ECSE 682 – F17
PROJECT REPORT

NGHIA DOAN

ID: 260-766-103

nghia.doan@mail.mcgill.ca

I. Training Neural Networks

The CIFAR-10 CNNs derived by OPAL are trained using 400 epochs with a batch size of 400 and the optimization algorithm is ADAM. OPAL uses Keras with Tensorflow backend, which is a very powerful python library for deep learning, however, with a typical CNN design space configuration (filter size: 3×3, number of filters: {16,32,64}, number of convolutional layers: {3,4,5}, max pooling filter size: {1×1, 2×2}), the pareto optimal results given by OPAL still do not satisfy the accuracy requirement. The best CNN designs generated by OPAL are shown in Table 1, and the network configuration with index 3 is chosen for the hardware implementation.

Table 1. Pareto optimal results given by OPAL for CIFAR-10 dataset

Conf.	Conv. Filter	Con. Pooling	Conv. Strides	Error rate	Est. HW. Cost
1	[32, 16, 32]	[2, 2, 2]	[1, 1, 2]	0.296960783355	5653.29
2	[32, 32, 16]	[2, 2, 2]	[1, 1, 2]	0.302745098577	3751.13
3	**[16, 16, 16]**	**[2, 2, 2]**	**[1, 1, 1]**	**0.30500000014**	**2982.06**
4	[64, 32, 32]	[2, 2, 2]	[2, 1, 2]	0.309803910115	2568.16
5	[32, 16, 32]	[2, 1, 2]	[2, 2, 2]	0.313823529903	1899.77
6	[16, 16, 16]	[2, 2, 2]	[1, 1, 1]	0.349705902969	1273.26
7	[16, 16, 16]	[2, 2, 2]	[1, 2, 1]	0.412941185867	1223.49

The network parameters compressed in the /ParseWeight/nghia_CNN.npz file can be extracted to mat files by running the /ParseWeight/ParseWeight.py script, after that all parameters should be stored to text files by running the /ParseWeight/ParseWeight.m script.

II. Fixed-point Model

The fixed-point model uses Q3.4 format to approximate all real-valued numbers, i.e. input feature maps, weight/bias values, and output feature maps. Note that the fixed-point model only simulates the computations happened in the convolutional and max pooling layers (included Relu), the final fully connected neural network and its followed activation layer are not considered. The fixed-point model can be evaluated by executing the FixedpointModel/Source/FixedPointConv.cpp script. The quantized output values before applying max pooling with Relu are written to the /HWImplementation/Bin folder, which then are used by the hardware model for asserting its computation correctness. In addition, the quantized values of input feature maps, and weight/bias values are also stored in the same folder as required inputs for the hardware implementation.

III. Hardware implementation

A. Architecture design

The computation of a convolutional layer with 4 input feature maps and 4 output filters is depicted in Fig. 1. This example is also used to explain the chosen hardware architecture. Note that the hardware implementation of this project is made flexible, which can compute any combinations of shapes and sizes of the input feature maps and the filters. Different configurations of the hardware design will be examined in part B for the competition.

As seen from Fig. 1, for each output of the output feature maps, the convolutional computation is as follow:

$$O(i,j,k) = B(i,j,k) + \left\{ \sum_{l=0}^{C_{in}-1} \left[\sum_{m=0}^{H_f-1} \sum_{n=0}^{W_f-1} I(i, S_h j + m, S_w k + n\,) F(m,n) \right] \right\} \tag{1}$$

Where $0 \le i < C_{out}$, $0 \le j < H_{out}$, and $0 \le k < W_{out}$, S_h and S_w are the size of the filter kernels in vertical and horizontal directions respectively. The computation inside the brackets "[]" is denoted as the 2-D convolution, whereas the computation inside the braces "{}" is denoted as the 3-D convolution.

1. Systolic array derivation

As any single output of the output feature maps can be computed by Eq. (1), we can rewrite it as follow when consider one output element only, i.e. by dropping the indices of O.

$$O = B + \sum_{i=0}^{C_{in}W_f\mathrm{H_f}-1} I(i)F(i) \tag{2}$$

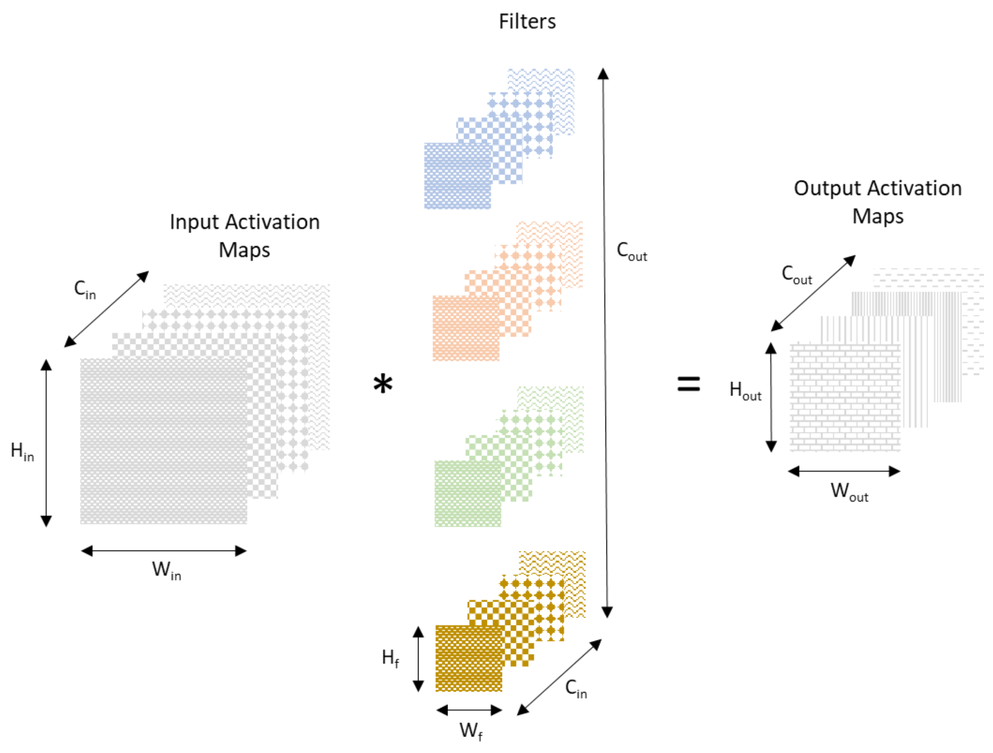


Figure 1. Convolutional layer computation (modified from the project question file)

