ECE564 ASIC DESIGN

Project Responsibilities: Hardware accelerator to implement a simplified Convolutional Neural Network algorithm
StudentID. : 200213223
Name : KENNY KADAYIL PAUL

Delay (ns to run provided provided example): 15170 ns
Clock period: 6.2 ns
# cycles":1517

Logic Area:

9313.19(um^2)

Memory: N/A

1/(delay.area) (ns^{-1}.um^{-2})

$1.1416 * 10^{-8}$

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

Hardware accelerator to implement a simplified Convolutional Neural Network algorithm

KENNY KADAYIL PAUL

1. Introduction

The hardware designed is a simplified version of a Convolutional Neural Network, a common algorithm used in machine learning. This has been widely used in areas such as image recognition and object classification which enables successfully identifying faces, objects apart from powering vision in robots and self driving cars. The most common architecture proposed for Convolutional Neural Networks are several phases of Convolution and Pooling followed by a multiple phases of Fully connected phase to render the output predictions.

Our hardware will be implementing two layers, the first layer is a feature extraction using filter vectors from an input SRAM which will be the convolution phase and the second layer is a fully connected layer to identify classes. The project input is a 12x12 array of 16-bit twos-complement numbers which can be the details of an image or part of it. The final output will be an 8x1 vector of 16-bit twos-complement which is a simplified version of the output class vector.

Convolution phase

The Convolution phase captures important caveats of image that helps in identifying specific features corresponding to a particular object. This sort of feature extraction is performed using pre determined set of four filter set having 9 elements each. To perform this the input image is first captured as blocks of 3x3 pixels. A dot product is performed on these 9 elements with each one of the four filters to obtain a total of 16 elements per filter having 32 bit width.

The dot product function is defined as

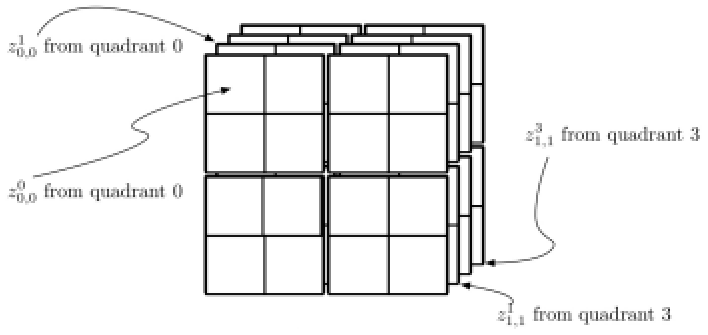
$$b = [b0, b1 ... b8]$$
$$a = [a0, a1 ... a8]$$

$$c = \sum_{n=0}^{8} a_i . b_i$$

For each 32 bit c, we perform the following min function and truncate the result z to 16-bits

$$z_i = \begin{cases} c_i, & c_i \geq 0 \\ 0, & otherwise \end{cases}$$

To meet with the project specification, 12x12 input is considered as four 6x6 quadrants and each quadrant is operated upon by separate datapaths to get the output of the first phase. This operation on each quadrant will form a 2x2x4 output array and the final output will be a 4x4x4 array as depicted below.



$z^1_{0,0}$ from quadrant 0
$z^0_{0,0}$ from quadrant 0
$z^3_{1,1}$ from quadrant 3
$z^1_{1,1}$ from quadrant 3

Fully connected phase

The 4x4x4 output from step one is merged into a single 64x1 array and stored in the SRAM to be used for computation of dot product again in the fully connected phase to get an output of 1x8 vector of 16 bit numbers. The multiplication vectors for this phase are supplied as an array of 64x8 in the SRAM memory. The computation for this phase is illustrated below.

$$m_i \;=\; [m_{i,0}, m_{i,1}, m_{i,2} .. m_{i,63}] \;\; where \;\; i = 1,2,3...8$$

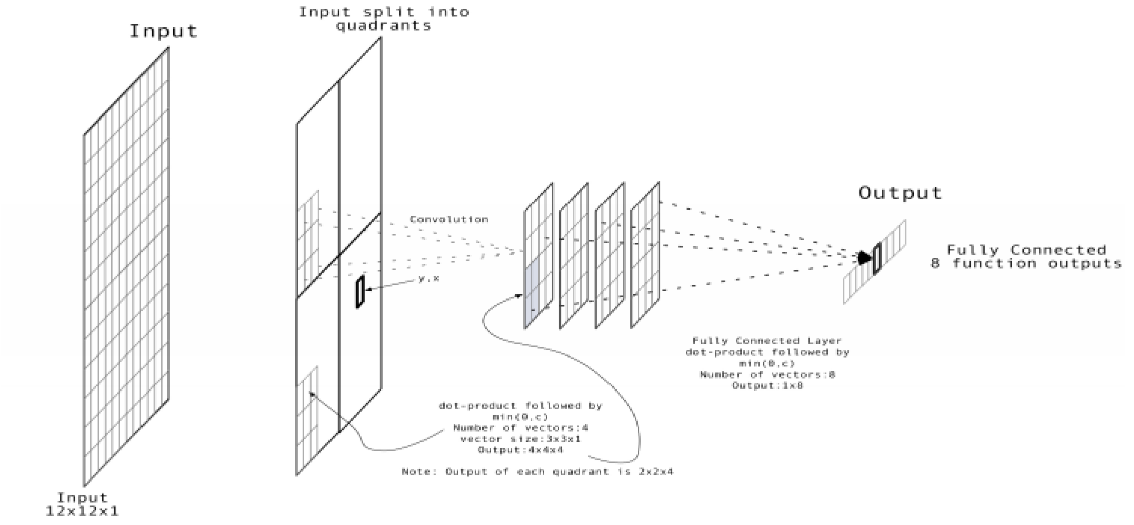
$$u \;=\; [u_0, u_1, u_2 .. u_{63}]$$

$$w_i \;=\; \sum_{n=0}^{63} m_i . u \quad where\, i \;=\; 1,2, \ldots .8$$

Element truncation is performed again to obtain the output vector as.

