

# Report

## Part1. Train VGG16 with quantization-aware training

1. Squeezing and Removing Batch Normalisation: We updated the following line in the VGG\_quant file to pick a layer in the middle and update its input and output channels to '8'.

**Original Command:** 'VGG16\_quant': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']

**Updated Command:** 'VGG16\_quant': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 8, '8i', 512, 'M', 512, 512, 512, 'M']

We made the above changes by selecting the layers corresponding to features [24] and [27]. After changing the layer in the above command, we also removed the batch normalization function from our selected layer by adding a statement in the \_make\_layer function for x=8i.

```
elif x == '8i':  
    layers += [QuantConv2d(in_channels, 8, kernel_size=3, padding=1),  
               nn.ReLU(inplace=True)]  
    in_channels = 8
```

2. The hyperparameters of the model were then updated to achieve an accuracy of 91% after updating the VGG model.

```
adjust_list = [20, 50]  
if epoch in adjust_list:  
    for param_group in optimizer.param_groups:  
        param_group['lr'] = param_group['lr'] * 0.1  
  
lr = 1e-2  
weight_decay = 1e-3  
epochs = 80
```

3. We grabbed the inputs of the layer corresponding to features[27] that was outputs[8]. It was then used in the rest of our project.

```
x_bit = 4  
x = save_output.outputs[8][0] # input of the 2nd conv layer  
x_alpha = model.features[27].act_alpha  
x_delta = x_alpha/(2**x_bit-1)
```

4. The difference between psum\_recovered and psum\_ref was calculated to see the impact of quantized sending input to the network.

```
difference = abs( output_ref[0] - output_recovered[0] )  
print(difference.mean()) ## It should be small, e.g., 2.3 in my trained model  
  
tensor(0.1779, device='cuda:0', grad_fn=<MeanBackward0>)
```

5. Comparing the prehooked input of the next layer with psum\_recovered.

```

r=nn.ReLU()
next_layer_in_computed = r(output_recovered)
next_layer_in_ref = save_output.outputs[9][0]
difference = abs( next_layer_in_computed - next_layer_in_ref)
print(difference.mean())

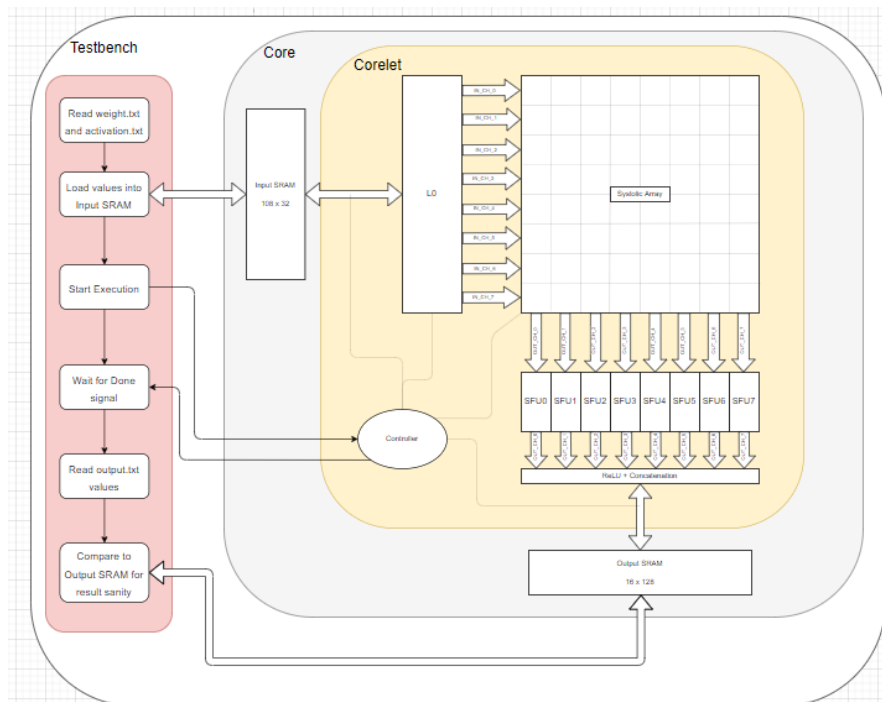
tensor(1.3314e-07, device='cuda:0', grad_fn=<MeanBackward0>)

```

The first part of the project involved updating the VGG model to make sure it completely fits in the Systolic Array that we have designed. Retraining was performed to achieve an accuracy closer to the original one. After that the inputs of the chosen layer were grabbed, processed, and compared to the input of the next layer to check if we are indeed performing what we intended. The activations, weights, and psum of the layer were then dumped in a file to check the functionality of our hardware.

