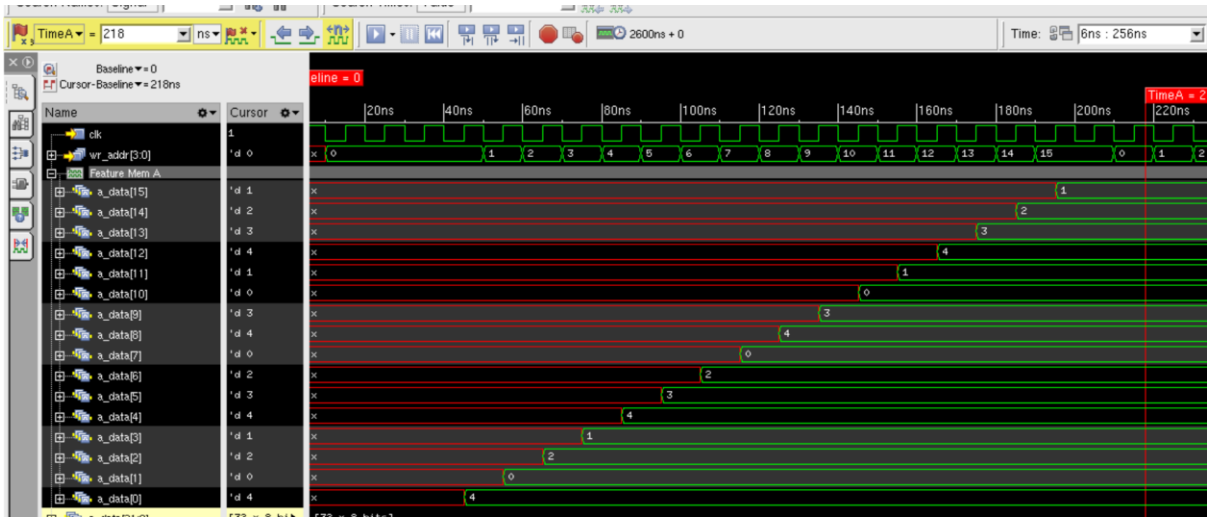
```
always @(negedge clk) begin
        if(rst)begin
                wr_addr <= 4'd0;
                data_in <= 8'd0;
                mem_acc <= 2'b00;
        end
        else if(start_a)begin
                wr_addr <= count;
                mem_acc <= 2'b01;
                data_in <= a[count];
        end
        else if(start_w)begin
                wr_addr <= count;
                mem_acc <= 2'b10;
                data_in <= w[count];
        end
        else if(start_p)begin
                rd_addr <= count;
                mem_acc <= 2'b11;
                p[count-1] <= data_out;
        end
```



Test Bench Writing A Matrix into the TPU

```
start_a = 1'b1;
#20
wait(count == 4'd0)
start_a = 1'b0;
```

```
always @(negedge clk) begin
        if(rst)begin
                wr_addr <= 4'd0;
                data_in <= 8'd0;
                mem_acc <= 2'b00;
        end
        else if(start_a)begin
                wr_addr <= count;
                mem_acc <= 2'b01;
                data_in <= a[count];
        end
        else if(start_w)begin
                wr_addr <= count;
                mem_acc <= 2'b10;
                data_in <= w[count];
        end
        else if(start_p)begin
                rd_addr <= count;
                mem_acc <= 2'b11;
                p[count-1] <= data_out;
        end
```