$$O_i \;=\; \begin{cases} w_i, & w_i \geq 0 \\ 0, & otherwise \end{cases}$$

The entire operation can be represented as follows.



The following sections describe the structure of hardware designed to execute the above described algorithm.

2. Micro-Architecture

The developed CNN system is developed under the following system requirements.

- Input image consisting of 144 elements each with a data width 16 bit stored in a one read+write ported sram memory.
- Phase one and phase two filter array consisting of a total of 548 elements with a data width of 16 bit stored in a one read ported sram memory.
- One read+write ported memory for storing the output array consisting of 8 elements each with a data width of 16 bit.

Critical Bottlenecks

The first bottleneck comes as critical path in the data plane which performs basic elemental arithmetic operation of multiplication and accumulation. This poses a challenge in terms of the delay-area requirements. The mathematical representation of data path function is represented as below.

$$M_i = M_{i-1} + (a_i \, . \, b_i)$$

The second bottleneck is in the SRAM memory where only one read/write port limits the data access to one read or one write per clock cycle.

The third bottleneck is posed in storing the intermediate 64 c_i values which is required for the fully connected phase computation. Storing in on chip registers increases the total number of cells required for the design but enables parallel access by other parts of the design while storing it in SRAM scatchpad requires overhead of additional clock cycles and sufficient hardware control logic to manage writing and re-reading from the system memory.

System Design

The aforementioned limitation in the memory allows only one computation i.e. either the z_i or w_i to proceed at a time. Hence the number of multiply accumulates in our design is fixed at one per clock cycle. This leads to a sequential form of execution wherein the computations for step one is performed before moving onto step two phase.

In order to overcome the first bottleneck in critical path, a three stage pipelined multiplier design is implemented for the dot product in the data plane. This enables to speed up the hardware and reduce the total time taken up for computation of the entire algorithm. The *optimize_register* command in synopsys along with the usage of designware multiplier generates the required pipeline for the multiplier addition logic.

Since total cell area is crucial to the cost of the hardware and based on the observations of how many registers are needed if the intermediate c_i are stored in the the chip it was observed that providing the control logic for writing out to memory and the resulting delay incurred for writing to the main memory fairs far better than the former approach.

Hence we have the following timing for each filter computation

Timing for convolution phase

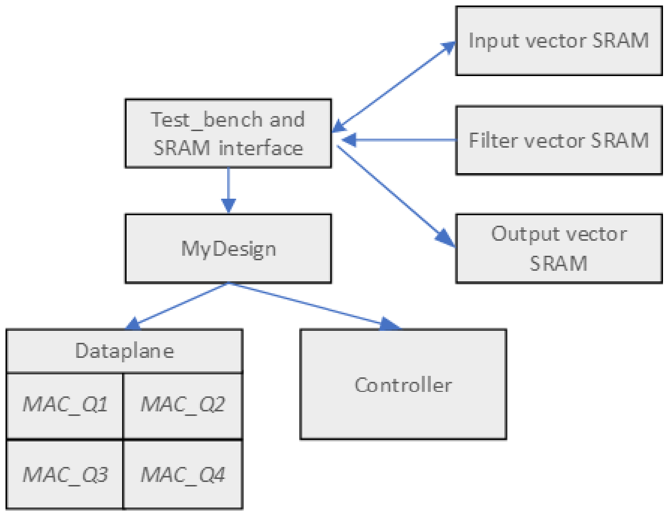
Clock	dim_addr	bvm_addr	dim_data	Clock	dim_addr	bvm_addr	dim_data	Clock	dim_addr	bvm_addr	dim_data	Clock	dim_addr	bvm_addr	dim_data
1	000	00		241	000	10		481	000	20		721	000	30	
2	001	01		242	001	11		482	001	21		722	001	31	
3	002	02		243	002	12		483	002	22		723	002	32	
4	010	03		244	010	13		484	010	23		724	010	33	
'				'				'				'			
'				'				'				'			
9	022	08		249	022	18		489	022	28		729	022	38	
10	-	-		250	-	-		490	-	-		730	-	-	
-	-	-		-	-	-		-	-	-		-	-	-	
15	100	-	Z0	255	110	-	Z16	495	120	-	Z32	735	130	-	Z48

Timing for fully connected phase

Clock	dim_addr	bvm_addr	dom_addr	dom_data	Clock	dim_addr	bvm_addr	dom_addr	dom_data
961	100	40			1443	100	200		
962	101	41			1444	101	201		
963	102	42			1445	102	202		
964	103	43			1446	103	203		
'					'				
'					'				
1024	13F	79			1506	13F	23F		
1025	-	-			1507	-	-		
-	-	-			-	-	-		
1029	-	-	00	W0	1512	-	-	07	W7