## Part2. Complete RTL core design connecting following blocks

1. Some modifications on the vanilla version were performed by us because of which a block diagram of our proposal has been given below. We got rid of the output FIFO by using internal registers within the SFU to store the intermediate results (we are doing an in place accumulation of the psums generated by systolic array). It sends the outputs directly to the Output SRAM post completion of the execution. The integration of the blocks was done as per the block diagram.



2. The files inside the filelist and the compilation output is given below. SFU is implemented inside the Corelet.

```
../src/fifo_mux_16_1.v
../src/fifo_mux_2_1.v
../src/fifo_mux_8_1.v
../src/fifo_depth64.v
../src/mac.v
../src/mac_tile.v
../src/mac_row.v
../src/mac_array.v
../src/l0.v
../src/corelet.sv
../src/sram.v
../src/core.v
../core_tb.sv
[pgangwar@ieng6-ece-03]:sim:672$ iveri filelist
[pgangwar@ieng6-ece-03]:sim:673$ █
```

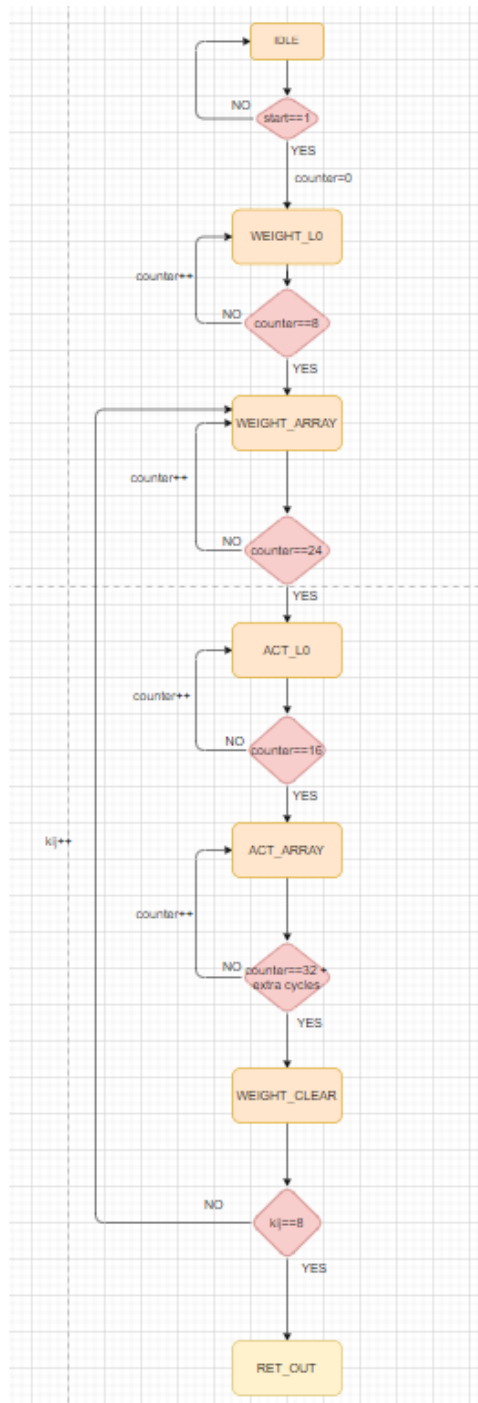
This part was about integrating the whole system by defining the interfaces and designing the controller.

### **Part3. Test bench generation to run following stages**

We created a controller logic which is sitting inside corelet to handle the entire execution end to end.

Hence, we did not use the testbench provided by the professor. (we created a simple custom testbench for validating the outputs with txt files)

Controller:



1. L1 scratchpad loading for weight and activation (e.g., from DRAM, which is emulated by your testbench) --> state=WEIGHT\_L0

The memory interface to I-SRAM is activated with the memory address calculated by the controller logic. The addressing acquires the appropriate weight address from I-SRAM depending on the present value of  $kij$ . Eight weights corresponding to the output channels are read from SRAM and loaded into the L0.

