Preliminary Report Format

/10

Project Responsibilities: Design, Implementation, Testing, Final Report
StudentID. Name 1: jajeffe2, John Jefferson III

Summary Risk Plan:
A major risk for this design is the size of the module being too large with this implementation of the design and taking very long to test and diagnose.

Another risk is not being able to efficiently and effectively utilize the designware libraries to produce the required mathematical output correctly.

I plan on mitigating these risks by writing the code very modularly for testing purposes and reading over the documentation provided in the designware links given.

Schedule:
Project Plan – 10/14/2017
Designware of DP – 10/18/2017
Pseudocode Design – 10/22/2017
Datapath (Math) – 10/24/2017
Controller (Memory) – 10/25/2017
Debugging – 10/30/2017
Optimization (Designware) – 11/4/2017
Final Report – 11/6/2017
Final Project – 11/6/2017

Brief Description of Mode of operation, including selected algorithms:

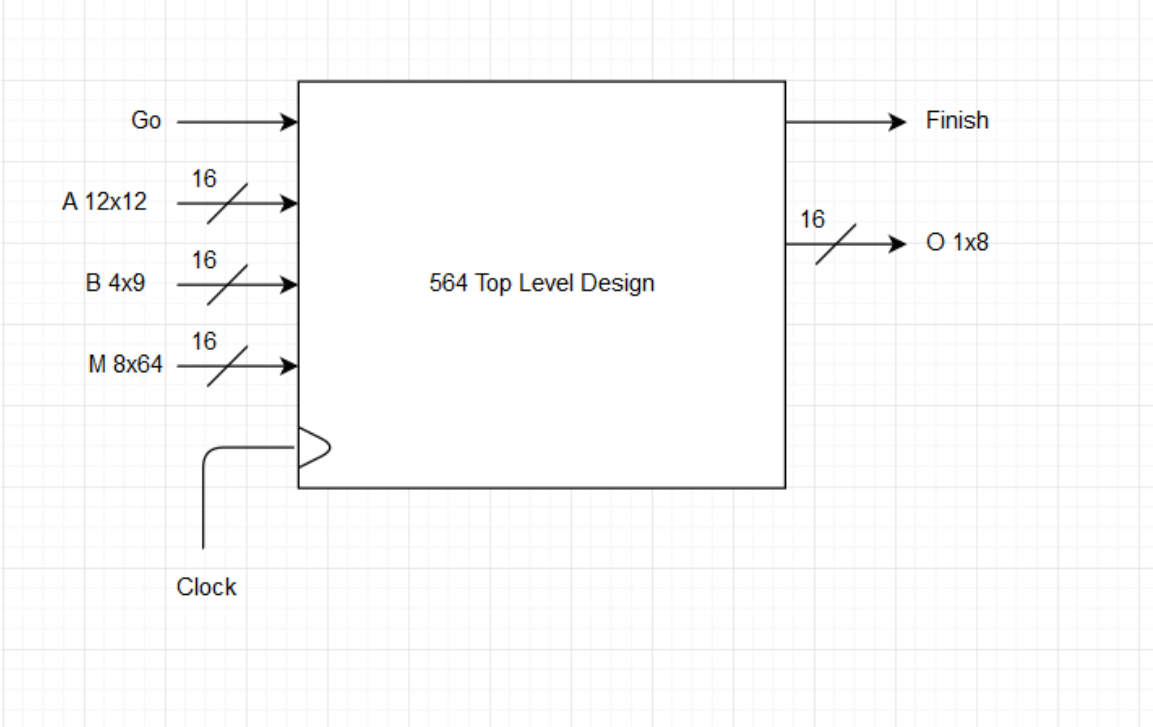
A method that could be used is to read the values from A into a buffer on the module from the beginning, giving up size constraints but maximizing speed. The data can be accessed from a buffer using indexing for passing into the Designware dot product instantiation of my choice and then rectified using a comparator. Some of the operations that are done, like the dot product, rectification, and matrix manipulation, can all be run in parallel with another level of module calls. The algorithm for both sets of the functions for the dot products will be the same but the sizing will be different. More testing needs to be done on the Designware library to optimize for sizing.

As for accessing memory, since I am reading the matrix A into a buffer from the beginning no lasting effects from the memory read to affect the module operation. Same for the output since I am writing to the SRAM only one time after the operation is finally complete and a new input array can be read in.

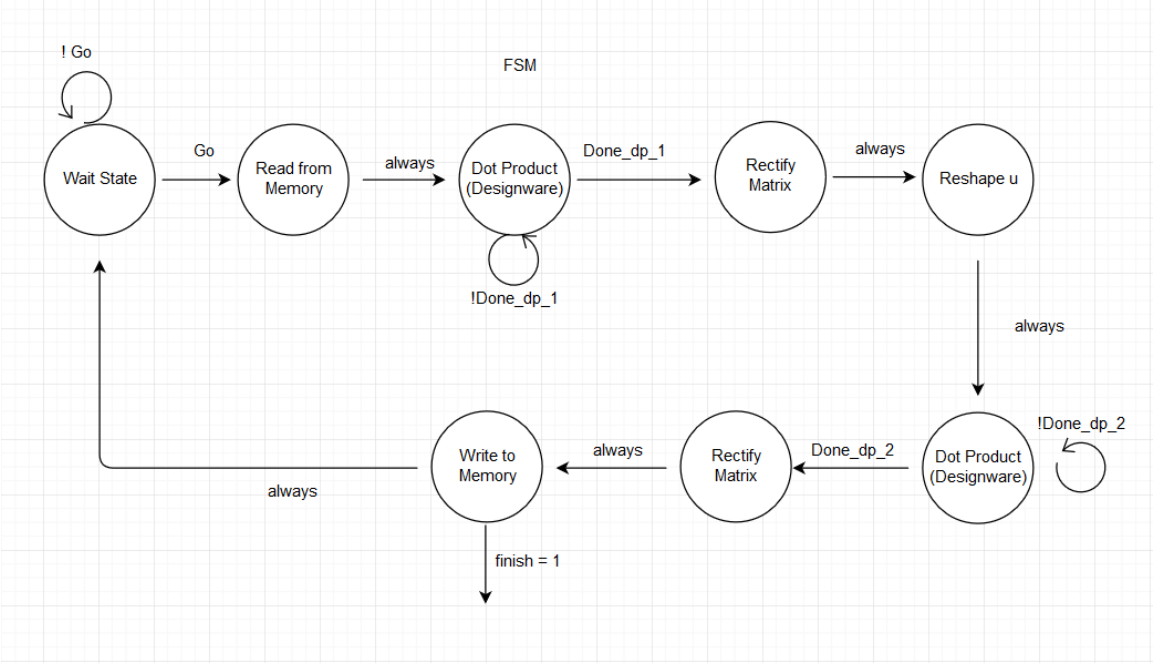
High level sketch.