Where i indicates the paired index between the input activation and the corresponding filter element in the convolution. Let us define $n = C_{in}W_fH_f$, Eq. (2) can be expressed as follow:

$$O = \mathrm{B} + \sum_{\mathrm{i}=0}^{n-1} I(i)F(\mathrm{i}) \tag{3}$$

$$O = \sum_{\mathrm{i}=0}^{\mathrm{n}} I(i)F(i) \tag{4}$$

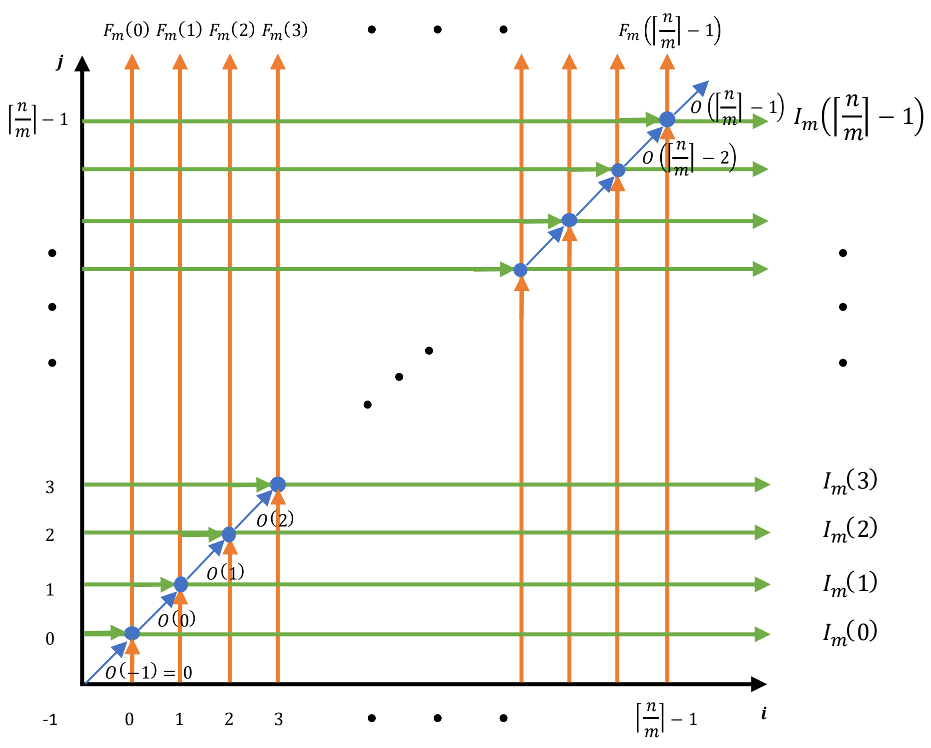
Where $I(n) = \mathrm{B}$, and $F(n) = 1$. One can notice that all required values of $I(i)$ and $F(i)$ are all independent and already available, theoretically Eq. (4) can be executed in one clock cycle. However, to derive a systolic array architecture [1], the data dependency needs to be determined as all PEs in the systolic array must "rhythmically" compute and transfer data to their neighbour PEs. Given a specific hardware implementation, assuming that a portion of m inputs of Eq. (4), $s.t. 1 \leq m \leq n$, can be computed in one clock cycle, Eq. (4) can be rewritten as:

$$O = \sum_{t=0}^{\left\lceil \frac{n}{m} \right\rceil - 1} \sum_{i=0}^{m-1} I(tm + i)F(tm + i) \tag{5}$$

$$O = \sum_{t=0}^{\left\lceil \frac{n}{m} \right\rceil - 1} G(I_m(t), F_m(t)) \tag{6}$$

$$O(t) = O(t-1) + G(I_m(t), F_m(t)) \quad \text{s.t. } 0 \leq t \leq \left\lceil \frac{n}{m} \right\rceil - 1 \tag{7}$$

Where $I_m(t)$ and $F_m(t)$ indicates m inputs needed for a PE at time t, which performs $G\big(I_m(t), F_m(t)\big) = \sum_{i=0}^{m-1} I(tm + i)F(tm + i)$. Eq. (7) assumes a data dependency between the current output value $O(t)$ and its previous value $O(t-1)$, with $O(-1) = 0$. The corresponding Dependence Graph [1] of Eq. (7) is illustrated in Fig. 2. A set of mapping vectors is selected as follow, the projection vector $d = (1,1)$, the processor space vector $p = (-1,1)$, and the scheduling vector $s = (0,1)$. Notice that all nodes $G(i,j)$ always satisfy $i = j$, hence any node $G(i,j)$ is mapped to a processor (PE) with the index: $p^TG = -i + j = 0$, which means only 1 processor (PE) is needed. The mapping conditions $p^Td = 0$ and $s^Td \neq 0$ are also satisfied. The edge mapping results are given in Table 2 and the