2. Kernel data loading to PE register (via L0) : --> state=WEIGHT\_ARRAY

The weights are transferred in a staggered fashion into the PE. Both the L0 and mac\_array logic are designed to take care of the staggering internally. Controller gives both the “rd” signal to L0 and sets the “inst\_w signal=’b01” for 8 clock cycles.

3. L0 data loading --> state=ACT\_L0

Once the weight is transferred to the mac\_array, we will read the appropriate activations from I-SRAM. The activations are placed after the weights (from address 72 onwards) in the SRAM. The controller selects the appropriate addressing mode between activation loading and weight loading based on the current state. Due to our special design, the controller sieves and acquires only 16 nij values corresponding to the current kij and loads them into the L0 block.

4. Execution with PEs --> state=ACT\_ARRAY

The controller sets the “inst\_w=’b10” for 16 cycles to the mac\_array. At the same time, it asserts “rd=’b1” for the L0. Due to this, the data being read from L0 will be transferred to the mac\_array with proper alignment with the instructions. The staggering of data and instruction is taken care of by mac\_array and L0 internally. After 32 clock cycles, the execution completes.

5. psum movement to L1 scratchpad (via OFIFO) -> N/A

We are not storing the psums in the SRAM, nor are we using the OFIFO. We designed SFU with register-based memory to sum up the psums in place immediately after they are generated.

6. a) Accumulation in SFU and store back to psum SRAM --> state=ACT\_ARRAY

We have not created an OFIFO. Instead, we are storing the data in local registers present in the SFU. Since the data we acquire from mac\_array is aligned, we can easily sum it up locally using an index pointer. After this, we move into WEIGHT\_CLEAR state.

b) Clearing the Weights stored in PE --> state=WEIGHT\_CLEAR

We pass a reset into the mac\_array block that will make load\_ready\_q of each mac\_tile to be 1. Essentially, we are making the PE ready to accept the next set of weights corresponding to the next kij. Then we go to ACT\_L0 state to load new kij, if any left. Otherwise, we go into the next state to transfer the outputs to memory.

7. ReLU in SFU and store back to psum SRAM --> state=RET\_OUT

If all the kij values from 0 to 8 have been covered then we know the data present in the SFU registers is the final post convolution output. Before sending data to the O-SRAM, we apply the ReLU function for each output channel using a comparator on the MSB. The addressing and CEN/WEN signals are taken care of by the controller. Since there are 16 outputs in our

case, the controller will only send 16 write cycles. We also pass a signal called done to let the testbench know that the execution is over, and the data has been stored in O-SRAM.

This part of the project was run with the next part that involves thorough verification of the whole system. We found the part of integrating everything together challenging and also a learning experience. We found a lot of bugs when we connected everything together in one shot so we went back and reconnected everything one at a time which made things work.