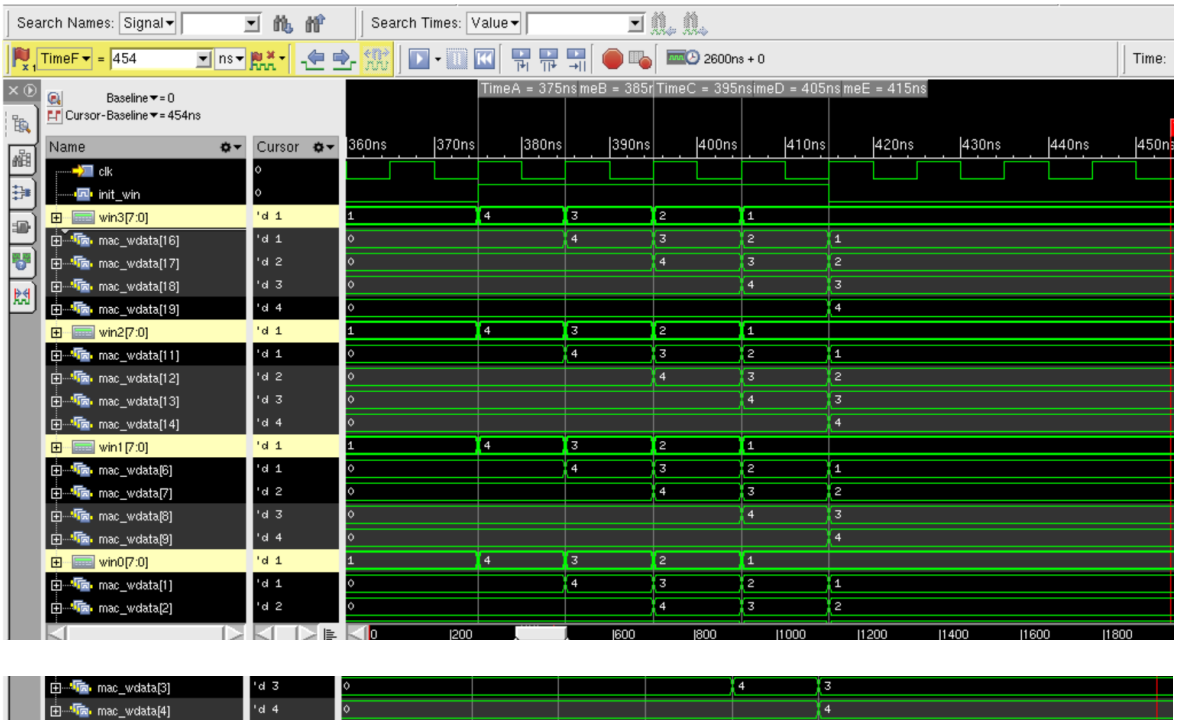


After the Start signal is given the TPU Starts loading the Weight matrix using 4 clock cycles. A counter is used to count upto the 4 cycles required to load the matrix.

```
always @(posedge clk or posedge rst) begin
        if(rst)begin
                init_cnt <= 2'd3;
                init_win <= 1'b0;
        end
        else if(busy && start && (init_cnt ==2'd3))begin
                init_cnt <= init_cnt+1;
                init_win <= 1'b1;
        end
        else if(init_win)begin
                if(init_cnt == 2'd3)
                        init_win <= 1'b0;
                else
                        init_cnt <= init_cnt+1;
        end
end
```

init_cnt is the counter and init_win is the datavalid.

```
assign mac_wdata[m*5] = w_data[(m*4) - init_cnt + 3];
assign mac_wdv[m*5] = init_win;
```



Input weight ports of the systolic array are highlighted. Init_win is on four 4 cycles and the values are flopped dwnwd

After the Init is done, the A matrix is started to load and the multiplication process starts. The A matrix is loaded across 7 cycles and they are counted up by using another counter called mul_cnt. The same counter is used to flush out the output products.

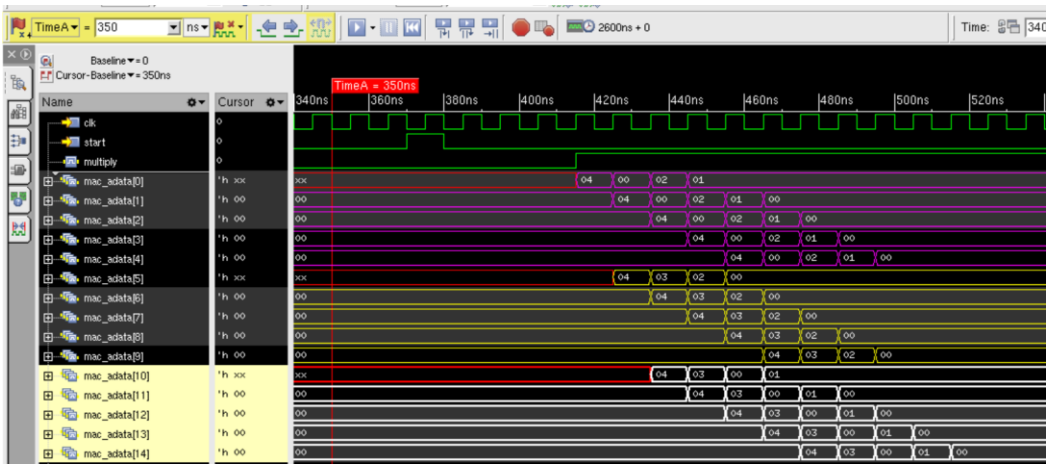
```
reg multiply;
always@(*) begin
        if(rst)
                multiply <= 1'b0;
        else if(busy && mac_wdv[4])
                multiply <=1'b1;
        else if(mul_cnt == 4'd10)
                multiply <=1'b0;
        else
                multiply <= multiply;
end

always@(posedge clk or posedge rst)begin
        if(rst)
                mul_cnt <= 4'd0;
        else if(multiply)
                mul_cnt <= mul_cnt +1;
        else
                mul_cnt <= 4'd0;
end
```