$F_m(0)$ $F_m(1)$ $F_m(2)$ $F_m(3)$
$F_m\left(\left\lceil\frac{n}{m}\right\rceil-1\right)$
j
$\left\lceil\frac{n}{m}\right\rceil-1$
$O\left(\left\lceil\frac{n}{m}\right\rceil-1\right)$
$I_m\left(\left\lceil\frac{n}{m}\right\rceil-1\right)$
$O\left(\left\lceil\frac{n}{m}\right\rceil-2\right)$
3
2
1
0
$O(2)$
$O(1)$
$O(0)$
$O(-1)=0$
$I_m(3)$
$I_m(2)$
$I_m(1)$
$I_m(0)$
-1
0
1
2
3
$\left\lceil\frac{n}{m}\right\rceil-1$
i

Fig. 2. The data dependence graph of Eq. (7)

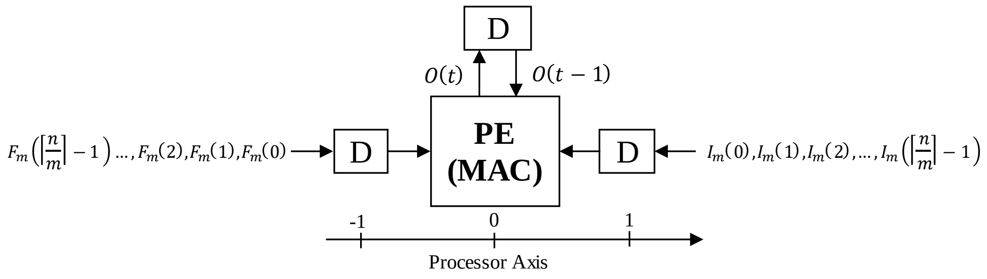
systolic array implementation of Eq. (7) with the mapping results obtained from Table 2 is depicted in Fig. 3. It is worth mentioning that the systolic array architecture has only 1 PE performing the multiplications and accumulations, which is also known as the MAC unit.

Table 2. Systolic array mapping results

Edge name	Edge direction ($p^T e$)	Num. Of Delays ($s^T e$)
Input feature map $I(1,0)$	-1	1
Filter map $F(0,1)$	1	1
Output $F(0,1)$	0	1

It is possible to use 1 MAC unit to perform the convolutional computation, doing so requires $\left\lceil\frac{n}{m}\right\rceil$ clock cycles for one output. However, to minimize the computation time, m needs to be increased, which means $2m$ number of inputs are fed into the MAC unit at the same time and the MAC unit must be able to process m multiplications and m additions within 1 clock cycle, which is impossible for large m. Because of this reason, instead the architecture in this project uses p parallel MACs to compute p different functions $G(I_m(t), F_m(t))$, each G function is completed in m clock cycles, then the partial sum is accumulated to calculate the final output O in $\left\lceil\frac{n}{pm}\right\rceil$ clock cycles, where $m = W_f \mathrm{H_f}$ and p can be determined based on the input channel size. The number of clock cycles needed to compute one output element given the proposed approach is: $T_{proposed} = \left\lceil\frac{n}{pm}\right\rceil(m+1)$.

The proposed design is faster than the systolic array approach when n and m are given if:



D
$O(t)$
$O(t-1)$
PE
(MAC)
$F_m\left(\left\lceil\frac{n}{m}\right\rceil-1\right)\ldots,F_m(2),F_m(1),F_m(0)$
D
D
$I_m(0),I_m(1),I_m(2),\ldots,I_m\left(\left\lceil\frac{n}{m}\right\rceil-1\right)$
-1
0
1
Processor Axis

Fig. 3. The derived systolic array architecture

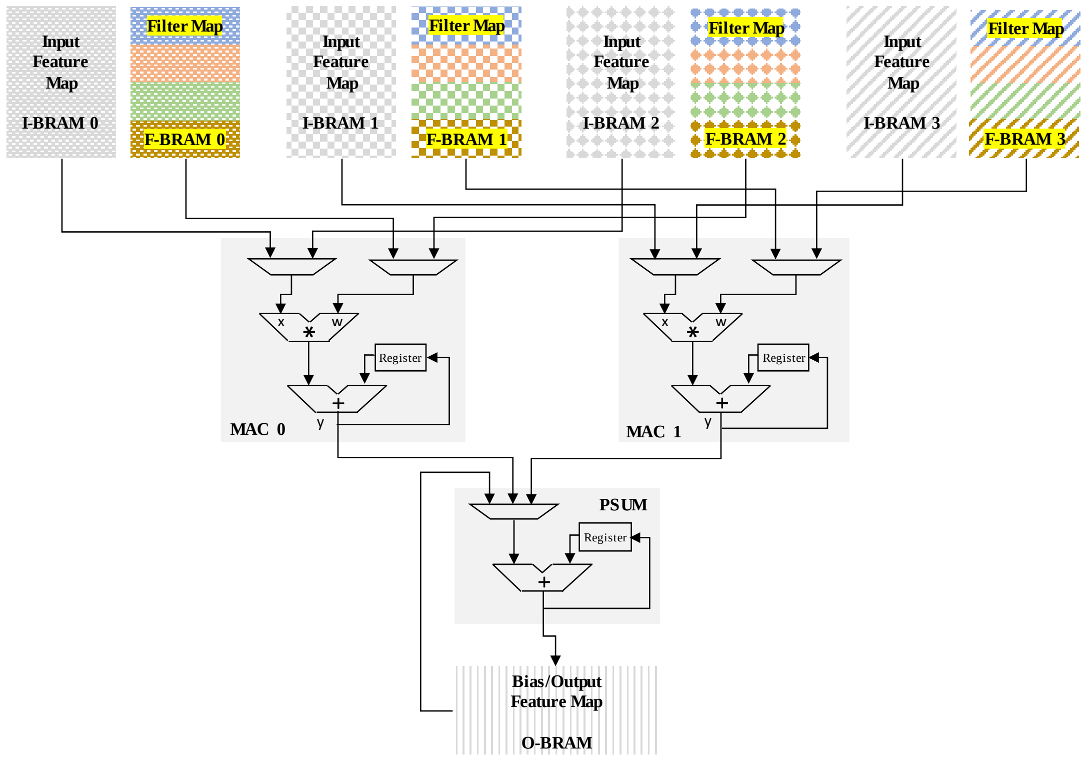


Fig. 4. The hardware architecture of the project

$$T_{proposed} = \left\lceil \frac{n}{pm} \right\rceil (m+1) < T_{sys.Array} = \left\lceil \frac{n}{m} \right\rceil \tag{8}$$