## Part4. Verification

All the three files, “activation.txt”, “weight.txt”, and “output.txt” were generated during the first part of the project and read in the testbench. After running the testbench, we saw that the systolic array output matches the one from our python file.

```
[pgangwar@ieng6-ece-03]:sim:683$ iveri filelist
[pgangwar@ieng6-ece-03]:sim:684$ irun
VCD info: dumpfile core_tb.vcd opened for output.
Checking the weights written into the I-SRAM from weight.txt
0-th read data is b99a999c --- Data matched
1-th read data is 1c307f7d --- Data matched
2-th read data is 992a0497 --- Data matched
3-th read data is 27bb9d46 --- Data matched
4-th read data is 3c92a9f7 --- Data matched
5-th read data is 7d091e4c --- Data matched
6-th read data is 75dcee69 --- Data matched
7-th read data is 179e79aa --- Data matched
8-th read data is d9999c9 --- Data matched
9-th read data is 1342c774 --- Data matched
10-th read data is ef5da1df --- Data matched
11-th read data is 31dec6a4 --- Data matched
12-th read data is 7dc29c75 --- Data matched
13-th read data is 3b71fa46 --- Data matched
14-th read data is 7cdfae79 --- Data matched
15-th read data is 0800710d --- Data matched
16-th read data is 9999cb99 --- Data matched
17-th read data is 374095be --- Data matched
18-th read data is 44a5a9e5 --- Data matched
19-th read data is 4acca591 --- Data matched
20-th read data is 3d67913d --- Data matched
21-th read data is 094adde7 --- Data matched
22-th read data is 7b11e979 --- Data matched
23-th read data is 29cfcd21c --- Data matched
24-th read data is 9999bdc4 --- Data matched
25-th read data is 1ca77b23 --- Data matched
26-th read data is 1d5101b4 --- Data matched
27-th read data is f750c54d --- Data matched
28-th read data is 0fcf1077 --- Data matched
29-th read data is 4d4e309b --- Data matched
30-th read data is a703d551 --- Data matched
31-th read data is d7f3e52d --- Data matched
32-th read data is 9c99d999 --- Data matched
33-th read data is 73e7211f --- Data matched
34-th read data is 737143e6 --- Data matched
35-th read data is 277617f7 --- Data matched
36-th read data is 400b6d73 --- Data matched
37-th read data is d077e092 --- Data matched
38-th read data is 2401d04f --- Data matched
39-th read data is 0300760d --- Data matched
40-th read data is aa999a9a --- Data matched
41-th read data is 7097d5ce --- Data matched
42-th read data is 7507ecc7 --- Data matched
43-th read data is 9c949417 --- Data matched
44-th read data is 3f7a773f --- Data matched
45-th read data is 9977c056 --- Data matched
46-th read data is 1e70e979 --- Data matched
47-th read data is fec446f0 --- Data matched
48-th read data is 98bb99dc --- Data matched
49-th read data is 9bc76bde --- Data matched
50-th read data is 2374f0bb --- Data matched
51-th read data is f790990d --- Data matched
52-th read data is 1eddd0f7 --- Data matched
53-th read data is 3ad6979b --- Data matched
54-th read data is e61da775 --- Data matched
55-th read data is 792df7c9 --- Data matched
56-th read data is 9999d999 --- Data matched
57-th read data is ca356be1 --- Data matched
58-th read data is 73795e10 --- Data matched
59-th read data is b7047f23 --- Data matched
60-th read data is 20cc5735 --- Data matched
61-th read data is 7f2707cf --- Data matched
62-th read data is 28cc5735 --- Data matched
63-th read data is 7f2707cf --- Data matched
64-th read data is ad40b640 --- Data matched
65-th read data is 70705702 --- Data matched
66-th read data is b9999999 --- Data matched
67-th read data is 3906ba3d --- Data matched
68-th read data is 7309abd0 --- Data matched
69-th read data is b4241ad6 --- Data matched
70-th read data is ee1d72d2 --- Data matched
71-th read data is f234d77d --- Data matched
72-th read data is b0409f6e --- Data matched
73-th read data is 093c5776 --- Data matched
Checking the activations written into the I-SRAM from activation.txt
0-th read data is 00000000 --- Data matched
1-th read data is 00000000 --- Data matched
2-th read data is 00000000 --- Data matched
3-th read data is 00000000 --- Data matched
4-th read data is 00000000 --- Data matched
5-th read data is 00000000 --- Data matched
6-th read data is 00000000 --- Data matched
7-th read data is 02330000 --- Data matched
8-th read data is 32260041 --- Data matched
9-th read data is 31350132 --- Data matched
10-th read data is 10120040 --- Data matched
11-th read data is 00000000 --- Data matched
12-th read data is 00000000 --- Data matched
13-th read data is 03020020 --- Data matched
14-th read data is 15030071 --- Data matched
15-th read data is 34520252 --- Data matched
16-th read data is 02200251 --- Data matched
17-th read data is 00000000 --- Data matched
18-th read data is 00000000 --- Data matched
19-th read data is 00000000 --- Data matched
20-th read data is 21000050 --- Data matched
21-th read data is 30520140 --- Data matched
22-th read data is 00110140 --- Data matched
23-th read data is 00000000 --- Data matched
24-th read data is 00000000 --- Data matched
25-th read data is 02120011 --- Data matched
26-th read data is 02020040 --- Data matched
27-th read data is 01530052 --- Data matched
28-th read data is 01220050 --- Data matched
29-th read data is 00000000 --- Data matched
30-th read data is 00000000 --- Data matched
31-th read data is 00000000 --- Data matched
32-th read data is 00000000 --- Data matched
33-th read data is 00000000 --- Data matched
34-th read data is 00000000 --- Data matched
35-th read data is 00000000 --- Data matched
Comparing the outputs written into the O-SRAM by the Systolic Array with the output.txt
0-th read data is 0000000009f0000005a0a47002c0000 --- Data matched
1-th read data is 000000ac0079000000b5007d00150000 --- Data matched
2-th read data is 000e00e00030001c00ab0050000a0000 --- Data matched
3-th read data is 0000009c000000300030004500000000 --- Data matched
4-th read data is 00000025f001a0250000000000300000 --- Data matched
5-th read data is 001600c001700c700000012004c0000 --- Data matched
6-th read data is 00ef0136001d00b0000600000000520000 --- Data matched
7-th read data is 000f0a90000000400390000000000000 --- Data matched
8-th read data is 0000007000000001e0000000000020000 --- Data matched
9-th read data is 000000fc001800a000000000000470000 --- Data matched
10-th read data is 000001150049004c000400000000f0000 --- Data matched
11-th read data is 0000009a00000000000039000000000000 --- Data matched
12-th read data is 0000003100130010000001000040000 --- Data matched
13-th read data is 000000c004000a500000000000000000 --- Data matched
14-th read data is 000000d6000b000760042000000500000 --- Data matched
15-th read data is 0000006f000000270023000000000000 --- Data matched
[pgangwar@ieng6-ece-03]:sim:685$
```

