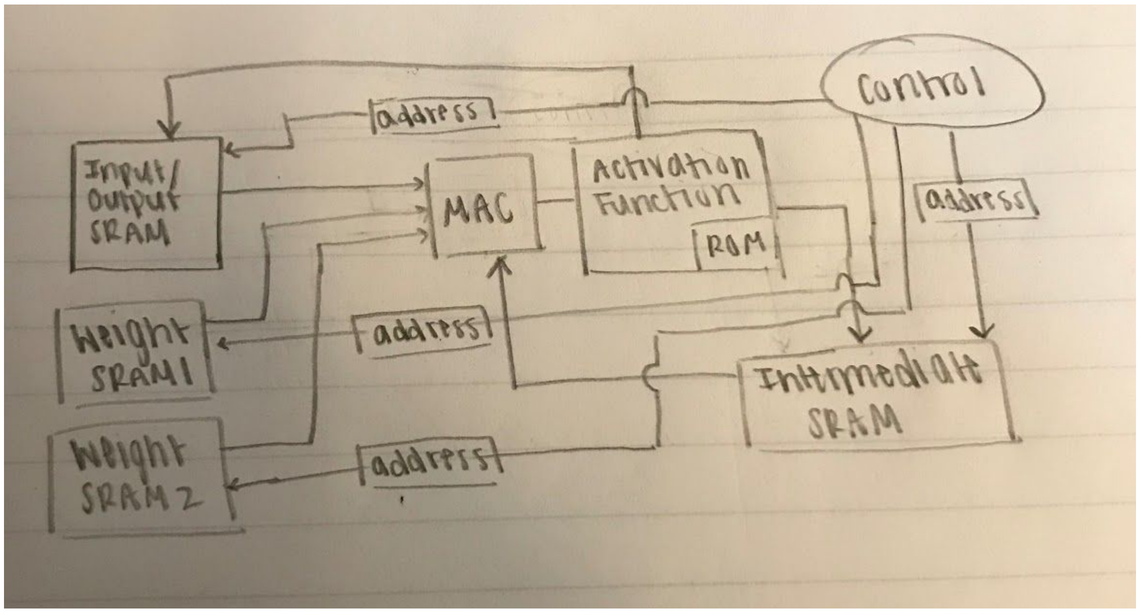
ESE 461 Final Project Report

Yuqi Liu Xingjue Liao Ginger Burrows

1.0 Introduction

The purpose of this lab was to create an accelerator for Multilayer Perceptrons using Verilog. The goal was to create a design with the smallest possible, area, power consumption, and execution time that still functioned properly. We were also required to meet certain design constraints, including limiting the number of multipliers to 64. We simulated our design to prove that it was able to perform the NN MLP computation accurately. We then synthesized the design using the Design Compiler to determine the area, power consumption, and execution time of our design. Finally, we used the Encounter tools to place and route our design. Our report will detail the system level architecture of our design, our design strategy, and results from our functional, as well as those from the completion of the design flow.

2.0 System Architecture



The above figure is the system level diagram of the hardware accelerator for multilayer perceptrons(MLP). The input matrix is stored into an SRAM module. This SRAM is partitioned as described in the SRAM section of the design strategy. This input SRAM will also be used to store the final result matrix after the two rounds of calculations have been performed. The two weight matrices are stored into two SRAM modules (also partitioned). The first weight SRAM is used for the first round of calculations and the second are used for the second round. There is also an intermediate SRAM that stores the intermediate result matrix from the first round of calculations and is then used as the input matrix for the second round.

During the first round of calculations, the control FSM continually fetches data from the input SRAM and the first weight SRAM and passes it to the MAC unit. This continues until a single MAC operation is done (one element of the intermediate result matrix has been calculated by multiplying all 784 inputs with weights and aggregating the result). Once the MAC operation is complete, the result is passed to the activation function unit and the intermediate result is stored in the intermediate SRAM.

During the second round of calculations, the control FSM continually fetches data from the intermediate SRAM and the second weight SRAM and passes it to the MAC unit. This again continues until a single MAC operation is done (one element of the final result matrix has been calculated by multiplying all 200 intermediate entries with weights and aggregating the result). Once the MAC is operation is complete, the result is again passed to the activation function unit and stored in the input/output SRAM.

3.0 Design Strategy

3.0.1 SRAM

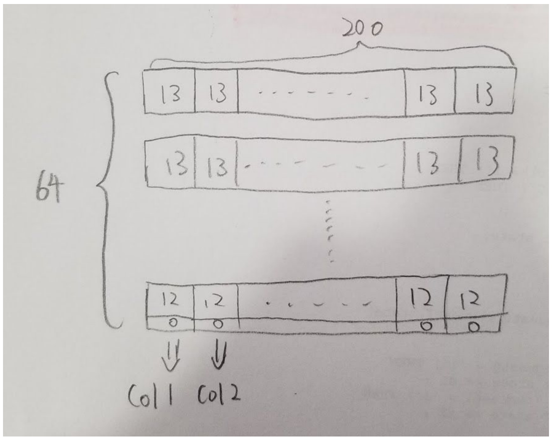
The basic SRAM module is implemented as a single-port RAM with asynchronous read. The input signals are address, data and write enable(we). Output ports is the data out. If we is asserted, the data from the input will be stored in the location pointed by the address at the rising edge. The data out port is always assigned the data pointed by the input address. This asynchronous read will reduce the number of clock cycles to fetch data to zero.

Due to the ease of fetching data, there will not be registers holding the data fetched from the SRAM. Instead, they will be directly passed to the MAC unit. Address registers need to be connected to the address input of the SRAM so that the read operation is synchronized with the clock.

In our design, we will have four large block of RAMs. Two blocks are for weight storage, one block is for storing input and output and the other block is used to store intermediate (Hidden Layer) results. Because 64 values are fetched during one clock cycle to do MAC calculation, all four large blocks are divided into 64 smaller partitions. All partitions within a large block are indexed by the same address register so that the complexity of the FSM will be reduced. By incrementing only one address register, a new batch of 64 values will come out.

The input RAM, which have 784 entries, needs to be divided into 64 partitions. As a result, the first sixteen partitions will have 13 entries and the rest 48 partitions will have 12 entries. Similarly, the intermediate RAM, which will hold the result of first round calculation, will provide input to the MAC in the second round of calculation. Therefore, it is partitioned in a similar manner where the first 8 partitions will have a size of 4 and the rest will have a size of 3.

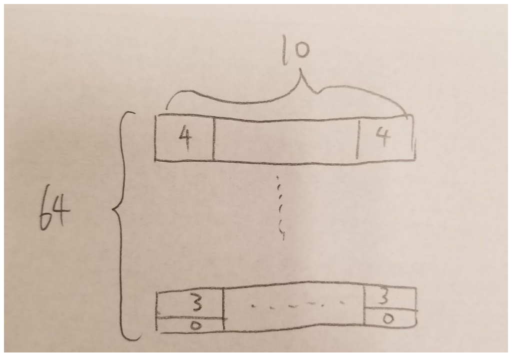
The partition for the weight SRAM is also similar. The diagram below illustrates how did we partition the SRAM storing the hidden layer weight:



Because the weight matrix is 784*200. We will perform 200 calculations to calculate each element in the result matrix. For each calculation it will take 13 rounds to finish multiplication of the input and all 784 entries in a column of weight matrix. We used 13 as the size of all subpartitions instead of a mix of 13 and 12. This results in all 64 partitions to have a size of 2600. We chose to implement all 64 partitions in the same size to simplify the operation of the control, despite using more resources.

