ECE 284 PROJECT

Chenyu Wu chw080@ucsd.edu
Yizheng Yu yiy003@ucsd.edu
Zian Wang ziw080@ucsd.edu
Haoliang Wei h5wei@ucsd.edu

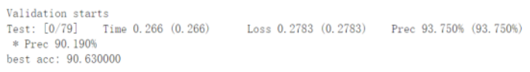
INTRODUCTION

The size of data set and number of calculations of machine learning grow rapidly these days. Therefore, the computational module deployed in machine learning model requires higher speed. In this project, a 2D systolic array computing architecture will be implemented and optimized for accelerating the machine learning training. After completing the basic functions of computation, we will optimize the latency and power consumption of the processor. The whole architecture will be discussed in this report.

Part1. VGG16 & RESNET quantization-aware training

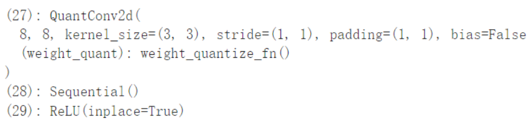
VGG16:

Fig.1 shows the accuracy of VGG16 is 90.63%, Fig.2 shows the 27th layer of the VGG16 model which is set to 8 input channels, 8 output channels without batch normalization and Fig.3 shows the calculated difference between recovered values and prehook values is 2.2645*10^{-7}.



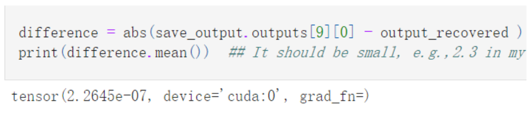
```
Validation starts
Test: [0/79]    Time 0.266 (0.266)      Loss 0.2783 (0.2783)    Prec 93.750% (93.750%)
 * Prec 90.190%
best acc: 90.630000
```

Fig. 1: Accuracy of VGG16



```
(27): QuantConv2d(
  8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
(28): Sequential()
(29): ReLU(inplace=True)
```

Fig. 2: VGG16 Model



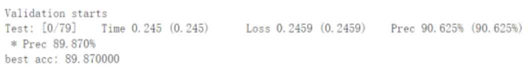
```
difference = abs(save_output.outputs[9][0] - output_recovered )
print(difference.mean())  ## It should be small, e.g.,2.3 in my
```

```
tensor(2.2645e-07, device='cuda:0', grad_fn=)
```

Fig. 3: Difference between psum_recoverd and actual values

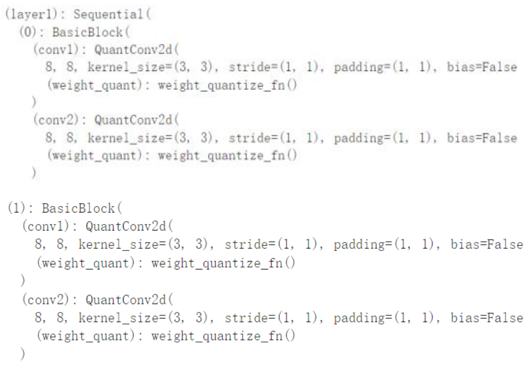
RESNET:

Same as VGG16, the accuracy of RESNET is 89.87%. Fig.5 shows the new RESNET model. And the difference is 3.4393*10^{-7}.



```
Validation starts
Test: [0/79]    Time 0.245 (0.245)      Loss 0.2459 (0.2459)    Prec 90.625% (90.625%)
 * Prec 89.870%
best acc: 89.870000
```

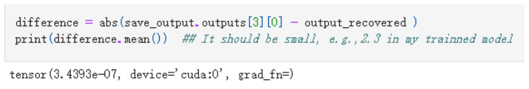
Fig. 4 Accuracy of RESNET



```
(layer1): Sequential(
  (0): BasicBlock(
    (conv1): QuantConv2d(
      8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )

(1): BasicBlock(
  (conv1): QuantConv2d(
    8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
```

Fig. 5 RESNET Model



```
difference = abs(save_output.outputs[3][0] - output_recovered )
print(difference.mean())  ## It should be small, e.g.,2.3 in my trainned model
```

```
tensor(3.4393e-07, device='cuda:0', grad_fn=)
```

Fig. 6 Difference between psum_recoverd and actual values

Part2. RTL code design

Fig.6 shows the entire circuit hierarchy and there's no compilation error.

