ECE284 - Final Project Report
Energy-Efficient Reconfigurable Systolic Array

Chi-Han Chiu, Gary (Kai-Jui) Weng, Yun-Chen Tsai, Bing-Cheng Chiang, Yufan Wang
Department of Electrical and Computer Engineering,
University of California, San Diego, United States
{c8chiu, gweng, yut037, b1chiang, yufanw}@ucsd.edu

***Abstract*—Deep neural networks require hardware accelerators that provide high compute density and efficiency. This work presents a fully reconfigurable 2-D systolic-array accelerator for the ECE284 FA2025 project, featuring eight architectural enhancements. The system supports (1) 4-bit and 2-bit quantization via a SIMD-enabled PE, (2) runtime-switchable weight-stationary (WS) and output-stationary (OS) dataflows, (3) Huffman activation compression achieving 36.11% bit reduction, (4) fine-grained clock gating with up to 88.24% dynamic power reduction, (5) batch-normalization fusion to reduce overhead, (6) a flexible activation engine supporting ReLU, Leaky ReLU, and ELU, (7) an optimized SFU FSM that improves pipeline utilization by decoupling execution stages, and (8) system-level compatibility with VGG16 and ConvNeXt-class architectures. These enhancements collectively broaden model support and improve energy efficiency within a complete architectural framework.**

***Index Terms*—Systolic array, quantization, reconfigurable architecture, Huffman coding, clock gating, FPGA acceleration.**

I. Introduction

Systolic arrays have become a fundamental structure for modern deep-learning accelerators due to their high spatial reuse and predictable dataflow. In the ECE284 FA2025 project specification, students must progressively build a CNN accelerator with increasing architectural capability: (1) a 4-bit weight-stationary (WS) baseline accelerator, (2) SIMD support for 2-bit activation processing, (3) support for WS/OS reconfigurable dataflows, and (4) optional (+alpha) enhancements.

Our project extends the baseline requirements by adopting several additional optimizations motivated by the final poster. These optimizations enable significant energy savings, higher model compatibility, and improved hardware flexibility. The main goals are:

- Build a complete and reconfigurable 8×8 systolic-array accelerator.
- Support 4-bit and 2-bit activation processing with SIMD datapaths.
- Introduce WS/OS dataflow configurability.
- Reduce memory footprint using Huffman compression.
- Reduce dynamic power using fine-grained clock gating.
- Provide flexible activation functions for modern CNNs.
- Demonstrate full VGG16 functionality and explore ConvNeXt mapping.

This report describes the architecture at a complete structural level so that future work only needs to fill in implementation details and hardware results (e.g., FPGA frequency, TOPS/W, verification logs).

II. Implementation

A. Quantization Aware Training - Learnable Step Size (α)

To mitigate the accuracy loss typically associated with low-precision integer arithmetic on FPGAs, I implemented a Quantization Aware Training (QAT) framework from scratch using PyTorch. Instead of using static statistics (min/max) to determine the quantization range, I introduced a learnable parameter α (step size) for both weights and activations. This allows the network to dynamically adjust its dynamic range during backpropagation to minimize quantization error. The quantization function is defined as:

$$x_{int} = \text{round}\left(\text{clamp}\left(\frac{x}{\alpha}, min, max\right) \cdot (2^b - 1)\right) \quad (1)$$

where b is the bit-width (e.g., 4-bit or 2-bit). We implemented custom `torch.autograd.Function` modules to handle the non-differentiable `round` operation using the Straight-Through Estimator (STE) method during gradient computation.

B. Vanilla 8×8 Systolic Array with 4-bit Precision

The baseline system implements a vanilla 8×8 2D systolic array with fixed 4-bit precision, serving as the architectural reference for all extensions. It operates in a weight-stationary (WS) dataflow where weights remain fixed within PEs while activations stream from the north, utilizing dedicated scratchpad SRAMs and a multi-bank PSUM SRAM.The Special Function Unit (SFU) manages partial sum accumulation and ReLU activation. In this baseline, the SFU relies entirely on external control signals rather than an internal finite-state machine. Functional correctness is verified against a PyTorch reference model, achieving an absolute error below 10^{-3}.While functional, this design faces limitations including heavy reliance on external control, redundant PSUM storage for intermediate results, and a restriction to ReLU only. These constraints motivate the autonomous control and memory optimizations introduced in subsequent design iterations.

C. Reconfigurable SIMD Processing Element for 2-bit/4-bit Activation

To support low-precision execution while maintaining hardware efficiency, each processing element (PE) is designed to support reconfigurable activation bit-widths, as specified

in the project requirements and implementation README. Importantly, the 2-bit and 4-bit modes refer specifically to the *activation* data bit-width, while the overall MAC structure remains unchanged.