By keeping all partitions at the same size, the weight address register will point to the first 64 entries from the next column in the weight matrix after incrementing the address register when calculation of one element in the output matrix is finished. If we were using different sizes of partitions, this alignment cannot be done using only one single address register indexing all partitions. The unused addresses in the partition will be initialized to zero so it will not affect the calculation.

The SRAM storing weight matrix for the second round is also partitioned in the similar manner with each subpartition having a size of 40.



3.0.2 Activation Function

We decides to implement the activation function using a look-up table which contains the numerical data of sigmoid. A ROM is used to hold the data. We chose a step size of 0.0625 which is 2^(-4) to calculate the numerical value of sigmoid from -6 to 6. The step size 0.0625 corresponds 0.00010000 in binary of the fixed-point representation. Therefore, the minimum resolution starts at bit 4.

One thing to notice is that sigmoid is centrosymmetric with respect to (0,0.5). Thus it is possible to only store half the values and perform arithmetic operations on them when the other half is required, which will save the area of ROM. We implemented this in our activation function design by storing sigmoid values for input 0 to +6. Since +6 is b0110.00000000 in fixed-point, using bit 10 to bit 4 of the output from MAC is enough to map 0 and all positive values to the discrete sigmoid values generated with a step size of 0.0625. When the input is larger than 6, a 1 is returned. When the input is less than -6, a 0 is returned. For the input in the range from -6 to 6, if the input is positive, then the corresponding data from the rom is outputted. If the input is negative, the input address to the ROM will be the negation of input value to fetch the sigmoid value corresponding to the opposite value of input. Then this output is subtracted from 1 (1.00000000 in fixed-point) to produce the actual result for the negative input.

3.0.3 MAC unit

The MAC unit takes in 128 16-bit inputs (64 data inputs from either the input or intermediate matrix, and 64 weight inputs) and produces a 16 bit output in one clock cycle. This output is the sum of the the multiplication of each of the inputs with the respective weight (the first input is multiplied by the first weight, the second by the second, and so on).

The input data and weights are packed into wire vectors, to allow for easier manipulation. A register, resultD/Q, is created to hold the intermediate 32 bit result of each round of 64 multiplications. A second 32 bit temporary register, temp, is created to hold the intermediate result of each multiplication. The resultQ register is truncated to 16 bits and assigned to the output register.

3.0.3.1 Sequential logic
At each clock cycle, if the reset signal is asserted, the intermediate result register, resultQ, is set to zero. Otherwise, resultQ is set to resultD. Thus, the resultD/Q registers act as a flip-flop.

3.0.3.2 Combinational logic
The temporary register, temp, is reset to zero. A for loop then iterates over the input data and weight vectors. During each iteration, one element of the data vector is multiplied with the corresponding element of the weight vector. The result of each multiplication is added to the temporary and register, temp, and stored back in temp. After the for loop, the data in the temp is added to that stored in the resultQ register and stored in the resultD register.

3.0.4 Control FSM

3.0.4.1 Control Signals
- inOutAddr: address of the input/output SRAM (this ram stores the input data for the first round, and the final result is written to it after the second round of calculations are complete)
- midAddr: intermediate SRAM address (this ram stores the results of the first round of calculations, these results then become the inputs in the second round of calculations)
- we: write enable that is shared between the intermediate SRAM and the input/output SRAM (it is a 2-d matrix where the first column corresponds to the intermediate SRAM and the second to the input/output SRAM)

- wAddr: weight SRAM address
- MAC_rst: signal to reset the MAC module
- currentState: stores the current state of the FSM
- nextState: stores the next state of the FSM
- wrCounter: keeps track of which ram partition to write
- round: stores which round of calculations the FSM is on
- Done: signals when the final result has been calculated (both rounds complete)

3.0.4.2 States

<u>INIT</u>

- Starts the mac calculation by setting the next state to MAC
- Resets the MAC module by asserting the MAC_rst signal

<u>MAC</u>

- Checks the round signal to determine if we are in round 0 (first round) or round 1 (second round)
- Round 0:
    - Increments the inOutAddr and wAddr
    - Checks to see if the mac calculations for one element of the intermediate result are complete by checking if the inOutAddr is equal to 12 (all 784 calculations have been completed when the inOutAddr reaches 12 because the MAC can perform 64 multiplications in one clock cycle. Thus, it takes 13 clock cycles to perform all 784 multiplications. One multiplication is done when the inOutAddr is equal to zero, so when the inOutAddr equals 12, thirteen cycles have passed.)
        - Passes 48 zeros to the mac to account for the uneven division of 64 into 784 (unused multipliers will just multiply zero in the last round)
        - Sets the next state to STORE
        - Resets the inOutAddr to zero

- Round 1:
    - Increment the midAddr and wAddr
    - Checks to see if the mac calculations for one element of the final result are complete by checking if midAddr is equal to 3 (all 200 calculations have been completed when the midAddr reaches 3 because the MAC can perform 64 multiplications in one clock cycle. Thus, it takes 4 clock cycles to perform all 200 multiplications. One multiplication is done when the midAddr is equal to zero, so when it reaches 3, 4 clock cycles have passed.)

            - Passes 56 zeros to the mac to account for the uneven division of 64 into 200
            - Sets the next state to STORE
            - Resets the midAddr to zero

<u>STORE</u>

- Asserts the write enable of the current (0-63, stored in wrCounter) mid or inout sram
- Sets wrData to ActFout (result of the activation function)
- Checks the round signal to determine if we are in round 0 (first round) or round 1 (second round)
- Round 0:
    - Sets the next state to MAC
    - Resets the MAC by asserting MAC_rst
    - Checks if all the intermediate results for the first round have been calculated and stored (first round done completely done) by checking if the wAddr is equal to 2600 (we know that all intermediate results have been calculated and stored when the wAddr is equal to 2600 because a total of 200 intermediate results need to be calculated and it takes 13 clock cycles to calculate each one. The wAddr is incremented by one each cycle, so when it reaches 2600 (200*13=2600), all 200 intermediate results have been calculated and stored.
        - Change the round register to 1
        - Clears the wAddr, wrCounter, and inOutAddr
    - If wAddr is not yet 2600
        - Sets the next state to the MAC
        - Resets the mac by asserting MAC_rst
        - Increments wrCounter
        - Checks if wrCounter has reached 63
            - Resets wrCounter to zero
            - Increments midAddr
- Round 1:
    - Checks if all the results from the second round have been stored (second round done completely done) by checking if the wAddr is equal to 40 (we know that all intermediate results have been calculated and stored when the wAddr is equal to 40 because a total of 10 intermediate results need to be calculated and it takes 4 clock cycles to calculate each one. The wAddr is incremented by one each cycle, so when it reaches 40 (10*4=40), all 10 results have been calculated and stored.