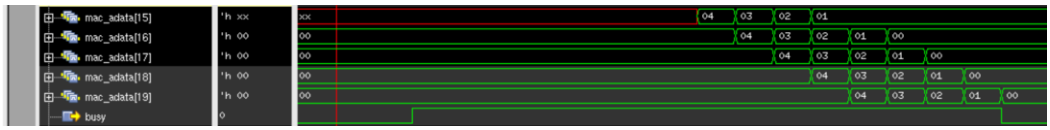
```
always @(*)begin
        if(rst)begin
                mac_radv[0] <= 1'b0;
        end
        else if(multiply && (mul_cnt<=3))begin
                mac_radv[0] <= 1'b1;
                mac_ad[0] <= a_data[mul_cnt];
        end
        else
                mac_radv[0] <= 1'b0;
end
```

```
always @(*) begin
        if(rst)begin
                mac_radv[1] <= 1'b0;
        end
        else if(multiply &&(mul_cnt>=1)&&(mul_cnt<=4))begin
                mac_radv[1] <= 1'b1;
                mac_ad[1] <= a_data[mul_cnt+4'd3];
        end
        else
                mac_radv[1] <= 1'b0;
end
```

```
always @(*) begin
        if(rst)begin
                mac_radv[2] <= 1'b0;
        end
        else if(multiply &&(mul_cnt>=2)&&(mul_cnt<=5))begin
                mac_radv[2] <= 1'b1;
                mac_ad[2] <= a_data[mul_cnt+4'd6];
        end
        else
                mac_radv[2] <= 1'b0;
end
```

```
always @(*) begin
        if(rst)begin
                mac_radv[3] <= 1'b0;
        end
        else if(multiply &&(mul_cnt>=3)&&(mul_cnt<=6))begin
                mac_radv[3] <= 1'b1;
                mac_ad[3] <= a_data[mul_cnt+4'd9];
        end
        else
                mac_radv[3] <= 1'b0;
end
```



