# ECE 464/564 Projects : Fall 2017

***Students work on these projects individually***. *Reminder: Collaboration is encouraged but the sharing of Verilog code is strictly forbidden. Also, group creation of one set of common code is forbidden. We will be running code comparison tools to look for even partial sharing of code.*

**Changes from previous versions:**

- More detail on memory interface for 464/EOL project
- Memory map modified for 564 project

**General information for both projects**

A top level module will be provided that instantiates:

1. Vector memory.
2. Input data memory
3. Output memory

A testbench will be provided that instantiates the top level module and:

4. Provides a one-cycle wide go signal to start your design.
5. Waits for your design to send the finish flag high.
6. Counts the number of clock cycles it took to finish.
7. It will check your output for correctness

Note, after completing one full calculation, your design should be ready for the go flag to go high again and start another calculation.

Your design is implemented in a separate file MyDesign.v. Please make sure to synthesize only MyDesign.v and NOT the test fixture, nor the SRAM.

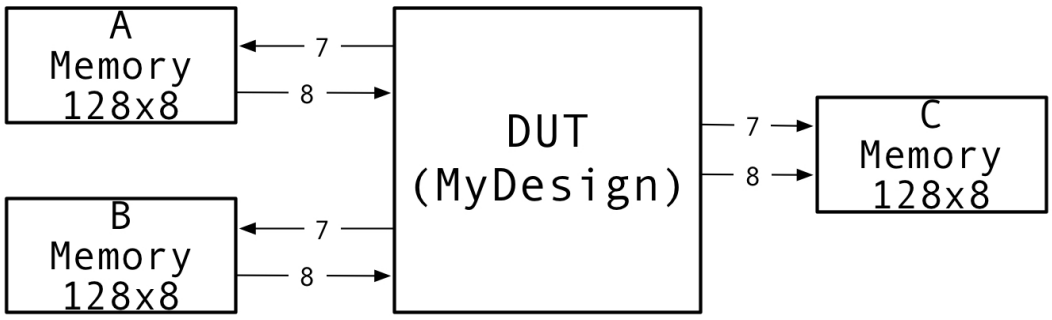
## ECE 464 and EOL Project

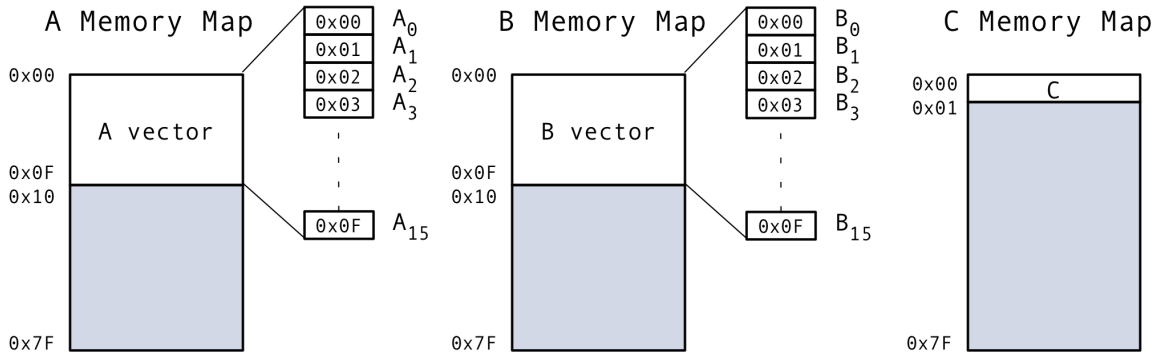
Your project is to design and implement hardware that conducts the convolution of two vectors. Reminder. The vectors will be 16 digits long, and will consist of 8-bit integers stored in a two's complement format. The result is a 16-bit number. The multipliers are to produce 16 bit outputs.
A reminder that a convolution involves multiplying the two vectors A and B to produce a single number C as follows.

$$C = \sum_{i=0}^{i=15} A[i].B[i]$$

Additional specifications:

- As shown in the block diagram, the vectors A and B will be stored in two separate SRAMs, each with their own interface to your design. The memories will show as instances A and B in modelsim.
- The result C will be produced as an output of your design and written into its 'C' memory. Only one location is used in the C memory.
- You are to detect overflow and send an overflow high if overflow is detected. Note this is a signed overflow. You can't just use carry_out. Overflow is to stay high until go is asserted again.
- You can use Synopsys DesignWare or design your own arithmetic units.
- A test fixture and SRAMs are given to you.
- Please register both the address and data registers to the SRAM. You read the data one clock cycle after the address changes. Please see the class recording.




