Available Online: 24 November 2026 DOI: 10.54254/2977-3903/2026.39296

FlexStream-CNN: a runtime-configurable streaming FPGA accelerator for CNN inference

Yuanping Luo

Electronic Information Engineering, Glasgow College, University of Electronic Science and Technology of China, Chengdu, China

18526129906@163.com

Abstract. Convolutional neural networks support edge-vision applications with different network shapes and latency requirements. Network-specific streaming accelerators exploit layer overlap, while shared-engine accelerators offer flexible scheduling across layers. FlexStream-Convolutional Neural Network (CNN) combines a streaming convolution-pooling front end with a folded fully connected back end for low-latency inference within a fixed LeNet-style layer sequence. Independent per-layer parallelism controls the reuse of fully connected arithmetic lanes. Runtime registers control convolution windows, stride, channel masks, pooling operations, input dimensions, and fully connected output counts. Geometry-derived vector lengths connect Flatten to the fully connected layers within fixed storage capacities. A serial configuration interface updates these controls between inferences while intermediate tensors remain on chip. Quantization-Aware Training (QAT) supplies integer weights and activations for compact storage and arithmetic, with wider accumulators and signed classification logits. Together, these mechanisms separate per-bitstream parallelism selection from runtime operator configuration within a fixed layer sequence and storage capacity. On this architecture, an output-window cycle model explains the measured latency-resource trade-off: nine routed configurations span a 24.6-fold frame-latency reduction, from 243.8 to 9.91 μs at 200 MHz, with the balanced reference point at 12.82 μs and 34.57 effective GOPS.

Keywords: FPGA, CNN accelerator, streaming architecture, runtime configurability, INT8 quantization, folding degree

1. Introduction

Convolutional Neural Networks (CNNs) support image classification, traffic-sign recognition, and other edge-vision tasks [1]. Different tasks require different network shapes and weights. A closed-loop control application also needs a short interval from input arrival to a classification result. FPGA acceleration offers spatial arithmetic and on-chip storage that can reduce this interval. The architectural challenge is to retain low latency while allowing a deployed accelerator to change supported operator parameters.

Dedicated streaming designs assign hardware to individual layers. FPQNet [2] combines quantized LeNet-5, channel parallelism, layer pipelining, and an OpenCAPI interface for low-latency classification. XNORCONV [3] combines binary convolution with conventional fully connected computation and an inter-

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

layer pipeline to reduce convolution arithmetic cost. Ji et al. [4] combine neural architecture search with a semi-parallel binary-CNN pipeline and evaluate multiple parallelism settings. FINN [5] generates quantized dataflow architectures from network descriptions and performance targets. These works establish the value of spatial pipelines and arithmetic reuse. Their generated network-specific structures motivate configuration mechanisms that operate within a deployed structure.

Shared-engine designs reuse a computing fabric across layers and networks. Eyeriss v2 [6] combines a hierarchical-mesh interconnect with flexible dataflow for compact and sparse CNNs in an Application-Specific Integrated Circuit (ASIC). SMA [7] integrates GPU-style Single-Instruction, Multiple-Data (SIMD) execution and systolic-array execution through temporal mode switching. The data-center FPGA platform [8] uses a common scheduling framework and supertile computation to process diverse CNN workloads. These architectures emphasize reuse and utilization. Their latency depends on workload size, scheduling, and memory traffic; shared execution alone does not imply low throughput or uniformly high latency.

Retargeting also depends on configuration granularity. fpgaConvNet [9] supports partitioned mappings with weight reloading and runtime-configurable interconnections in a reference architecture. Other designs use generated datapaths [5, 10], compiled instructions [11-13], or runtime kernel parameters [14]. Dynamic Partial Reconfiguration (DPR) changes hardware partitions by loading partial bitstreams [15, 16]. Register-based configuration instead retains the implemented datapath and changes its control values. It requires compatible tensor shapes and an explicit weight-loading path. Published runtime-programmable designs [17, 18] provide useful flexibility references, but their reported performance does not establish the end-to-end switching latency of this implementation.

FlexStream-CNN separates two design choices. Per-bitstream Fully Connected (FC) parallelism determines how many neurons execute concurrently. Runtime registers determine supported operator settings inside the fixed layer sequence. A streaming convolution-pooling front end overlaps feature extraction, and folded FC layers reuse their arithmetic lanes over successive neuron groups. We evaluate the FC latency-resource trade-off on a common device and clock target, then distinguish register-update cost from complete model replacement.

1.1. Contributions

1. FC folding and per-layer parallelism. We define neuron-to-lane scheduling and an output-window cycle model for independent FC parallelism. Nine routed configurations validate the model and expose the latency-resource trade-off. The results distinguish the minimum-latency setting from a balanced setting with lower estimated energy.