As $\left\lceil \frac{n}{pm} \right\rceil (m+1) < \left(\frac{n}{pm}+1\right)(m+1)$ and $\left\lceil \frac{n}{m} \right\rceil > \frac{n}{m}$, if p is a solution of Eq. (9) then p is also a solution of Eq. (8).

$$\left(\frac{n}{pm}+1\right)(m+1) \quad < \frac{n}{m} \tag{9}$$

$$\frac{n(m+1)}{n-m(m+1)} \quad < p \qquad \text{with } m(m+1) < n \tag{10}$$

It is clear that designing p MAC units that performs m multiplications and m additions in m clock cycles is more practically than designing 1 MAC unit that performs the same amount of computation in 1 clock cycle. In addition, with p satisfies Eq. (10) the proposed approach is also faster than the systolic array approach.

2. Hardware architecture

For the purpose of illustration, Fig. 4 shows the architecture that implements the convolutional computation in Fig. 1. As the design exploits the parallelism existing in each input channel, which means each MAC unit oversees the 2D convolution in some specific input channels. It leads to a data partitioning requirement for the filter and input values such that each coupled I-BRAM-i and F-BRAM-i store the input values and filter values for only one input channel, respectively, and hence they are depicted with the same texture. In addition, the filter blocks that have the same color indicating that they are in the same output channel. Given the 4 input channels and only 2 MACs, for each output channel, the applied scheduling is that the first and second input channels are concurrently convolved with their corresponding filters in MAC 0 and MAC 1, respectively, their outputs are then accumulated and stored in the PSUM registers. At the second iteration, the third and fourth input channels are computed in MAC 0 and MAC 1, respectively, and their outputs are also accumulated to PSUM registers. Once all convolutional computation of all input channels is done, the PSUM reads the bias values stored in the output BRAM (O-BRAM) and accumulates it to the partial sum. The output of the PSUM unit is then truncated before writing back to O-BRAM. The same approach is sequentially applied for all other output channels. All MAC units and BRAMs are scheduled and their data flow is driven by an FSM, which acts like the main Controller of the design. It should be noted that thanks to the presence of MUX components, no redundant data needs to be duplicated. However, in the real implementation, all filter values, i.e. all output channels and input channels, are stored in a BRAM that connects to one MAC unit. This causes a data duplication of the filter values when more than one MAC unit are used.

Fig. 5 shows a complete computation required for the first output in the second layer, which has 16 input channels with the filter kernel is equal 3×3, where only 4 MAC units are made use of. Obviously, the computation requires 4 iterations, in each iteration the partial sum of the convolution between 4 input channels and their corresponding filters are concurrently executed. The first iteration happens from Cursor 1 at 140000ps to Cursor 2 at 240000ps, which is 10 clock cycles, one clock cycle is needed for the state transition from convolutional computation to partial sum update. Once the first partial sum update is finished, Cursor 3 at 290000ps, all MAC unit are reset and start the next iteration of the convolution again. After the last iteration, Cursor 8 at 920000ps, the current value of the partial sum is: $psum = (0 - 3 + 0 + 3 + 0) + (-2 - 7 + 0 + 9) + (-21 + 0 + 0 + 0) + (0 - 37 + 5 + 1) = -52$, the bias value is then added up to the $psum = -52 - 2 = -54$. This value is then stored to the same O-BRAM address where the corresponding bias value is read from at Cursor 10 at 970000ps, finishing a complete computation for one output feature map element.

Fig. 6,7 and 8 show the experimental results of the same configuration applied for all 3 convolution layers. Note that the output values of the fixpoint model and that of the hardware implementation are completely identical, which are check by an assert function in the testbench module. The VHDL codes of the hardware implementation can be found in the folder named HWImplementation. To evaluate the hardware implementation, include all VHDL files into a MODELSIM project then simulate the HWImplementation/controller_tb.vhd file. Please make sure that the layer configurations in the HWImplementation/controller.vhd file is matched with that of the testbench module for a correct output assertion.

