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*—We present a reconfigurable 2-D systolic-array accelerator for ECE284 FA2025 featuring eight key enhancements, including SIMD-based 4/2-bit quantization, switchable dataflows, and fused batch-normalization. The design achieves 36.11**

***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:

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

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 (Part 1) implements a vanilla 2D systolic array accelerator with fixed 4-bit activation and weight precision. This design serves as the architectural reference for all subsequent extensions.

The accelerator consists of an 8×8 MAC array operating in a weight-stationary (WS) dataflow, activation and weight scratchpad SRAMs, a multi-bank PSUM SRAM, and an SFU responsible for accumulation and ReLU activation. In WS dataflow, weights remain fixed within each processing element while activations are streamed from the north, minimizing weight movement and simplifying control.

The SFU collects partial sums from the output FIFO, performs accumulation across kernel positions, applies ReLU, and manages PSUM memory access. In this baseline implementation, the SFU is externally controlled and does not contain an internal finite-state machine.

Functional correctness is verified against a PyTorch-based reference model. All partial sums match the golden outputs with an absolute error below 10^{-3}.

While functionally correct, this vanilla design exposes several limitations. The SFU relies on extensive external control, PSUM memory stores redundant intermediate results across kernel indices, and the post-processing stage is limited to ReLU. These constraints motivate later design iterations that introduce autonomous SFU control, reduced PSUM memory footprint, and extended post-processing support.

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.

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.

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

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 is an effective technique for reducing dynamic power consumption by preventing unnecessary clock toggling in idle hardware components. In theory, a design employing clock gating should consume less power than a design without clock gating, as dynamic power is directly proportional to switching activity. In our work, clock gating is applied to minimize redundant transitions within processing elements (PEs) and memory blocks during sparse computations.

Our initial implementation follows the output-stationary clock gating architecture introduced in the course lecture slides. This approach disables clock signals to PEs whose outputs are not updated, thereby reducing switching activity when the computation results are zero or unchanged. To further improve power efficiency, we extend this design to support weight-stationary clock gating, enabling additional opportunities to gate clocks when weights remain inactive due to sparsity.

To quantify the power reduction, we estimate the effective clock gating ratio using Bayesian analysis on the sparsity propagation through the multiply-accumulate (MAC) operations.

Poster results show:

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

In practice, this value is affected by FPGA routing overhead and the inability to fully shut down all gated components; therefore, the measured power reduction may be slightly lower.

Experimental results from our poster demonstrate a maximum power reduction of approximately 88.24%, consistent with the predicted sparsity level. The effectiveness of clock gating depends on sparsity patterns and hardware implementation details; however, the benefit remains stable across layers exhibiting high activation sparsity (approximately 86%) and moderate weight sparsity (approximately 16%). As future work, we aim to achieve more aggressive power savings by enabling finer-grained clock gating and fully shutting down idle transistors to further reduce leakage and residual switching power.

B. Alpha 3: Huffman Compression

To reduce memory footprint and bandwidth consumption, our design incorporates Huffman compression for activation data, supported by a dedicated hardware Huffman decoder. As shown in the activation histogram on the poster, the activation distribution exhibits strong skew, with a small subset of values occurring with significantly higher frequency. This property makes the data highly amenable to entropy-based compression.

The histogram on the poster shows strong skew in activation distribution, enabling Huffman coding with a bit reduction of:

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

This reduces SRAM footprint and bandwidth pressure.

The decoder processes a serial Huffman-encoded bit stream and reconstructs the original symbols on-the-fly before the data enters the systolic array. Internally, the decoder is realized as a finite state machine (FSM) that traverses a hardcoded Huffman tree, with each input bit guiding a left or right transition. Upon reaching a leaf node, the corresponding 8-bit symbol is produced, and a one-cycle char_valid signal indicates the availability of valid decoded data.

The decoder operates synchronously, consuming one bit per clock cycle, and immediately resets to the root state after each symbol is decoded. This design enables seamless integration into the accelerator datapath without requiring large buffers or complex control logic. By placing the decompression stage before the systolic array, compressed activations can be stored and transferred efficiently, while computation proceeds on fully reconstructed data.