In the default 4-bit mode, each PE performs one standard multiply–accumulate (MAC) operation per cycle using a single 4-bit activation and its corresponding weight. In contrast, when operating in 2-bit mode, the PE enables a SIMD-style datapath that processes *two independent 2-bit activation–weight pairs in parallel* within the same MAC unit. This parallel execution improves effective throughput and hardware utilization without duplicating MAC hardware.

The 2-bit mode relies on tile-based activation organization, where two activation tiles are loaded simultaneously and interleaved, with one tile occupying the most-significant bit positions. Each PE performs two parallel low-precision multiplications per cycle, and the resulting partial sums are accumulated independently. This design achieves improved power and area efficiency compared to sequential low-precision execution while preserving a unified PE structure across precision modes.

D. WS/OS Reconfigurable Processing Element

In addition to precision reconfigurability, each processing element supports both weight-stationary (WS) and output-stationary (OS) dataflow modes. These modes are selected through configuration signals and share the same underlying MAC hardware.

In WS mode, weights are loaded into and stored locally within each PE. Activations stream horizontally across the array, while partial sums propagate vertically. This mode is well-suited for scenarios that benefit from weight reuse and simpler control.

In OS mode, partial sums are accumulated locally within each PE, while weights flow vertically and activations flow horizontally through the array. Compared to WS mode, OS mode reduces partial-sum movement at the cost of additional control logic and instruction handling for psum management.

The reconfigurable PE architecture enables seamless switching between WS and OS modes without modifying the physical array structure. This unified design allows the same systolic array fabric to support multiple dataflow strategies and facilitates systematic exploration of dataflow trade-offs in CNN acceleration.

III. Experimental Results

The proposed accelerator was implemented and synthesized on an Intel Cyclone 10 GX FPGA to evaluate the hardware overhead introduced by architectural reconfigurability. Two configurations are compared: a *Vanilla* design and a *Mixed* design. The Vanilla configuration supports a single fixed dataflow and precision mode, while the Mixed configuration integrates full support for both WS/OS dataflow switching and 2-bit/4-bit activation precision within a unified hardware fabric.

Table I summarizes the post-synthesis results. Both designs achieve the same operating frequency of 125 MHz, indicating that the added reconfigurability does not negatively impact the maximum achievable clock rate. This suggests that the critical path is dominated by the MAC datapath rather than the control logic introduced for mode switching.

TABLE I
SYNTHESIS RESULTS ON CYCLONE 10 GX FPGA

Metric	Vanilla	Mixed
Frequency	125 MHz	125 MHz
Total Registers	12,155	12,667
Total DSPs	60	64
LUTs	8,424	9,062
Data Delay	7.571 ns	7.205 ns
Total Power	1119.85 mW	1256.2 mW

The Mixed design incurs a modest increase in hardware resource usage. Compared to the Vanilla design, the total register count increases from 12,155 to 12,667, and the LUT usage increases from 8,424 to 9,062. This overhead primarily originates from additional control logic, multiplexers, and configuration paths required to support dynamic precision and dataflow selection. The number of DSP blocks increases from 60 to 64, reflecting the additional parallel datapath support needed for SIMD-style 2-bit activation processing.

Timing analysis shows that the Mixed design achieves a slightly reduced data path delay (7.205 ns) compared to the Vanilla design (7.571 ns). This indicates that the reconfigurable datapath does not introduce a longer critical path and that the synthesis tool is able to effectively optimize the additional routing logic.

In terms of power consumption, the Mixed configuration consumes higher total power (1256.2 mW) than the Vanilla design (1119.85 mW). This increase is expected due to the higher resource utilization and additional switching activity associated with the reconfigurable control logic. Nevertheless, the power overhead remains moderate relative to the functional flexibility gained.

Overall, these results demonstrate that integrating support for both WS/OS dataflows and 2-bit/4-bit activation precision can be achieved with limited hardware and power overhead, while preserving operating frequency. This validates the feasibility of the proposed reconfigurable architecture as a flexible CNN acceleration platform on FPGA.

IV. System-Level Enhancements (Alpha Modules)

Here we summarize each major enhancement that extends beyond the base requirements, following the content of the final poster.