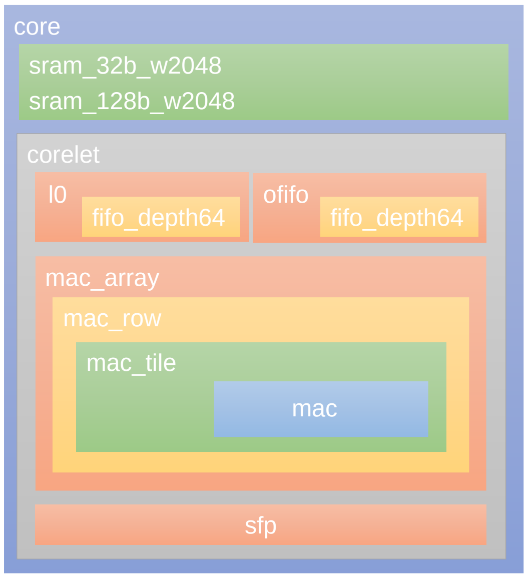
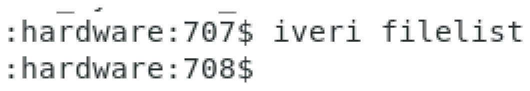


Fig. 6: Circuit hierarchy

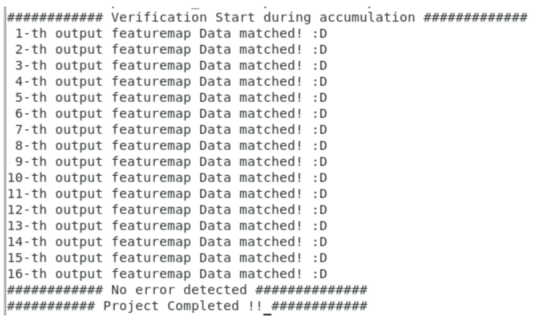


```
:hardware:707$ iveri filelist
:hardware:708$
```

Fig. 7 Compilation

Part3. Test bench generation and verification results

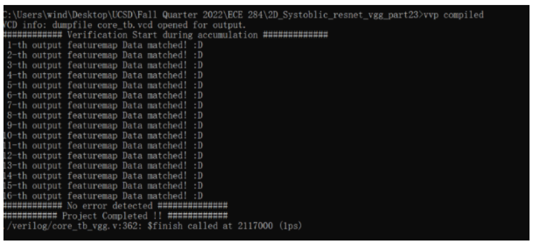
With VGG16 model, we have 8 input channels, 8 output channels, input feature map with 6*6 after padding and output feature map with 4*4. The comparation results are shown in Fig. 8. We make all 10 steps working separately, such as reading data to memory, loading data to fifos, execution and etc. You will see we make some of those steps working in parallel.



```
############ Verification Start during accumulation #############
 1-th output featuremap Data matched! :D
 2-th output featuremap Data matched! :D
 3-th output featuremap Data matched! :D
 4-th output featuremap Data matched! :D
 5-th output featuremap Data matched! :D
 6-th output featuremap Data matched! :D
 7-th output featuremap Data matched! :D
 8-th output featuremap Data matched! :D
 9-th output featuremap Data matched! :D
10-th output featuremap Data matched! :D
11-th output featuremap Data matched! :D
12-th output featuremap Data matched! :D
13-th output featuremap Data matched! :D
14-th output featuremap Data matched! :D
15-th output featuremap Data matched! :D
16-th output featuremap Data matched! :D
############ No error detected ##############
########### Project Completed !!_############
```

Fig. 8 VGG16 Comparation results

As for RESNET model, we still have 8 input and output channels. But we have 34*34 feature maps with padding. So we pick only 3*10 of them to calculate the first 8 output results. In addition to the residual x, all results are matched as shown in Fig. 9.



```
:\Users\wind\Desktop\UCSD\Fall Quarter 2022\ECE 284\2D_Systoblic_resnet_vgg_part23>vvp compiled
CD info: dumpfile core_tb.vcd opened for output.
############ Verification Start during accumulation #############
 1-th output featuremap Data matched! :D
 2-th output featuremap Data matched! :D
 3-th output featuremap Data matched! :D
 4-th output featuremap Data matched! :D
 5-th output featuremap Data matched! :D
 6-th output featuremap Data matched! :D
 7-th output featuremap Data matched! :D
 8-th output featuremap Data matched! :D
 9-th output featuremap Data matched! :D
10-th output featuremap Data matched! :D
11-th output featuremap Data matched! :D
12-th output featuremap Data matched! :D
13-th output featuremap Data matched! :D
14-th output featuremap Data matched! :D
15-th output featuremap Data matched! :D
16-th output featuremap Data matched! :D
############ No error detected ##############
########### Project Completed !! ############
/verilog/core_tb_vgg.v:362: $finish called at 2117000 (1ps)
```

Fig. 9 RESNET comparation results

Part4. Mapping on FPGA (Cyclone IV)

After finishing the design, we map our design on FPGA and get the circuit specifications as shown in Table 1.