2. Bounded runtime-configurable streaming architecture. Runtime geometry connects Flatten to FC1, and FC output counts determine downstream vector lengths and Argmax termination. Core-level simulation verifies resolution and class-count changes within fixed capacities. Separate serial-transaction tests characterize configuration-write cost.

3. Quantized storage and arithmetic. An INT8 QAT and export path supplies signed weights and unsigned hidden activations, with wider accumulation and final logits. Software evaluation reports accuracy for five datasets without attributing cross-network accuracy differences solely to quantization.

1.2. Organization

Section II reviews background and related work. Section III defines the configurable quantized model. Section IV details the hardware architecture. Section V reports the experimental results. Section VI discusses what the results support; Section VII outlines future work; and Section VIII concludes.

2. Background and related work

2.1. Three core operations of a CNN

A convolution layer computes a sliding weighted sum over the input feature map:

$$y_{k,i,j} = \sum_c \sum_{u,v} w_{k,c,u,v} \cdot x_{c,i\cdot s+u,\, j\cdot s+v} + b_k \tag{1}$$

A pooling layer downsamples by reducing each window to one value, taking the maximum or the mean:

$$y_{k,i,j} = \max_{u,v \in W} x_{k,i\cdot s+u,j\cdot s+v}, \tag{2}$$

where W is the pooling window, replaced by a mean for average pooling. A fully connected layer computes an affine map:

$$y_j = \sum_i w_{j,i}\, x_i + b_j. \tag{3}$$

Convolution and FC layers perform Multiply-Accumulate (MAC) operations. Max pooling performs comparisons, while average pooling performs addition and scaling. Their window, stride, and dimension parameters determine both computation and tensor shape.

2.2. LeNet-style layer sequence

LeNet [1] alternates convolution and pooling before FC classification. The implemented sequence is Conv1, Pool1, Conv3, Pool2, Flatten, FC1, FC2, FC3, and Argmax. The compact sequence permits a detailed analysis of FC scheduling and front-end overlap. AlexNet [19] and VGG [20] also use convolution and FC computation, but extending their depth requires additional stages or a different scheduling architecture.

2.3. INT8 quantization

Low-bit representations reduce parameter storage and arithmetic operand widths. DoReFa-Net [21] studies low-bit training. Binarized Neural Networks (BNN) [22] use binary weights and activations. Jacob et al. [23] describe integer-only inference. These methods motivate a compact integer datapath. Accuracy retention requires comparison at the same network capacity and task; bit width alone does not establish an accuracy advantage.

2.4. Streaming and shared-engine organizations

Layer pipelines overlap production and consumption of feature-map samples, subject to buffering and rate constraints. Shared engines schedule multiple operators onto a reusable array. Neither organization uniquely determines frame latency: pipeline fill, FC input buffering, output serialization, and inter-layer scheduling all contribute. On-chip weight generation [24] offers another approach to parameter bandwidth. The relevant comparison is therefore the implemented schedule and storage boundary, rather than a universal maximum-versus-sum latency rule.

2.5. Related work and design focus

Prior work provides several forms of flexibility. Runtime configuration and instruction-driven designs [12-14, 17, 18] change computation without replacing the full device configuration. fpgaConvNet [9] combines subgraph reuse, weight reloading, and configurable interconnections. Precision and heterogeneous designs [25-29] explore arithmetic alternatives, while dynamic parallelism [30] and multi-network scheduling [31] explore other resource-allocation dimensions. BCNN-NAS [4] already evaluates multiple parallelism points,

so a parameter sweep by itself is not a unique contribution. This work focuses on a specific FC scheduling model and its resource-latency consequences inside a register-configurable streaming front end. The measured folding sweep and the bounded comparisons below connect that model to implementation results.

3. Configurable quantized model

3.1. Network structure and configuration scope

The evaluated union-capacity configuration accepts 32 × 32 × 3 input and uses Conv1 and Conv3 capacities of three channels each. With 5 × 5 convolution, stride one, and 2 × 2 stride-two pooling, the feature-map sizes are 28^2, 14^2, 10^2, and 5^2. Flatten serializes 25 samples per channel into 75 FC1 inputs. The evaluated FC dimensions are 75–120–84–43, yielding 221,592 MACs per frame. FC1 reserves capacity for 1,024 inputs; this storage capacity is distinct from the active input length.

The current RTL supports runtime FC output counts and input geometry within the same layer sequence. Registers 0x20–0x22 set the FC output counts; each preceding count determines the next FC input length, and the final count determines Argmax termination. Width and height registers 0x0C–0x0D drive geometry propagation through both convolution-pooling stages. Flatten and FC1 use the resulting spatial area and active channel count to agree on vector length. Core-level simulation checks of these mechanisms are complete.