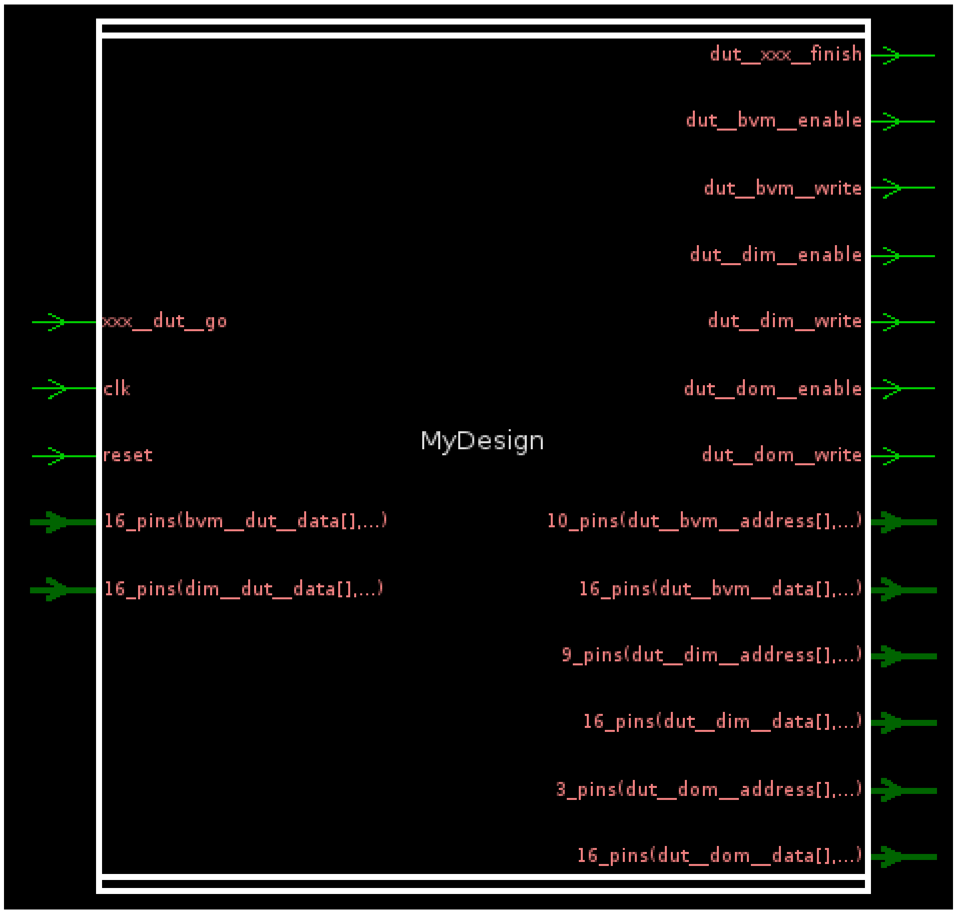
3. Interface Specification

The project contains one top level module having two sub modules. One for the data path and one for the controller.



The purpose of the testbench module is described in the verification section.

MyDesign module is interfaced with SRAM and receives an external clock, go and global reset. The module also gives out finish signal indicating completion of the CNN operation. It instantiates the Data plane and the controller required for the design. In addition to this it also contains some glue logic to direct the output of the corresponding MAC module to the scratchpad memory. Below is MyDesign input and output signals and table listing of all the signals in the design and their width and function.

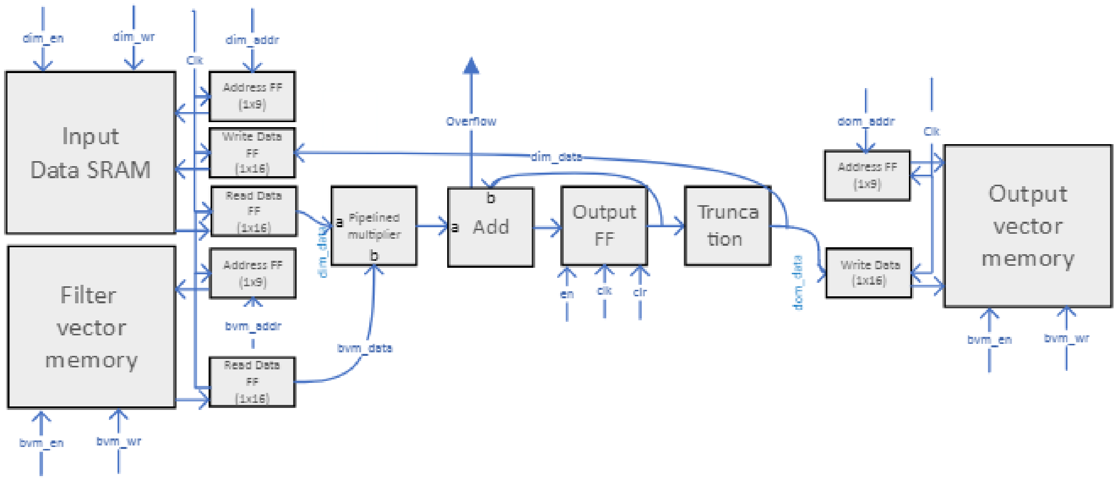


MyDesign
xxx__dut__go
clk
reset
16_pins(bvm__dut__data[],...)
16_pins(dim__dut__data[],...)
dut__xxx__finish
dut__bvm__enable
dut__bvm__write
dut__dim__enable
dut__dim__write
dut__dom__enable
dut__dom__write
10_pins(dut__bvm__address[],...)
16_pins(dut__bvm__data[],...)
9_pins(dut__dim__address[],...)
16_pins(dut__dim__data[],...)
3_pins(dut__dom__address[],...)
16_pins(dut__dom__data[],...)

