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*—This work presents a fully reconfigurable 2-D systolic-array accelerator featuring eight architectural enhancements: (1) 4-bit and 2-bit quantization via SIMD-enabled PEs, (2) runtime-switchable weight-stationary (WS) and output-stationary (OS) dataflows, (3) Huffman activation compression (36.11% bit reduction), (4) fine-grained clock gating (up to 88.24% dynamic power reduction), (5) batch-normalization fusion, (6) flexible activation functions (ReLU, LeakyReLU, ELU, GELU), (7) autonomous SFU FSM for improved pipeline utilization, and (8) VGG16 and ConvNeXt compatibility. These enhancements improve energy efficiency and model support within a unified architectural framework.**

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

I. Introduction

Systolic arrays have emerged as a fundamental architecture for CNN acceleration due to their regular structure, high spatial reuse, and predictable dataflow patterns. However, existing designs often face trade-offs between flexibility and efficiency, particularly when supporting multiple precision modes, dataflow strategies, and modern network architectures.

This work addresses these challenges by presenting a fully reconfigurable 8×8 systolic-array accelerator that integrates multiple architectural enhancements within a unified hardware framework. Our contributions include:

1) A reconfigurable processing element (PE) architecture supporting both 4-bit and SIMD-style 2-bit activation processing, enabling precision scaling without hardware duplication.
2) Runtime-switchable weight-stationary (WS) and output-stationary (OS) dataflow modes, allowing optimization for different workload characteristics.
3) System-level optimizations including Huffman-based activation compression, fine-grained clock gating, and batch-normalization fusion, collectively improving energy efficiency and memory utilization.
4) An autonomous finite-state machine (FSM) based Summation and Function Unit (SFU) that reduces memory requirements and simplifies control complexity.
5) Comprehensive software–hardware co-design demonstrating compatibility with VGG16 and ConvNeXt architectures under low-precision quantization.

The remainder of this paper is organized as follows: Section II describes the baseline architecture and reconfigurable design. Section III presents experimental results. Section IV details system-level enhancements. Section V concludes with a summary and future directions.

II. Architecture and Implementation

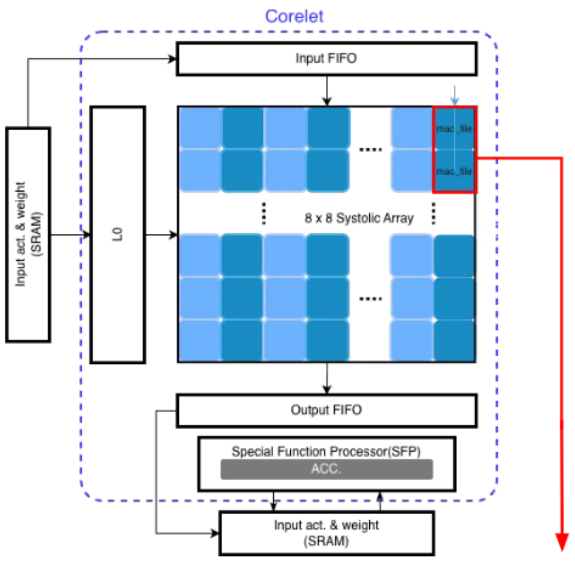
Corelet
Input FIFO
mac_tile
mac_tile
8 x 8 Systolic Array
L0
Input act. & weight (SRAM)
Output FIFO
Special Function Processor(SFP)
ACC.
Input act. & weight (SRAM)

Fig. 1. Hierarchical core architecture showing the testbench (`core_tb.v`), main core (`core.v`), and corelet (`corelet.v`) modules. The diagram illustrates data flow from external memory (XMEM) through L0 FIFO, MAC Array, SFU, and Output FIFO, with bidirectional communication between SFU and Partial Sum Memory (PMEM).

A. Quantization Aware Training

To mitigate accuracy degradation from limited representational range, we implement a Quantization Aware Training (QAT) framework with learnable step size α for both weights and activations, enabling adaptive dynamic range adjustment during backpropagation.