Note: Do not use

## ECE 564-001 Project

Your project is to design and implement hardware that performs two stages of operations on an input array and generate an output vector.

This is a simplified version of a Convolutional Neural Network, a common algorithm used in machine learning. We will be implementing two layers, the first layer is a feature extraction from an input and the second layer is a fully connected layer to identify classes.

This example is completely contrived and the task is to perform operations on provided data so do not get concerned about the meaning of any values generated.

We are essentially performing a number of dot-products of two vectors. These vectors are extracted from the inputs and from provided vectors.

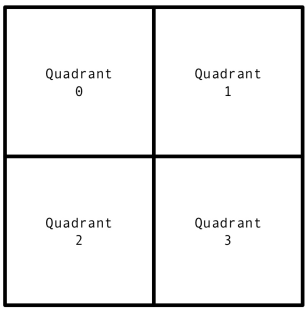
Remember, the dot-product of two vectors [x] and [y] is:

$$x = [x_0, x_1 ... x_{n-1}]$$
$$y = [y_0, y_1 ... y_{n-1}]$$
$$z = \sum_{n=0}^{n-1} x_i \cdot y_i$$

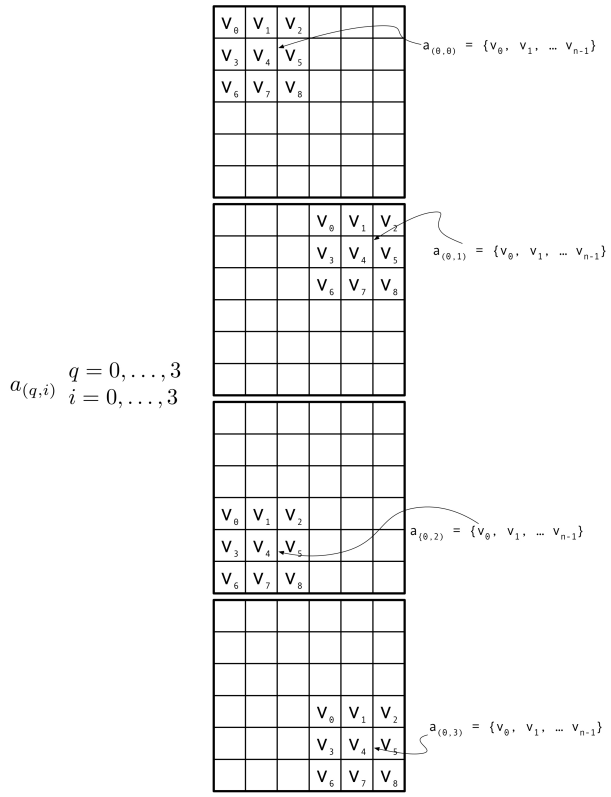
The project input is a 12x12 array of 16-bit twos-complement numbers. The final output will be an 8x1 vector of 16-bit twos-complement numbers stored in memory. Storage details will be described later. The entire operation will be performed with two steps.

### Step One

The first step is to first perform dot-products on regions of the inputs using four separate step-one vectors. We will refer to these as the $[b]_{0..3}$ vectors and they will be supplied in the "Filter Vector memory" as shown in the block diagram and memory map. Each of these [b] vectors are 9x1.
You will consider the 12x12 input as four 6x6 quadrants and operate on each of these quadrants in an identical manner.




From each quadrant, you will form four 1x9 vectors, as shown in the figure below. We will refer to these vectors as [a(q,i)].