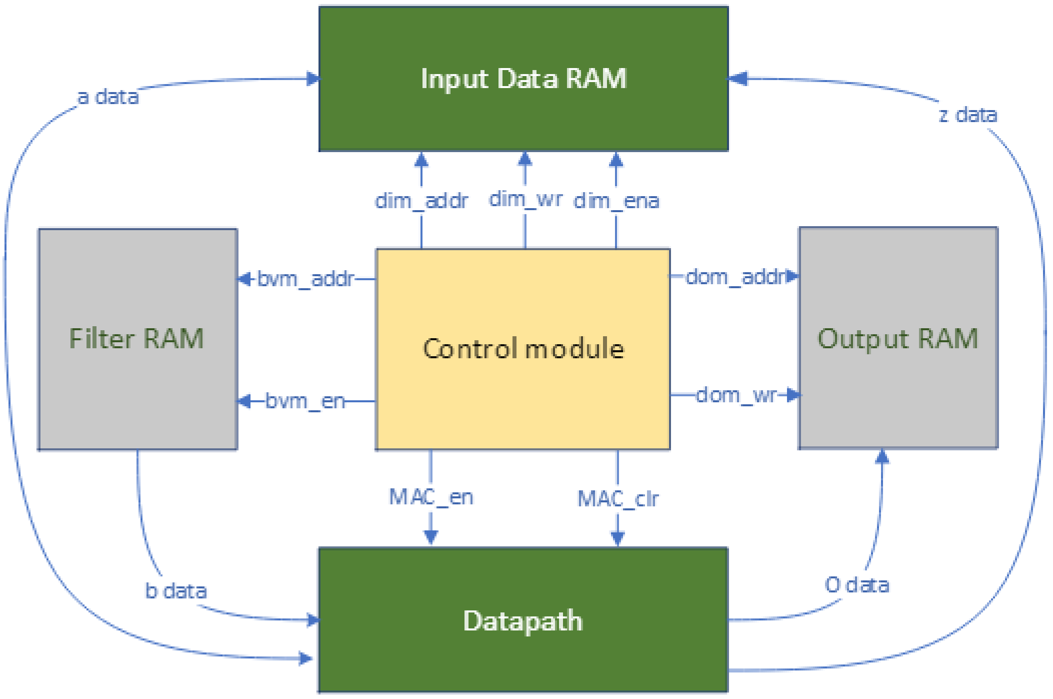
SIGNALS	WIDTH	FUNCTION
clk	1	Clock to all modules
reset	1	Reset to all modules
dut__xxx__finish	1	Finish asserted from MyDesign
xxx__dut__go	1	Signal to start machine from testbench
dut__bvm__address	10	Address to filter memory
dut__bvm__enable	1	Enable for filter memory
dut__bvm__write	1	Write enable for filter memory
dut__bvm__data	16	Data from MyDesign to filter memory
bvm__dut__data	16	Data from filter memory to MyDesign
dut__dim__address	9	Address to input vector memory
dut__dim__enable	1	Enable for input vector memory

dut__dim__write	1	Write signal for input vector memory
dut__dim__data	16	Data from MyDesign to input vector memory
dim__dut__data	16	Data from input vector memory to MyDesign
dut__dom__address	3	Address to output vector memory
dut__dom__data	16	Data from MyDesign to output vector memory
dut__dom__enable	1	Enable for output vector memory
dut__dom__write	1	Write signal for output vector memory
enableq1	1	Enable of quadrant 1 mac
enableq2	1	Enable of quadrant 2 mac
enableq3	1	Enable of quadrant 3 mac
enableq4	1	Enable of quadrant 4 mac
clear	1	Clear for all mac
counter_start	1	Internally generated machine start signal
sys_counter	12	Global clock cycle counter for MyDesign
block_counter	8	Counter for address calculation management
filter_counter	4	Counter for filter1
filter2_counter	8	Counter for filter2
array_counter	2	Three element counter
write_address	9	Write address for scratch pad memory
quad_count	4	Quadrant counter
temp_dut_dim_address	9	Temporary register for input vector memory address
temp_dut_bvm_address	10	Temporary register for filter memory address
current_state	3	Current state of FSM in controller
next_state	3	Next state of FSM in controller
dataA	16	A input of multiplier register
dataB	16	B input of multiplier register
mult_prod	32	Product output of multiplier
C	32	Output FF of MAC

The Datapath consists of a pipelined multipler, a 16 bit adder and a truncation function as given in the block diagram section which we call as the MAC. The pipelined multiplier module and adder module used is *DW02_mult_3_stage* and 32 bit adder both of which are taken from synopsys designware library. The computation for the four quadrants requires four instantiations of this MAC which are named as MAC_Q1, MAC_Q2, MAC_Q3 and MAC_Q4.



