# ECE 284
# Project Report

**Fall 2024**


***k-furthest-neighbors***


| | |
|---|---|
| Anushka Chaudary | **A69029714** |
| Arjit Verma | **A69030364** |
| Chainika Shah | **A69030115** |
| Hariram Ramakrishnan | **A69030260** |
| Shail Vaidya | **A69030307** |

# 1. Introduction

This project aims to implement a reconfigurable AI accelerator utilizing 2D systolic array architecture, enhanced with features such as sparsity-aware clock gating and Max Pooling, and optimized for compute time. The design is validated using a VGGNet model trained with quantization-aware training with the CIFAR-10 dataset.

# 2. VGG16 quantization-aware training

The **27th layer of the VGG16 mode**l was modified to have 8 input and 8 output channels in its convolutional layer, aligning it with the requirements for mapping onto an 8x8 2D systolic array. Additionally, we removed the batch normalization layer that typically follows the convolution, simplifying the structure to just "conv -> relu." By implementing these changes and training the model, we successfully achieved an accuracy of **92%.**

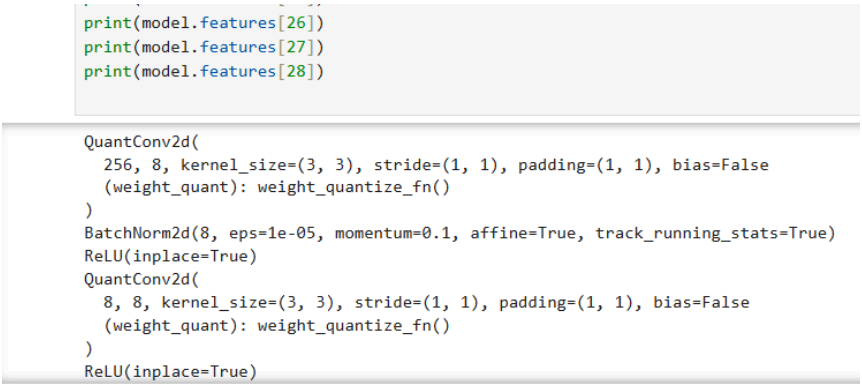


*Fig 1: VGG16 network with modified layers*

# 3. “Vanilla” Version

## 3.1. RTL Design

In addition to the mac_array, L0 and OFIFO, we developed the RTL code for the SFU module to perform the accumulation and ReLU operations, and instantiated all the required modules at the corelet hierarchy. The corelet was then instantiated in the core hierarchy, along with the **pmem** SRAM, of size 512 words x 128 bits, to store the partial sums, and the **xmem** SRAM, of size 256 words x 32 bits, to store the input weight and activations. The FIFO depth for the L0 and OFIFO were also reduced to 16 to save area.

## 3.2. Testbench

The testbench, acting as the control unit, performs the tasks of loading weight and activation data from the TXT files into the SRAMs, and then loading the L0 with first the weights, followed by the activations. Consecutive loops were used as much as possible to maximize efficiency.

Once the computations are complete, the testbench loads the output partial sums to the OFIFO, and then to the PSUM memory. It also implements the cherry-picking algorithm in order to use the SFU and calculates the final output feature value and compares it to the expected output, delivering appropriate messages for the comparison. Here, in addition to the final output, the PSUMs are also compared to the expected PSUM, with results displayed accordingly.

### 3.3. FPGA Mapping

Table 1 shows the results of the FPGA mapping. We are able to see the area benefit gained through FIFO depth reduction, resulting in only 4132 registers needed for the entire 'Vanilla' version of the design.