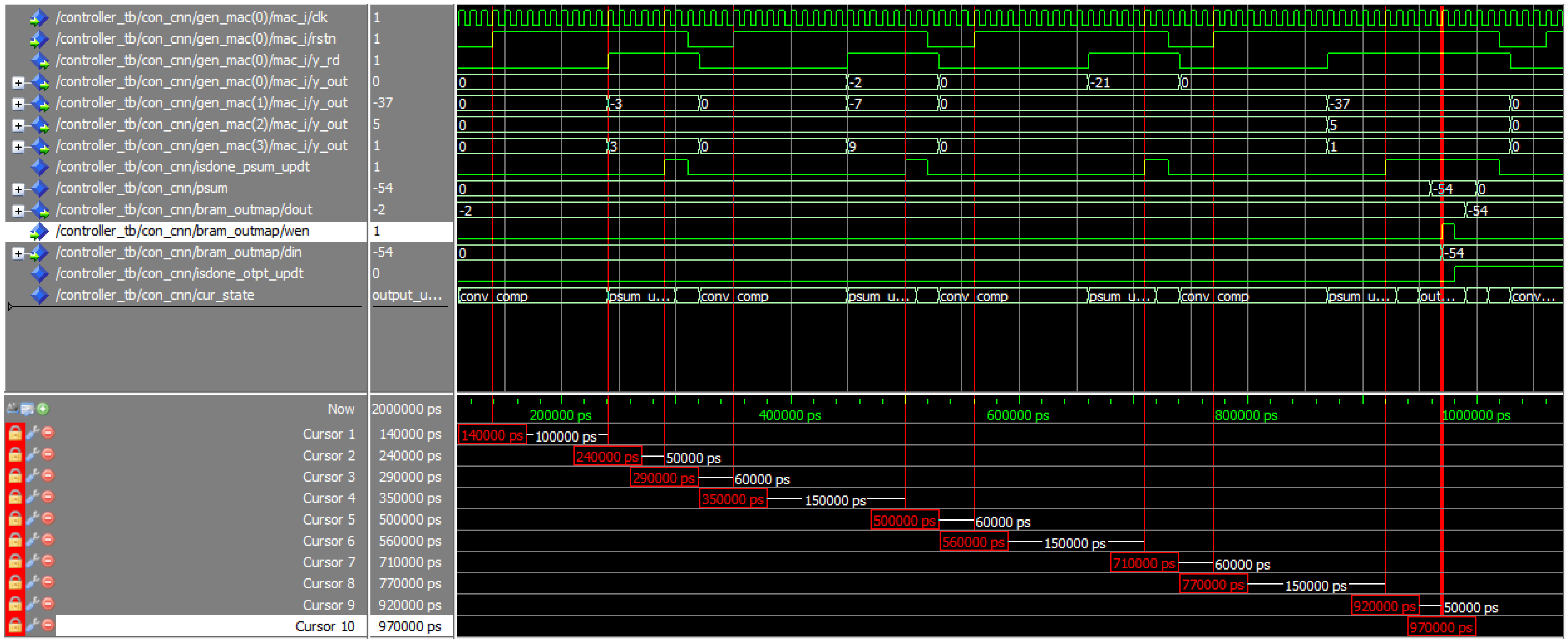
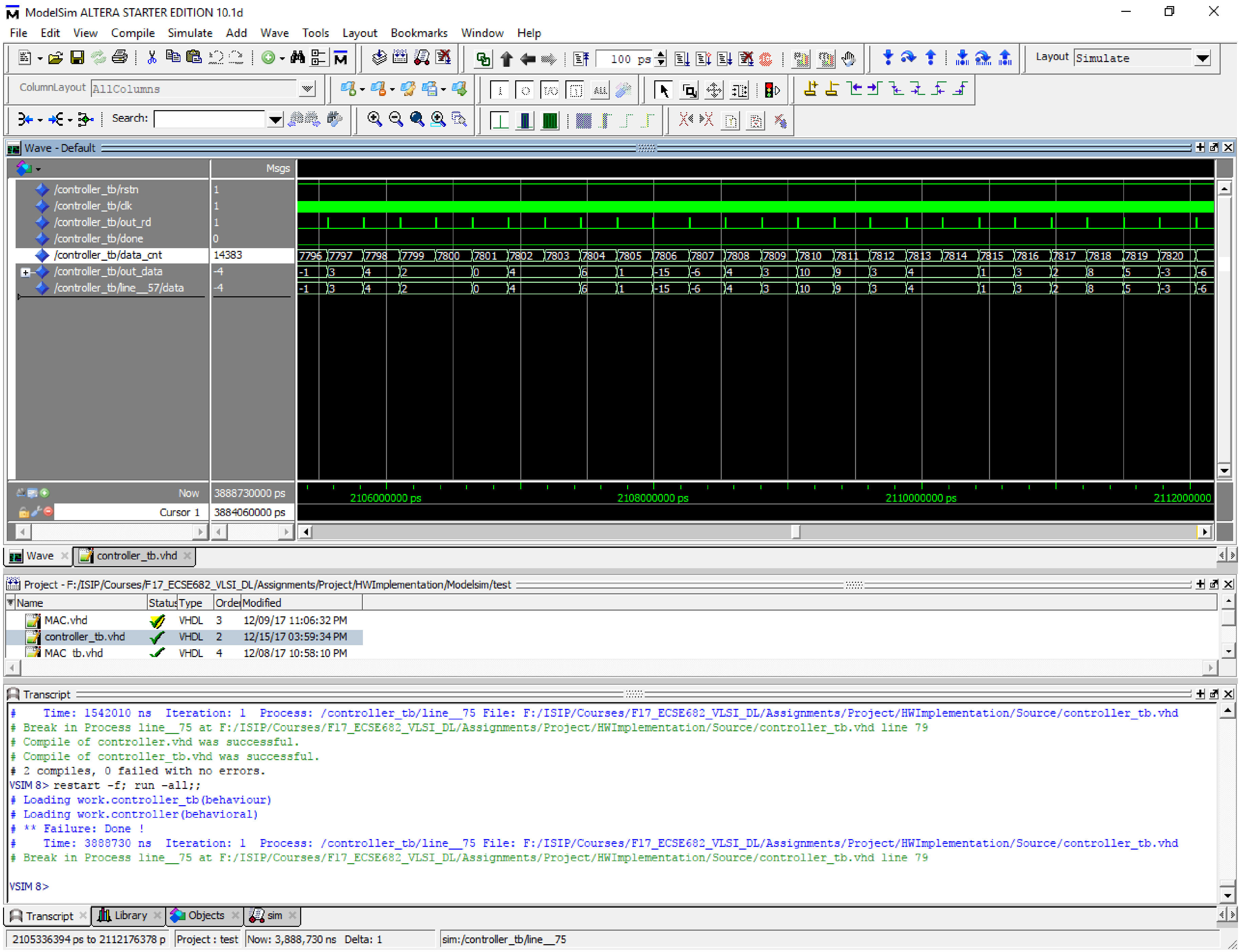