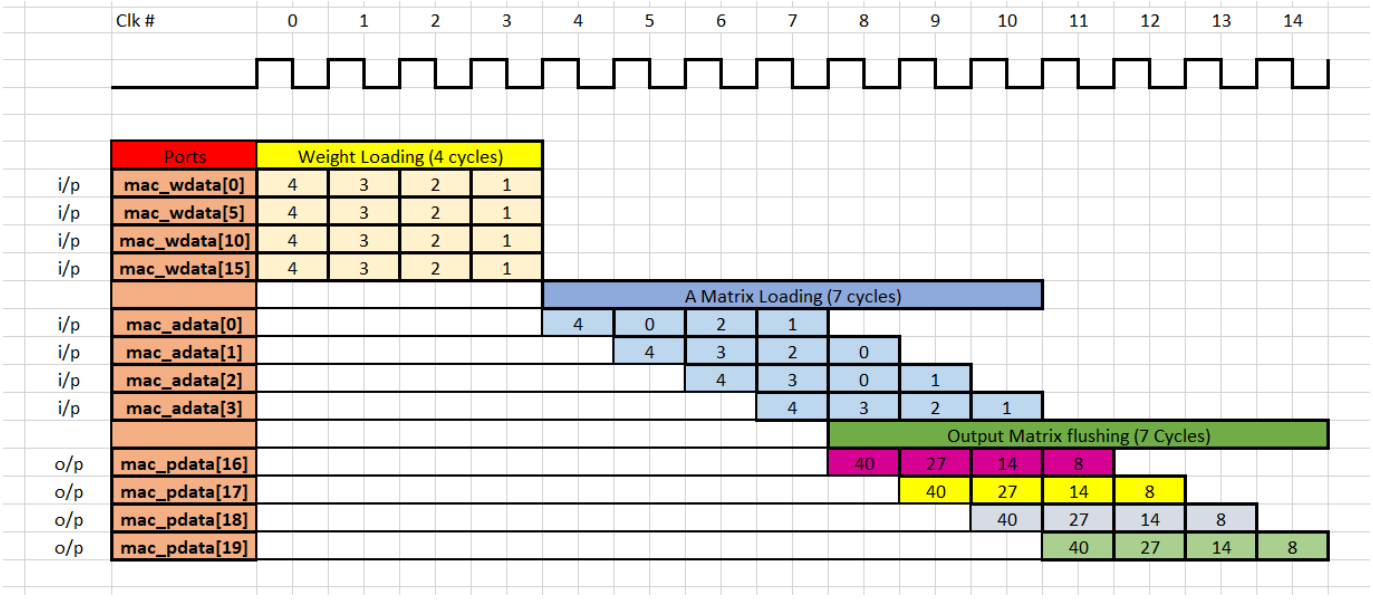


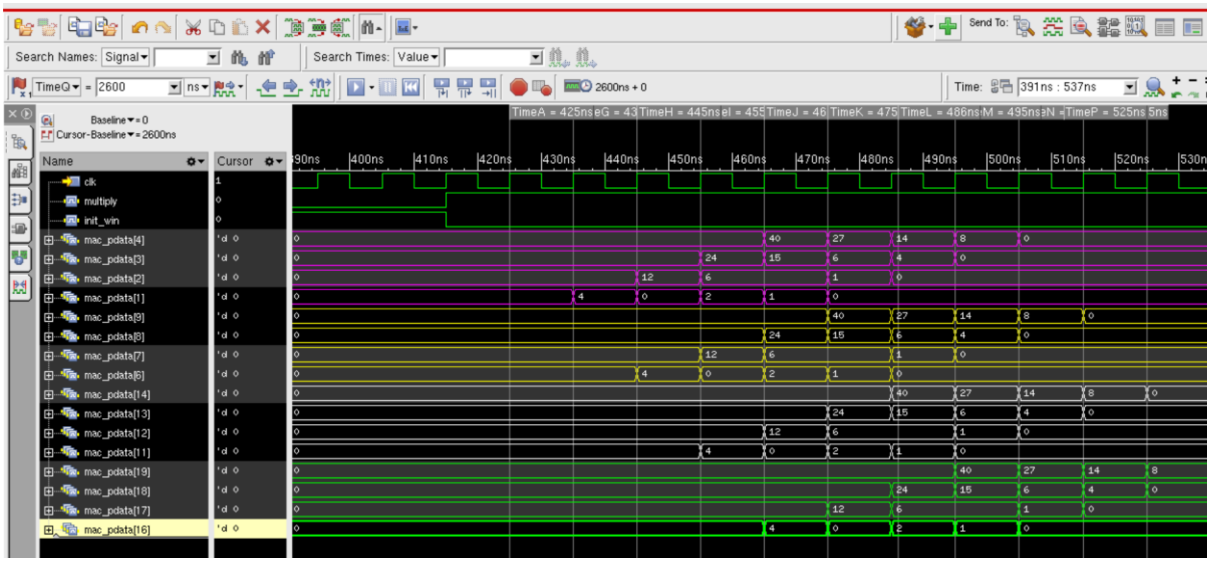
mac_adata[0] starts followed by mac_adata[5],mac_adata[10],mac_adata[15]

The values loaded into these port is determined by the mapping provided in question 2.

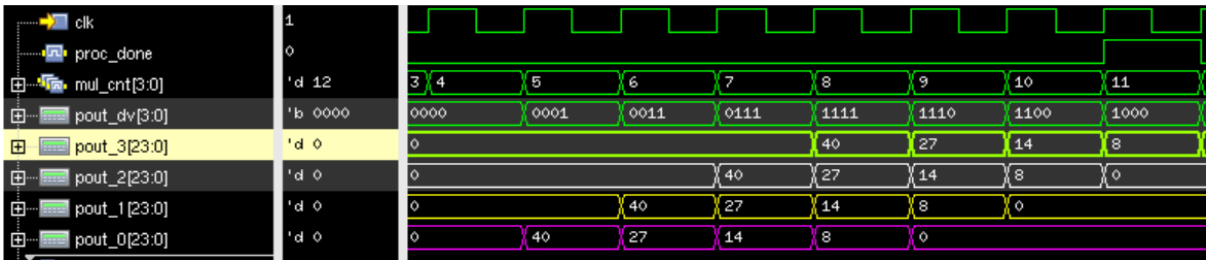
After 4 cycles of A being loaded, we will get the first Product output in parallel. These products are then routed to the quantization and activation units respectively