## 4. Reconfigurable Version

### 4.1. Reconfigurable PE & Instruction Map

The PE was enhanced to enable output stationary mode computation. This required the addition of an extra instruction bit, with the new instruction mapping shown in Table 2. The MSB indicates if the present computation refers to weight or output stationary mode mapping, and the PE is ready to receive weight and activation inputs and modify the calculation to use the internal registers accordingly.

| **Cyclone IV FPGA Mapping of Vanilla Version** | |
|---|---|
| Fmax | 122.2 MHz |
| Switching Activity | 20% |
| PVT | ss-1.2V-100C |
| Thermal Dynamic Power | 18.76mW |
| Total Logic Elements | 6886 |
| Total Registers | 4132 |
| Total Operations | 8x8x2 = 128 |
| GOPs/s | 16 |
| TOPs/W | 0.834 |

*Table 1: FPGA Mapping Results*

| **instr[2:0]** | **Description** |
|---|---|
| 000 | IDLE |
| 001 | W_LOAD |
| 010 | W_EXEC |
| 011 | NOT_USED |
| 100 | RESET |
| 101 | O_SHIFT |
| 110 | O_EXEC |
| 111 | IDLE |

*Table 2: Reconfigurable PE Instruction Map*

## 4.2. Design & Testbench Changes

Output stationary mode required the use of IFIFO in order to provide the weight inputs to the PE array in a pipelined, staggered fashion. The output ports of the IFIFO are connected to the in_n ports in the PE, with the unused bits ignored in the computation. The testbench handles the transfer of data from SRAM xmem to the IFIFO. The SRAM is made to be dual-port in order to read the weights and activations simultaneously, to populate the IFIFO and L0 in parallel. Once loaded, the computation is enabled, and the output pixel values are sent to the SFU, which performs only the ReLU operation in this mode. The output from the SFU is compared against the reference outputs, and the appropriate message is displayed.

# 5. Alphas - Design & Verification Improvements

## 5.1. Compute Time Optimization

Our design implements three enhancements(alphas) that are focused towards overall cycle count reduction in order to achieve complete pipelining. This completely utilizes the 2-D systolic array without any IDLE cycles during both mapping modes - weight and output stationary. This is accomplished through the use of synchronous pipelined  soft reset mode, pipelined output shift, and testbench enhancement.

### 5.1.1 Synchronous Soft Reset

The soft reset instruction is provided to the PE to reset the weights and load the next weight (from the next *kij*) for next computation. This ensures that the PE is active in each cycle, allowing for completely pipelined operation. This eliminates the need for a hard reset of the entire array in order to load the next set of weights. This signal can be seen propagating down the array in Fig. 2, while Fig. 3 depicts the overall picture of the reset signal propagating across the array, pipelined between the execute and load instructions.