/controller_tb/con_cnn/gen_mac(0)/mac_i/clk 1
/controller_tb/con_cnn/gen_mac(0)/mac_i/rstn 1
/controller_tb/con_cnn/gen_mac(0)/mac_i/y_rd 1
/controller_tb/con_cnn/gen_mac(0)/mac_i/y_out 0
/controller_tb/con_cnn/gen_mac(1)/mac_i/y_out -37
/controller_tb/con_cnn/gen_mac(2)/mac_i/y_out 5
/controller_tb/con_cnn/gen_mac(3)/mac_i/y_out 1
/controller_tb/con_cnn/isdone_psum_updt 1
/controller_tb/con_cnn/psum -54
/controller_tb/con_cnn/bram_outmap/dout -2
/controller_tb/con_cnn/bram_outmap/wen 1
/controller_tb/con_cnn/bram_outmap/din -54
/controller_tb/con_cnn/isdone_otpt_updt 0
/controller_tb/con_cnn/cur_state output_u...
conv_comp
psum_u...
out...
conv...
Now 2000000 ps
200000 ps
400000 ps
600000 ps
800000 ps
1000000 ps
Cursor 1 140000 ps
Cursor 2 240000 ps
Cursor 3 290000 ps
Cursor 4 350000 ps
Cursor 5 500000 ps
Cursor 6 560000 ps
Cursor 7 710000 ps
Cursor 8 770000 ps
Cursor 9 920000 ps
Cursor 10 970000 ps
100000 ps
50000 ps
60000 ps
150000 ps
60000 ps
150000 ps
60000 ps
150000 ps
50000 ps

Fig. 5. Convolutional computation for one output feature map element with 4 MAC units, 16 input channels and 3×3 filter kernel



```
/controller_tb/rstn          1
/controller_tb/clk           1
/controller_tb/out_rd        1
/controller_tb/done          0
/controller_tb/data_cnt      14383
/controller_tb/out_data      -4
/controller_tb/line__57/data -4

data_cnt:  7796 7797 7798 7799 7800 7801 7802 7803 7804 7805 7806 7807 7808 7809 7810 7811 7812 7813 7814 7815 7816 7817 7818 7819 7820
out_data:  -1 3 4 2 0 4 6 1 -15 -6 4 3 10 9 3 4 1 3 2 8 5 -3 -6
line__57/data: -1 3 4 2 0 4 6 1 -15 -6 4 3 10 9 3 4 1 3 2 8 5 -3 -6

Now     3888730000 ps
Cursor 1 3884060000 ps
```

Project - F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Modelsim/test

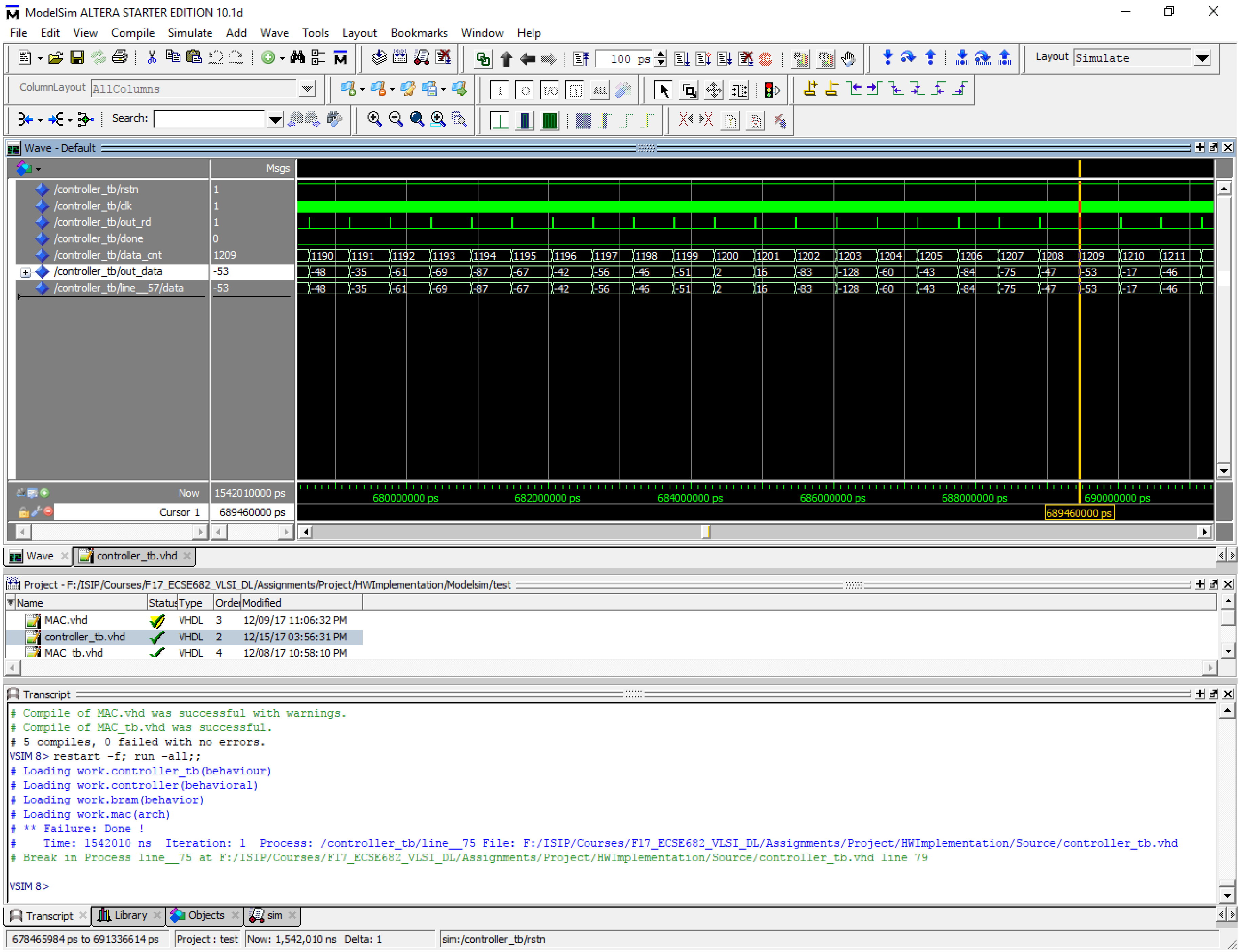
Name	Status	Type	Order	Modified
MAC.vhd	✓	VHDL	3	12/09/17 11:06:32 PM
controller_tb.vhd	✓	VHDL	2	12/15/17 03:59:34 PM
MAC_tb.vhd	✓	VHDL	4	12/08/17 10:58:10 PM

```
#    Time: 1542010 ns  Iteration: 1  Process: /controller_tb/line__75 File: F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd
# Break in Process line__75 at F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd line 79
# Compile of controller.vhd was successful.
# Compile of controller_tb.vhd was successful.
# 2 compiles, 0 failed with no errors.
VSIM 8> restart -f; run -all;;
# Loading work.controller_tb(behaviour)
# Loading work.controller(behavioral)
# ** Failure: Done !
#    Time: 3888730 ns  Iteration: 1  Process: /controller_tb/line__75 File: F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd
# Break in Process line__75 at F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd line 79

VSIM 8>
```

```
2105336394 ps to 2112176378 p   Project : test   Now: 3,888,730 ns  Delta: 1   sim:/controller_tb/line__75
```