	Column 1 - Output	Column 2 - Output	Column 3 - Output	Column 4 - Output
Clk #4	1*4 + 2*4 + 3*4 + 4*4 = 40			
Clk #5	1*0 + 2*3 + 3*3 + 4*3 = 27	1*4 + 2*4 + 3*4 + 4*4 = 40		
Clk #6	1*2 + 2*2 +3*0 + 4*2 = 14	1*0 + 2*3 + 3*3 + 4*3 = 27	1*4 + 2*4 + 3*4 + 4*4 = 40	
Clk #7	1*1 + 2*0 +3*1 + 4*1 = 8	1*2 + 2*2 +3*0 + 4*2 = 14	1*0 + 2*3 + 3*3 + 4*3 = 27	1*4 + 2*4 + 3*4 + 4*4 = 40
Clk #8		1*1 + 2*0 +3*1 + 4*1 = 8	1*2 + 2*2 +3*0 + 4*2 = 14	1*0 + 2*3 + 3*3 + 4*3 = 27
Clk #9			1*1 + 2*0 +3*1 + 4*1 = 8	1*2 + 2*2 +3*0 + 4*2 = 14
Clk #10				1*1 + 2*0 +3*1 + 4*1 = 8



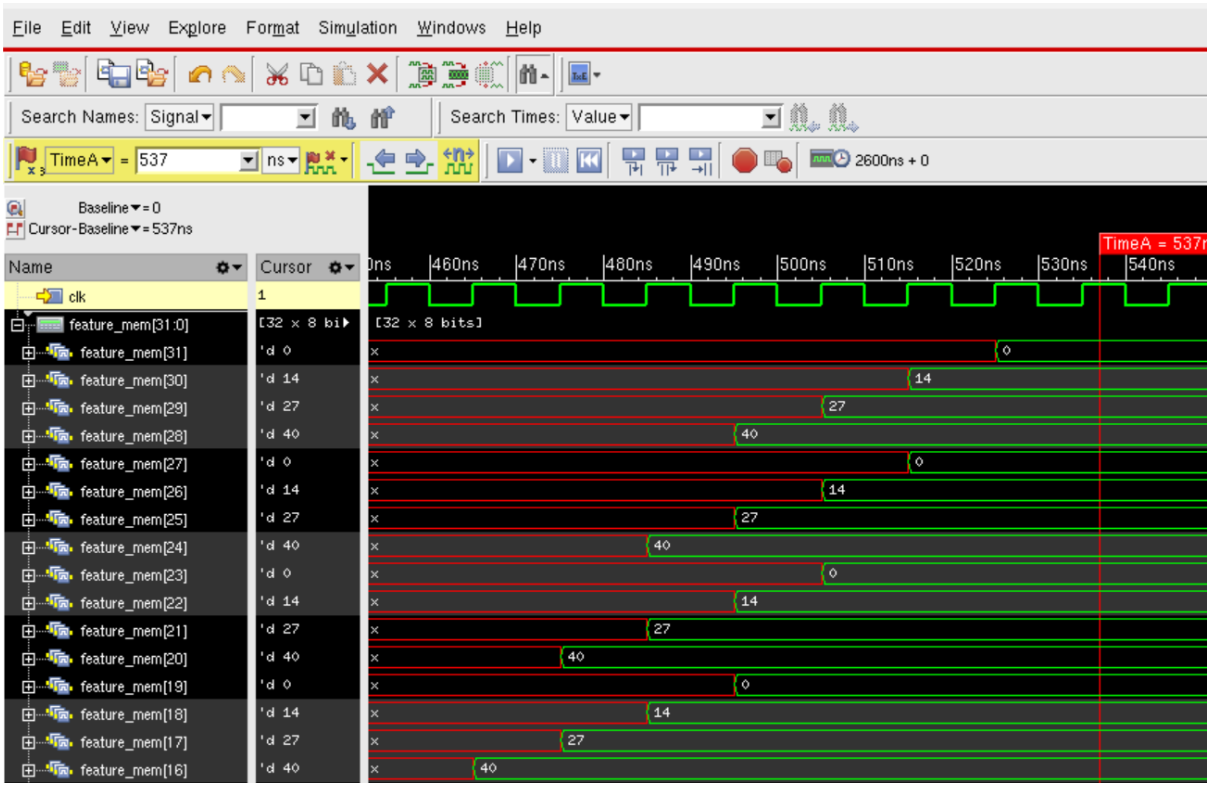


Note that the colout coding of the outputs are the same for easy comparison.