The nine-point routed evaluation uses the current RTL with its default register values, so the active FC dimensions remain 75–120–84–43 while the runtime-dimension controls are present in the netlist. The tables in Section V therefore report the runtime-configurable design, not a separate fixed-output variant. Figure 1 shows the reference tensor sizes with the runtime controls that apply to each stage (registers 0x00–0x3F): the current RTL activates image dimensions at 0x0C–0x0D, retires the explicit FC1-length writes at 0x06–0x07, and adds runtime FC output counts at 0x20–0x22 with downstream length linkage. The controller supplies compatible weights and frame dimensions and holds configuration stable throughout inference.

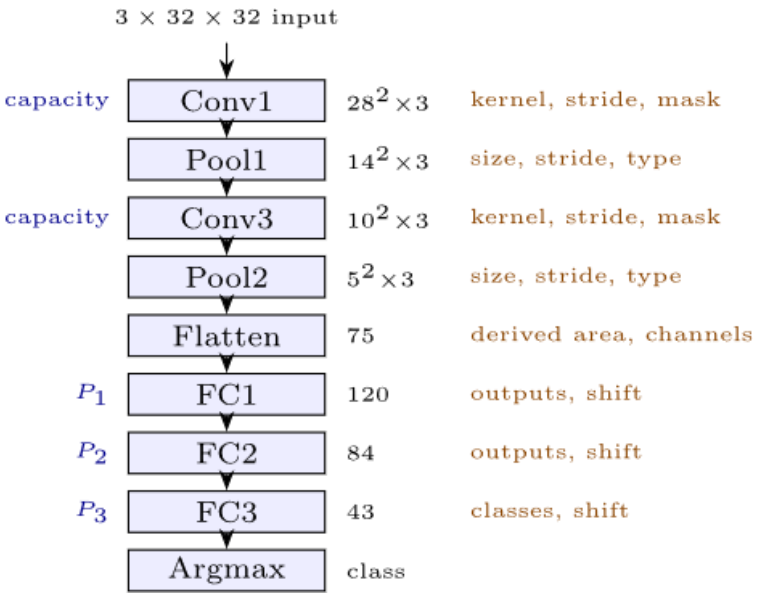

Figure 1. Reference tensor sizes and current runtime controls

3.2. Quantization and integer export

The inspected export path uses tensor-wise DoReFa normalization [21]. For a weight tensor W, it first computes

$$u = \frac{tanhW}{2max|tanhW|+\epsilon} + \frac{1}{2},$$
$$q = 2round(255u)/255 - 1, \tag{4}$$
$$W_q = clip_{[-128,127]}(round(127q)).$$

Here $\epsilon = 10^{-8}$ prevents a zero normalization denominator. The export uses NumPy rounding at each stage and clips the result to signed INT8. Bias export applies $clip_{[-128,127]}(round(127b))$. Tensor-wise normalization is a model transformation; its equivalence to a floating-point network requires separate accuracy evaluation.

The hardware multiplies unsigned 8-bit activations by signed 8-bit weights and accumulates in a wider signed word. It shifts signed 8-bit biases into the accumulator domain. For the common configured shift $s > 0$, requantization applies

$$z = (acc + 2^{s-1}) >>> s, \quad x_{hidden} = min(255, max(0, z)). \tag{5}$$

The operator $>>>$ denotes arithmetic right shift. In the absence of fixed-width overflow, the positive half-LSB offset rounds to the nearest integer with ties toward positive infinity. At $s = 0$, the offset is zero and the datapath performs no shift. Hidden-layer ReLU clips to unsigned INT8; the final FC layer retains wider signed requantized logits for Argmax. The reported tests configure $s = 7$. Matching labels does not establish equality of every logit or equivalence to floating-point inference.

Configuration writes and weight writes use distinct register addresses. A write changes a control value immediately, so the controller must finish the current inference before updating its configuration. Model replacement also requires compatible weight addresses, bias values, and input preprocessing. The documented interface does not make weights available merely by changing a dimension.

3.3. Software accuracy and published references

The software QAT study evaluates CIFAR-10, SVHN, MNIST, FashionMNIST, and GTSRB. Their input channels, spatial sizes, and class counts differ. The 16/32 model uses 16 Conv1 and 32 Conv3 output channels. We report the final models' test accuracy after ten training epochs. On CIFAR-10 and SVHN, the two-convolution capacity model reaches 63.66% and 87.65%, far below published large-network results of 80.1% [5] and 97.6% [28]; Table 1 therefore reports the three datasets whose task scale matches this capacity. These software tasks provide accuracy context; they are not five demonstrated end-to-end hardware switches. The archived hardware comparison uses CIFAR-10 stimuli in the evaluated 32 × 32 tensor layout and fixed-output FC baseline. In that baseline, the reference-export script selects test frames with a positive maximum task logit so that unused zero-valued output slots do not win Argmax. This restricted test set supports the observed label agreement, not unrestricted multi-task numeric equivalence.