        - Sets the next state to DONE
        - Clears the wAddr, wrCounter, and inOutAddr
    - If wAddr is not yet 40
        - Sets the next state to the MAC
        - Resets the mac by asserting MAC_rst
        - Increments wrCounter
        - Checks if wrCounter has reached 63
            - Resets wrCounter to zero
            - Increments inOutAddr

<u>DONE</u>

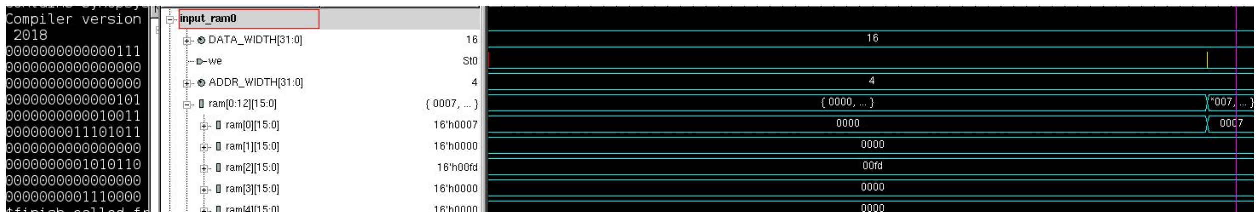
- Asserts the done signal

4.0 Functional Validation

To verify the behavioral model of the design, we created a testbench which will initialize the input/output ram as well as weight srams. The initialization is done carefully to make sure that the unused address gets initialized to zero. In the testbench, only one image is loaded for simulation and the testbench will check if done signal from the device is asserted. If done is high, the result will then be printed to the console. The following code snippet shows this conditional logic:

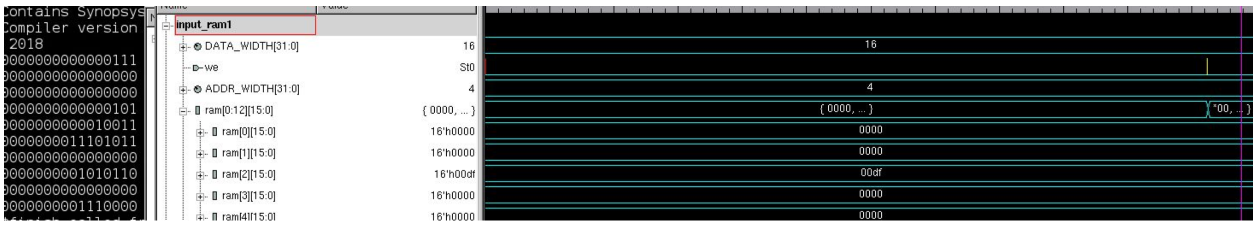
```
if(done) begin
        $display("%b",DUT.input_ram0.ram[0]);
        $display("%b",DUT.input_ram1.ram[0]);
        $display("%b",DUT.input_ram2.ram[0]);
        $display("%b",DUT.input_ram3.ram[0]);
        $display("%b",DUT.input_ram4.ram[0]);
        $display("%b",DUT.input_ram5.ram[0]);
        $display("%b",DUT.input_ram6.ram[0]);
        $display("%b",DUT.input_ram7.ram[0]);
        $display("%b",DUT.input_ram8.ram[0]);
        $display("%b",DUT.input_ram9.ram[0]);
```

To verify the above lines, we checked the simulation waveform to make sure that the value printed is indeed the value in the ram. Screenshots showing printed result on the left and the value in the RAM in the waveform are shown below.



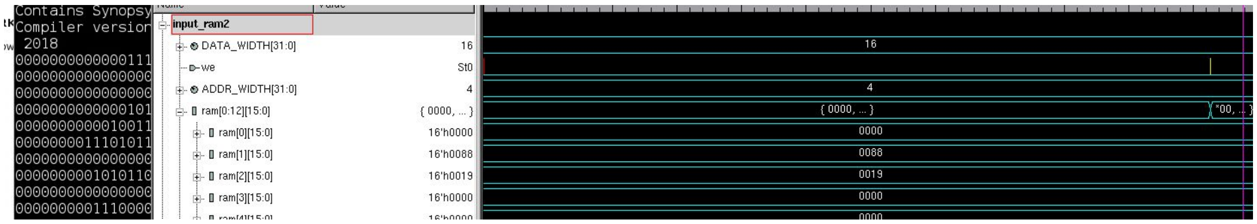
```
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
```

```
input_ram0
  DATA_WIDTH[31:0]   16
  we                 St0
  ADDR_WIDTH[31:0]   4
  ram[0:12][15:0]    { 0007, ... }
    ram[0][15:0]     16'h0007
    ram[1][15:0]     16'h0000
    ram[2][15:0]     16'h00fd
    ram[3][15:0]     16'h0000
```



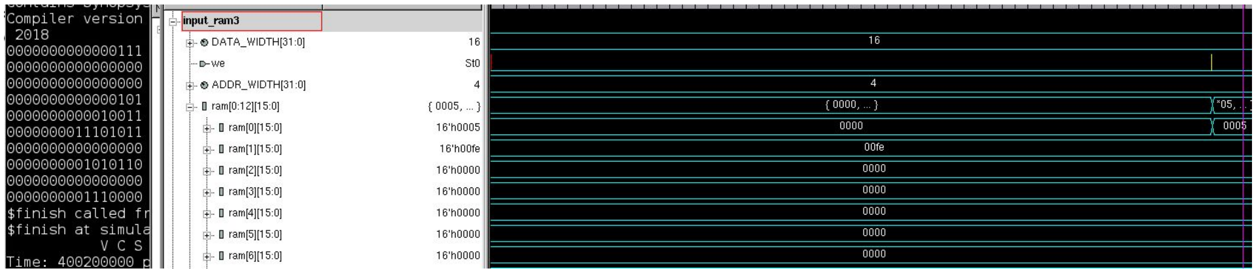
```
Contains Synopsys
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
```

```
input_ram1
  DATA_WIDTH[31:0]        16              16
  we                      St0
  ADDR_WIDTH[31:0]        4               4
  ram[0:12][15:0]         { 0000, ... }   { 0000, ... }   '00, ... }
    ram[0][15:0]          16'h0000        0000
    ram[1][15:0]          16'h0000        0000
    ram[2][15:0]          16'h00df        00df
    ram[3][15:0]          16'h0000        0000
    ram[4][15:0]          16'h0000        0000
```



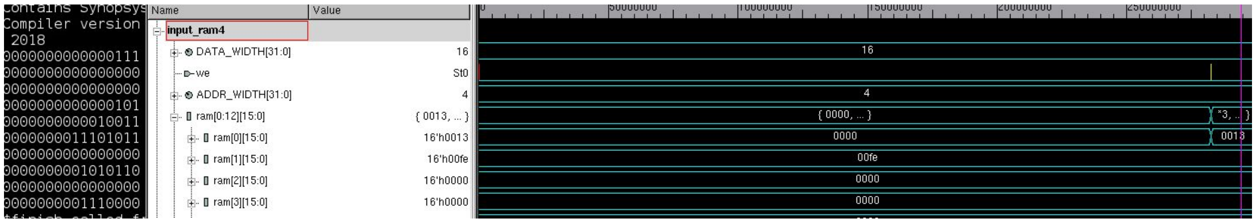
```
Contains Synopsy
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
```

```
input_ram2
  DATA_WIDTH[31:0]        16              16
  we                      St0
  ADDR_WIDTH[31:0]        4               4
  ram[0:12][15:0]         { 0000, ... }   { 0000, ... }   '00, ... }
    ram[0][15:0]          16'h0000        0000
    ram[1][15:0]          16'h0088        0088
    ram[2][15:0]          16'h0019        0019
    ram[3][15:0]          16'h0000        0000
    ram[4][15:0]          16'h0000        0000
```