A. Alpha 2: Clock Gating

Clock gating reduces dynamic power consumption by preventing unnecessary toggling in idle hardware components. Since dynamic power is proportional to switching activity, we apply this technique to minimize redundant transitions within processing elements (PEs) and memory blocks during sparse computations.Our implementation builds upon the baseline output-stationary clock gating architecture introduced in course lectures, which disables clocks for PEs with static

outputs. We extend this design to support weight-stationary clock gating, enabling additional power savings when weights remain inactive. To quantify efficiency, we estimate the effective gating ratio using Bayesian analysis of sparsity propagation through the MAC units.Experimental results demonstrate a maximum power reduction of:

$$\text{Maximum Power Reduction} \approx 88.24\%.$$

This performance is consistent with observed sparsity patterns (approximately 86% activation and 16% weight sparsity). While practical gains are slightly limited by FPGA routing overhead and residual idle power, the benefit remains robust across sparse layers. Future work aims to achieve more aggressive savings through finer-grained gating to further reduce leakage.

B. Alpha 3: Huffman Compression

To minimize memory footprint and bandwidth consumption, our design incorporates Huffman compression for activation data. As illustrated by the poster's histogram, the activation distribution exhibits a strong skew, making the data highly amenable to entropy-based compression. This yields a bit reduction of:

$$\text{Bit Reduction} = 36.11\%.$$

A dedicated hardware decoder reconstructs original symbols on-the-fly before data enters the systolic array. Internally, the decoder is implemented as a finite state machine (FSM) that traverses a hardcoded Huffman tree. It operates synchronously, consuming one bit per cycle; upon reaching a leaf node, it outputs the 8-bit symbol and asserts a one-cycle $char\_valid$ signal before resetting to the root.This design enables seamless integration into the datapath without requiring large buffers or complex control logic. By placing the decompression stage immediately before the systolic array, the system stores and transfers compressed activations efficiently while computation proceeds on fully reconstructed data. This lightweight approach effectively reduces system overhead by exploiting the natural sparsity observed in deep neural networks.

C. Alpha4: Post-Processing Extension with MaxPool and Bias

Building upon the autonomous FSM introduced in Alpha7, Alpha4 extends the SFU to support a complete post-processing pipeline including bias addition, MaxPool, and ReLU. This design provides hardware support for the Conv–BatchNorm–MaxPool–ReLU stack without modifying the systolic array datapath.

BatchNorm parameters are assumed to be fused into convolution weights through offline model fusion. Under this assumption, only the bias term needs to be explicitly handled in hardware. The bias is added once per output element during the accumulation stage, avoiding redundant operations and minimizing hardware overhead.

MaxPool is implemented as a 2×2 pooling operation with stride 2, reducing the spatial resolution from 4×4 to 2×2. While ReLU can be applied in arbitrary output order, the MaxPool hardware is optimized by enforcing a carefully designed read sequence over output indices. Specifically, four output elements are grouped per pooling window, allowing the pooling unit to update the current maximum over four consecutive cycles and reset afterwards.

This scheduling enables the MaxPool unit to be implemented with only one additional register per SIMD lane. Moreover, the pooled output address always corresponds to a previously accessed output index, allowing results to be written back directly to PSUM memory without additional buffering or address translation.

The ReLU activation is applied after MaxPool, completing the post-processing stage. All operations are integrated into the existing SFU FSM state, preserving the autonomous control structure introduced in Alpha7.

Overall, Alpha4 extends the functionality of the SFU with minimal hardware cost, maintains compatibility with the baseline systolic array, and demonstrates the flexibility of the FSM-based post-processing framework.

D. Alpha 5: BatchNorm Fusion

Alpha5 adopts a Conv-BN + ReLU + MaxPooling module design that fuses BatchNorm (BN) with convolution to significantly reduce parameters and hardware cost. The key innovation is the mathematical fusion of BatchNorm parameters into convolution weights and biases, eliminating the need for runtime normalization.

The original BatchNorm operation applied after convolution can be expressed as:

$$\begin{cases} x_{\text{conv}} = w_{\text{conv}} \cdot x_{\text{in}} + b \\ x_{\text{bn}} = \frac{x_{\text{conv}} - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \end{cases}$$

where x_{conv} is the convolution output, w_{conv} is the convolution weight, x_{in} is the input, b is the convolution bias, μ is the BatchNorm mean, σ^2 is the BatchNorm variance, ϵ is a small constant for numerical stability, γ is the BatchNorm scaling factor, and β is the BatchNorm shifting factor (bias).