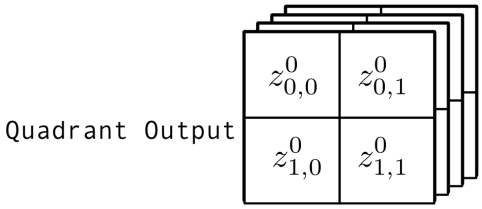

You will then perform a dot-product of each of these $[a_{(q,i)}]$ vectors against each of the four supplied step one $[b]_{0..3}$ vectors.

$$b = [b_0, b_1 ... b_8]$$
$$a = [a_0, a_1 ... a_8]$$
$$c = \sum_{n=0}^{8} a_i \cdot b_i$$

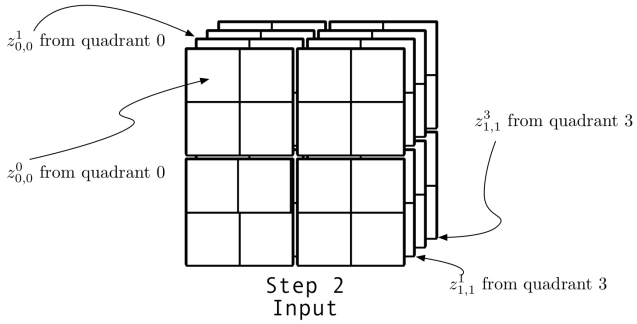
This will create four int32's c values which we will form into consider to be a 2x2 array. For each [c], perform the following min function and truncate the result z to 16-bits.

$$z_{i,j}^b = f(c_{i,j}^b) = \begin{cases} c_i & , c_i \geq 0 \\ 0 & , otherwise \end{cases} \text{ where } b = 0 \ldots 3, i = 0, 1, j = 0, 1$$

Each quadrant will require 16 dot products.
This operation on each quadrant will form a 2x2x4 output array.




We will now take combine the four 2x2 quadrant outputs and form a 4x4x4 array as shown in the figure below which will be the input to step two.




## Step Two

In step two, we again will be to perform dot products on this step one output but we will use eight supplied [m] vectors. These [m] vectors are supplied in the "Filter Vector memory" as shown in the block diagram and memory map.

These eight [m] vectors have 64 elements and you will perform a dot product of each [$m_i$], i=0..7 vector(s) against a 64-element vector formed from the 4x4x4 output of step one. To reshape the 4x4x4 output of step 1 to create a 1x64 [u] vector, you will extract the elements in row-major fashion starting at the first layer.

Again each dot-product will be followed by the function f(x) as shown below.

$$m_i = [m_{i,0}, m_{i,1} ... m_{i,63}]$$
$$u = [u_0, u_1 ... u_6 3]$$
$$w_i = \sum_{n=0}^{63} m_i \cdot u, i = 0, \ldots, 7$$

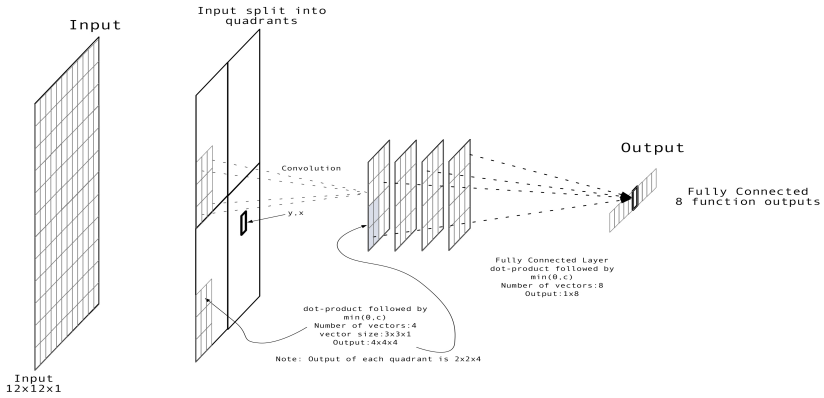
$$O_i = f(w_i) = \begin{cases} w_i & , w_i \geq 0 \\ 0 & , otherwise \end{cases} \text{ where } i = 0 \ldots 7$$

This will result in eight outputs, O and this will be the output of our system.