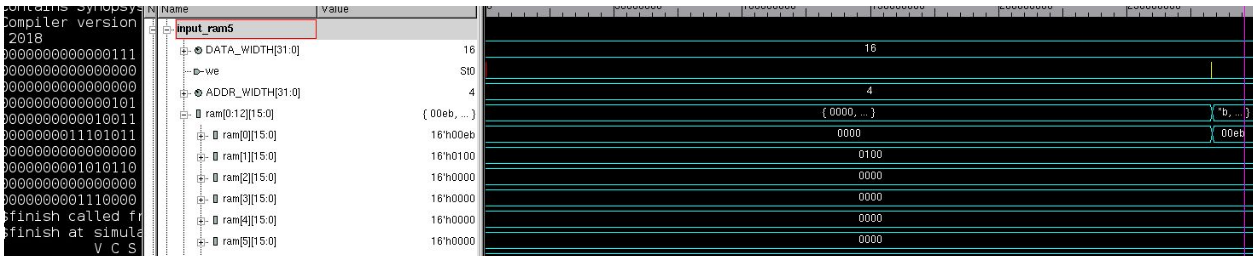
```
Contains Synopsy
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
$finish called fr
$finish at simula
          V C S
Time: 400200000 p
```

```
input_ram3
  DATA_WIDTH[31:0]        16              16
  we                      St0
  ADDR_WIDTH[31:0]        4               4
  ram[0:12][15:0]         { 0005, ... }   { 0000, ... }   '05, ... }
    ram[0][15:0]          16'h0005        0000            0005
    ram[1][15:0]          16'h00fe        00fe
    ram[2][15:0]          16'h0000        0000
    ram[3][15:0]          16'h0000        0000
    ram[4][15:0]          16'h0000        0000
    ram[5][15:0]          16'h0000        0000
    ram[6][15:0]          16'h0000        0000
```



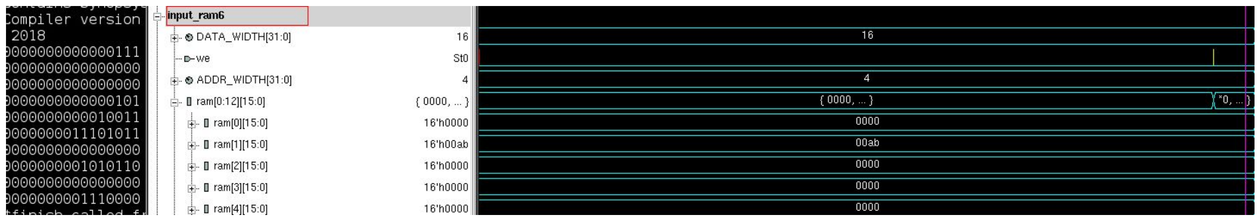
```
Contains Synopsys
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
$finish called fr
```

```
Name                      Value           0    50000000    100000000    150000000    200000000    250000000
input_ram4
  DATA_WIDTH[31:0]        16              16
  we                      St0
  ADDR_WIDTH[31:0]        4               4
  ram[0:12][15:0]         { 0013, ... }   { 0000, ... }   '3, ... }
    ram[0][15:0]          16'h0013        0000            0013
    ram[1][15:0]          16'h00fe        00fe
    ram[2][15:0]          16'h0000        0000
    ram[3][15:0]          16'h0000        0000
```



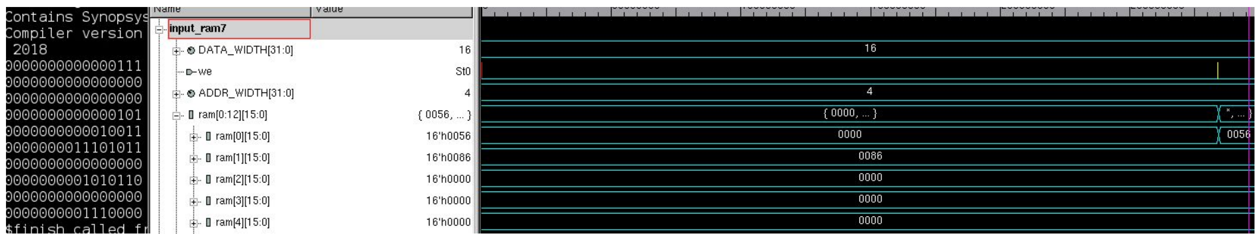
```
Contains Synopsys
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
$finish called fr
$finish at simula
          V C S
```

```
Name                      Value
input_ram5
  DATA_WIDTH[31:0]        16              16
  we                      St0
  ADDR_WIDTH[31:0]        4               4
  ram[0:12][15:0]         { 00eb, ... }   { 0000, ... }   'b, ... }
    ram[0][15:0]          16'h00eb        0000            00eb
    ram[1][15:0]          16'h0100        0100
    ram[2][15:0]          16'h0000        0000
    ram[3][15:0]          16'h0000        0000
    ram[4][15:0]          16'h0000        0000
    ram[5][15:0]          16'h0000        0000
```



```
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
```

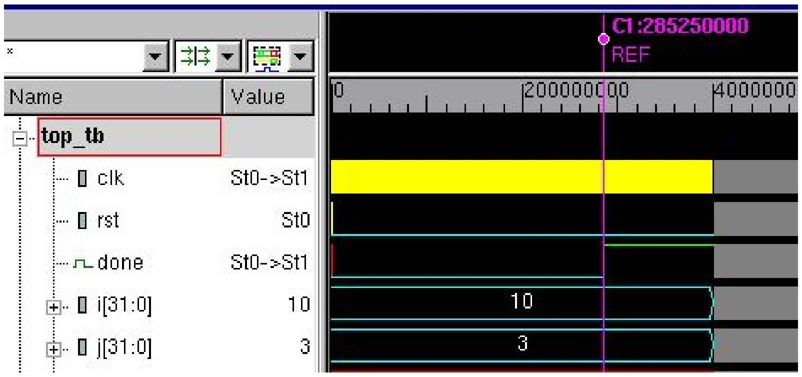
```
input_ram6
  DATA_WIDTH[31:0]    16
  we                  St0
  ADDR_WIDTH[31:0]    4
  ram[0:12][15:0]     { 0000, ... }
    ram[0][15:0]      16'h0000
    ram[1][15:0]      16'h00ab
    ram[2][15:0]      16'h0000
    ram[3][15:0]      16'h0000
    ram[4][15:0]      16'h0000
```



```
Contains Synopsys
Compiler version
 2018
0000000000000111
0000000000000000
0000000000000000
0000000000000101
0000000000010011
0000000011101011
0000000000000000
0000000001010110
0000000000000000
0000000001110000
$finish called fr
```

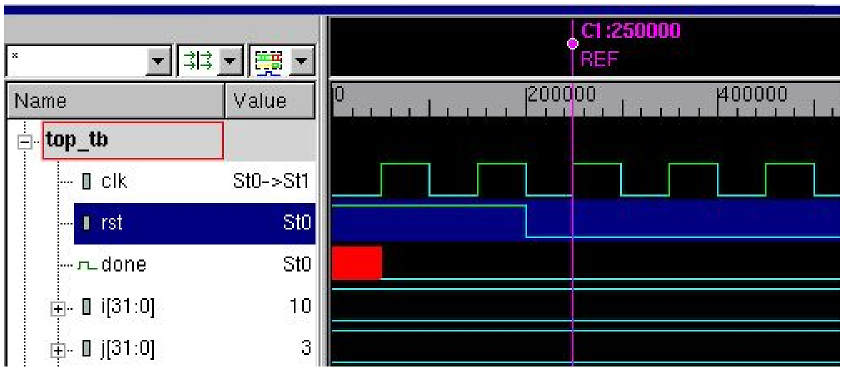
```
Name                  Value
input_ram7
  DATA_WIDTH[31:0]    16
  we                  St0
  ADDR_WIDTH[31:0]    4
  ram[0:12][15:0]     { 0056, ... }
    ram[0][15:0]      16'h0056
    ram[1][15:0]      16'h0086
    ram[2][15:0]      16'h0000
    ram[3][15:0]      16'h0000
    ram[4][15:0]      16'h0000
```

The ten printed values match the values from all 10 rams at address 0. Because the done signal is asserted when the calculation is done, the end time of calculation can be located by looking at the time when done goes high, which is shown below.