On the following pages

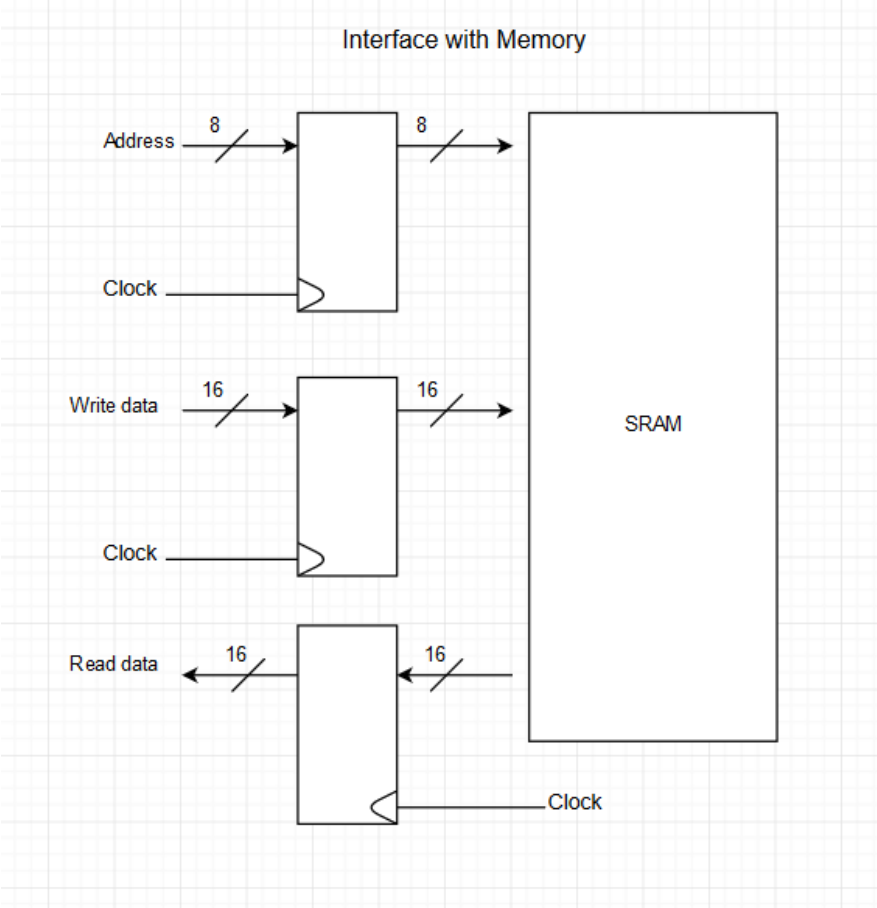
Top Level:



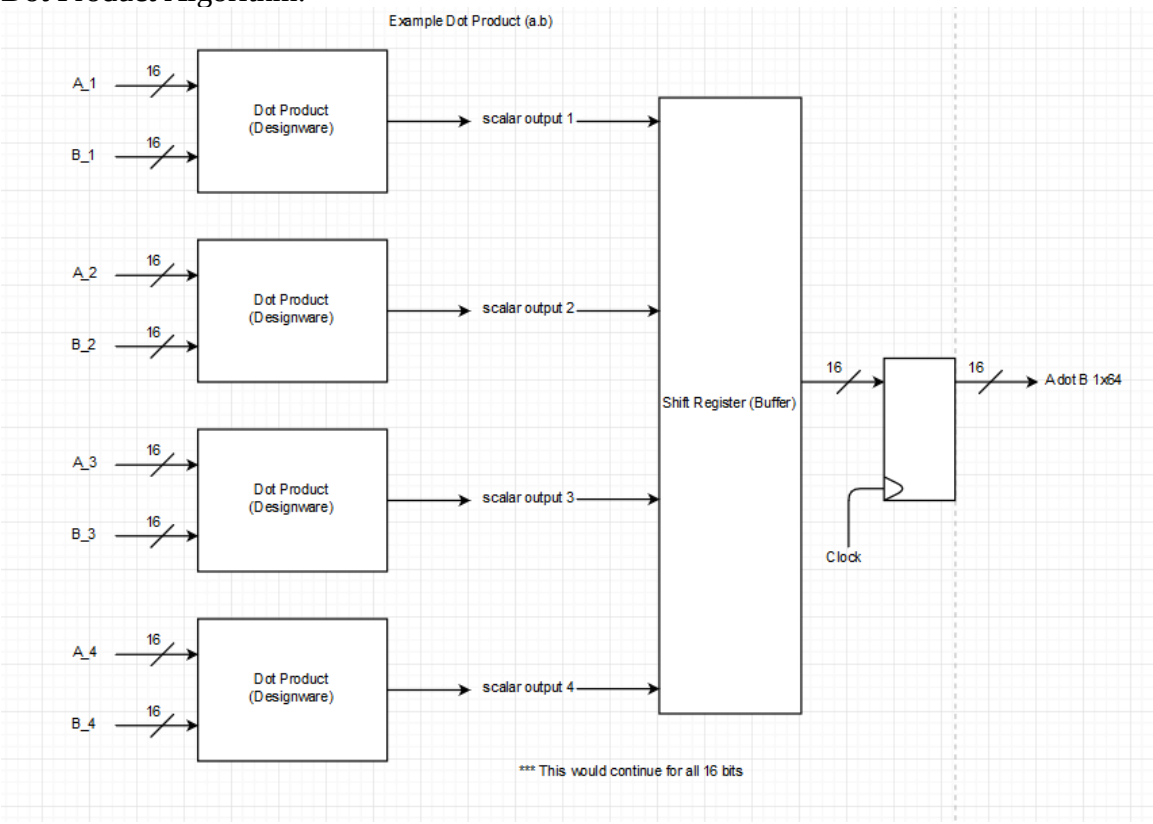
Finite State Machine:



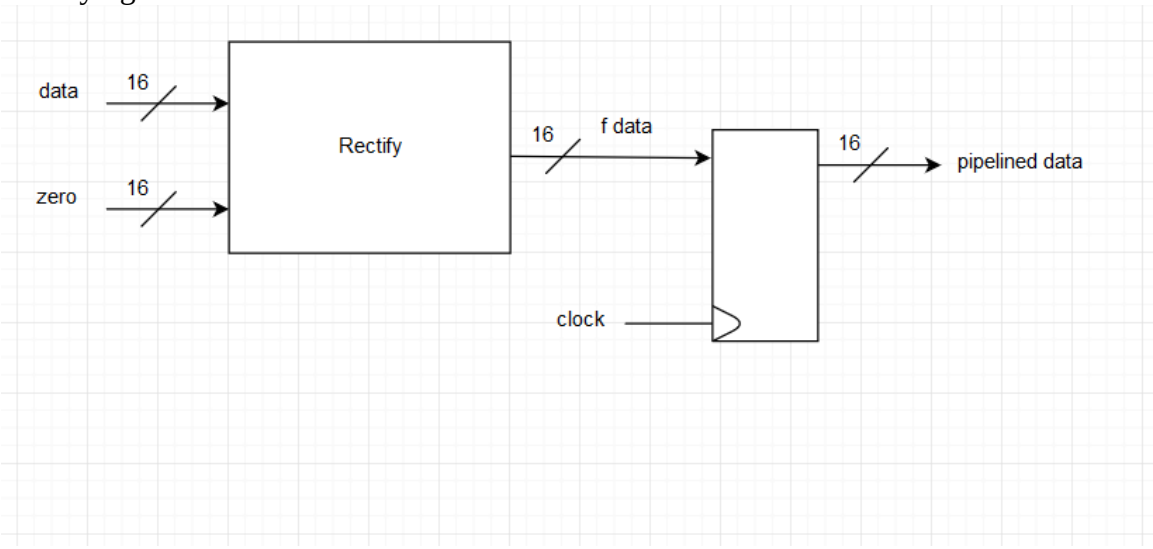
Memory Interface:



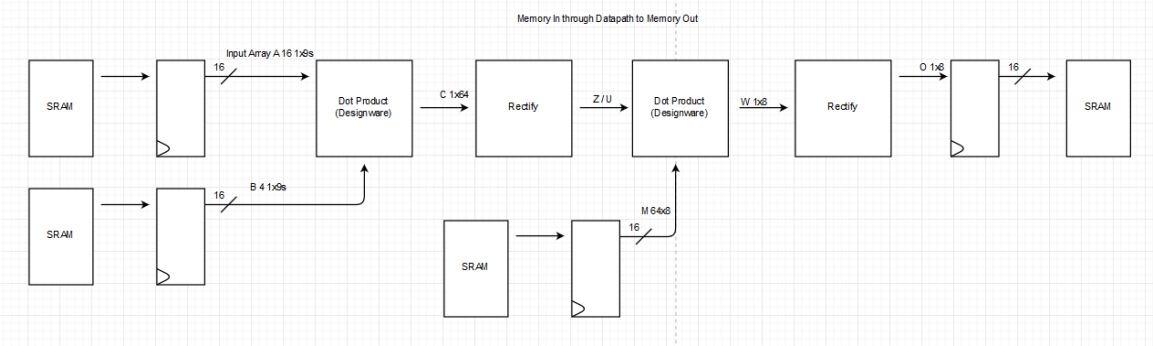
Dot Product Algorithm:



Rectifying Function:



End to End Design:



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
        sign_C_1_1  = sign(C(1,1)) ;
        C_1_1  = dec2bin(abs(C(1,1)),32) ;
        C_1_1  = C_1_1(1:16) ;
        C_1_1  = bin2dec(C_1_1) ;
        C(1,1)  = sign_C_1_1*C_1_1;
        sign_C_1_2  = sign(C(1,2)) ;
```

```
        C_1_2  = dec2bin(abs(C(1,2)),32) ;
        C_1_2  = C_1_2(1:16) ;
        C_1_2  = bin2dec(C_1_2) ;
        C(1,2)  = sign_C_1_2*C_1_2;
        sign_C_2_1  = sign(C(2,1)) ;
        C_2_1  = dec2bin(abs(C(2,1)),32) ;
        C_2_1  = C_2_1(1:16) ;
        C_2_1  = bin2dec(C_2_1) ;
        C(2,1)  = sign_C_2_1*C_2_1;
        sign_C_2_2  = sign(C(2,2)) ;
        C_2_2  = dec2bin(abs(C(2,2)),32) ;
        C_2_2  = C_2_2(1:16) ;
        C_2_2  = bin2dec(C_2_2) ;
        C(2,2)  = sign_C_2_2*C_2_2;

        Z = max(0,C)

        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
```

Testbench Code for Testing:

```
//-----------------------------------------------------------------------------------------------
// To run simulation
//
// vlog -sv ece564_project_tb_top.v
// vsim -c -do "run 1us; 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

//----------------------------------------------------------------------
//----------------------------------------------------------------------
// SV Interface to DUT
interface dut_ifc(
                input bit clk   );
```

```
//------------------------------------------------------------------------------
// Control
//
logic               dut__xxx__finish   ;
logic               xxx__dut__go       ;

//------------------------------------------------------------------------------
// Filter-vector memory
//
logic   [ 8:0]        dut__bvm__address  ;
logic   [15:0]        dut__bvm__data     ;  // write data
logic   [15:0]        bvm__dut__data     ;  // read data
logic               dut__bvm__enable   ;
logic               dut__bvm__write    ;

//------------------------------------------------------------------------------
// Input data memory
//
logic   [ 8:0]        dut__dim__address  ;
logic   [15:0]        dut__dim__data     ;  // write data
logic   [15:0]        dim__dut__data     ;  // read data
logic               dut__dim__enable   ;
logic               dut__dim__write    ;


//------------------------------------------------------------------------------
// Output data memory
//
logic   [ 2:0]        dut__dom__address  ;
logic   [15:0]        dut__dom__data     ;  // write data
logic               dut__dom__enable   ;
logic               dut__dom__write    ;


//--------------------------------
// General
//
logic               reset          ;

clocking cb_test @(posedge clk);

  default input #1 output #1;

  output      reset             ;

  input       dut__xxx__finish   ;
  output      xxx__dut__go       ;
  input       dut__bvm__address  ;
  input       dut__bvm__data     ;
  input       bvm__dut__data     ;
  input       dut__bvm__enable   ;
  input       dut__bvm__write    ;
  input       dut__dim__address  ;
  input       dut__dim__data     ;
  input       dim__dut__data     ;
  input       dut__dim__enable   ;
  input       dut__dim__write    ;
  input       dut__dom__address  ;
  input       dut__dom__data     ;
  input       dut__dom__enable   ;
  input       dut__dom__write    ;

endclocking : cb_test

clocking cb_dut @(posedge clk);

  default input #1 output #1;
endclocking : cb_dut

modport DUT (
```

```
   clocking   cb_dut             ,

   input      reset              ,

   output     dut__xxx__finish   ,
   input      xxx__dut__go       ,
   output     dut__bvm__address  ,
   output     dut__bvm__data     ,
   output     bvm__dut__data     ,
   output     dut__bvm__enable   ,
   output     dut__bvm__write    ,
   output     dut__dim__address  ,
   output     dut__dim__data     ,
   output     dim__dut__data     ,
   output     dut__dim__enable   ,
   output     dut__dim__write    ,
   output     dut__dom__address  ,
   output     dut__dom__data     ,
   output     dut__dom__enable   ,
   output     dut__dom__write