Table 1. Software test accuracy and published references

Dataset	Input	QAT 16/32	Published
MNIST	1×28²	98.70%	98.4% [3]; 98.8% [2]
FashionMNIST	1×28²	89.28%	91.6% [32]
GTSRB	3×32²	84.75%	n/a

Note: 16/32 denotes the Conv1/Conv3 output-channel capacity of our INT8 QAT models. Input is channels × spatial area; n/a denotes no corresponding result. The FashionMNIST reference is the dataset's official repository benchmark for a two-convolution CNN [32].

Table 1 reports the INT8 QAT model at the 16/32 channel capacity. On MNIST, the 98.70% result falls within the published LeNet-class band of 98.4%–98.8% for the same task. On FashionMNIST, the 89.28% result sits 2.3 points below the dataset's official two-convolution CNN benchmark of 91.6% [32], a gap consistent with the smaller capacity and integer-only arithmetic. The published networks and quantization settings still differ from ours, so the table provides task context without a matched-network ranking. A paired floating-point/INT8 comparison at identical capacity must establish the quantization loss on each dataset.

4. Hardware architecture

4.1. Overall streaming architecture

Figure 2 places the streaming convolution-pooling front end, the buffered FC back end, and the configuration and storage planes in one view. Valid strobes qualify data, and frame/row synchronization marks feature-map boundaries; the evaluated modules do not expose a general ready-based backpressure protocol. Line buffers retain convolution history. Flatten and the FC input memories buffer complete vectors before FC computation. Intermediate tensors stay on chip, while an external controller supplies configuration and weights between inferences.

Figure 2 shows the same accelerator as three planes with separate physical interfaces. The external serial interface feeds a configuration manager, which drives the register file at the top; the registers carry the runtime operator and shape fields (0x00–0x3F) to every compute stage over a dedicated control bus, so operator settings change without touching the datapath layout. The parameter interface fills an on-chip parameter store, from which the convolution kernels read line-buffer weights and the folded FC layers read weight and bias memories. The image interface supplies the pixel stream to the first convolution stage, and the stages then hand feature maps forward through pooling, Flatten, and the FC back end to the label output. Keeping the three planes separate is what allows the same bitstream to be re-configured for a different tensor shape or class count between inferences.

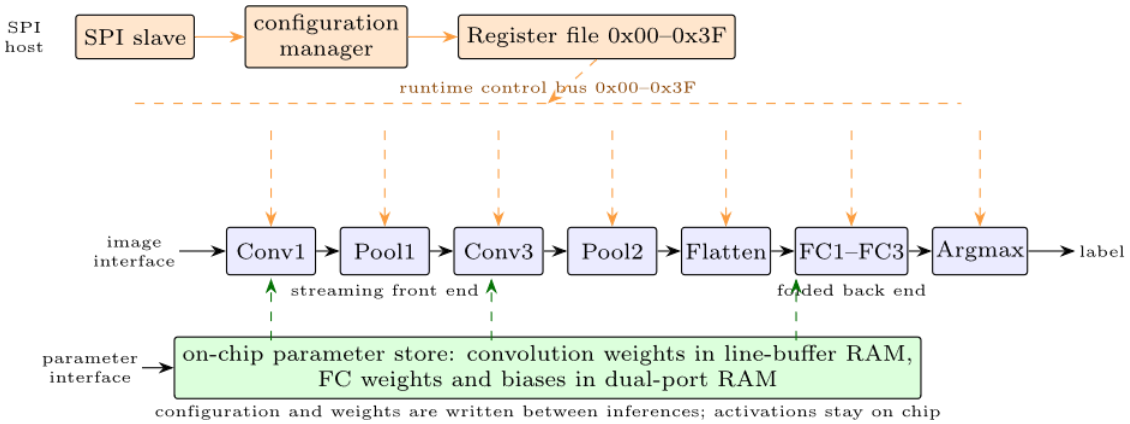

Figure 2. Hardware planes: configuration and memory management at the top, the streaming datapath in the middle, and the on-chip parameter store at the bottom

4.2. Configuration path: 16 SCLK per register write

Each Serial Peripheral Interface (SPI) write carries an 8-bit command and an 8-bit data byte, requiring 16 Serial-Clock (SCLK) rising edges. The command contains the read/write bit and register address. Figure 3 shows one transaction. The archived 16-register-write test totals 256 SCLK; this transaction count includes decoded fields and does not establish that all fields alter the datapath. Additional writes for masks, shifts, addresses, or the new FC-output controls add their own transaction cost.