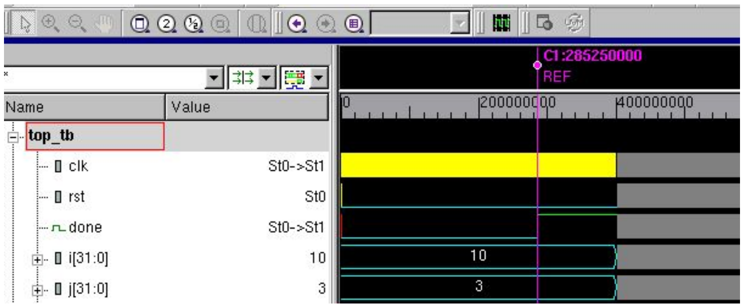
```
Name          Value
top_tb
  clk         St0->St1
  rst         St0
  done        St0->St1
  i[31:0]     10
  j[31:0]     3
```

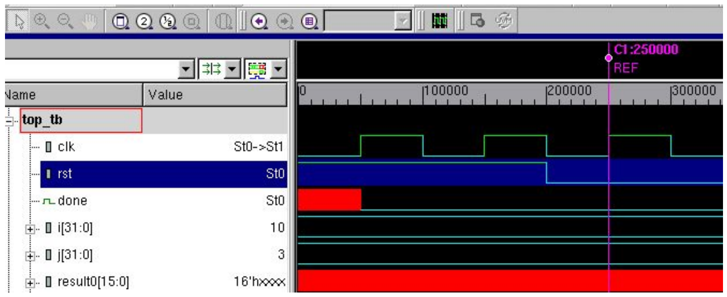
Also, the starting time of the calculation is the first clock edge after reset, which is shown below.



```
Name          Value
top_tb
  clk         St0->St1
  rst         St0
  done        St0
  i[31:0]     10
  j[31:0]     3
```

To verify that our design will take the same time to calculate the result for all kinds of inputs, we ran the simulation for test image 2 and checked the starting and ending time of calculation. They are shown below:





It can be seen that the starting and ending time are still the same. Therefore, the total execution time will be the product of 10 and the number of clock cycles taken to calculate the result of one image.

We are using clock period of 100ns in our testbench, so the number of clock cycles it takes to calculate result for one image is (285250-250)/100 = 2850. Thus, to finish a batch of ten images, the calculation time is 2850*10=28500 clock cycles.

To verify the correctness of the result, we used the MATLAB code to generate the fixed-point representation and the double precision results. We adjusted the step size in sigmoid function to be the same as our design, 0.0625. The following table shows the result obtained for test image 1.

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
111	110	0.027344	0.023438	0.00390625	0.02538
0	0	0	0	0	0.000165
0	1	0	0.003906	0.00390625	0.002188
101	101	0.019531	0.019531	0	0.020615
10011	10010	0.074219	0.070313	0.00390625	0.072613
11101011	11101110	0.917969	0.929688	0.01171875	0.925973
0	0	0	0	0	0.001963

1010110	1010110	0.335938	0.335938	0	0.330501
0	0	0	0	0	0.001595
1110000	1110000	0.4375	0.4375	0	0.42495

It can be seen that the simulation result is close to the MATLAB result. Some of them are the same and some of them differ at the least significant bit. The difference between these two is very small. The double precision value is listed in the table just serving as a reference. Because we only have 16 bits, we will never achieve the accuracy of double precision result.

In addition, the MATLAB result is more accurate than the simulation result as for some entries it is closer to the double precision result. The main reason is that in the MATLAB code the results are produced by converting the final result in double precision to fixed-point so the calculation to get final result is still done in double precision. Therefore, it will be more accurate since we are only using 16-bit fixed point numbers where we need to truncate each multiplication and there is no rounding, leading to lower accuracy.

The result of other 9 test images are shown below:
Image 2

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
0	0	0	0	0	0.001168917
0	0	0	0	0	0.001030884
0	0	0	0	0	0.001157012
10	10	0.0078125	0.0078125	0	0.006612708
10	10	0.0078125	0.0078125	0	0.007171836
11001010	11001111	0.7890625	0.80859375	0.01953125	0.803933274
0	0	0	0	0	6.77974E-05
10111000	10111011	0.71875	0.73046875	0.01171875	0.725811045
1	1	0.00390625	0.00390625	0	0.003822837
11101010	11101110	0.9140625	0.9296875	0.015625	0.924879788

Image 3

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result

1	1	0.00390625	0.00390625	0	0.004421294
1	1	0.00390625	0.00390625	0	0.002684225
0	0	0	0	0	0.001567244
1011	1011	0.04296875	0.04296875	0	0.043446035
10	10	0.0078125	0.0078125	0	0.009912801
10110101	10110101	0.70703125	0.70703125	0	0.705414916
0	0	0	0	0	0.000283459
1100001	1100001	0.37890625	0.37890625	0	0.379021242
1	1	0.00390625	0.00390625	0	0.002475842
10001000	10010000	0.53125	0.5625	0.03125	0.5505723

Image 4

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
0	0	0	0	0	0.00176575
0	0	0	0	0	2.65058E-05
0	0	0	0	0	0.000799586
1100	1011	0.046875	0.04296875	0.00390625	0.043544816
101	101	0.01953125	0.01953125	0	0.02026008
1111	1110	0.05859375	0.0546875	0.00390625	0.055503648
0	0	0	0	0	0.001509848
100110	100010	0.1484375	0.1328125	0.015625	0.140787364
10	10	0.0078125	0.0078125	0	0.006558333
10011	10010	0.07421875	0.0703125	0.00390625	0.071622865

Image 5

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
1	1	0.00390625	0.00390625	0	0.003352182
0	0	0	0	0	0.001794856
0	0	0	0	0	0.001215611
11	11	0.01171875	0.01171875	0	0.011625015

11	10	0.01171875	0.0078125	0.00390625	0.009794687
10111110	11000001	0.7421875	0.75390625	0.01171875	0.74509786
0	0	0	0	0	0.000142226
10111000	10111011	0.71875	0.73046875	0.01171875	0.726902795
10	10	0.0078125	0.0078125	0	0.007836488
1111	1111	0.05859375	0.05859375	0	0.05565888

Image 6

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
11	11	0.01171875	0.01171875	0	0.013147029
0	0	0	0	0	0.000749909
0	1	0	0.00390625	0.00390625	0.002493816
110	110	0.0234375	0.0234375	0	0.022985404
100	100	0.015625	0.015625	0	0.015922969
10111000	10111011	0.71875	0.73046875	0.01171875	0.725344937
0	0	0	0	0	0.001294672
10010000	10011000	0.5625	0.59375	0.03125	0.571773778
0	0	0	0	0	0.001093273
10000100	10001100	0.515625	0.546875	0.03125	0.531138138