);
  modport TB  (clocking cb_test );

  function void loadRam( int              mem_inst    ,
                         int              config_addr ,
                         logic [15:0   ]  config_data );

    if (mem_inst == 0)
      tb_top.dim_mem.mem[config_addr]  = config_data ;
    else if (mem_inst == 1)
      tb_top.bvm_mem.mem[config_addr]  = config_data ;
    else if (mem_inst == 2)
      tb_top.dom_mem.mem[config_addr]  = config_data ;

  endfunction

endinterface : dut_ifc

typedef virtual  dut_ifc    vDutIfc;
//
//

//-----------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------
//  This class generates input data and predicts result(s)

package operation;


    //-----------------------------------------------------------------------------------------------------
    // Class - base_operation
    //
    // Contains the inputs array, b-vectors and weights
    // Generates expected quadrant and output results
    //
    //

    class base_operation ;

        //-----------------------------------------------------------------------------------------------------
        // for Debug
        time             timeTag         ;
        int              tId             ;  // transaction number
        int              tolerance = 2   ;  // +/- expected value
        //-----------------------------------------------------------------------------------------------------
        // All process data from input to output
        //  - input and b/m vectors will randomized although you will see consistentcy from run to run
```

```
rand  shortint signed    inputArray       [12] [12]            ;  // 12x12 input

      shortint signed    ROI              [2]  [2]  [6] [6]    ;  // input array quadrant
rand  shortint signed    bVectors         [4]  [9]             ;  // 1x9 vectors

      shortint signed    quadrants        [4]  [2]  [2] [2] [2] ;  // each quadrant is 4x2x2

      shortint signed    quadrantsMerged  [4]  [4]  [4]        ;

rand  shortint signed    mVectors         [8]  [64]            ;
      shortint signed    outputArray      [8]                  ;

bit  [7:0]               outputStatus                          ;

rand  shortint signed    scratchPad       [256]                ;  // 256 entry scratch pad

//------------------------------------------------------------------------------------------------------------
// Randomization help



//------------------------------------------------------------------------------------------------------------
// Temporary variable used to calculate expected results

logic [15:0] tmpVec [9] ;
int          tmpInt     ;


//------------------------------------------------------------------------------------------------------------
// New

function new ();
          this.tId         = tId    ;
          this.timeTag     = $time  ;
endfunction


//------------------------------------------------------------------------------------------------------------
// Pre randomize

function void pre_randomize();              //1 -> Turns on the constraint, 0-> Turns off the constraint
   this.c_range .constraint_mode(1) ;
endfunction : pre_randomize


//------------------------------------------------------------------------------------------------------------
// Constraints
//
// consstrain input and b/m vectors
constraint c_range {
   foreach (inputArray[i,j]) {
     inputArray [i][j] inside {[-20000:20000]} ;
   }
   foreach (bVectors[i,j]) {
     bVectors [i][j] inside {[-20000:20000]} ;
   }
   foreach (mVectors[i,j]) {
     mVectors [i][j] inside {[-20000:20000]} ;
   }
}


//------------------------------------------------------------------------------------------------------------
// Post randomize

function void post_randomize();

   // WHen we randomized, we use this method to now calculate expected results from the values after they are randomized

   //-------------------------------------------------------------------------------------------------------
```

```
        // Calculate expected results for each step and the 1x8 output vector

        // extract regions of interest from input array for each quadrant
        for (int quadY=0; quadY<2; quadY++)
          begin
            for (int quadX=0; quadX<2; quadX++)
              begin
                for (int row=0; row<6; row++)
                  begin
                    for (int col=0; col<6; col++)
                      begin
                        ROI [quadY][quadX][row][col] = inputArray[quadY*6+row][quadX*6+col];
                      end
                  end
              end
          end

        // Now create quadrant data from the ROI
        //  -  dot-product of the 9-element b-vectors with each of the four 3x3 sections of the ROI
        //  - 16 dot-products creating a 4x2x2 output quadrant array
        //
        $display("%0d %0d", $size(inputArray), $size(bVectors));
        for (int quadY=0; quadY<2; quadY++)
          begin
            for (int quadX=0; quadX<2; quadX++)
              begin
                // for each quadrant, dot-product of each corner of ROI with each b-vector
                for (int layer=0; layer<4; layer++)
                  begin
                    // the quadrant if broken into the four 3x3 corners
                    for (int row=0; row<2; row++)
                      begin
                        for (int col=0; col<2; col++)
                          begin
                            tmpInt = 0 ;
                            for (int e=0; e<$size(bVectors[layer]); e++)
                              begin
                                // multiple-accumulate the 1x9 b-vector against the flattens 3x3 section of the ROI
                                tmpInt    +=                                    bVectors[layer][e]*ROI[quadY][quadX][row*3+e/3][col*3+e%3];
                                //$display("%0d %0d %0d %0d %0d = %0d %0d x %0d %0d", layer, quadY, quadX, row, col, layer, e,
row*3+e/3, col*3+e%3) ;
                              end
                            // trancate and save in quadrant array
                            quadrants [layer][quadY][quadX][row][col] = (tmpInt < 0)  ? 0 : tmpInt[31:16] ;
                          end
                      end
                  end
              end
          end

        // Now merge the four quadrants into a 4x4x4 array for output processing
        //
        for (int layer=0; layer<4; layer++)
          begin
            for (int quadY=0; quadY<4; quadY++)
              begin
                for (int quadX=0; quadX<4; quadX++)
                  begin
                    quadrantsMerged [layer][quadY][quadX] = quadrants [layer][quadY/2][quadX/2][quadY%2][quadX%2];
                  end
              end
          end


        // Create output using dot-product of each 64-element m-vector with the flattened 4x4x4 array
        //
        for (int m=0; m<8; m++)
          begin
            tmpInt = 0;
            for (int layer=0; layer<4; layer++)
```

```
        begin
          for (int Y=0; Y<4; Y++)
            begin
              for (int X=0; X<4; X++)
                begin
                  tmpInt += mVectors[m][layer*16+Y*4+X]*quadrantsMerged [layer][Y][X];
                end
            end
        end
      outputArray[m] = (tmpInt < 0) ? 0 : tmpInt[31:16] ;
    end

  // we now have the expected output result and intermediate values for debug

  // Status
  outputStatus = 8'd0;

endfunction : post_randomize


//------------------------------------------------------------------------------------------------------
// Methods


function void calculateResult();