The controller generates signals to interface with the SRAM simulator through the testbench module. The input, filter and output memory addresses are generated at appropriate clock cycles so that the the data plane receives the input data for computation. After every MAC computation phase there is a write to either the scratchpad memory or the output memory with the computed value. In addition to this the control module also provides MAC enable and MAC clear signals. The MAC enable signal is generated at the start of every c_i or w_i computation. The MAC clear signal is given to clear the output flip flop in the MAC after the write cycle of every z_i or O_i value.



4. Technical Implementation

The controller signals are implemented using FSM based counters. So depending on the global counter, the control signals *dut__bvm__enable, dut__bvm__write, dut__dim__enable, dut__dim__write, dut__dom__enable, dut__dom__write, enableq1, enableq2, enableq3, enableq4, clear* and based on these the MAC operates.

<u>State0</u>

The controller waits for the assertion of the go signal. Once it is received it goes to stage two.

<u>State 1</u>

This state provides the address data to both input vector memory and filter memory and makes the *dut__bvm__enable* and *dut__dim__enable high*. The FSM remains in this state until the *filter_counter* becomes equal to 9, it then proceeds on to state2. A total of 64 c_i values are generated after completing all the iterations of this state.

<u>State 2</u>

This is the writing phase of z_i values of the convolution phase onto the input vector memory scratch pad which resides from locations 0x100-0x1FF. These values are written in row major fashion. It remains in this state until the *filter_counter* becomes 14 at which state it rolls back to state 1. Another check performed here is whether the global counter has reached 959, if yes it proceeds onto state 3

<u>State3</u>

At this stage we know that the convolution phase has finished and we can start the fully connected phase. This stage fetches the z_i vector which we have written in state 2 and convolves it with the second filter vectors. For each value of w_i generated the controller remains in this stage until the *filter2_counter* has reached the value 64. Once the w is generated it proceeds to state 4 which is writing phase of w_i values.

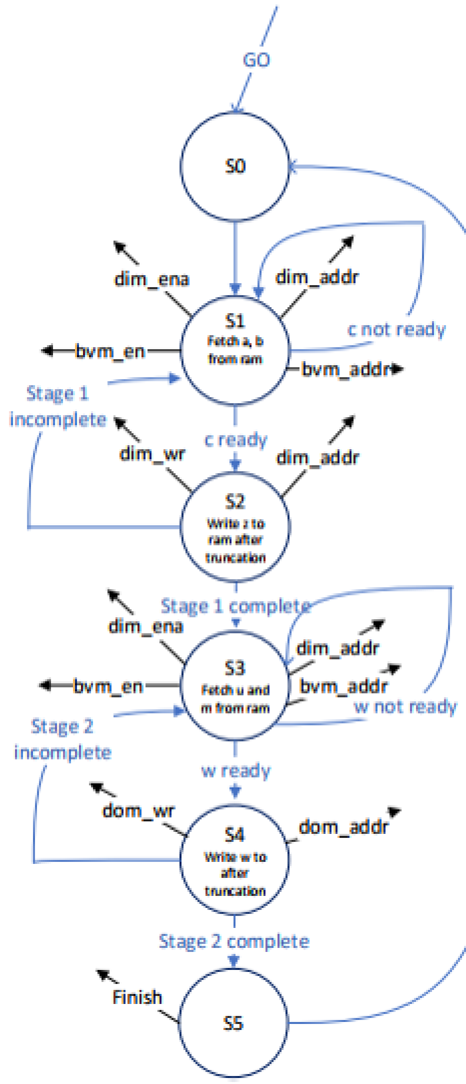
<u>State4</u>

This stage writes the O_i values into the output vector memory. When the *filter2_counter* has reached 68, this state has finished and concurrently it checks the global counter to see whether all w values are generated which happens at the value 1511. If yes, the FSM proceeds to Stage6 otherwise it loops back to state 3.

<u>State5</u>

This stage is kept to assert the finish signal and go back to state 0 for the next go signal.

The transition diagram for the FSM is given below.