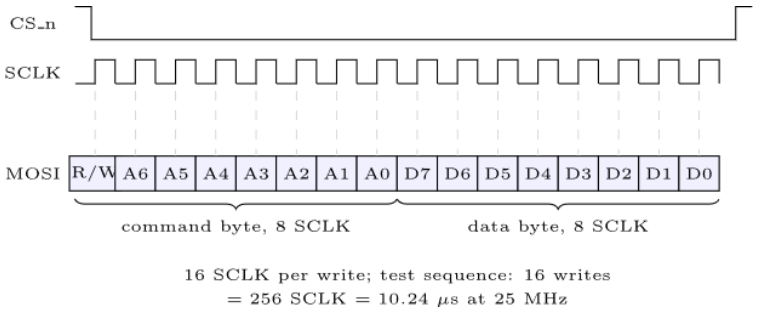

Figure 3. SPI write waveform: one 8-bit command plus one 8-bit data byte, 16 SCLK rising edges

4.3. Configurable convolution layer

The convolution datapath combines line buffers with a maximum-size sliding window (Figure 4). Row merging aligns stored samples from earlier rows with the current row. Kernel selection uses the newest $K \times K$ subwindow at offset $K_{max} - K$ and masks inactive taps. Input-channel masks exclude channel contributions from the sum. Output-channel masks qualify the outgoing valid strobes. The fixed-range tap arithmetic operates spatially in parallel; this is distinct from the folded FC neuron scheduling described later. Stride counters select which completed windows emit outputs.

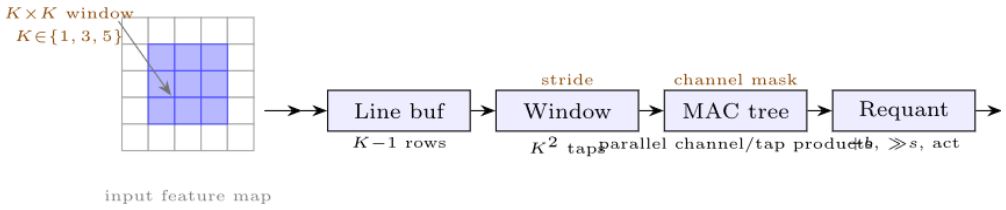

Figure 4. Window selection and convolution datapath

Figure 5 traces local runtime controls to their consumers. Kernel size selects a subwindow and valid boundaries, stride controls row/column phase selection, and channel masks affect summation or output validity. The current core propagates the resulting geometry to Flatten and FC1. This propagation changes active lengths while storage capacities remain fixed.

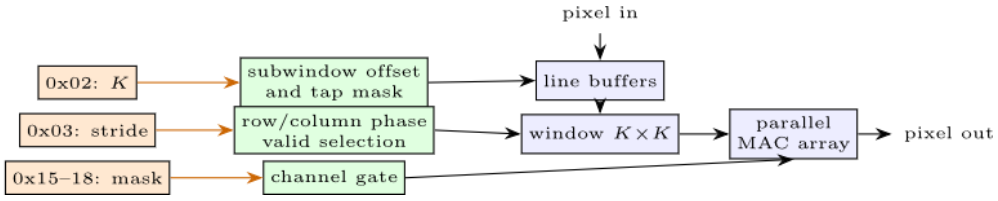

Figure 5. Runtime controls and their datapath consumers

An inactive input channel contributes zero to each tap sum. For output channel k, the integer accumulation is

$$S_k = b_{k,scaled} + \sum_{c<C_{max}} m_c \sum_{u,v<K} x_{c,u,v} w_{k,c,u,v}. \tag{6}$$

Here $m_c \in \{0,1\}$ selects an input channel. A separate output mask suppresses valid strobes for inactive output channels. The current Flatten also uses the active Conv3 output count to set serialization length. Legal configurations require positive channel counts within capacity and consistent counts across connected layers. Masking changes arithmetic and valid events; it leaves allocated storage unchanged.

4.4. Configurable pooling layer

The pooling mode selects a maximum or an average over the active window. Window size selects the included samples, and row/column phase counters select stride positions. These operators consume valid input samples and emit downsampled values. The evaluated end-to-end configuration uses max pooling with a 2 × 2 window and stride two; other operator settings require shape-compatible downstream use.