  // Note: result is calculated in post_randomize

endfunction : calculateResult


// Displays inputs, intermediate values and output
function void displayAll();

  string  fileName;
  int     fd            ;

  fileName = "projects564.log";
  fd = $fopen (fileName , "w");
  $fwrite(fd, "clear all\n\n");

  $display("@%0t :INFO: Input Array ", $time);
  $display("IA = [");
  $fdisplay(fd, "IA = [");
  for (int y=0; y<12; y++)
    begin
      for (int x=0; x<12; x++)
        begin
          $write("%6d  ", inputArray[y][x]);
          $fwrite(fd, "%6d  ", inputArray[y][x]);
        end
        $write(";\n");
        $fwrite(fd, ";\n");
    end
  $write("]\n");
  $fwrite(fd, "]\n");

  `ifdef  TB_DISPLAY_INTERMEDIATE
    for (int quadY=0; quadY<2; quadY++)
        begin
          for (int quadX=0; quadX<2; quadX++)
            begin
              $display("@%0t :INFO: ROI : {%0d,%0d} ", $time, quadY, quadX);
              $display("ROI[%0d] = [", (quadY*2+quadX)+1);
              $fwrite(fd, "ROI(%0d,:,:) = [\n", (quadY*2+quadX)+1);
              for (int row=0; row<6; row++)
                begin
                  for (int col=0; col<6; col++)
                    begin
                      $write("%6d  ", ROI[quadY][quadX][row][col]);
```

```
                $fwrite(fd, "%6d  ", ROI[quadY][quadX][row][col]);
              end
            $write(";\n");
            $fwrite(fd, ";\n");
          end
        $write("]\n\n");
        $fwrite(fd, "]\n\n");
      end
    end
`endif

$display("@%0t :INFO: B-vectors ", $time);
$display("B = [");
$fwrite(fd, "B = [\n");
for (int layer=0; layer<4; layer++)
  begin
    for (int x=0; x<9; x++)
      begin
        $write("%6d  ", bVectors[layer][x]);
        $fwrite(fd, "%6d  ", bVectors[layer][x]);
      end
      $write(";\n");
      $fwrite(fd, ";\n");
  end
$write("]\n\n");
$fwrite(fd, "]\n\n");

`ifdef  TB_DISPLAY_INTERMEDIATE
$display("@%0t :INFO: Layer Quadrant Arrays ", $time);
for (int layer=0; layer<4; layer++)
    begin
      for (int quadY=0; quadY<2; quadY++)
        begin
          for (int quadX=0; quadX<2; quadX++)
            begin
              $display("@%0t :INFO: Layer %0d, Quadrant {%0d,%0d}", $time, layer, quadY, quadX);
              $display("Q(%0d,:,:) = [", layer);
              $fwrite(fd, "Q_%0d_%0d(%0d,:,:) = [\n", quadY+1, quadX+1, layer+1);
              for (int y=0; y<2; y++)
                begin
                  for (int x=0; x<2; x++)
                    begin
                      $write("%6d  ", quadrants[layer][quadY][quadX][y][x]);
                      $fwrite(fd, "%6d  ", quadrants[layer][quadY][quadX][y][x]);
                    end
                    $write("\n");
                    $fwrite(fd, "\n");
                end
              $write("]\n\n");
              $fwrite(fd, "]\n\n");
            end
        end
    end
`endif

$display("@%0t :INFO: M-vectors ", $time);
$display("M = [");
$fwrite(fd, "M = [\n");
for (int cls=0; cls<8; cls++)
  begin
    for (int x=0; x<64; x++)
      begin
        $write("%6d  ", mVectors[cls][x]);
        $fwrite(fd, "%6d  ", mVectors[cls][x]);
      end
      $write("\n");
      $fwrite(fd, "\n");
  end
$write("]\n\n");
$fwrite(fd, "]\n\n");
```

```
    `ifdef  TB_DISPLAY_INTERMEDIATE
    $display("@%0t :INFO: Layers of Merged Array ", $time);
    for (int layer=0; layer<4; layer++)
       begin
         $display("@%0t :INFO: Layer %0d ", $time, layer);
         $display("Qm = [");
         $fwrite(fd, "Qm(%0d,:,:) = [\n", layer+1);
         for (int y=0; y<4; y++)
           begin
             for (int x=0; x<4; x++)
               begin
                 $write("%6d  ", quadrantsMerged[layer][y][x]);
                 $fwrite(fd, "%6d  ", quadrantsMerged[layer][y][x]);
               end
               $write("\n");
               $fwrite(fd, "\n");
           end
         $write("]\n\n");
         $fwrite(fd, "]\n\n");
       end
    `endif

    `ifdef  TB_DISPLAY_EXPECTED
    $display("@%0t :INFO: Output Vector", $time);
    $display("O = [");
    $fwrite(fd, "O = [\n");
    for (int cls=0; cls<8; cls++)
      begin
        $write("%6d  ", outputArray[cls]);
        $fwrite(fd, "%6d  ", outputArray[cls]);
      end
    $write("]\n\n");
    $fwrite(fd, "]\n\n");
    `endif

    $fclose(fd);

endfunction

// Displays inputs, intermediate values and output
function void generateMatlab();

    string  matlabFileName;
    int     fd          ;

    matlabFileName = "projects564.m";
    fd = $fopen (matlabFileName , "w");
    $fwrite(fd, "clear all\n\n");

    $fwrite(fd,"%% IA is the Input Array, 12x12 of int16's\n");
    $fdisplay(fd, "IA = [");
    for (int y=0; y<12; y++)
      begin
        for (int x=0; x<12; x++)
          begin
            $fwrite(fd, "%6d  ", inputArray[y][x]);
          end
          $fwrite(fd, ";\n");
      end
    $fwrite(fd, "]\n");

    $fwrite(fd,"%% B contains the 1x9 step 1 filters\n");
    $fwrite(fd, "B = [\n");
    for (int layer=0; layer<4; layer++)
      begin
        for (int x=0; x<9; x++)
          begin
            $fwrite(fd, "%6d  ", bVectors[layer][x]);
          end
```

```
            $fwrite(fd, ";\n");
          end
        $fwrite(fd, "]\n\n");

        $fwrite(fd,"%% M contains the 1x64 step 2 filters\n");
        $fwrite(fd, "M = [\n");
        for (int cls=0; cls<8; cls++)
          begin
            for (int x=0; x<64; x++)
              begin
                $fwrite(fd, "%6d  ", mVectors[cls][x]);
              end
              $fwrite(fd, "\n");
          end
        $fwrite(fd, "]\n\n");