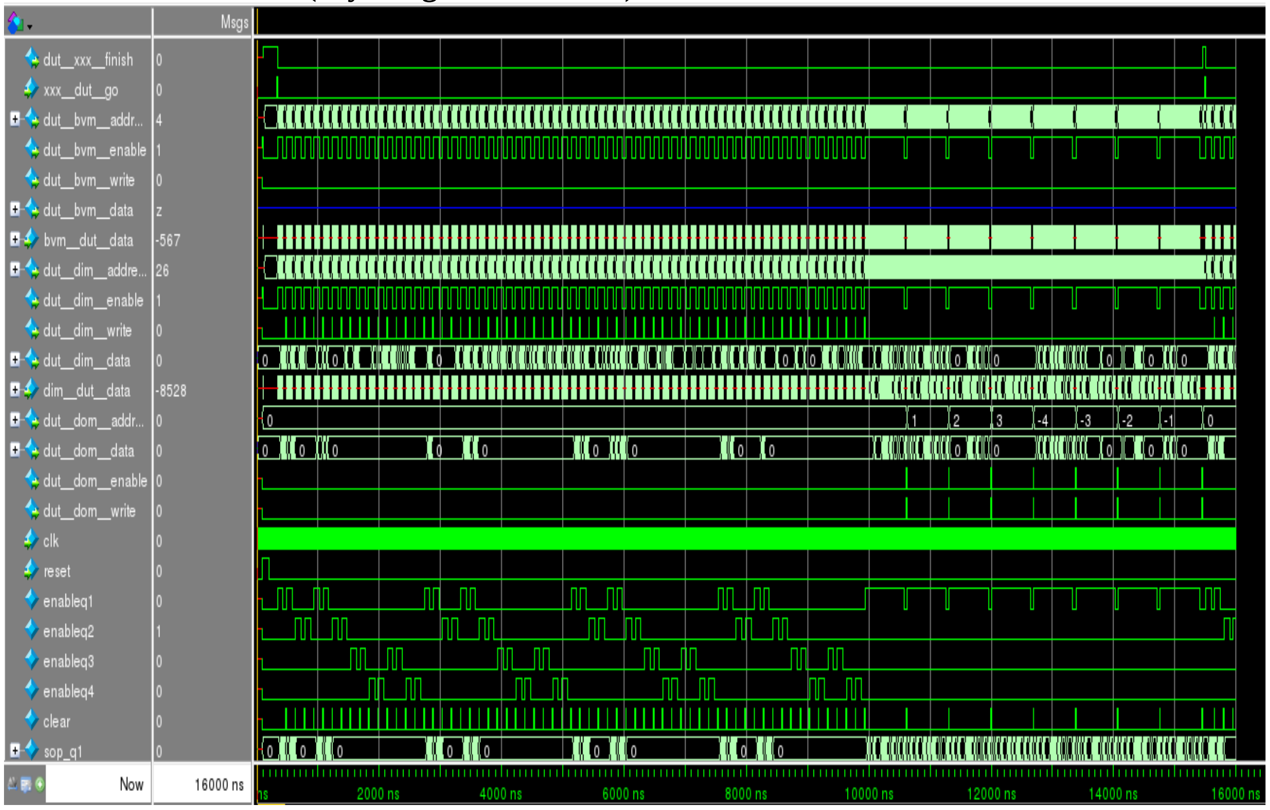


5. Verification

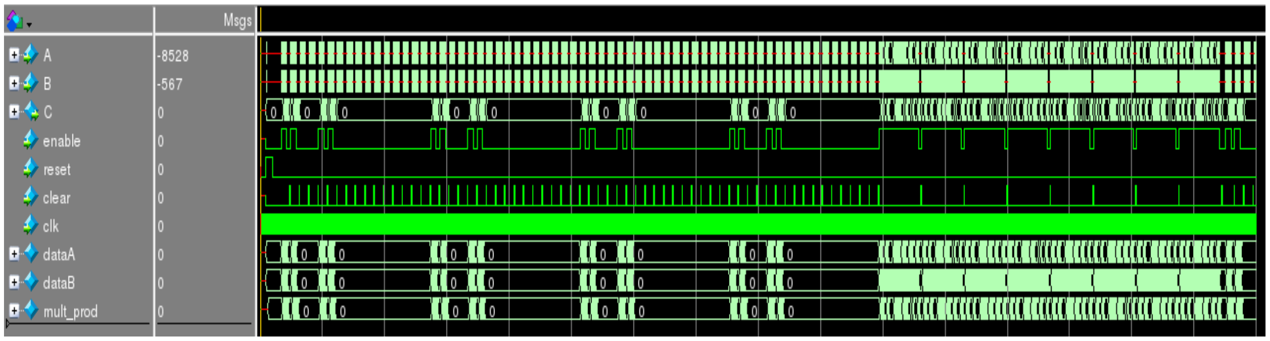
Verification functionality is given by the test bench which sits above MyDesign. It has the following two capabilities. First it accommodates the SRAM simulator where all the required data values for the machine to work are stored. The second is once the MyDesign gives the finish signal, the completion of computation is inferred and the content of SRAM is dumped into a text file and verified with the outputs obtained from MATLAB model of the design. In this manner it checks for the correctness of the design.