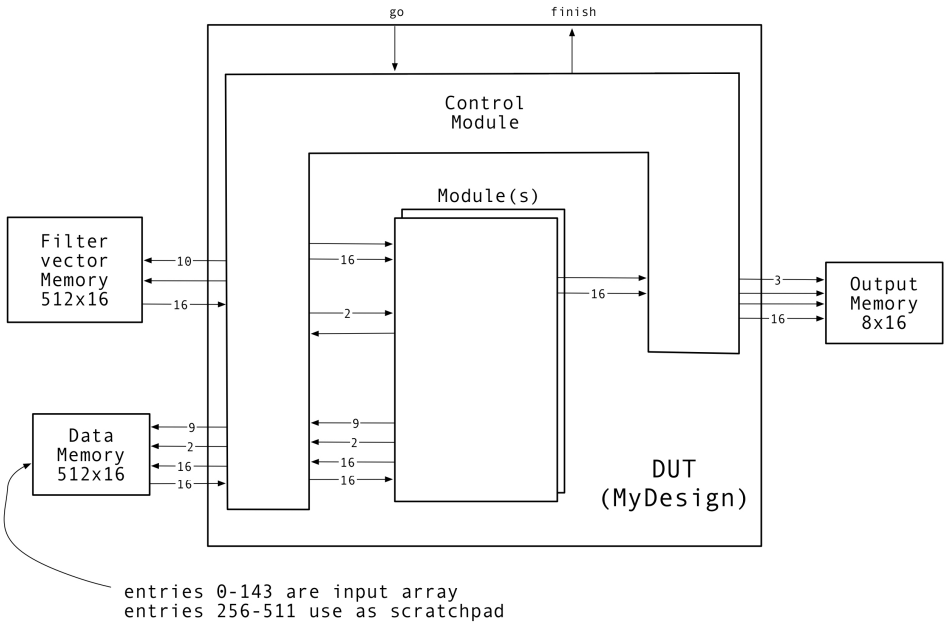
You will be provided with matlab code that demonstrates all the steps. That code is attached to the end of this document.

## Summary

Below is an overall flow diagram of our system. If you do a search for neural networks, you will see similar images but likely much more complicated with many more layers.

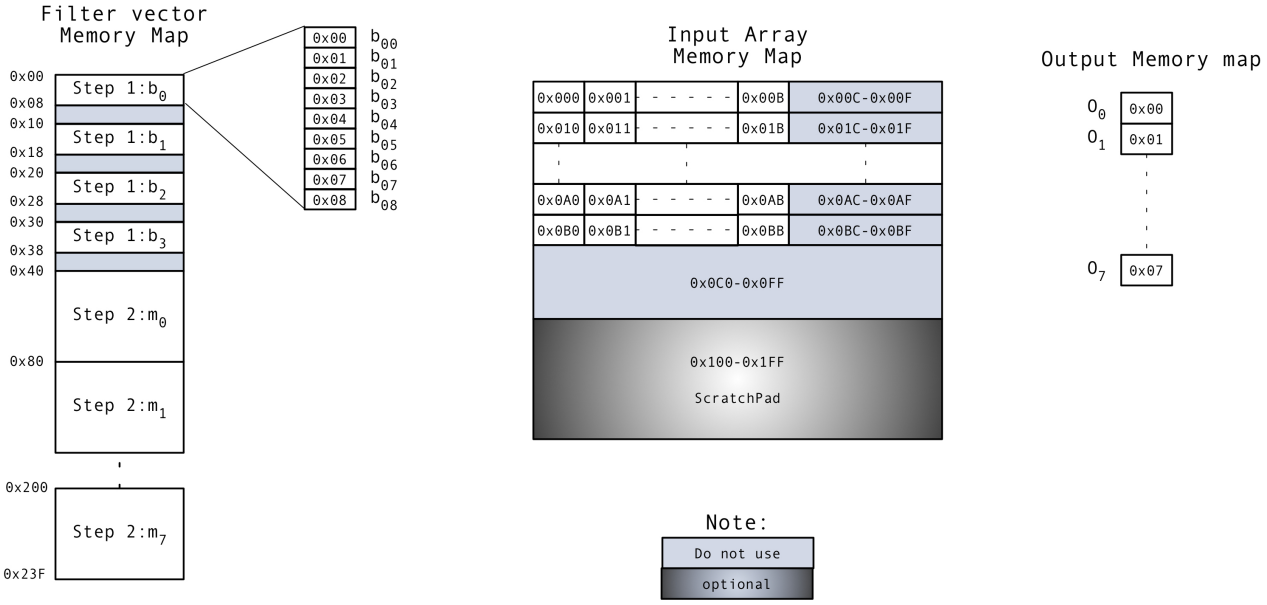



Below is a block diagram.