The quantization function maps floating-point values x to b-bit integers:

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

where $b \in \{2, 4\}$ is the target bit-width. The non-differentiable `round` operation is handled using the Straight-Through Estimator (STE), implemented via custom `torch.autograd.Function` modules that pass gradients through unchanged during backpropagation.

B. Baseline Architecture

The baseline system implements a vanilla 8×8 2D systolic array with 4-bit precision, operating in weight-stationary (WS) dataflow mode. As illustrated in Fig. 1 and Fig. 2, the architecture consists of a hierarchical structure: the testbench (`core_tb.v`) instantiates the main core (`core.v`), which contains external memory (XMEM), a corelet processing unit (`corelet.v`), and partial sum memory (PMEM).

In WS mode, weights are pre-loaded and stored locally within each processing element (PE), while activations stream horizontally across the array from north to south. Partial sums propagate vertically from west to east, accumulating results as they traverse the array. The architecture employs dedicated scratchpad SRAMs for weight storage and a multi-bank PSUM SRAM for intermediate results. The Special Function Unit (SFU) manages partial sum accumulation and ReLU activation, controlled entirely by external testbench signals.

Functional correctness is verified against a PyTorch reference model, achieving mean absolute error below 10^{-3} across all test cases. While functional, this baseline design exhibits several limitations: (1) heavy dependence on external control signals, (2) redundant PSUM storage for intermediate kernel iterations, and (3) restriction to ReLU activation only. These constraints motivate the autonomous control and memory optimizations introduced in subsequent design iterations.

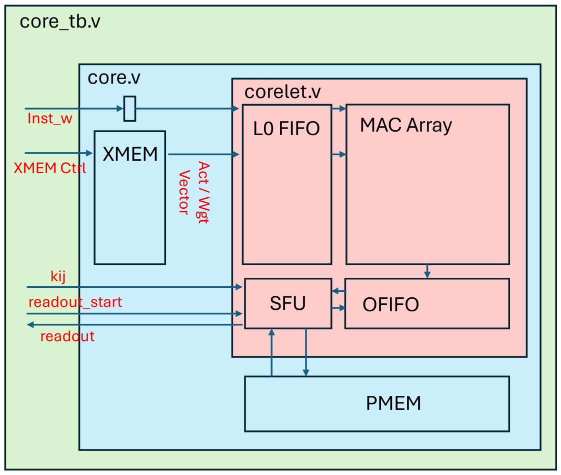

Fig. 2. Corelet architecture block diagram showing the data flow from Input Activation & Weight SRAM through L0 local memory, Input FIFO, 8×8 Systolic Array (composed of `mac_tile` units), Output FIFO, and Special Function Processor (SFP) with Accumulator. The highlighted `mac_tile` column indicates the reconfigurable processing elements supporting WS/OS dataflows and 2-bit/4-bit SIMD modes.

C. Reconfigurable SIMD Processing Element

To support precision scaling while maintaining hardware efficiency, each processing element (PE) is designed with reconfigurable activation bit-widths. Importantly, the 2-bit and 4-bit modes refer specifically to *activation* data bit-width, while weight precision and the overall MAC structure remain 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. 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 implementation relies on tile-based activation organization: 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. The SIMD parallelism effectively doubles the throughput per PE in 2-bit mode without proportionally increasing area or power consumption.

D. WS/OS Reconfigurable Dataflow

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, enabling runtime switching without reconfiguration overhead.

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, such as convolutional layers with large kernel sizes.

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. This trade-off is beneficial for layers with high activation reuse or when minimizing off-chip memory bandwidth.

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, facilitating systematic exploration of dataflow trade-offs in CNN acceleration and enabling workload-specific optimization.

III. Experimental Results

A. FPGA Synthesis Results

To evaluate the hardware overhead introduced by architectural reconfigurability, we synthesized the accelerator on an Intel Cyclone 10 GX FPGA using Quartus Prime. Two configurations are compared: a *Vanilla* design supporting a single fixed dataflow and precision mode, and a *Mixed* design integrating full support for both WS/OS dataflow 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

and 2-bit/4-bit activation precision within a unified hardware fabric.