6. Results achieved

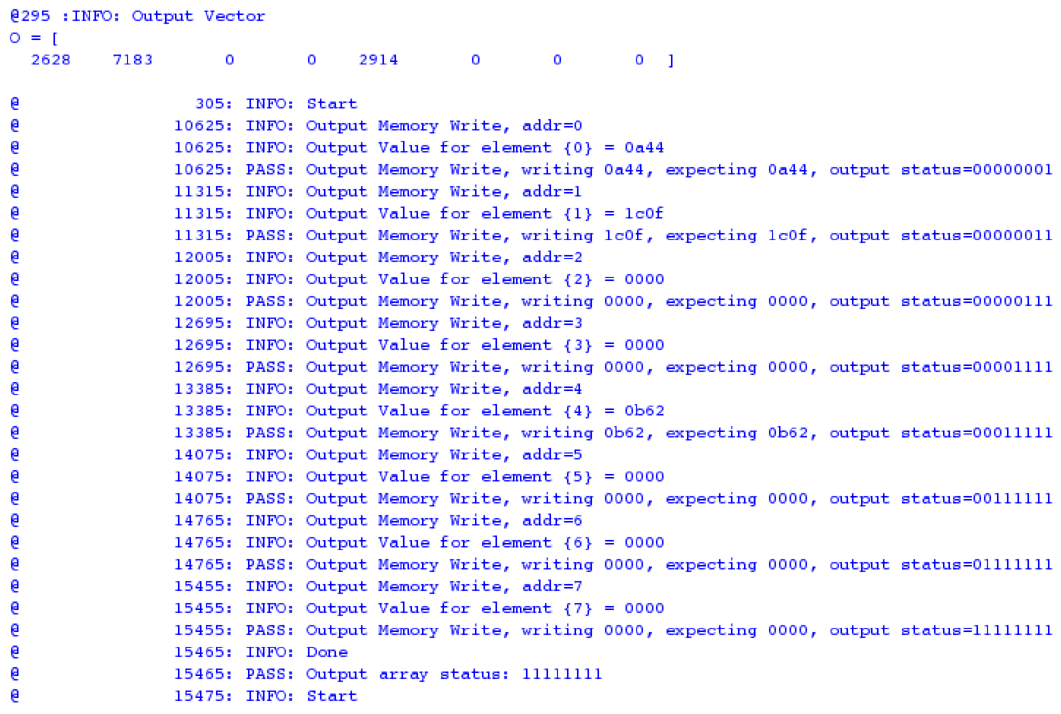
Simulation waveforms(MyDesign + controller):



Simulation waveforms(MAC):



Output transcript



```
@295 :INFO: Output Vector
O = [
   2628    7183       0       0    2914       0       0       0  ]

@                 305: INFO: Start
@               10625: INFO: Output Memory Write, addr=0
@               10625: INFO: Output Value for element {0} = 0a44
@               10625: PASS: Output Memory Write, writing 0a44, expecting 0a44, output status=00000001
@               11315: INFO: Output Memory Write, addr=1
@               11315: INFO: Output Value for element {1} = 1c0f
@               11315: PASS: Output Memory Write, writing 1c0f, expecting 1c0f, output status=00000011
@               12005: INFO: Output Memory Write, addr=2
@               12005: INFO: Output Value for element {2} = 0000
@               12005: PASS: Output Memory Write, writing 0000, expecting 0000, output status=00000111
@               12695: INFO: Output Memory Write, addr=3
@               12695: INFO: Output Value for element {3} = 0000
@               12695: PASS: Output Memory Write, writing 0000, expecting 0000, output status=00001111
@               13385: INFO: Output Memory Write, addr=4
@               13385: INFO: Output Value for element {4} = 0b62
@               13385: PASS: Output Memory Write, writing 0b62, expecting 0b62, output status=00011111
@               14075: INFO: Output Memory Write, addr=5
@               14075: INFO: Output Value for element {5} = 0000
@               14075: PASS: Output Memory Write, writing 0000, expecting 0000, output status=00111111
@               14765: INFO: Output Memory Write, addr=6
@               14765: INFO: Output Value for element {6} = 0000
@               14765: PASS: Output Memory Write, writing 0000, expecting 0000, output status=01111111
@               15455: INFO: Output Memory Write, addr=7
@               15455: INFO: Output Value for element {7} = 0000
@               15455: PASS: Output Memory Write, writing 0000, expecting 0000, output status=11111111
@               15465: INFO: Done
@               15465: PASS: Output array status: 11111111
@               15475: INFO: Start
```

Timing of the Design:

Setup Time:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : MyDesign
Version: K-2015.06-SP1
Date   : Sun Nov 12 14:46:57 2017
****************************************

Operating Conditions: slow   Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm
Wire Load Model Mode: top

  Startpoint: FILTER1_Q2/clk_r_REG862_S7
            (rising edge-triggered flip-flop clocked by clk)
  Endpoint: FILTER1_Q2/clk_r_REG639_S4
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max
```