Overall, the combination of Huffman compression and lightweight hardware decoding effectively reduces memory and bandwidth overhead with minimal control complexity, making it well-suited for sparse and skewed activation distributions commonly 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:

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

where $x_{\textbf{conv}}$ is the convolution output, $w_{\textbf{conv}}$ is the convolution weight, $x_{\textbf{in}}$ 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:

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

where the fused parameters are:

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

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:

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

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:

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

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 function module that supports multiple activation functions (ReLU, ELU, LeakyReLU, and GELU) through a single hardware module with runtime reconfiguration capability. The `ActivationFunction` module provides a unified interface that multiplexes between four activation functions based on a 2-bit mode selection signal:

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

where $m = \textbf{act\_func\_mode}[1:0]$ is the mode selection signal.

ReLU is implemented directly in hardware: $f_{\text{ReLU}}(x) = \max(0, x)$, requiring only a sign bit check and a multiplexer. LeakyReLU uses a small negative slope $\alpha = 0.01$ for negative inputs, approximated using bit-shift: $\alpha \approx 1/64 = 2^{-6}$, resulting in leakyrelu_out $= x$ if $x \geq 0$, else $x \gg 6$ for negative values. This approximation uses a simple arithmetic right shift, avoiding multiplication hardware while maintaining reasonable accuracy.

ELU uses an exponential function for negative inputs: $f_{\text{ELU}}(x) = x$ if $x > 0$, else $\alpha \cdot (e^x - 1)$ where $\alpha = 1.0$. Hardware approximation uses piecewise linear segments: $f_{\text{ELU}}(x) \approx x$ if $x \geq 0$, -1 if $x < -4$, or $(x \gg 2) - 1$ if $-4 \leq x < 0$. This piecewise linear approximation avoids expensive exponential computation while providing reasonable accuracy for the typical input range.

GELU is defined as $f_{\text{GELU}}(x) = 0.5 \cdot x \cdot (1 + \tanh(\sqrt{2/\pi} \cdot (x + 0.044715 \cdot x^3)))$. Hardware approximation (simplified): $f_{\text{GELU}}(x) \approx 0.5 \cdot x$ if $x > 0$, else 0. Implementation uses gelu_temp $= x \gg 1$ (right shift by 1 = multiply by 0.5), with gelu_out $=$ gelu_temp if $x \geq 0$, else 0.

The unified activation function module uses a multiplexer-based architecture to select the output based on the mode signal. All activation functions operate on quantized 16-bit signed integers $x \in \{-32768, -32767, \ldots, 32767\}$. The approximations are designed to work within this quantization range: ReLU is exact for all quantized values, LeakyReLU has approximation error $\epsilon < 2^{-6}$ for negative values, ELU uses piecewise linear approximation with error depending on input range, and GELU uses simplified approximation suitable for inference.

Compared to a ReLU-only implementation, the unified module adds: 1 arithmetic right shifter per lane for LeakyReLU, 1 right shifter + 1 subtractor + comparison logic per lane for ELU, 1 right shifter + comparison logic per lane for GELU, and a 4-to-1 multiplexer per lane (2-bit select). Total additional area is $O(col \times psum\_bw)$ for logic, $O(\log_2(4) \times col)$ for multiplexer control. All activation functions are combinational, completing within a single clock cycle and maintaining pipeline throughput. The module integrates seamlessly with the SFU pipeline: SFU_output $= f_{\text{act}}(\text{psum}, \text{act\_func\_mode})$, and is compatible with all dataflow modes (WS and OS), all SIMD modes (2-bit and 4-bit), and all existing SFU FSM states, making it a drop-in replacement for the original ReLU module while providing 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.

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.

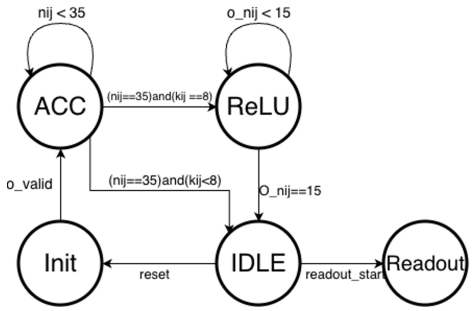

Fig. 1. Alpha 7 Finite State Machine.

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. 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:

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%

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.