Table 1: FPGA mapping report

Paramete	Value
Frequency/MHz	127.32
Logic elements	17293
Register	12098
Pins	451
Total power/mW	335.78
Dynamic pow/mW	33.04
Static pow/mW	119.59
I/O pow/mW	183.15

Part4. Alpha

Up till now, we have built the based 2D systolic architecture machine learning accelerator, which achieves the calculations for both VGG16 convolution model and RESNET model. However, all steps are processed separately, which wastes a lot of time and power. Based on that, we try to optimize the design in three ways listed below.

- **Dual port ram design**

In previous design, accumulations need psum memory to spit out the value stored after mac array execution, and memory needs to store the accumulated final results. This would cause conflicts in the signal port RAM since we cannot read and write simultinuously. So, we design a dual port RAM. It helps write and read at the same time, which highly optimize the latency. This RAM has both write and read address. The write and read enable are controlled by the separated signal (WEN and REN). The structure is shown in Fig 10.

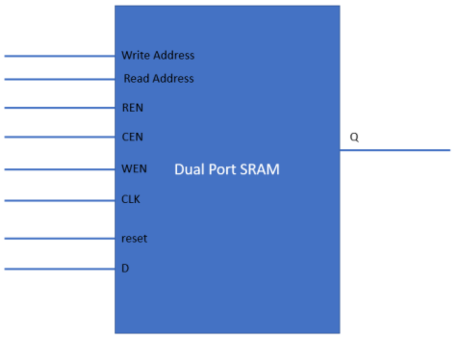


Fig. 10 Dual port RAM structure

- **Ofifo-memory controller**

In the previous design, write and read signals are given by the testbench. Ofifo produces output each time the execution is done. However, in real design, ofifo should spit out output while executing. So, in this case, we add a control unit. Once ofifo_valid is enabled, it means the data is ready in ofifo. So, the control unit can send write signal to the psum memory and it can catch the data from the ofifo. Also, CTRL unit can send add signal to the memory to let the memory provide the data for sfp unit calculating the convolution data. The controller structure is clearly shown in Fig. 11.

- **Memory double buffering**

The third optimization we have done is memory double buffering for loading weights into the memory and writing into l0 fifo simultaneously. In order to achieve this, we add two more RAM instances to store weights. So, like Ping-pong, during one cycle, we load the weights from outside into one of the RAMs and write weights in the other RAM into l0 fifo. Then continue to finish the calculations. The circuit structure is shown in Fig. 11.

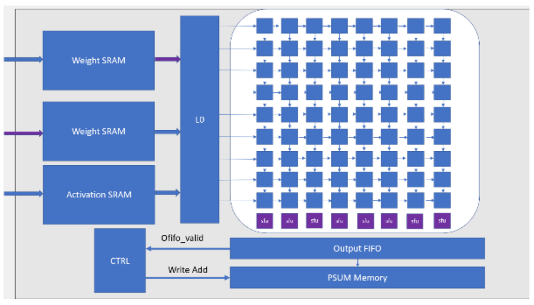
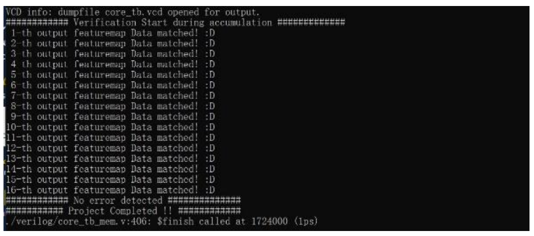


Fig. 11 Process structure with controller and memory double buffering

After doing that, we test the results in new circuit, which has a dual-port RAM, a controller and memory double buffer. The comparation results are shown in Fig. 12. In addition, we map the optimized circuit into FPGA again, and get the specifications as shown in Table 2.



```
VCD info: dumpfile core_tb.vcd opened for output.
############ Verification Start during accumulation #############
 1-th output featuremap Data matched! :D
 2-th output featuremap Data matched! :D
 3-th output featuremap Data matched! :D
 4-th output featuremap Data matched! :D
 5-th output featuremap Data matched! :D
 6-th output featuremap Data matched! :D
 7-th output featuremap Data matched! :D
 8-th output featuremap Data matched! :D
 9-th output featuremap Data matched! :D
10-th output featuremap Data matched! :D
11-th output featuremap Data matched! :D
12-th output featuremap Data matched! :D
13-th output featuremap Data matched! :D
14-th output featuremap Data matched! :D
15-th output featuremap Data matched! :D
16-th output featuremap Data matched! :D
############ No error detected ##############
########### Project Completed !! ############
./verilog/core_tb_mem.v:406: $finish called at 1724000 (1ps)
```