```
Point                                      Incr       Path
-----------------------------------------------------------------------
clock clk (rise edge)                          0.0000     0.0000
clock network delay (ideal)                    0.0000     0.0000
FILTER1_Q2/clk_r_REG862_S7/CK (SDFF_X2)              0.0000     0.0000 r
FILTER1_Q2/clk_r_REG862_S7/Q (SDFF_X2)               0.4571     0.4571 r
FILTER1_Q2/U78/ZN (AND2_X4)                          0.3335     0.7906 r
FILTER1_Q2/U1/A[5] (mac_3_DW02_mult_3_stage_1)        0.0000     0.7906 r
FILTER1_Q2/U1/mult_97/a[5] (mac_3_DW_mult_tc_1)       0.0000     0.7906 r
FILTER1_Q2/U1/mult_97/U566/ZN (INV_X8)               0.0870     0.8776 f
FILTER1_Q2/U1/mult_97/U743/ZN (XNOR2_X2)             0.3092     1.1868 f
FILTER1_Q2/U1/mult_97/U1204/ZN (INV_X4)              0.2376     1.4243 r
FILTER1_Q2/U1/mult_97/U1202/ZN (NAND3_X4)            0.1787     1.6031 f
FILTER1_Q2/U1/mult_97/U720/ZN (OAI22_X2)             0.3951     1.9982 r
FILTER1_Q2/U1/mult_97/U175/S (HA_X1)                 0.6439     2.6421 r
FILTER1_Q2/U1/mult_97/U974/ZN (XNOR2_X2)             0.3835     3.0255 r
FILTER1_Q2/U1/mult_97/U172/S (FA_X1)                 0.7660     3.7916 f
FILTER1_Q2/U1/mult_97/U1050/ZN (XNOR2_X2)            0.2987     4.0903 f
FILTER1_Q2/U1/mult_97/U1241/ZN (XNOR2_X2)            0.2586     4.3489 f
FILTER1_Q2/U1/mult_97/product[9] (mac_3_DW_mult_tc_1)
                                          0.0000     4.3489 f
FILTER1_Q2/U1/PRODUCT[9] (mac_3_DW02_mult_3_stage_1)
                                          0.0000     4.3489 f
FILTER1_Q2/add_48/B[9] (mac_3_DW01_add_2)            0.0000     4.3489 f
FILTER1_Q2/add_48/U207/ZN (NAND2_X1)                 0.1740     4.5229 r
FILTER1_Q2/add_48/U164/ZN (INV_X1)                   0.0875     4.6104 f
FILTER1_Q2/add_48/U238/ZN (AOI21_X4)                 0.3186     4.9290 r
FILTER1_Q2/add_48/U235/ZN (OAI21_X4)                 0.1060     5.0350 f
FILTER1_Q2/add_48/U170/ZN (NAND2_X2)                 0.1124     5.1474 r
FILTER1_Q2/add_48/U167/ZN (NAND2_X2)                 0.0684     5.2157 f
FILTER1_Q2/add_48/U154/ZN (NAND2_X2)                 0.1102     5.3260 r
FILTER1_Q2/add_48/U245/ZN (NAND2_X4)                 0.0747     5.4007 f
FILTER1_Q2/add_48/U160/ZN (AOI21_X4)                 0.1580     5.5587 r
FILTER1_Q2/add_48/U156/ZN (XNOR2_X2)                 0.3098     5.8685 r
FILTER1_Q2/add_48/SUM[14] (mac_3_DW01_add_2)          0.0000     5.8685 r
FILTER1_Q2/clk_r_REG639_S4/D (DFFR_X2)               0.0000     5.8685 r
data arrival time                                    5.8685

clock clk (rise edge)                          6.2000     6.2000
clock network delay (ideal)                    0.0000     6.2000
clock uncertainty                             -0.0500     6.1500
FILTER1_Q2/clk_r_REG639_S4/CK (DFFR_X2)              0.0000     6.1500 r
library setup time                            -0.2814     5.8686
data required time                                   5.8686
-----------------------------------------------------------------------
data required time                                   5.8686
```

```
  data arrival time                                    -5.8685
  ---------------------------------------------------------------------------
  slack (MET)                                           0.0001
```

Holdtime check:

```
****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : MyDesign
Version: K-2015.06-SP1
Date   : Sun Nov 12 14:44:16 2017
****************************************

Operating Conditions: fast   Library: NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm
Wire Load Model Mode: top

  Startpoint: CONTROL/clk_r_REG827_S4
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: clk_r_REG828_S5
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: min

  Point                                   Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                  0.0000     0.0000
  clock network delay (ideal)            0.0000     0.0000
  CONTROL/clk_r_REG827_S4/CK (DFF_X1)    0.0000     0.0000 r
  CONTROL/clk_r_REG827_S4/Q (DFF_X1)     0.0571     0.0571 r
  CONTROL/quad_count[3] (controller)     0.0000     0.0571 r
  clk_r_REG828_S5/D (DFFR_X1)            0.0000     0.0571 r
  data arrival time                                 0.0571

  clock clk (rise edge)                  0.0000     0.0000
  clock network delay (ideal)            0.0000     0.0000
  clock uncertainty                      0.0500     0.0500
  clk_r_REG828_S5/CK (DFFR_X1)           0.0000     0.0500 r
  library hold time                     -0.0088     0.0412
  data required time                                0.0412
  -----------------------------------------------------------
  data required time                                0.0412
  data arrival time                          -0.0571
  -----------------------------------------------------------
  slack (MET)                                 0.0159
```

Area of the Design:

```
****************************************
Report : area
Design : MyDesign
Version: K-2015.06-SP1
Date   : Sun Nov 12 15:14:59 2017
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

   NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm (File:
/afs/eos.ncsu.edu/lockers/research/ece/wdavis/tech/nangate/NangateOpenCellLibrary_PD
Kv1_2_v2008_10/liberty/520/NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nld
m.db)

Number of ports:                         1136
Number of nets:                          7393
Number of cells:                         6043
Number of combinational cells:           5527
Number of sequential cells:               502
Number of macros/black boxes:               0
Number of buf/inv:                        847
Number of references:                      24

Combinational area:             6710.116009
Buf/Inv area:                    473.746002
Noncombinational area:          2603.076021
Macro/Black Box area:              0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                9313.192030
```

7. Conclusions

Hardware accelerator to implement a simplified Convolutional Neural Network algorithm was designed. Below is the summary of the results obtained

Clock period = 6.2ns
Number of cycle to perform the operation = 1517 cycles
Execution time = 1517 * 6.2 = 9405.4ns
Area obtained = 9313.192030(um^2)
Performance parameter = area * number of cycles * clock period
= 9313.192030*1517*6.2
= 87594296.318962 ns*um^2