This part helped us verify the functionality of our Systolic Array by comparing it with the software's output.

## Part5. Mapping on FPGA

The output of various required tasks is shown below.

Tasks				
Compilation				
	Task	Time		
	▼ Compile Design			
✓	▶▶ Analysis & Synthesis	00:00:43		
✓	▶▶ Fitter (Place & Route)	00:02:00		
✓	▶▶ Assembler (Generate programming files)	00:00:09		
✓	▶▶ Timing Analysis	00:00:12		
	▶▶ EDA Netlist Writer			
	■ Edit Settings			

Slow 1200mV 100C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	123.85 MHz	123.85 MHz	clk	

Power Analyzer Status	Successful - Sat Nov 27 00:02:02 2021
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	ECE284-Final_Project
Top-level Entity Name	corelet
Family	Cyclone IV GX
Device	EP4CGX150DF31I7AD
Power Models	Final
Total Thermal Power Dissipation	256.48 mW
Core Dynamic Thermal Power Dissipation	16.94 mW
Core Static Thermal Power Dissipation	119.24 mW
I/O Thermal Power Dissipation	120.30 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

In this part of the project, we mapped the Corelet module that was designed to be synthesizable on the FPGA. The Corelet module did not have any memory instantiated within it and was just composed of logical blocks to allow it to be mapped to the FPGA. After running Place and Route on our design, we measured its power and maximum operating frequency in the slow corner. The key takeaway from this part was learning how to map the design on FPGA and measuring key design parameters, like frequency and power dissipation.

## Part6. Alpha

We implemented a few design optimizations over the baseline vanilla version.

1. A Mealy-FSM based controller inside the Corelet was designed that took care of
  - a. the scheduling of the appropriate operations in L0 and systolic array
  - b. memory address generation and interfacing with input and output RAM
  - c. synchronization among different modules for the proper functioning of the design.
2. Designed a Sieving logic in the controller that only read the useful activation data from the I-SRAM. That means the only the psums that were actually utilized for calculating the final output corresponding to each kij we're computed. It helped by
  - a. removing the unnecessary computations performed by the Systolic Array
  - b. Since there are lesser partial sums, we saved the memory required for storing them
  - c. reducing the energy consumption and latency as there are lesser memory and systolic array operations

3. Removing the unnecessary computation also simplified the addressing required for storing the partial sums and allowed for a parallelized structure for the output computation. Due to the index alignment in the psums provided after input Sieving, we were able to perform in-place accumulation of psums corresponding to different kij values in the SFU with just one copy of register array corresponding to the total number of outputs. Through this
  - a. we saved the multiple write and read operations to the SRAM
  - b. Saved on the overhead of storing each psum separately in the memory for every kij. (Memory requirement became less by a factor of 9)
  - c. Reduced latency in calculating final output. The final outputs are generated within 1 clock cycle after systolic array completes execution.