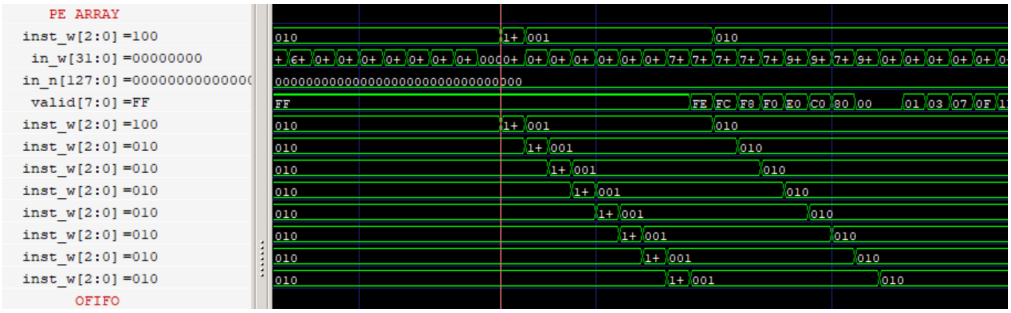


*Fig 2: Execution, Soft Reset & Kernel load Instruction propagates down the PEs*

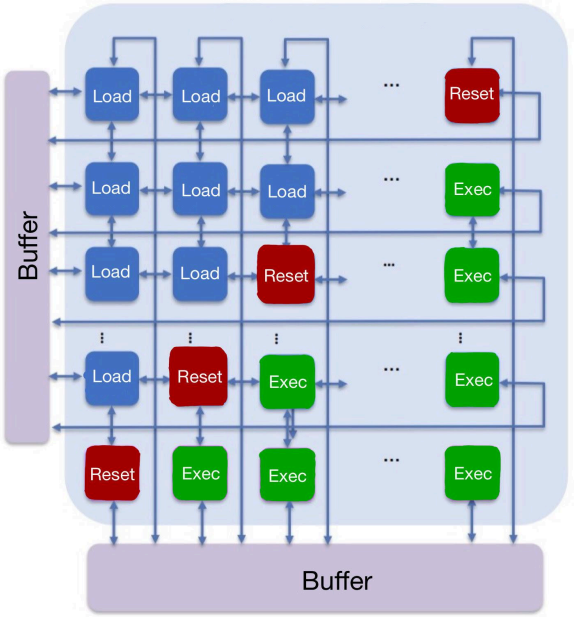


*Fig 3: Execution, Soft Reset & Kernel loading pipelined across PEs*

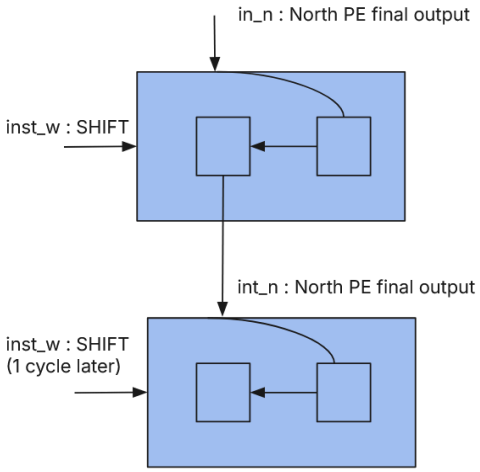


*Fig 4: SHIFT instruction: Input stored in current PE; Output sent to South PE*

### 5.1.2 Pipelined Output Shifting

To shift out the calculated output pixel values in the most efficient manner, each PE is designed to receive the SHIFT instruction as soon as the computation is completed, independent of the state of the neighboring PEs. As shown in Fig. 4, the output pixel value from the north is captured, while the current output pixel is sent south. This means the PE can begin shifting in the immediate next cycle after computation, without any IDLE cycle.

### 5.1.3. Testbench Optimization

The testbench functions as the control unit in our project, and we have designed it to sequence instructions with no idle cycles, making full use of the pipelining implemented in the PEs. This leads to an overall cycle count of **586 cycles** in weight stationary and **47 cycles** in output stationary per output feature calculation.

## 5.2. Sparsity-Aware Clock Gating

The design has sparsity aware-clock gating to save dynamic power. This is achieved inside the PEs which accept a weight-zero and activation-zero flag, preventing the switching of the partial sum registers if either of these flags are high. The PE rows were enhanced to dynamically

disable the computation if the incoming weight or activation is zero, which prevents unnecessary toggling and dynamic power consumption.

## 5.3. Max Pooling Layer Implementation

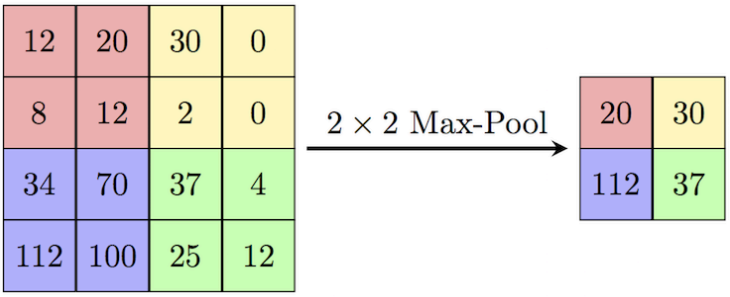


*Fig 5: 2x2 MaxPool Operation*

The SFU is capable of implementing a MaxPool2d layer. When *max_pool_en* signal is asserted, the SFU computes the result of the MaxPool operation along with ReLU, with any kernel size and stride. This is verified for the MaxPool2d(2,2) present in VGG16, by enabling the MaxPool mode. The max pooling operation happens over 4 cycles of cherry-picked input data. The result is provided by the SFU as soon as the *max_pool_en* is pulled low.

# 6. Conclusion

We have presented all the details pertaining to the developed 2-D Systolic Array based AI accelerator design. The project completes all the requirements for the “vanilla” version, enabling weight-stationary mode computation and verification, as well as the reconfigurable version. This enables output-stationary mode computation which has also been verified with weight and activation data from VGG16 neural network. In addition, we have presented enhancements that are twofold - leading to overall compute time reduction through reduced cycle count, and additional features to help save power and implement other layers used in neural networks.