4.5. Fully connected folding and parallelism

Let L denote the active input-vector length, N the active output-neuron count, and P the synthesis-time lane parallelism. Each FC layer buffers its input vector and assigns output neuron k to lane $k \bmod P$. The P lanes process one neuron group concurrently, then the state machine advances the group base by P. For output count N,

$$R = \lceil N/P \rceil, \quad N_{tail} = N - (R-1)P. \tag{7}$$

R is the folding-round count, not the exact reciprocal of P. The final group activates only its N_{tail} lanes. Results leave a single output port in neuron order, so full arithmetic parallelism still includes serialization.

The cycle model describes the interval from the first valid FC output to the last, inclusive. Let this output-window span be $W(L,N,P)$. For the evaluated state machine,

$$W(L,N,P) = (R-1)(2L+8+P) + N_{tail}. \tag{8}$$

Each nonfinal group emits P values. Before the next group emits, weight loading, input replay, and control transitions contribute an effective inter-group preparation term $2L+8$. This term uses the measured clock-boundary convention and includes overlapping transitions; it is not a sum of separately rounded phase durations. At $R=1$, $W=N$, but the layer still collects its input and prepares the first group before producing any output.

For example, FC3 has $L=84$ and $N=43$. With $P=32$, two groups produce 32 and 11 results, yielding $W=219$ cycles. With $P=64$, one group produces 43 results, yielding $W=43$. Both cases still perform the first-group preparation. Increasing parallelism removes repeated preparations between output groups while retaining first-group preparation and serial output. Figure 6 separates these intervals.

With dimensions fixed at 75–120–84–43, changing 32/32/32 to 64/64/32 reduces the three FC output-window spans from 594/580/219 to 278/332/219. Their sum falls by 564 cycles, matching the end-to-end reduction from 3,128 to 2,564. The folding sweep below combines this scheduling benefit with resource and estimated-energy costs.

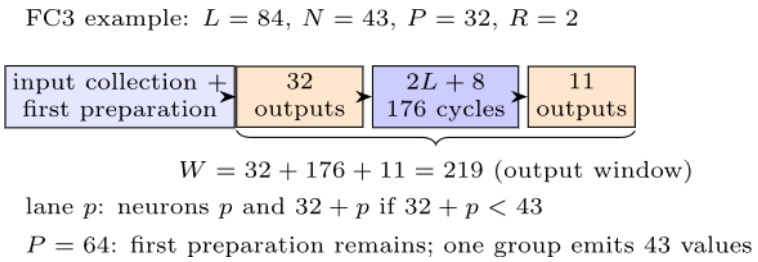

Figure 6. FC preparation, neuron groups, and output serialization

4.6. Runtime geometry and classification dimensions

Runtime geometry supplies one shared vector-length contract for Flatten and FC1. For each dimension d, the core applies $d' = \lfloor (d - K)/s \rfloor + 1$ through Conv1, Pool1, Conv3, and Pool2. At each stage, K and s denote that operator's window size and stride. With positive final width w_2, height h_2, and active Conv3 channels c_2, Flatten serializes $w_2 h_2 c_2$ values in channel-major order. FC1 consumes the same derived count. This removes independent software programming of the FC1 length; registers 0x06–0x07 remain for interface compatibility but no longer drive FC1.

The demonstrated runtime domain comprises the 32 × 32, 28 × 28, and 24 × 24 configurations verified in simulation. General configurations must fit the implemented line buffers, the encoded per-channel area threshold, FC1 storage, and its nonzero 10-bit length field. The current row buffers have 64 addresses; the area threshold uses the low eight bits of the allocated Flatten capacity. Physical storage capacity alone therefore does not establish support for every geometry. Input dimensions must match the supplied frame. Zero dimension fields select synthesis-time image defaults. Area clipping and arbitrary channel combinations remain unvalidated. All controls must remain stable during a frame.

FC output counts select active neurons within each layer's output capacity. Zero or excessive effective counts select the layer capacity. A nonzero 0x22 overrides the legacy FC3 count at 0x08–0x09; writing zero to 0x22 restores the legacy source. The next FC layer consumes the preceding active count. Argmax latches the final count at the first logit and terminates after that many valid results. Its first-logit initialization also handles all-negative signed logits.

Activation selection applies after each convolution and before pooling. The current two-bit selector chooses Binary, ReLU, or an integer-code clamp to $[0, 6]$. The clamp threshold is the code value six; equivalence to real-valued ReLU6 requires a corresponding quantization scale. FC hidden layers retain fixed ReLU with unsigned 8-bit saturation, and FC3 retains wider signed requantized logits without activation.