Table I summarizes the post-synthesis results. Both designs achieve the same maximum 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 modest resource increases: registers 12,155 to 12,667 (4.2%), LUTs 8,424 to 9,062 (7.6%), and DSPs 60 to 64 (6.7%), primarily from additional control logic and configuration paths. Timing analysis shows a slightly reduced data path delay (7.205 ns vs 7.571 ns), indicating effective synthesis optimization. Power consumption increases from 1119.85 mW to 1256.2 mW (12.2%), representing a moderate overhead relative to the functional flexibility gained.

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, validating the feasibility of the proposed reconfigurable architecture as a flexible CNN acceleration platform on FPGA.

IV. SYSTEM-LEVEL ENHANCEMENTS (ALPHA MODULES)

A. Alpha 2: Fine-Grained Clock Gating

Clock gating reduces dynamic power by preventing unnecessary clock toggling in idle components. We extend baseline clock gating to support WS mode. The gating logic disables clocks for PEs and memory blocks when inputs remain static. The effective gating ratio is estimated using Bayesian analysis of sparsity propagation through MAC units.

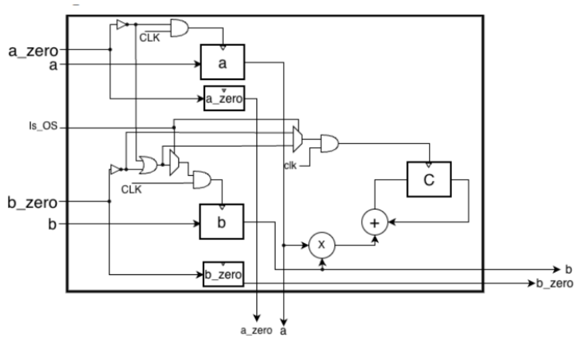

Fig. 3. Clock gating implementation showing gating logic and power reduction analysis. The design supports both WS and OS modes, with gating decisions based on input sparsity patterns.

Results demonstrate up to 88.24% power reduction, consistent with observed sparsity patterns (86% activation, 16% weight sparsity). The power reduction is achieved through fine-grained gating at the PE level, with gating decisions based on input data characteristics.

B. Alpha 3: Huffman-Based Activation Compression

We incorporate Huffman compression for activation data, exploiting the skewed distribution of quantized activations (strong skew toward zero and small values). Our implementation achieves 36.11% bit reduction. A dedicated FSM-based decoder traverses a hardcoded Huffman tree, reconstructing symbols on-the-fly (one bit per cycle) before data enters the systolic array. Upon reaching a leaf node, it outputs the 8-bit symbol and asserts `char_valid` before resetting. By placing decompression immediately before the array, the system stores compressed activations efficiently while computation proceeds on fully reconstructed data.

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 activation. This design provides hardware support for the Conv–BatchNorm–MaxPool–ReLU stack commonly found in modern CNNs without modifying the systolic array datapath.

BatchNorm parameters are assumed to be fused into convolution weights through offline model fusion (detailed in Section IV-D). 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, as illustrated in Fig. 4. 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.

D. Alpha 5: BatchNorm Fusion

BatchNorm layers are commonly used in modern CNNs to stabilize training and improve convergence. However, at inference time, BatchNorm operations introduce additional computation and memory overhead. Alpha5 adopts a Conv-BN