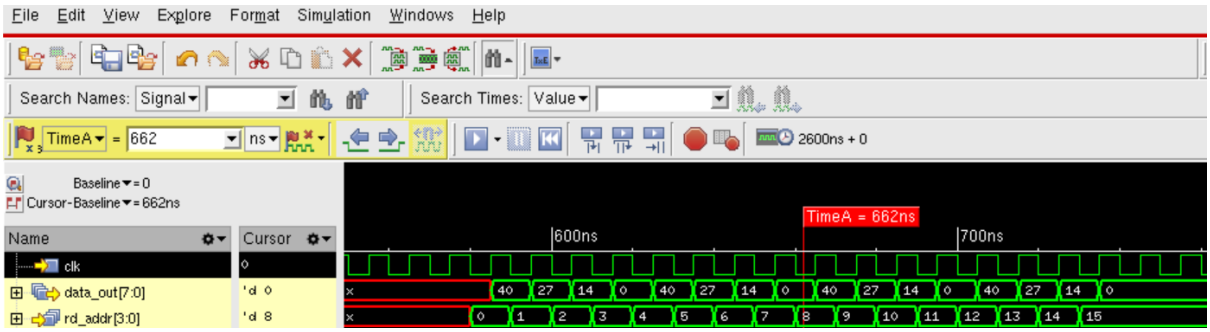
Once all the outputs are flushed out, the proc_done signal goes up which indicates to the testbench the output is ready to be read out. After the outputs are flushed, it is taken parallel into the quantization and activation unit of the TPU. The output of which are then stored into the feature memory as shown below

```
activ a1(.acout(act_data),
        .dv_acout(proc_done),
        .acin(quant_dout),
        .dv_acin(quant_dvout),
        .rst(rst)
);


quant q1(.rst(rst),
        .dv_pin(sys_dv),
        .pin(sys_data),
        .qout(quant_dout),
        .dv_qout(quant_dvout)
);
```



Note that the output value of 8 has now become 0 due to the activation unit. (8<10).

The testbench after detecting the busy signal has gone down, then readsout the feature memory for display.




```
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ iverilog tb_tpu_top.v tpu_top.v quant.v mac.v activ.v
skanda@LAPTOP-3RMLJVKC:~/asic_design/midterm$ vvp a.out

 W_matrix =   1   2   3   4
              1   2   3   4
              1   2   3   4
              1   2   3   4

 A_Matrix =   4   0   2   1
              4   3   2   0
              4   3   0   1
              4   3   2   1

 P_Matrix =  40  27  14   0
             40  27  14   0
             40  27  14   0
             40  27  14   0
VCD info: dumpfile wave_tpu_top.vcd opened for output.
```

```
-bash-4.2$ /network/rit/lab/ceashpc/software/cadence/XCELIUM1909/tools.lnx86/bin/64bit/xrun -gui -access rwc /network/rit/home
/sb914816/final_midterm/tb_tpu_top.v tpu_top.v mac.v quant.v activ.v
TOOL:   xrun(64)        19.09-s001: Started on Mar 19, 2024 at 19:58:54 EDT
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
Recompiling... reason: checksum check failure for module worklib.stimulus:v (VST).
file: /network/rit/home/sb914816/final_midterm/tb_tpu_top.v
        module worklib.stimulus:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Top level design units:
                stimulus
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.stimulus:v <0x75cfcc2e>
                        streams:  21, words: 26664
                worklib.tpu_top:v <0x043738af>
                        streams: 113, words: 32382
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                Instances  Unique
                Modules:               20       5
                Registers:            180      45
                Scalar wires:         102       -
                Expanded wires:        96       4
                Vectored wires:        84       -
                Always blocks:        136      25
                Initial blocks:         4       4
                Cont. assignments:     41      15
                Pseudo assignments:    39      15
        Writing initial simulation snapshot: worklib.stimulus:v
```