5. Experimental evaluation

5.1. Setup and metric definitions

The evaluation targets a KCU116 board with an xcku5p-ffvb676-2-e device (216,960 LUTs, 1,824 DSP48E2, 480 BRAM tiles) in Vivado 2017.4. All nine FC-parallelism points use the current RTL, which includes runtime FC output counts and geometry, one tensor shape, and a 200 MHz constraint. RTL simulation counts frame cycles C from the input-frame trigger to the label-valid event; with $f = 200$ MHz,

$$T = C/f, \quad F_{eff} = f/C, \quad G_{eff} = 443,184/T/10^9 \tag{9}$$

where F_{eff} assumes nonoverlapping frame processing and GOPS use 221,592 MACs per frame at two operations per MAC. Power is a Low-confidence vectorless post-route estimate and $E_{est} = P_{est}T$. Archived

timing reports cover constrained core paths and leave SPI and external I/O timing incomplete. These numbers are interpreted against the architecture in the discussion below; the runtime-dimension controls carry the core-level simulation evidence of Section V-B.

5.2. Correctness and runtime-dimension evidence

The archived baseline suite reports 30/30 checks; output labels agree with the integer reference on ten frames at PAR = 1 and two frames at each other evaluated point, covering the tested inputs rather than every intermediate value.

Vivado xsim tests exercise the current dimension controls. A core-level six-phase test switches FC3 among 43, 10, and 3 classes and restores 43, then reduces both hidden output dimensions from four to two; zero weights and class-index biases yield labels 42, 9, and 2 without timeout. An FC unit test checks six output-count settings at $P = 1, 2, 4$, including zero and excessive-count fallback: across 18 phases all 84 expected values match and no excess outputs follow. A separate core test changes only frame dimensions under fixed weights, 5 × 5 convolutions, and 2 × 2 stride-two max pooling: with two Conv3 channels, 32 × 32, 28 × 28, and 24 × 24 inputs give 50, 32, and 18 Flatten values, each equal to the first FC1 sum in the constant-value chain, with final label nine; a fourth phase checks zero width/height fallback. Fifteen activation cases cover Binary, ReLU, integer clamp, and the reserved selector.

These tests drive core configuration ports directly. They do not exercise the complete SPI-to-core path or real-weight inference on the evaluated datasets, and their cycle counters start after input-frame transmission, so we exclude them from latency comparisons.

5.3. Folding-degree design space

The design-space variable is the per-layer parallelism tuple $P_1/P_2/P_3$, fixed in each bitstream. Table 2 retains all nine points. For the fixed dimensions, the model in (7) predicts output-window spans. The measured frame-count relation is $C = 1,735 + \sum_{i=1}^{3} W(L_i, N_i, P_i)$, which reproduces the simulated frame counts. This validation covers the listed tuples and frame format, not arbitrary new network shapes.

Table 2. Nine-point FC folding ladder of the current RTL AT 200 MHZ

P1/P2/P3	R1/R2/R3	DSP48	Cycles	LUT	BRAM	T (μs)	Eff. GOPS	Est. W	Est. μJ
1/1/1	120/84/43	3	48,760	58,560	37	243.80	1.82	1.125	274.28
2/2/2	60/42/22	6	25,168	59,323	38	125.84	3.52	1.151	144.84
4/4/4	30/21/11	12	13,284	60,524	44	66.42	6.67	1.197	79.50
8/8/8	15/11/6	24	7,554	62,683	52	37.77	11.73	1.262	47.67
16/16/16	8/6/3	48	4,680	67,137	72	23.40	18.94	1.411	33.02
32/32/32	4/3/2	96	3,128	74,430	112	15.64	28.34	1.746	27.31
64/64/32	2/2/2	160	2,564	87,559	160	12.82	34.57	2.060	26.41
64/64/64	2/2/1	192	2,388	95,513	182	11.94	37.12	2.175	25.97
128/128/64	1/1/1	319	1,982	119,119	284.5	9.91	44.72	2.728	27.03

Note: RTL simulation supplies cycles; routing reports supply LUT/BRAM/DSP. FF grows 17,736 to 60,621 across the ladder. Eff. denotes serial-processing effective GOPS; Est. denotes Low-confidence vectorless estimates. All derived values use 200 MHz. Bold identifies 64/64/32. Runtime output counts keep oversubscribed lanes realized: the last row carries 319 DSP48 against a nominal lane sum of 320.

Figure 7 uses configuration index $j = 1, \ldots, 9$ in the row order of Table 2. The index separates the nearby 160- and 192-lane settings without treating them as equally spaced hardware capacities. Rings identify 64/64/32. Panel (a) shows simulated cycles and the dashed 1,735-cycle remainder; panel (b) shows estimated energy derived at 200 MHz. The energy curve carries no independent uncertainty bound because the power reports provide Low-confidence vectorless estimates.