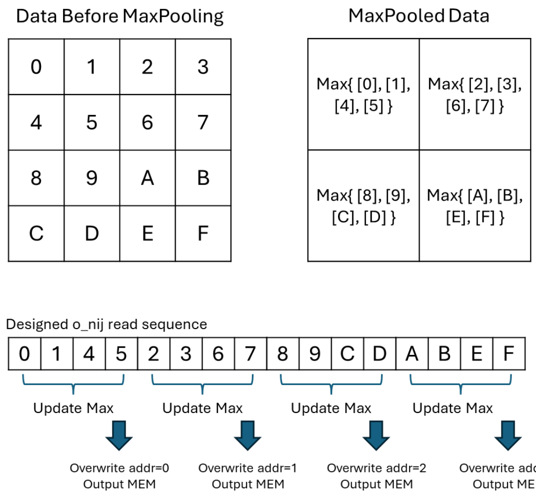
Data Before MaxPooling
MaxPooled Data
Max{ [0], [1], [4], [5] }
Max{ [2], [3], [6], [7] }
Max{ [8], [9], [C], [D] }
Max{ [A], [B], [E], [F] }
Designed o_nij read sequence
Update Max
Overwrite addr=0 Output MEM
Overwrite addr=1 Output MEM
Overwrite addr=2 Output MEM
Overwrite addr=3 Output MEM

Fig. 4. MaxPool(2×2, stride=2) hardware implementation showing the input 4×4 data grid, the resulting 2×2 pooled output, and the optimized `o_nij` read sequence. The sequence groups four consecutive elements per pooling window (e.g., 0, 1, 4, 5 for the top-left window), enabling the MaxPool unit to update the maximum over four cycles and write directly to the corresponding output memory address without additional buffering.

fusion strategy that mathematically combines BatchNorm with convolution, eliminating runtime normalization and reducing both parameters and hardware cost.

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 μ, σ^2, γ, and β are BatchNorm parameters, and ϵ is a small constant for numerical stability.

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:

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

BN parameters are fixed at inference and fused into adjacent convolution layers, eliminating runtime normalization (4.3% computation reduction, 25% hardware cost reduction). The fused parameters are 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, LeakyReLU, ELU, and GELU via runtime reconfiguration, enabling a single hardware design to support various network architectures.

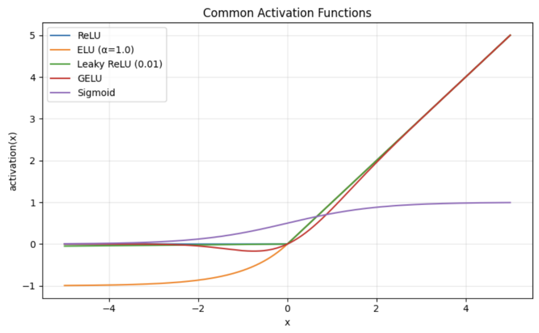

Fig. 5. Common activation functions supported by the flexible activation unit. The plot shows ReLU, ELU ($\alpha = 1.0$), LeakyReLU (slope=0.01), GELU, and Sigmoid functions over the input range $[-5, 5]$. Each function exhibits distinct characteristics: ReLU provides simple thresholding, ELU offers smooth negative responses, LeakyReLU allows small negative gradients, and GELU provides a smooth approximation.

A 2-bit control signal $m = \text{act\_func\_mode}[1:0]$ selects the active function:

$$f_{\text{act}}(x, m) = \begin{cases} f_{\text{ReLU}}(x) = \max(0, x) & \text{if } m = 2'b00 \\ f_{\text{ELU}}(x) & \text{if } m = 2'b01 \\ f_{\text{LeakyReLU}}(x) & \text{if } m = 2'b10 \\ f_{\text{GELU}}(x) & \text{if } m = 2'b11 \end{cases}$$

ReLU is implemented efficiently as $\max(0, x)$ using a simple comparator and multiplexer. 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.

The multiplexer-based architecture adds minimal overhead (shifters and comparison logic per lane). All functions are combinational and complete within one clock cycle, integrating seamlessly with the SFU pipeline and remaining compatible with all dataflow and SIMD modes.

F. Alpha7: FSM Optimization in SFU

Alpha7 redesigns the SFU with an internal FSM for autonomous operation, improving timing robustness and PSUM memory coordination. The FSM pipelines PSUM accumulation (ACC), post-processing (ReLU), and coordinated memory operations. By internally managing accumulation loops, the SFU eliminates fine-grained testbench control, preventing race conditions and simplifying integration.