        $fwrite(fd,"%% Extract the four 6x6 regions of the input array, we will calls these Quadrants\n");
        $fwrite(fd,"Quadrant0 = IA(1:6,1:6)\n");
        $fwrite(fd,"Quadrant = Quadrant0\n");
        $fwrite(fd,"Quadrant(:,:,2) = IA(1:6,7:12)\n");
        $fwrite(fd,"Quadrant(:,:,3) = IA(7:12,1:6)\n");
        $fwrite(fd,"Quadrant(:,:,4) = IA(7:12,7:12)\n");
        $fwrite(fd,"\n");
        $fwrite(fd,"%% Perform a dot-product of the b-vector with each corner of the ROI\n");
        $fwrite(fd,"\n");
        $fwrite(fd,"C = zeros(2,2);\n");
        $fwrite(fd,"for layer = 1:4\n");
        $fwrite(fd,"    for QuadrantN = 1:4\n");
        $fwrite(fd,"        \n");
        $fwrite(fd,"        A_1_1  = [Quadrant(1,1:3,QuadrantN) Quadrant(2,1:3,QuadrantN) Quadrant(3,1:3,QuadrantN)];\n");
        $fwrite(fd,"        A_1_2  = [Quadrant(1,4:6,QuadrantN) Quadrant(2,4:6,QuadrantN) Quadrant(3,4:6,QuadrantN)];\n");
        $fwrite(fd,"        A_2_1  = [Quadrant(4,1:3,QuadrantN) Quadrant(5,1:3,QuadrantN) Quadrant(6,1:3,QuadrantN)];\n");
        $fwrite(fd,"        A_2_2  = [Quadrant(4,4:6,QuadrantN) Quadrant(5,4:6,QuadrantN) Quadrant(6,4:6,QuadrantN)];\n");
        $fwrite(fd,"        C(1,1) = A_1_1*B(layer,:)';\n");
        $fwrite(fd,"        C(1,2) = A_1_2*B(layer,:)';\n");
        $fwrite(fd,"        C(2,1) = A_2_1*B(layer,:)';\n");
        $fwrite(fd,"        C(2,2) = A_2_2*B(layer,:)';\n");
        //$fwrite(fd,"        C(1,1) = [Quadrant(1,1:3,QuadrantN) Quadrant(2,1:3,QuadrantN)
Quadrant(3,1:3,QuadrantN)]*B(layer,:)';\n");
        //$fwrite(fd,"        C(1,2) = [Quadrant(1,4:6,QuadrantN) Quadrant(2,4:6,QuadrantN)
Quadrant(3,4:6,QuadrantN)]*B(layer,:)';\n");
        //$fwrite(fd,"        C(2,1) = [Quadrant(4,1:3,QuadrantN) Quadrant(5,1:3,QuadrantN)
Quadrant(6,1:3,QuadrantN)]*B(layer,:)';\n");
        //$fwrite(fd,"        C(2,2) = [Quadrant(4,4:6,QuadrantN) Quadrant(5,4:6,QuadrantN)
Quadrant(6,4:6,QuadrantN)]*B(layer,:)';\n");
        $fwrite(fd,"        \n");
        $fwrite(fd,"        %% apply f\(x\) and truncate\n");

        for (int x=1; x<3; x++)
          begin
            for (int y=1; y<3; y++)
              begin
                //$fwrite(fd,"        C_%0d_%0d__orig = C(%0d,%0d) ;\n",x,y,x,y);
                $fwrite(fd,"        sign_C_%0d_%0d  = sign(C(%0d,%0d)) ;\n",x,y,x,y);
                $fwrite(fd,"        C_%0d_%0d  = dec2bin(abs(C(%0d,%0d)),32) ;\n",x,y,x,y);
                $fwrite(fd,"        C_%0d_%0d  = C_%0d_%0d(1:16) ;\n",x,y,x,y);
                $fwrite(fd,"        C_%0d_%0d  = bin2dec(C_%0d_%0d) ;\n",x,y,x,y);
                $fwrite(fd,"        C(%0d,%0d)  = sign_C_%0d_%0d*C_%0d_%0d;\n",x,y,x,y,x,y);
              end
          end

        $fwrite(fd,"\n");
        //$fwrite(fd,"        Z = max(0,C)/(2^16)\n");
        $fwrite(fd,"        Z = max(0,C)\n");
        $fwrite(fd,"        \n");
        $fwrite(fd,"        Zq(:,:,layer,QuadrantN) = Z;\n");
        $fwrite(fd,"        \n");
        $fwrite(fd,"    end\n");
        $fwrite(fd,"end\n");
        $fwrite(fd,"\n");
```

```
        $fwrite(fd,"%% Now operate on the four 2x2x4 quadrants as if it were a 4x4x4 array\n");
        $fwrite(fd,"%% Our matlab quadrant array is organized as (y,x,ROI,layer)\n");
        $fwrite(fd,"%% So lets just merge them\n");
        $fwrite(fd,"%% Remember, a layer is generated from each B vector\n");
        $fwrite(fd,"for layer = 1:4\n");
        $fwrite(fd,"    Zmerged(:,:,layer) = [Zq(:,:,layer,1) Zq(:,:,layer,2) ;\n");
        $fwrite(fd,"                          Zq(:,:,layer,3) Zq(:,:,layer,4) ]\n");
        $fwrite(fd,"end\n");
        $fwrite(fd,"\n");
        $fwrite(fd,"%% Create a 1x64 vector from a merged array\n");
        $fwrite(fd,"%% We create the vector using row major layer by layer\n");
        $fwrite(fd,"%%\n");
        $fwrite(fd,"%% use reshape on the transpose because reshape uses column major\n");
        $fwrite(fd,"U = [];\n");
        $fwrite(fd,"for layer = 1:4\n");
        $fwrite(fd,"    U = [U reshape(Zmerged(:,:,layer)', [1 16])];\n");
        $fwrite(fd,"end\n");
        $fwrite(fd,"\n");
        $fwrite(fd,"%% Now dot-product Qv with each output m-vector\n");
        $fwrite(fd,"W = [];\n");
        $fwrite(fd,"for o = 1:8\n");
        $fwrite(fd,"    W = [W M(o,:)*U'];\n");
        $fwrite(fd,"end\n");
        $fwrite(fd,"\n");
        $fwrite(fd,"%% Now apply our f(x)\n");
        $fwrite(fd,"O = max(0,W);\n");
        $fwrite(fd,"%% truncate\n");
        $fwrite(fd,"O = O/(2^16);\n");
        $fwrite(fd,"%% Expected output\n");
        $fwrite(fd,"O\n");
        $fwrite(fd,"\n");

        $fclose(fd);

      endfunction

  endclass


endpackage

//------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------
// This function generates the data object and then responds to memory read requests from the DUT
//

import operation::*;

class generator;

  vDutIfc   DutIfc  ;
  base_operation op  ;

  int  max_ops      ;