You are required to use a common module for the step-one quadrant operations.
You can implement step-two as you wish.

Here are the storage memory maps




## Comments

Additional specifications:

- You can use Synopsys DesignWare or design your own arithmetic units.
- A test fixture and SRAMs will be provided.
- If you choose to store some of your temporary data in local memory, use the scratchpad area on the Input memory

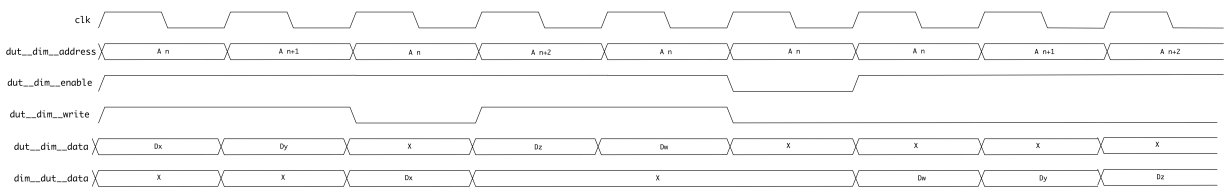
Note: Again, when multiplying int16s you can truncate the resulting int32 to int16

# Interfaces

## Signals

| Direction | Type | Width (Bus) | Name | Comment |
|---|---|---|---|---|
| Control | | | | |
| output | reg | | dut__xxx__finish | High when DUT is ready for a ‘go’. Deassert after ‘go’ |
| input | wire | | xxx__dut__go | Pulsed |
| Filter Vector Memory | | | | |
| output | reg | [9:0] | dut__bvm__address | |
| output | reg | | dut__bvm__enable | High for Read |
| output | reg | | dut__bvm__write | Low for Read |
| output | reg | [15:0] | dut__bvm__data | Write Data |
| input | wire | [15:0] | bvm__dut__data | Read Data (do not use) |
| Input Data memory | | | | |
| output | reg | [8:0] | dut__dim__address | |
| output | reg | | dut__dim__enable | High for Read and Write |
| output | reg | | dut__dim__write | High for write |
| output | reg | [15:0] | dut__dim__data | Write Data |
| input | wire | [15:0] | dim__dut__data | Read Data |
| Output Data Memory | | | | |
| output | reg | [2:0] | dut__dom__address | |
| output | reg | [15:0] | dut__dom__data | Write Data |
| output | reg | | dut__dom__enable | High for Write |
| output | reg | | dut__dom__write | High for write |
| General | | | | |
| input | wire | | clk | |
| input | wire | | reset | Active high |

## Example RAM Signaling




## Compile and Simulation

```
vlog -sv ece564_project_tb_top.v
vlog -sv MyDesign.v
vsim -c -do "run 2us; quit" tb_top
```

You can display the expected intermediate and output results by adding defines to vlog

```
vlog -sv +define+TB_DISPLAY_INTERMEDIATE+TB_DISPLAY_EXPECTED ece564_project_tb_top.v
```