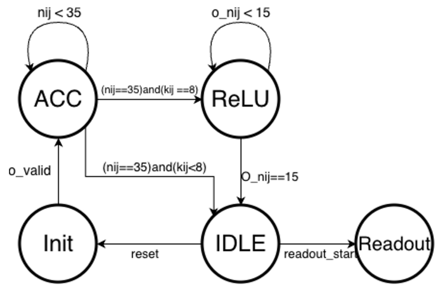

Fig. 6. SFU finite state machine (FSM) with five states: Init (waits for `o_valid`), ACC (accumulates PSUMs while `nij` < 35), ReLU (applies activation while `o_nij` < 15), IDLE (waits for `readout_start`), and Readout (outputs results). State transitions are controlled by loop counters (`nij`, `kij`, `o_nij`) and external signals, enabling autonomous operation with minimal control overhead.

The FSM-based design significantly reduces PSUM memory by accumulating directly into output-space addresses, reducing storage from $nij \times och \times kij \times psum\_bw$ to $o\_nij \times och \times psum\_bw$ (5.06 MB to 0.25 MB, 95.1% reduction), establishing a structured control framework for advanced post-processing operations.

G. Alpha 8: ConvNeXt Model Support

To explore modern efficient architectures beyond traditional CNNs, we implemented a quantized version of ConvNeXt, a modern architecture that bridges the gap between CNNs and Vision Transformers. Compared to VGG16, ConvNeXt introduces several key architectural advantages that make it superior for resource-constrained edge devices:

- **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.

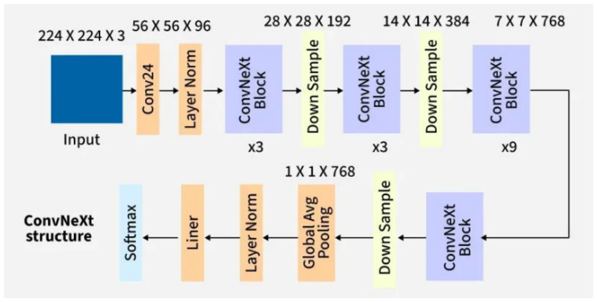

Fig. 7. ConvNeXt network architecture showing the complete data flow from input (224×224×3) through initial Conv24 and LayerNorm layers, four stages of ConvNeXt Blocks with downsampling (producing 56×56×96, 28×28×192, 14×14×384, 7×7×768 feature maps), and the classification head with Global Average Pooling, LayerNorm, Linear layer, and Softmax activation.

TABLE II
ACCURACY COMPARISON OF QUANTIZED MODELS ON CIFAR-10

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%

The software stack includes a dedicated extraction module that reshapes quantized tensors into memory layouts compatible with our accelerator (tile size $t = 8$, supporting OS and WS formats). Evaluation on CIFAR-10 (Table II) shows 4-bit quantization yields competitive accuracies: VGG16 achieves 92.13% (4-bit) and 90.67% (2-bit), while ConvNeXt achieves 89.70% with 60.5% smaller model size (13.4MB vs 33.9MB).

ConvNeXt offers a better accuracy–model size trade-off (40% of VGG16 parameters). Future challenges include LayerNorm complexity (hardware-expensive dynamic mean/variance calculation) and unsigned quantization constraints (negative values clipped to zero).

V. CONCLUSION

This work presents a fully reconfigurable 8×8 systolic-array accelerator supporting 4-bit and SIMD-style 2-bit activation processing, as well as runtime-switchable WS/OS dataflows. System-level enhancements include fine-grained clock gating (88.24% power reduction), Huffman compression (36.11% bit reduction), batch-normalization fusion (4.3% computation, 25% hardware cost reduction), flexible activation functions, and an autonomous FSM-based SFU (95.1% memory reduction). FPGA synthesis shows the reconfigurable design achieves the same frequency (125 MHz) as the baseline with moderate resource/power increases. Software–hardware co-design demonstrates competitive accuracy (92.13% VGG16, 89.70% ConvNeXt at 4-bit) with significant model size reduction. This project provides an extensible framework for exploring precision scaling, dataflow selection, and system-level optimization in CNN accelerators.