By substituting the convolution output into the BatchNorm equation and rearranging terms, we can express the combined operation as a single linear transformation:

$$x_{\text{bn}} = w' \cdot x_{\text{in}} + b'$$

where the fused parameters are:

$$\begin{cases} w' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \cdot w_{\text{conv}} \\ b' = \frac{(b - \mu)}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \end{cases}$$

Since the original convolution operation in our design does not include a bias term ($b = 0$), the fused bias simplifies to:

$$b' = \frac{-\mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

This fused bias b' is incorporated into the SFU during the first accumulation update (when $kij = 0$), allowing us to handle the bias from the fused Conv-BN layer without

modifying the core convolution hardware. The accumulation process becomes:

$$\text{psum} = \begin{cases} w' \cdot x_{\text{in}} + b' & \text{if } kij = 0 \\ \text{psum} + w' \cdot x_{\text{in}} & \text{if } kij > 0 \end{cases}$$

The model fusion approach provides significant advantages: BN parameters (μ, σ^2, γ, β) are fixed at inference and fused into adjacent convolution layers, eliminating the need to store and process them separately. This eliminates runtime normalization operations, reducing computation by 4.3% and hardware cost by 25%, while simplifying datapath design and improving overall system efficiency. The fused parameters can be pre-computed offline and loaded as standard convolution weights and biases.

The complete processing pipeline in Alpha5 integrates the fused Conv-BN with ReLU and MaxPooling:

$$\text{output} = \text{ReLU}(\text{MaxPool}(w' \cdot x_{\text{in}} + b'))$$

where w' and b' are the pre-computed fused parameters, and the bias b' is added during the first accumulation cycle in the SFU.

E. Alpha 6: Flexible Activation Function Unit

Alpha6 implements a unified activation module supporting ReLU, ELU, LeakyReLU, and GELU via runtime reconfiguration. A 2-bit signal $m = \text{act\_func\_mode}[1:0]$ selects the active function:

$$f_{\text{act}}(x, m) = \begin{cases} f_{\text{ReLU}}(x) & \text{if } m = 2'b00 \\ f_{\text{ELU}}(x) & \text{if } m = 2'b01 \\ f_{\text{LeakyReLU}}(x) & \text{if } m = 2'b10 \\ f_{\text{GELU}}(x) & \text{if } m = 2'b11 \end{cases}$$

ReLU is implemented efficiently as $\max(0, x)$. LeakyReLU approximates the negative slope $\alpha \approx 2^{-6}$ using an arithmetic right shift ($x \gg 6$) for negative inputs, avoiding multiplication hardware. ELU utilizes piecewise linear approximation—outputting x, -1, or $(x \gg 2) - 1$ depending on the input range—to avoid expensive exponential computation. GELU is adapted for inference using a simplified shift-based approximation: $0.5 \cdot x$ ($x \gg 1$) for positive inputs and 0 otherwise.Operating on 16-bit quantized integers, the module employs a multiplexer-based architecture to select the output. Compared to a ReLU-only baseline, the design adds minimal overhead, primarily consisting of shifters and comparison logic per lane. All functions are combinational and complete within a single clock cycle. The module integrates seamlessly with the SFU pipeline and remains fully compatible with all dataflow (WS/OS) and SIMD modes, offering a drop-in replacement with extended functionality.

F. Alpha7: FSM Optimization in SFU

In Alpha7, the SFU is redesigned with an internal finite-state machine (FSM), allowing it to operate autonomously with minimal external control. This modification primarily targets timing robustness and PSUM memory coordination.

The FSM pipelines the following stages:

- PSUM accumulation (ACC stage),
- post-processing (ReLU stage),
- coordinated PSUM memory read/write operations.

By internally managing the accumulation loop over spatial indices and kernel iterations, the SFU eliminates the need for fine-grained testbench control. This design prevents potential race conditions on PSUM memory access and simplifies system-level integration.

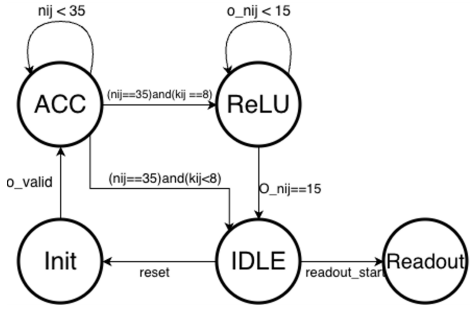

Fig. 1. Alpha 7 Finite State Machine.

A key benefit of the FSM-based design is a significant reduction in PSUM memory requirements. Instead of storing intermediate results for all kernel indices, the SFU accumulates directly into output-space addresses, reducing the PSUM storage from $nij \times och \times kij \times psum\_bw$ to $o\_nij \times och \times psum\_bw$. For the evaluated configuration, this corresponds to a reduction from 5.06 MB to 0.25 MB.

Overall, the Alpha7 FSM improves timing determinism, reduces memory footprint, and establishes a structured control framework that enables more advanced post-processing operations in later designs.