Image 7

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
0	0	0	0	0	0.000722922
0	0	0	0	0	0.000671659
0	0	0	0	0	0.000564984
10010	10001	0.0703125	0.06640625	0.00390625	0.071474217
1	1	0.00390625	0.00390625	0	0.005074094
11001010	11001111	0.7890625	0.80859375	0.01953125	0.79940924
0	0	0	0	0	0.000188982
10010000	10010100	0.5625	0.578125	0.015625	0.572439559

1000	1000	0.03125	0.03125	0	0.029355711
1101100	1101000	0.421875	0.40625	0.015625	0.405473289

Image 8

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
1	1	0.00390625	0.00390625	0	0.003166724
100	11	0.015625	0.01171875	0.00390625	0.014029622
0	0	0	0	0	0.000505686
1001	1001	0.03515625	0.03515625	0	0.034793079
101	101	0.01953125	0.01953125	0	0.018674223
10101110	10110001	0.6796875	0.69140625	0.01171875	0.687880709
0	0	0	0	0	0.000119463
1100100	1100100	0.390625	0.390625	0	0.386139837
1	1	0.00390625	0.00390625	0	0.002856056
10010100	10011000	0.578125	0.59375	0.015625	0.584489891

Image 9

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result
10	10	0.0078125	0.0078125	0	0.007163063
0	0	0	0	0	0.000459605
1	1	0.00390625	0.00390625	0	0.002591593
1010	1001	0.0390625	0.03515625	0.00390625	0.035595406
0	0	0	0	0	0.001350742
10101010	10101110	0.6640625	0.6796875	0.015625	0.684538658
0	0	0	0	0	0.000818984
1101100	1101100	0.421875	0.421875	0	0.414736513
0	0	0	0	0	0.000352124
10000000	10000100	0.5	0.515625	0.015625	0.498509293

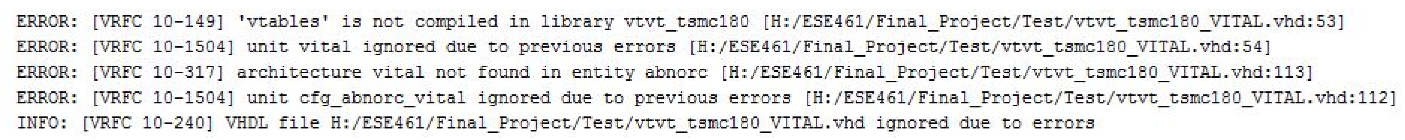
Image 10

Simulation-Fixed	MATLAB Fixed	Simulation Result	MATLAB result	Difference	Double Result

10	10	0.0078125	0.0078125	0	0.007922493
0	0	0	0	0	0.001619618
0	0	0	0	0	0.000924234
11	11	0.01171875	0.01171875	0	0.012046606
110	110	0.0234375	0.0234375	0	0.024431677
1011001	1011001	0.34765625	0.34765625	0	0.344438669
0	0	0	0	0	0.000199357
100010	11111	0.1328125	0.12109375	0.01171875	0.125583565
100	100	0.015625	0.015625	0	0.014777035
111	111	0.02734375	0.02734375	0	0.025511154

All results from 10 test images display similar characteristics and the error is small. Therefore, our design is functioning properly.

We tried to use the netlist after place and route to do the post-implementation simulation with the library files. Because the library files are in VHDL and we don’t know how to use VCS to compile both verilog and VHDL at the same time, we tried to use Vivado to simulate the files as its simulator supports mixed-language simulation, though we are not implementing it on an FPGA. However, although the Vivado tool recognizes the netlist and the library, the simulation failed due to the error below and we don’t know how to solve it:



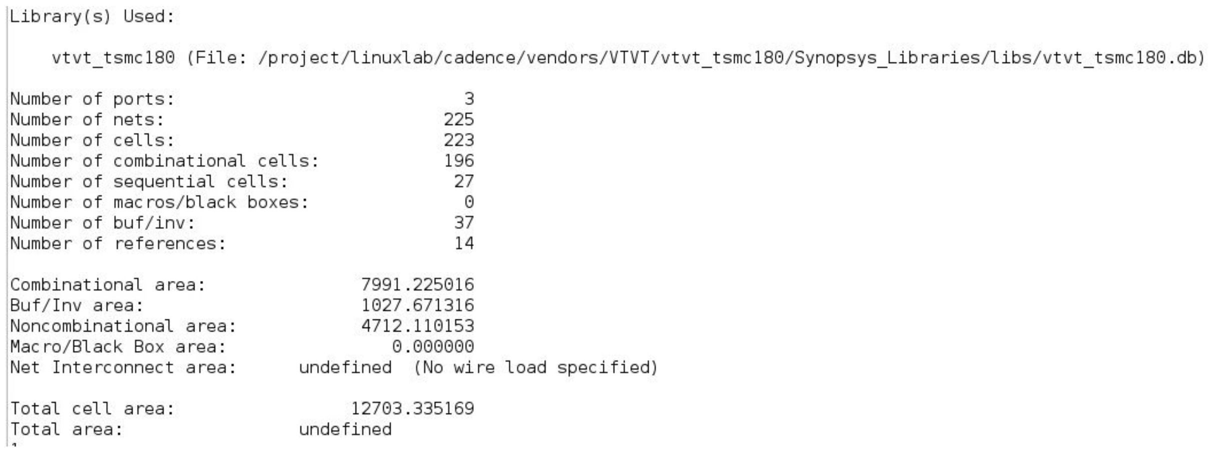
```
ERROR: [VRFC 10-149] 'vtables' is not compiled in library vtvt_tsmc180 [H:/ESE461/Final_Project/Test/vtvt_tsmc180_VITAL.vhd:53]
ERROR: [VRFC 10-1504] unit vital ignored due to previous errors [H:/ESE461/Final_Project/Test/vtvt_tsmc180_VITAL.vhd:54]
ERROR: [VRFC 10-317] architecture vital not found in entity abnorc [H:/ESE461/Final_Project/Test/vtvt_tsmc180_VITAL.vhd:113]
ERROR: [VRFC 10-1504] unit cfg_abnorc_vital ignored due to previous errors [H:/ESE461/Final_Project/Test/vtvt_tsmc180_VITAL.vhd:112]
INFO: [VRFC 10-240] VHDL file H:/ESE461/Final_Project/Test/vtvt_tsmc180_VITAL.vhd ignored due to errors
```

5.0 Complete Design Flow

Synthesis

1. Area

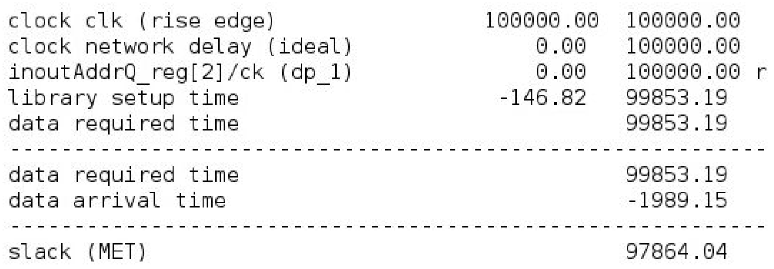
   Our accelerator’s area is 1.27033 mm^2 , the screenshot for the area report is shown below:



```
Library(s) Used:

    vtvt_tsmc180 (File: /project/linuxlab/cadence/vendors/VTVT/vtvt_tsmc180/Synopsys_Libraries/libs/vtvt_tsmc180.db)

Number of ports:                            3
Number of nets:                           225
Number of cells:                          223
Number of combinational cells:            196
Number of sequential cells:                27
Number of macros/black boxes:               0
Number of buf/inv:                         37
Number of references:                      14

Combinational area:               7991.225016
Buf/Inv area:                     1027.671316
Noncombinational area:            4712.110153
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 12703.335169
Total area:                 undefined
```