  // Temp
  logic [ 8:0] tmp_dim_addr       ;
  logic [ 8:0] tmp_bvm_addr       ;
  logic [ 2:0] tmp_dom_addr       ;
  logic [ 7:0] tmp_inputArray_addr ;
  logic [15:0] tmp_dom_data       ;

  integer tId                   ;
  integer addr;

  function new (
            input  vDutIfc DutIfc
            );

    this.DutIfc  =  DutIfc  ;
```

```
  endfunction

  task init();
    $display("@%t: INFO: Initialize memories", $time);
    op = new();
    assert(op.randomize());

    //------------------------------------------------------------------------------------------------
    // Load RAM with random data from op object

    $display("@%t: INFO: Initialize Memories",  $time);

    // Input array
    for (int y=0; y<12; y++)
      begin
        for (int x=0; x<12; x++)
          begin
            addr = y*'h10+x;
            DutIfc.loadRam(0,addr,op.inputArray[y][x]);
            //$display("@%t: DEBUG: Load DIM: Row=%0d, Col=%0d, addr=%4h, data=%4h",  $time, y, x, addr, op.inputArray[y][x]);
          end
      end
    // B-vectors
    for (int y=0; y<4; y++)
      begin
        for (int x=0; x<9; x++)
          begin
            addr = y*'h10+x;
            DutIfc.loadRam(1,addr,op.bVectors[y][x]);
            //$display("@%t: DEBUG: Load b-vectors: Row=%0d, Col=%0d, addr=%4h, data=%4h",  $time, y, x, addr,
op.bVectors[y][x]);
          end
      end
    // M-vectors
    for (int y=0; y<8; y++)
      begin
        for (int x=0; x<64; x++)
          begin
            addr = y*'h10+x+'h40;
            DutIfc.loadRam(1,addr,op.mVectors[y][x]);
            //$display("@%t: DEBUG: Load m-vectors: Row=%0d, Col=%0d, addr=%4h, data=%4h",  $time, y, x, addr,
op.mVectors[y][x]);
          end
      end

    // memories initialized
    //------------------------------------------------------------------------------------------------

  endtask : init

  task run();       //running all three tasks in parallel
    $display("@%t: INFO: Memory Checker running", $time);
    //op = new();
    //assert(op.randomize());
    op.displayAll();
    op.generateMatlab();

    // Keep track of number of times we hit go
    max_ops = 3;
    tId = 0;

    fork
      // Output memory
      begin
        forever
          begin
            @(posedge DutIfc.cb_test);
            if (DutIfc.cb_test.dut__dom__enable && DutIfc.cb_test.dut__dom__write)
              begin
```

```
            tmp_dom_addr = DutIfc.cb_test.dut__dom__address ;
            tmp_dom_data = DutIfc.cb_test.dut__dom__data    ;
            $display("@%t: INFO: Output Memory Write, addr=%h", $time, tmp_dom_addr);
            $display("@%t: INFO: Output Value for element {%0d} = %h", $time, tmp_dom_addr, tmp_dom_data);
            if (($signed(tmp_dom_data)  >  op.outputArray[tmp_dom_addr]+op.tolerance) | ($signed(tmp_dom_data)  <
op.outputArray[tmp_dom_addr]-op.tolerance))
              begin
                op.outputStatus[tmp_dom_addr] = 1'b0 ;
                $display("@%t: ERROR: Output Memory Write, writing %h, expecting %h, tolerance = %0d, output status=%8b", $time,
tmp_dom_data, op.outputArray[tmp_dom_addr], op.tolerance, op.outputStatus);
              end
            else
              begin
                op.outputStatus[tmp_dom_addr] = 1'b1 ;
                $display("@%t: PASS: Output Memory Write, writing %h, expecting %h, output status=%8b", $time, tmp_dom_data,
op.outputArray[tmp_dom_addr], op.outputStatus);
              end
          end
        end
    end
    // Generate GO
    begin
      forever
        begin
          @(DutIfc.cb_test);
          if (DutIfc.cb_test.dut__xxx__finish)
            begin
              $display("@%t: INFO: Start", $time);
              @(DutIfc.cb_test);
              @(DutIfc.cb_test);
              DutIfc.xxx__dut__go  =  1;
              @(DutIfc.cb_test);
              DutIfc.xxx__dut__go  =  0;
              @(DutIfc.cb_test);
              tId++ ;
            end
          while(~DutIfc.cb_test.dut__xxx__finish)
            begin
              @(DutIfc.cb_test);
            end
          if (tId == max_ops)
            begin
              $display("@%t: INFO: Done", $time);
              if (|(~op.outputStatus))
                begin
                  $display("@%t: ERROR: A bad or no output data was written to the output array: %8b", $time, op.outputStatus);
                end
              else
                begin
                  $display("@%t: PASS: Output array status: %8b", $time, op.outputStatus);
                end
              break;
            end
          //check();
        end
    end
  join_any
  @(DutIfc.cb_test.dut__xxx__finish);
 endtask: run


endclass

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// Environment
//
class Environment;

  //vTBIfc  vDut_ifc ;
```

```
  vDutIfc   DutIfc   ;

  generator  mem_gen ;

  function new (
             //input  vTBIfc dut_if
             input  vDutIfc DutIfc
             );

    this.DutIfc  =  DutIfc  ;

  endfunction

  task build();

    mem_gen = new (DutIfc) ;

  endtask

  task reset();

    mem_gen.init() ;

    DutIfc.xxx__dut__go  =  0;
    DutIfc.reset  =  0;
    repeat (10) @(DutIfc.cb_test);
    DutIfc.reset  =  1;
    repeat (10) @(DutIfc.cb_test);
    DutIfc.reset  =  0;
    repeat (10) @(DutIfc.cb_test);

  endtask

  task run();

    mem_gen.run() ;
    $display("@%t: INFO: Memory Generator done", $time);

  endtask: run

  task wrap_up();

  endtask

endclass

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// Test
program automatic test(
                  dut_ifc  DutIfc
                  //input logic reset
                  );

  Environment env ;

  initial
    begin
      env = new (.DutIfc  ( DutIfc )
              );

      env.build     ( );
      env.reset     ( );
      env.run       ( );
      env.wrap_up   ( );

      $finish;

    end
```

```
endprogram

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// Tesbench
// - instantiate the DUT and testbench

module tb_top ();


  parameter CLK_PHASE=5  ;

  //------------------------------------------------------------------------------
  // b-vector memory
  wire [ 8:0]         dut__bvm__address  ;
  wire [15:0]          dut__bvm__data     ;  // write data
  wire [15:0]          bvm__dut__data     ;  // read data
  wire                dut__bvm__enable   ;
  wire                dut__bvm__write    ;

  //------------------------------------------------------------------------------
  // Input data memory
  wire [ 8:0]         dut__dim__address  ;
  wire [15:0]          dut__dim__data     ;  // write data
  wire [15:0]          dim__dut__data     ;  // read data
  wire                dut__dim__enable   ;
  wire                dut__dim__write    ;