Fig. 6. Output comparison of the fixed-point model and hardware implementation of Conv. Layer 0



```
/controller_tb/rstn          1
/controller_tb/clk           1
/controller_tb/out_rd        1
/controller_tb/done          0
/controller_tb/data_cnt      1209
/controller_tb/out_data      -53
/controller_tb/line__57/data -53
Now      1542010000 ps
Cursor 1  689460000 ps

data_cnt:  1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211
out_data:  -48 -35 -61 -69 -87 -67 -42 -56 -46 -51 2 16 -83 -128 -60 -43 -84 -75 -47 -53 -17 -46
line__57/data: -48 -35 -61 -69 -87 -67 -42 -56 -46 -51 2 16 -83 -128 -60 -43 -84 -75 -47 -53 -17 -46
```

Project - F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Modelsim/test

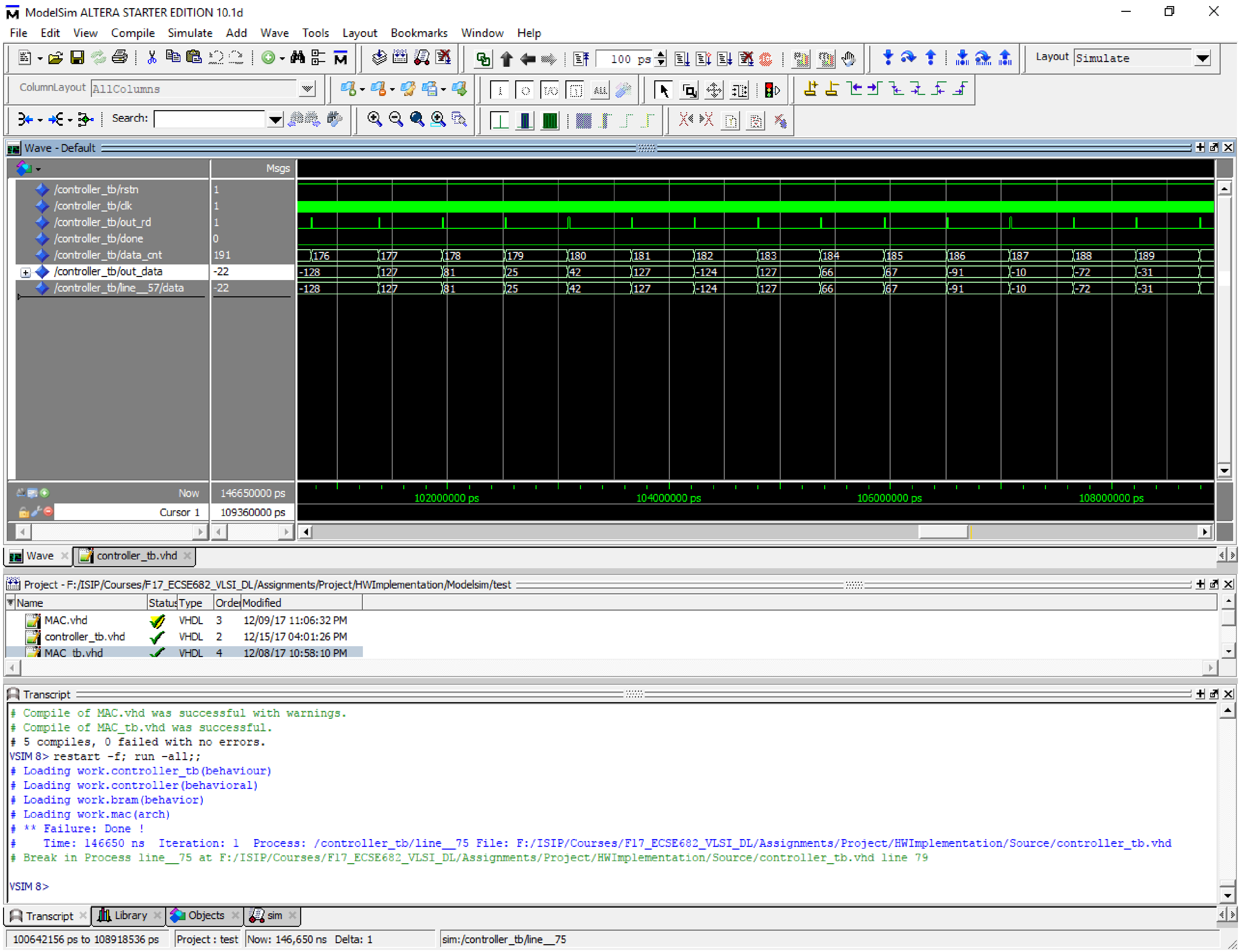
Name	Status	Type	Order	Modified
MAC.vhd	✓	VHDL	3	12/09/17 11:06:32 PM
controller_tb.vhd	✓	VHDL	2	12/15/17 03:56:31 PM
MAC_tb.vhd	✓	VHDL	4	12/08/17 10:58:10 PM

```
# Compile of MAC.vhd was successful with warnings.
# Compile of MAC_tb.vhd was successful.
# 5 compiles, 0 failed with no errors.
VSIM 8> restart -f; run -all;;
# Loading work.controller_tb(behaviour)
# Loading work.controller(behavioral)
# Loading work.bram(behavior)
# Loading work.mac(arch)
# ** Failure: Done !
#    Time: 1542010 ns  Iteration: 1  Process: /controller_tb/line__75 File: F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd
# Break in Process line__75 at F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd line 79

VSIM 8>
```

```
678465984 ps to 691336614 ps   Project : test   Now: 1,542,010 ns  Delta: 1   sim:/controller_tb/rstn
```