2. Timing

   The max timing report is shown below:

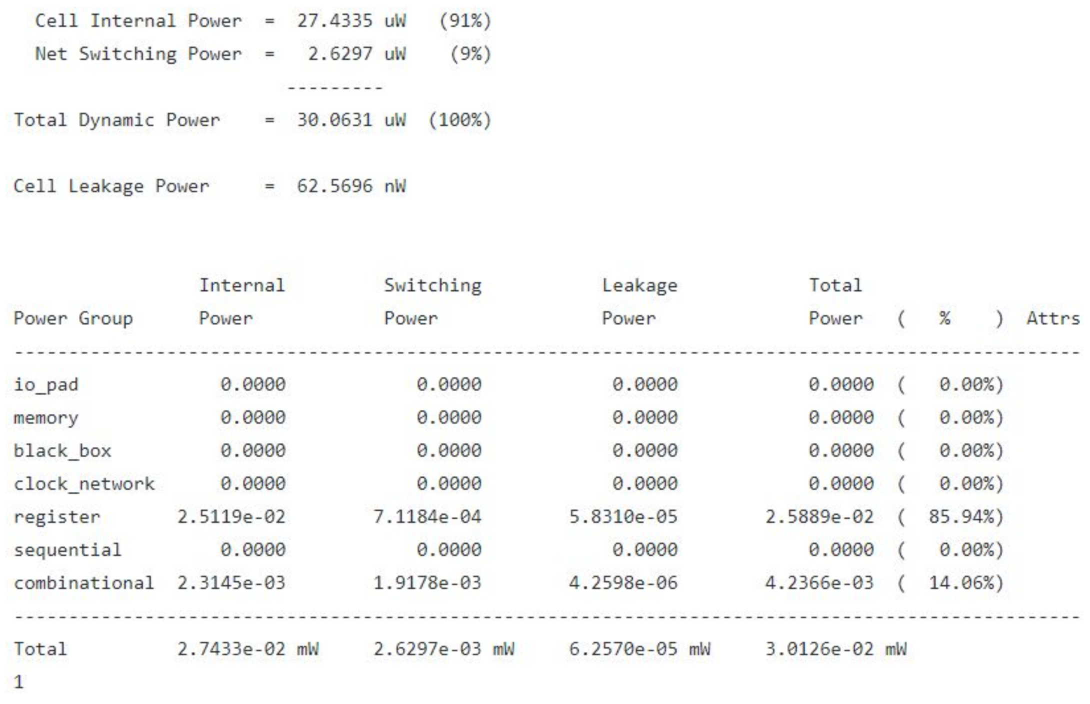


```
clock clk (rise edge)               100000.00  100000.00
clock network delay (ideal)              0.00  100000.00
inoutAddrQ_reg[2]/ck (dp_1)              0.00  100000.00 r
library setup time                    -146.82   99853.19
data required time                              99853.19
-----------------------------------------------------------
data required time                              99853.19
data arrival time                               -1989.15
-----------------------------------------------------------
slack (MET)                                     97864.04
```

   The timing is satisfied and the critical path delay is 1989.15 ps. This means the maximum clock frequency is 1/(1989.15e-12) = 502 MHz. Taking 1989.15ps as the maximum clock period and 28500 number of clock cycles to finish all 10 images, the execution time will be 1989.15 * 28500 = 56.69 us.

3. Power

   The power report is shown below:

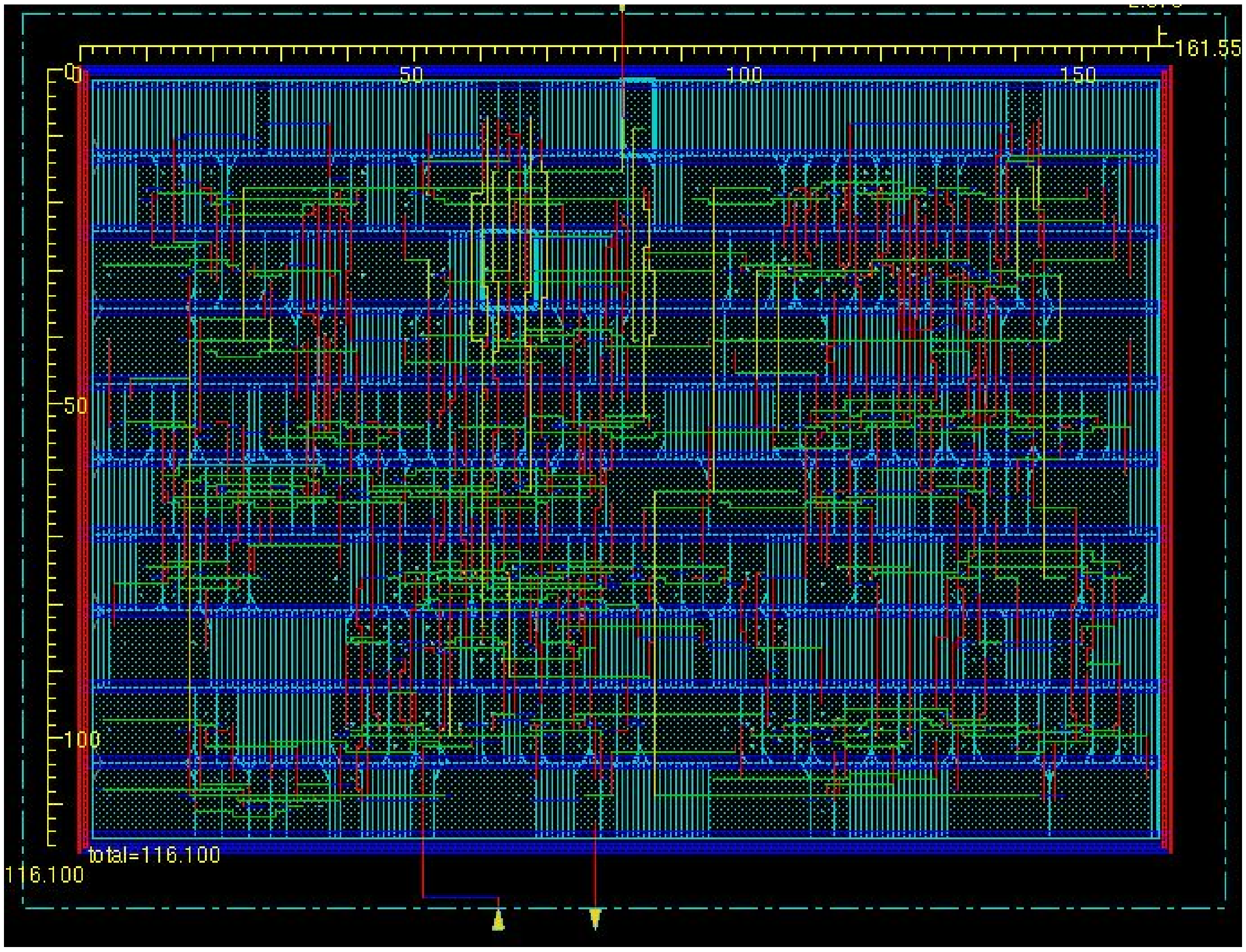


```
  Cell Internal Power  =  27.4335 uW   (91%)
  Net Switching Power  =   2.6297 uW    (9%)
                         ---------
Total Dynamic Power    =  30.0631 uW  (100%)

Cell Leakage Power     =  62.5696 nW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000              0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000              0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000              0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000              0.0000             0.0000  (   0.00%)
register       2.5119e-02       7.1184e-04          5.8310e-05         2.5889e-02  (  85.94%)
sequential         0.0000           0.0000              0.0000             0.0000  (   0.00%)
combinational  2.3145e-03       1.9178e-03          4.2598e-06         4.2366e-03  (  14.06%)
--------------------------------------------------------------------------------------------------
Total          2.7433e-02 mW    2.6297e-03 mW       6.2570e-05 mW      3.0126e-02 mW
1
```