**Example Matlab Code**

```
clear all

% IA is the Input Array, 12x12 of int16's
IA = [
 12579   -3150   15584   18357   17791   -2942   11576  -10913  -17007    5883  -16281    4586  ;
 17282   -2288   13897  -15256   14622   -8114    4530     354   10408   -2975   -9824   -4870  ;
-15713   -1399    5228   -7920   16784  -13830  -19651    7785  -11515  -14584    9978  -18716  ;
  7204   10017    2416    6906   11613    5368  -14010    1150  -14404  -11842    -988   -3131  ;
 -1294  -16093   14075     674   17598  -10939   12993  -10468    2774   14857  -10516    8477  ;
  1046  -16696    7316   -8374   17078    2488   -6603   -3255   16833   -5312  -17074  -14117  ;
 -7497   10285   15998   17310   -6240  -10261   11635   -8166   -1298   -9664   19437  -10346  ;
  1803   -7383    3043   11441   -8204  -15498    3125    6197    6087  -14690   10023     350  ;
  4652   15106    9792    6845  -14539     526   -2263  -11101  -19570   -9435   14624  -13441  ;
-12763  -17856   -2607  -17508  -12639    9767   -9820   18726  -18579   11375   -3556    2414  ;
 -6442    8947  -13008    6905   -4020   -2138   15799  -15314    5561   10729    5826   -4808  ;
-18724   18980   14963    4153  -13879   17325    -539  -17869  -11901    8844  -17182   18942  ;
]
% B contains the 1x9 step 1 filters
B = [
-13435   18765   -5643  -17177   -6822  -10447    9571   14312   -1344  ;
 16548   13336   10838   18696   -3168  -17654   12477  -12215    9841  ;
 17993   16168   16844   11142    8397   -9046   -1313  -16437    -664  ;
 -3392    5206   13799  -15760  -19918   -2012   19412    -681    6108  ;
]

% M contains the 1x64 step 2 filters
M = [
  6552    -800   -3633    6451  -14670    2147   14429   16916   -7814    9927   -7934    8302   12070   15606   -5184  -19316
7053  -11087   -2160  -19321  -17148  -17229   -5819   17906    7947   10611     778   16963   -9775    8800  -11410   -9511
16682   12481    3719    -377   -3268     500  -13696   -5190   -2846   -4759   12944   15168   13738   13869    4613  -10473   -
9678    7692  -14696   19406   16754   13321   12251    1078    6424    4556   -6934  -13128   13072    1582   14382  -12906
 12481    3719    -377   -3268     500  -13696   -5190   -2846   -4759   12944   15168   13738   13869    4613  -10473   -9678
7692  -14696   19406   16754   13321   12251    1078    6424    4556   -6934  -13128   13072    1582   14382  -12906   -1719
13430    4099   17376   15135      82   -3912   12554   17096   -2388   -7901    1226     192   -1372   19951    9621   -8650   -
3577   10042    5175  -10450   -8363   19708   11848  -15060   -5477    6772  -17179    5035   -9371   10397   14399    4601
  4099   17376   15135      82   -3912   12554   17096   -2388   -7901    1226     192   -1372   19951    9621   -8650   -3577
10042    5175  -10450   -8363   19708   11848  -15060   -5477    6772  -17179    5035   -9371   10397   14399    4601    4653
3141   16705  -11752   -1081    4460   18390  -18435  -14048  -14382    -956   -7805   14318   18415  -14857  -16649    5785
9625   17440   -2441   -8158      60  -17944    7089    3595   -6087   17463    5771    4848    6052  -14606   11214  -18554
 16705  -11752   -1081    4460   18390  -18435  -14048  -14382    -956   -7805   14318   18415  -14857  -16649    5785    9625
17440   -2441   -8158      60  -17944    7089    3595   -6087   17463    5771    4848    6052  -14606   11214  -18554   -9802
16875     -74   18206   17451    2992   15689    6901   -6276   -4785    8315    2677   -6945  -18381  -10808  -15845   17088
15546   12952  -10774   -4272   -8985   -1767  -12301   -8910     720   12795  -16903   -5940  -10449   13352   16556   -3178
   -74   18206   17451    2992   15689    6901   -6276   -4785    8315    2677   -6945  -18381  -10808  -15845   17088   15546