Fig. 7. Output comparison of the fixed-point model and hardware implementation of Conv. Layer 1



```
/controller_tb/rstn          1
/controller_tb/clk           1
/controller_tb/out_rd        1
/controller_tb/done          0
/controller_tb/data_cnt      191     176  177  178  179  180  181   182  183  184  185  186  187  188  189
/controller_tb/out_data      -22     -128 127  81   25   42   127  -124  127  66   67   -91  -10  -72  -31
/controller_tb/line__57/data -22     -128 127  81   25   42   127  -124  127  66   67   -91  -10  -72  -31

Now       146650000 ps
Cursor 1  109360000 ps
```

Project - F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Modelsim/test

Name	Status	Type	Order	Modified
MAC.vhd	✓	VHDL	3	12/09/17 11:06:32 PM
controller_tb.vhd	✓	VHDL	2	12/15/17 04:01:26 PM
MAC_tb.vhd	✓	VHDL	4	12/08/17 10:58:10 PM

```
# Compile of MAC.vhd was successful with warnings.
# Compile of MAC_tb.vhd was successful.
# 5 compiles, 0 failed with no errors.
VSIM 8> restart -f; run -all;;
# Loading work.controller_tb(behaviour)
# Loading work.controller(behavioral)
# Loading work.bram(behavior)
# Loading work.mac(arch)
# ** Failure: Done !
#    Time: 146650 ns  Iteration: 1  Process: /controller_tb/line__75 File: F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd
# Break in Process line__75 at F:/ISIP/Courses/F17_ECSE682_VLSI_DL/Assignments/Project/HWImplementation/Source/controller_tb.vhd line 79

VSIM 8>
```

```
100642156 ps to 108918536 ps   Project : test   Now: 146,650 ns  Delta: 1   sim:/controller_tb/line__75
```

Fig. 8. Output comparison of the fixed-point model and hardware implementation of Conv. Layer 2

B. Performance and utilization comparison of different designs

Table 3. Performance and hardware utilization of various designs at the first convolution layer

Conf.	H_{out} (W_{out})	C_{out}	Num. MAC	Block memory (bits)	Mem. Accesses (MBytes)	Latency (clock cycles)	Logic utilization (ALMs, max 9430 ALMs)	Error rate	Weighted Cost
1	30	32	16	365568	201.6	806400	553	0.297	0.754
3	**30**	**16**	**16**	**153600**	**100.8**	**403200**	**531**	**0.305**	**0.511**
4	15	64	22	427776	100.8	403200	9231	0.310	0.75

For the competition, the CNNs with configuration indices 1,3, and 4 from Table 1 are chosen. The synthesis results given in Table 3 only consider for the first convolutional layer, as with the current network sizes of those CNNs, the first convolution layer requires the largest number of data accesses and computation among all layers. H_{out} and C_{out} are obtained as in [2], given the stride values, input feature map size (32×32), and filter kernel size (3×3). The error rates are rounded from the results given by OPAL. Because the implemented design only exploits parallelism at the level of input channel, the number of MAC units for all configurations are set to the largest possible number of C_{in} in all three layers, allowing that the design can be fit into the given chip. The block memory sizes and logic utilization are obtained from Quartus. The latency is obtained by measuring the computation time between two consecutive output elements from the testbench module, i.e. the time between two consecutive rising edges of the out_rd signal as in Fig. 6,7, and 8, then this number is multiplied by the total number of output elements in the first convolution layer. The memory accesses are calculated as $Mem\ Accesses = C_{out}H_{out}W_{out}(2C_{in}H_fW_f + 2)$ as for each output elements, the input feature map and the filter values each is read from the memory $C_{in}H_fW_f$ times, and the bias value is read from O-BRAM once before the output is written back the O_BRAM at the same address. The weighted cost is the average value of four substituent costs namely memory accesses, latency, logic utilization and error rate. Each substituent code is normalized with the maximum value in the same column before being accounted for the weighted cost. As a result, Configuration 3 is chosen for the competition as it has the smallest weighted cost. Please also noted that, unlike the input feature BRAMs, the memory for filter values are increased when the number of MACs are increased, therefore Configuration 4 design has the same memory accesses as that of Configuration 3 design, but more redundant memory is consumed by the filter values.

REFERENCE

[1] Parhi, Keshab K. VLSI digital signal processing systems: design and implementation. John Wiley & Sons, 2007.
[2] Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).