The total power is about 30uW

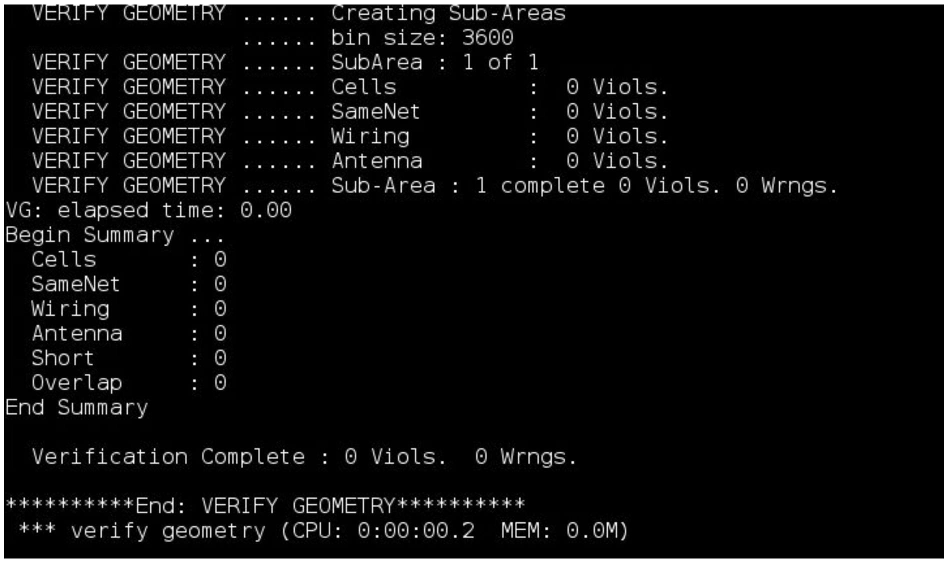
6.0 Place and Routing

The schematic of the circuit after the P&R:



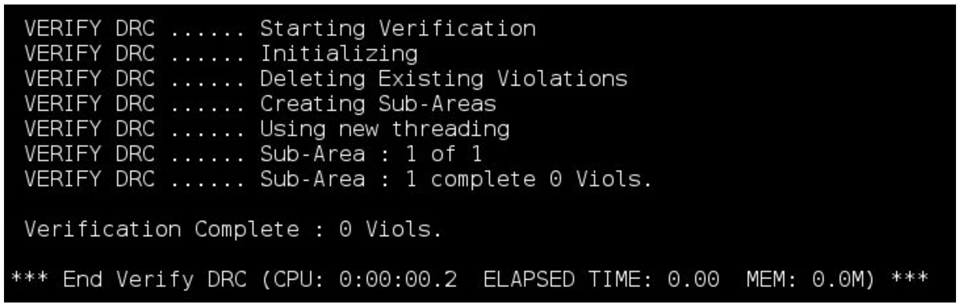
The total area of the circuit board is 1.8756 mm^2 . It's bigger because of the filler and the buffers added.

The geometry verification:



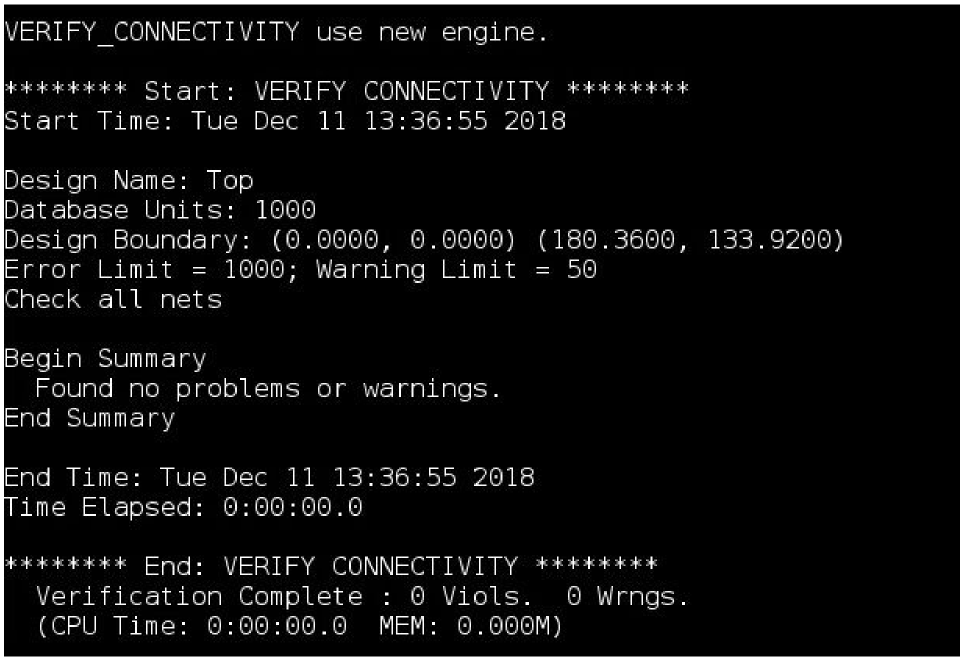
```
  VERIFY GEOMETRY ...... Creating Sub-Areas
                  ...... bin size: 3600
  VERIFY GEOMETRY ...... SubArea : 1 of 1
  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:00.2  MEM: 0.0M)
```

The DRC verification:



```
 VERIFY DRC ...... Starting Verification
 VERIFY DRC ...... Initializing
 VERIFY DRC ...... Deleting Existing Violations
 VERIFY DRC ...... Creating Sub-Areas
 VERIFY DRC ...... Using new threading
 VERIFY DRC ...... Sub-Area : 1 of 1
 VERIFY DRC ...... Sub-Area : 1 complete 0 Viols.

 Verification Complete : 0 Viols.

*** End Verify DRC (CPU: 0:00:00.2  ELAPSED TIME: 0.00  MEM: 0.0M) ***
```

The connection verification:



```
VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Tue Dec 11 13:36:55 2018

Design Name: Top
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (180.3600, 133.9200)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Tue Dec 11 13:36:55 2018
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)
```

There is no violation in the verification.

7.0 Conclusion

In summary, we were able to design a functional MLP accelerator. Our design had only a small amount of error, which is acceptable when taking into account the fact that we are using 16-bit fixed point numbers where we need to truncate each multiplication and there is no rounding. Our design was also very far from exceeding the maximum timing, area, and power requirements. The total execution time was required to be less than 10ms and our design took only 56.69 us. The area of our design was required to be less than 1000 mm^2 and the are of our design was only 1.27033 mm^2. The power of our design was required to be under 10W and the total power of our design was only 30uW.

We were also able to complete the place and route of our design and verified that there were no violations. We attempted to to complete post-layout verification simulation by using the netlist after place and route to do the post-implementation simulation with the library files. However, we encountered errors that we were not able to solve because the library files were written in VHDL. This issue is something to explore further in the future.