Fig. 12 Printed results with all optimizations

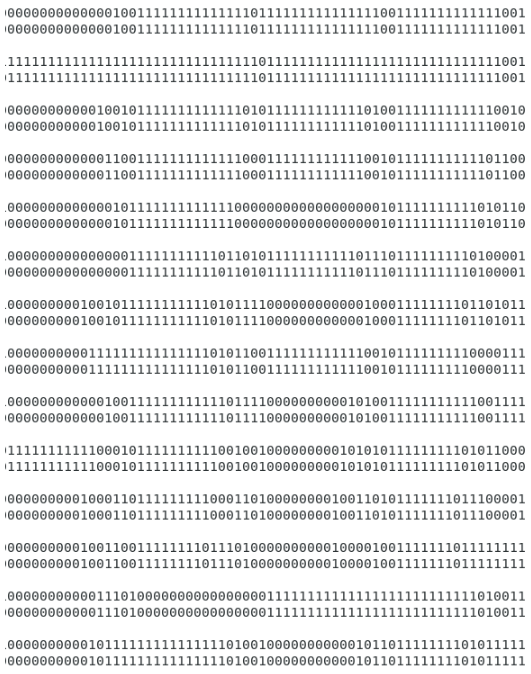
Table 2 New circuit FPGA mapping report

Paramete	Value
Frequency/MHz	121.82
Logic elements	16531
Register	12000
Pins	459
Total power/mW	333.24
Dynamic pow/mW	26.52
Static pow/mW	119.55
I/O pow/mW	187.17

Compared with the original circuit, we have lower dynamic power consumption. However, all the difference is very slight. Therefore, after analyzation, we think the advantages of these optimizations are only helping reduce the latency of the process but not the frequency of the clock and other specifications of the circuit.

Additional verification

In addition to basic verification for 8*8 case, we further verify the 4*4 condition. The results are still matching as shown in fig. 13. Even though we get the correct results for first 4 output channels. There're still issues for the idle ports. The reason for those errors is that we don't modify the testbench accordingly. But it can still prove that our design logic is totally correct.



```
0000000000000100111111111111110111111111111111100111111111111111001
0000000000000100111111111111110111111111111111100111111111111111001

1111111111111111111111111111111011111111111111111111111111111111001
1111111111111111111111111111111011111111111111111111111111111111001

0000000000001001011111111111111010111111111111110100111111111111110010
0000000000001001011111111111111010111111111111110100111111111111110010

0000000000001100111111111111110001111111111111100101111111111111101100
0000000000001100111111111111110001111111111111100101111111111111101100

0000000000000101111111111111100000000000000000000010111111111111010110
0000000000000101111111111111100000000000000000000010111111111111010110

0000000000000000111111111111011010111111111111101110111111111110100001
0000000000000000111111111111011010111111111111101110111111111110100001

0000000001001011111111111111010111000000000000001000111111111101101011
0000000001001011111111111111010111000000000000001000111111111101101011

0000000000111111111111111110101100111111111111110010111111111110000111
0000000000111111111111111110101100111111111111110010111111111110000111

0000000000000100111111111111110111100000000000101001111111111111001111
0000000000000100111111111111110111100000000000101001111111111111001111

1111111111100010111111111111001001000000000010101011111111111101011000
1111111111100010111111111111001001000000000010101011111111111101011000

0000000001000110111111111111000110100000000010011010111111111011100001
0000000001000110111111111111000110100000000010011010111111111011100001

0000000001001100111111111101110100000000000010000100111111111011111111
0000000001001100111111111101110100000000000010000100111111111011111111

0000000000011101000000000000000000001111111111111111111111111111111010011
0000000000011101000000000000000000001111111111111111111111111111111010011

0000000000101111111111111111110100100000000000001011011111111110101111
0000000000101111111111111111110100100000000000001011011111111110101111
```

Fig. 13 Results compared in 4*4 condition

Conclusion

For the ECE284 final project, we demonstrate a methodology to implement a 2D systolic architecture machine learning accelerator. Also, some techniques such as dual-port RAM, memory double buffering and etc. are being applied to the design. It turns out to produce a marginal improvement in circuit specifications. After analyzation, we found that only the latency is benefit from those techniques, which makes a lot sense. Otherwise, the design also passes tests in other extreme conditions. So, we strongly believe it's a good design now.