12952  -10774   -4272   -8985   -1767  -12301   -8910     720   12795  -16903   -5940  -10449   13352   16556   -3178   18727  -
17934   17382  -12850   12813   18484   -9414    -694   13048   16368    3355  -18129   -9544   12317   12066   12346  -16019
7482   -5032  -13298   -5429   -7977   -8828    -423   17239    1692   17681   -5198   17701    1989     334  -19044    1651
 17382  -12850   12813   18484   -9414    -694   13048   16368    3355  -18129   -9544   12317   12066   12346  -16019    7482
-5032  -13298   -5429   -7977   -8828    -423   17239    1692   17681   -5198   17701    1989     334  -19044    1651   14863
7012  -15411   -5684    3241    9164  -18961  -18871  -11400    5662    5821   14534    9726   -8166   17357   -8168  -11369
418   18802    7888   -3270   -8276    1313  -13767  -19231   -6484   10269   12214   12310   18349    4042   -2632    2873
-15411   -5684    3241    9164  -18961  -18871  -11400    5662    5821   14534    9726   -8166   17357   -8168  -11369     418
18802    7888   -3270   -8276    1313  -13767  -19231   -6484   10269   12214   12310   18349    4042   -2632    2873   14739
11319   18443  -18816   17807  -17250    9880   14769   -6547   19831    1443  -13984   18314   16567  -18416   -5596   13145
11921    4800    8479  -16934  -14679   -9020    6263   -9520   15951   -3107   -7797   -2818   19111   18960  -11103  -16974
 18443  -18816   17807  -17250    9880   14769   -6547   19831    1443  -13984   18314   16567  -18416   -5596   13145   11921
4800    8479  -16934  -14679   -9020    6263   -9520   15951   -3107   -7797   -2818   19111   18960  -11103  -16974  -19153   -
6163   -7099   -5274  -12097    4769  -15841  -14077   16502   -4121   -9038  -13169   16474  -11986   19217   -1112    3041
17322   15436    4818   -5102    6492    7659   -9416   12056   12759   11542  -18679    6085   19202    5663    6544  -18406
]

% Extract the four 6x6 regions of the input array, we will calls these Quadrants
Quadrant0 = IA(1:6,1:6)
Quadrant = Quadrant0
Quadrant(:,:,2) = IA(1:6,7:12)
Quadrant(:,:,3) = IA(7:12,1:6)
Quadrant(:,:,4) = IA(7:12,7:12)

% Perform a dot-product of the b-vector with each corner of the ROI

C = zeros(2,2);
for layer = 1:4
    for QuadrantN = 1:4

        A_1_1  = [Quadrant(1,1:3,QuadrantN) Quadrant(2,1:3,QuadrantN) Quadrant(3,1:3,QuadrantN)];
        A_1_2  = [Quadrant(1,4:6,QuadrantN) Quadrant(2,4:6,QuadrantN) Quadrant(3,4:6,QuadrantN)];
        A_2_1  = [Quadrant(4,1:3,QuadrantN) Quadrant(5,1:3,QuadrantN) Quadrant(6,1:3,QuadrantN)];
```

```
        A_2_2  = [Quadrant(4,4:6,QuadrantN) Quadrant(5,4:6,QuadrantN) Quadrant(6,4:6,QuadrantN)];
        C(1,1) = A_1_1*B(layer,:)';
        C(1,2) = A_1_2*B(layer,:)';
        C(2,1) = A_2_1*B(layer,:)';
        C(2,2) = A_2_2*B(layer,:)';

        % apply f(x) and truncate
        Z = max(0,C)/(2^16)

        Zq(:,:,layer,QuadrantN) = Z;

    end
end

% Now operate on the four 2x2x4 quadrants as if it were a 4x4x4 array
% Our matlab quadrant array is organized as (y,x,ROI,layer)
% So lets just merge them
% Remember, a layer is generated from each B vector
for layer = 1:4
    Zmerged(:,:,layer) = [Zq(:,:,layer,1) Zq(:,:,layer,2) ;
                          Zq(:,:,layer,3) Zq(:,:,layer,4) ]
end

% Create a 1x64 vector from a merged array
% We create the vector using row major layer by layer
%
% use reshape on the transpose because reshape uses column major
U = [];
for layer = 1:4
    U = [U reshape(Zmerged(:,:,layer)', [1 16])];
end

% Now dot-product Qv with each output m-vector
W = [];
for o = 1:8
    W = [W M(o,:)*U'];
end

% Now apply our f(x)
O = max(0,W);
% truncate
O = O/(2^16);
% Expected output
O
```