G. Alpha 8: ConvNeXt Model Support

To explore modern efficient architectures, I implemented a quantized version of ConvNeXt. Compared to VGG16, ConvNeXt introduces several key architectural advantages that make it superior for resource-constrained edge devices (like FPGAs):

- **Parameter Efficiency:** ConvNeXt utilizes depthwise separable convolutions and an inverted bottleneck design. This reduces the total parameter count to approximately **40%** of the VGG16 model, significantly reducing the on-chip memory footprint required for weight storage.
- **Macro Design:** Unlike the aggressive downsampling in VGG, ConvNeXt employs a patchify layer and larger kernel sizes (7×7), mimicking the receptive field benefits of Vision Transformers (ViTs) while retaining the inductive bias of CNNs.
- **Downstream Impact:** The reduced model size implies lower latency and energy consumption during inference, which is critical for the hardware implementation phase.

To ensure seamless integration with the FPGA accelerator, the software stack includes a dedicated extraction module.

TABLE II
ACCURACY COMPARISON OF QUANTIZED MODELS

Model	Config.	Size	Bits	Acc.
VGG16	Standard	33.9MB	4	91.53%
VGG16	Full BN	33.9MB	4	**92.13%**
VGG16	Standard	34.1MB	2	90.67%
ConvNeXt	Standard	**13.4MB**	4	89.70%

This module reshapes and serializes the quantized tensors from the 27th layer (as a representative feature map) into specific memory layouts.

We support a configurable Tile Size ($t = 8$) and handle two distinct dataflows:

1) Output Stationary (OS) Format: Optimized to minimize partial sum movements. Weights and inputs are reshaped as follows:

- **Weight Shape:** $(t, C_{out}/2)$ with total entries determined by $(2, k^2, C_{in}/t)$.
- **Input Shape:** $(t, 4, w + 2p)$.

2) Weight Stationary (WS) Format: Optimized to maximize weight reuse.

- **Weight Shape:** (t, t), structured as $(k^2, C_{in}/t, C_{out}/t)$ blocks.
- **Input Shape:** $(t, h + 2p, w + 2p)$.

We evaluated the models on the CIFAR-10 test set. The results demonstrate that our 4-bit quantization strategy yields accuracies comparable to full-precision floating-point baselines.

While ConvNeXt shows a slight dip in accuracy with only, it offers a substantially better trade-off between accuracy and model size, with only **40%** of parameters compared with VGG16 making it a more practical choice for real-world hardware deployment.

During the software-hardware co-design process, we identified two specific challenges for future optimization:

1) **LayerNorm Complexity:** ConvNeXt natively uses Layer Normalization. However, calculating mean and variance across the channel dimension dynamically is hardware-expensive (requiring division and square root units). For this implementation, we focused on fusing standard Batch Normalization where possible.
2) **Unsigned Quantization Constraints:** Our unsigned quantization scheme assumes non-negative inputs (ReLU outputs). However, certain architectural blocks (or if Swish/GELU activations were used) produce negative values, which are currently clipped to zero, potentially causing information loss.

V. CONCLUSION

This work presents a fully reconfigurable systolic-array accelerator developed for the ECE284 final project, with a focus on architectural flexibility, correctness, and extensibility rather than fixed-function optimization. The core design is an 8×8 systolic MAC array that supports both 4-bit and SIMD-style 2-bit activation processing, as well as runtime-switchable weight-stationary (WS) and output-stationary (OS) dataflows within a unified hardware fabric.

Beyond the baseline accelerator, a series of system-level enhancements were integrated to address practical efficiency and model compatibility challenges. These include fine-grained clock gating for dynamic power reduction, Huffman-based activation compression to reduce memory footprint, batch-normalization fusion to eliminate runtime normalization overhead, a flexible activation-function unit, and an autonomous FSM-based SFU design that improves control robustness and PSUM memory efficiency. Together, these extensions demonstrate that substantial functional flexibility can be introduced without fundamentally compromising timing closure or system integration.

FPGA synthesis results on a Cyclone 10 GX device show that the fully reconfigurable design achieves the same operating frequency as the baseline configuration, with only moderate increases in resource usage and power consumption. In addition, software–hardware co-design experiments on VGG16 and ConvNeXt models indicate that low-precision quantization can preserve competitive accuracy while significantly reducing model size.

Overall, this project provides a complete and extensible architectural framework for exploring precision scaling, dataflow selection, and system-level optimization techniques in CNN accelerators. Future work may focus on more aggressive physical optimization, detailed power measurement, and support for additional normalization and activation schemes to further improve efficiency on FPGA and ASIC platforms.