  //------------------------------------------------------------------------------
  // Output data memory
  wire [ 2:0]         dut__dom__address  ;
  wire [15:0]          dut__dom__data     ;  // write data
  wire                dut__dom__enable   ;
  wire                dut__dom__write    ;

  //------------------------------------------------------------------------------
  // General
  //
  reg      clk              ;
  wire     reset            ;
  reg      xxx__dut__go     ;
  wire     dut__xxx__finish ;


  //------------------------------------------------------------------------------
  // Interface
  //
  dut_ifc  dut_if( .clk  (clk));

  //------------------------------------------------------------------------------
  //------------------------------------------------------------------------------
  //------------------------------------------------------------------------------
  // DUT
  //  - use interface for connection to DUT
  dut_wrapper dut_wrapper (

     .dut_if            ( dut_if.DUT         ),
     .clk               ( clk                )

     );

  sram  #(.ADDR_WIDTH  ( 9),
          .DATA_WIDTH  (16))
         bvm_mem  (
          .address      ( dut_if.dut__bvm__address  ),
          .write_data   ( dut_if.dut__bvm__data     ),
          .read_data    ( dut_if.bvm__dut__data     ),
          .enable       ( dut_if.dut__bvm__enable   ),
          .write        ( dut_if.dut__bvm__write    ),
```

```
        .clock        ( clk                     )
       );

 sram  #(.ADDR_WIDTH  ( 9),
        .DATA_WIDTH  (16))
        dim_mem (

        .address     ( dut_if.dut__dim__address  ),
        .write_data  ( dut_if.dut__dim__data     ),
        .read_data   ( dut_if.dim__dut__data     ),
        .enable      ( dut_if.dut__dim__enable   ),
        .write       ( dut_if.dut__dim__write    ),

        .clock       ( clk                     )
       );

 sram  #(.ADDR_WIDTH  ( 3),
        .DATA_WIDTH  (16))
       dom_mem  (

        .address     ( dut_if.dut__dom__address  ),
        .write_data  ( dut_if.dut__dom__data     ),
        .read_data   (                           ),
        .enable      ( dut_if.dut__dom__enable   ),
        .write       ( dut_if.dut__dom__write    ),

        .clock       ( clk                     )
       );



 //-------------------------------------------------------------------------------
 //-------------------------------------------------------------------------------
 //-------------------------------------------------------------------------------
 //-------------------------------
 // Testbench
 //
 test  tb  (
          .DutIfc  ( dut_if.TB )

          );

 //-------------------------------------------------------------------------------
 //  clk
 initial
   begin
      clk                  = 1'b0;
      forever # CLK_PHASE clk = ~clk;
   end


 //-------------------------------------------------------------------------------
 //-------------------------------------------------------------------------------
 // Stimulus

 //-------------------------------------------------------------------------------


endmodule

//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
// DUT wrapper
module dut_wrapper (
          dut_ifc                  dut_if  ,
          input  wire               clk
);

  dut dut(
```

```
        .dut__xxx__finish   ( dut_if.dut__xxx__finish   ),
        .xxx__dut__go       ( dut_if.xxx__dut__go       ),
        .dut__bvm__address  ( dut_if.dut__bvm__address  ),
        .dut__bvm__data     ( dut_if.dut__bvm__data     ),
        .bvm__dut__data     ( dut_if.bvm__dut__data     ),
        .dut__bvm__enable   ( dut_if.dut__bvm__enable   ),
        .dut__bvm__write    ( dut_if.dut__bvm__write    ),
        .dut__dim__address  ( dut_if.dut__dim__address  ),
        .dut__dim__data     ( dut_if.dut__dim__data     ),
        .dim__dut__data     ( dut_if.dim__dut__data     ),
        .dut__dim__enable   ( dut_if.dut__dim__enable   ),
        .dut__dim__write    ( dut_if.dut__dim__write    ),
        .dut__dom__address  ( dut_if.dut__dom__address  ),
        .dut__dom__data     ( dut_if.dut__dom__data     ),
        .dut__dom__enable   ( dut_if.dut__dom__enable   ),
        .dut__dom__write    ( dut_if.dut__dom__write    ),

        .reset              ( dut_if.reset              ),
        .clk                ( clk                       )
       );



endmodule

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// DUT

module dut (

        //------------------------------------------------------------------------------
        // Control
        //
        output reg              dut__xxx__finish   ,
        input  wire             xxx__dut__go       ,

        //------------------------------------------------------------------------------
        // b-vector memory
        //
        output reg  [ 8:0]      dut__bvm__address  ,
        output reg              dut__bvm__enable   ,
        output reg              dut__bvm__write    ,
        output reg  [15:0]      dut__bvm__data     ,  // write data
        input  wire [15:0]      bvm__dut__data     ,  // read data

        //------------------------------------------------------------------------------
        // Input data memory
        //
        output reg  [ 8:0]      dut__dim__address  ,
        output reg              dut__dim__enable   ,
        output reg              dut__dim__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   ,
        output reg              dut__dom__write    ,


        //--------------------------------
        // General
        //
        input  wire             clk            ,
```

```
        input  wire                reset

        );

 //------------------------------------------------------------------------
 //
 //<<<<----  YOUR CODE HERE    ---->>>>
 //
 `include "v564.vh"
 //
 //------------------------------------------------------------------------

endmodule
```

Memory:

```
/*****************************************************************************************************

   File name   : sram.v
   Author      : Lee Baker
   Affiliation : North Carolina State University, Raleigh, NC
   Date        : Oct 2017
   email       : lbbaker@ncsu.edu

   Description : Generic SRAM


*****************************************************************************************************/

//`timescale 1ns/10ps

module sram     #(parameter  ADDR_WIDTH      = 8   ,
                  parameter  DATA_WIDTH      = 16  )
                (
                //---------------------------------------------------------
                //
                input  wire [ADDR_WIDTH-1:0  ]  address     ,
                input  wire [DATA_WIDTH-1:0  ]  write_data  ,
                output reg  [DATA_WIDTH-1:0  ]  read_data   ,
                input  wire                     enable      ,
                input  wire                     write       ,

                input  clock
                );


   //---------------------------------------------------------
   // Associative memory

   bit  [DATA_WIDTH-1 :0  ]     mem     [int] = '{default: 'X};


   //---------------------------------------------------------
   // Read

   always @(*)
     begin
       #4 read_data   =  mem [address] ;
     end

   //---------------------------------------------------------
   // Write

   always @(posedge clock)
     begin
       if (write)
         mem [address] = write_data ;
```

```
    end
  //-------------------------------------------------------------


endmodule
```