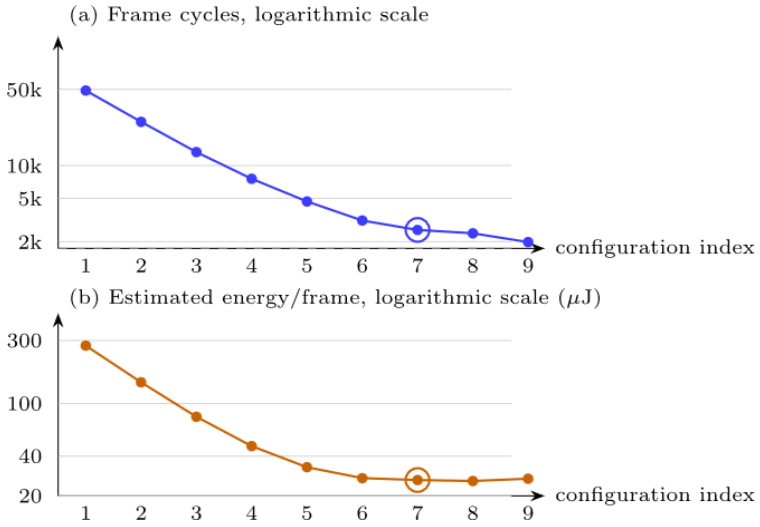

Figure 7. FC parallelism reduces cycles with diminishing returns

Increasing FC parallelism removes repeated group preparations: cycles fall from 48,760 to 1,982 (24.6-fold) while the measured remainder stays 1,735 cycles, which covers front-end traversal, Flatten serialization, first-group FC work, and interface timing. LUT use rises from 58,560 to 119,119 and BRAM from 37 to 284.5 tiles. DSP counts now track the nominal lane sums: runtime output counts keep oversubscribed lanes realized, so the last two rows carry 192 and 319 DSP48s against lane sums of 192 and 320, where a fixed-output synthesis would realize only 171 and 247. All nine points meet setup and hold constraints on the constrained core paths at 200 MHz, so their common-clock latency differences follow their cycle differences. Estimated energy falls to its minimum at 64/64/64 (25.97 μJ), stays within 2% at the deployable point (26.41 μJ), and rises at 128/128/64 (27.03 μJ); given Low-confidence vectorless power, the two mid points are indistinguishable. The discussion below interprets this curve, its diminishing returns, and the objective-dependent choice of a parallelism tuple.

5.4. Latency and throughput

Output-window spans count first-to-last valid events inclusively, while the Argmax tail counts elapsed cycles from the final FC3 result to the label event. The reference configuration produces the label-valid event after 2,564 simulated frame cycles. At the common 200 MHz clock, this derives to 12.82 μs, 78,003 effective frames/s, and 34.57 effective GOPS under serial frame processing. The routed netlist's Worst Negative Slack (WNS) of $+0.041$ ns suggests 201.7 MHz from $1/(5 - 0.041)$ ns; this is a WNS-based extrapolation, not a board-measured maximum clock. The archived reduced-mode test reports 2,314 frame cycles, but that observation does not establish general shape switching or correctness of all masked tensor layouts.

Measured output windows separate observed timing from complete stage execution: the frame-trigger-to-Flatten-first-output interval is 1,071 cycles, the three FC output spans total 829 cycles, and the remaining 664 cycles of the 2,564-cycle frame hold the residual work and boundary timing.

The FC first-output offsets from the preceding stage's last output total 591 cycles, of which the leading preparation terms $2L + 8$ account for 582 plus three boundary cycles per transition. With the 75-cycle Flatten span and the two-cycle classification tail, $664 = 75 + 591 + 2 - 4$, where the subtraction converts four inclusive spans to elapsed intervals. The front-end average is $198,900/1,071 = 185.7$ MAC/cycle and the post-front-end average $22,692/1,493 = 15.2$ MAC/cycle, characterizing schedule partitions rather than peak utilization.

5.5. Register updates and model loading

The archived SPI transaction test writes 16 registers in 256 SCLK, i.e. 10.24 μs at 25 MHz. This counts the register-write sequence; it neither verifies every decoded control nor measures a complete external model switch, which also depends on clock-domain crossing, configuration consistency, weight supply, and the readiness of the next input.

Weight loading shares the external SPI interface: each weight write at 0x10 carries one payload byte preceded by a command byte, bias writes use 0x13, and the loader also sets channel and address registers with no demonstrated auto-increment transfer. For the approximately 23,400-byte payload used in the earlier estimate, even data writes alone require at least

$$T_{SPI,data} \geq \frac{23{,}400 \times 16}{25 \times 10^6} = 14.976\ ms.$$

This is an ideal lower bound excluding address, channel, and control overhead. It is not a measured model-load duration. The earlier 0.115 ms figure assumes one payload byte per 202.6 MHz internal load cycle. The evaluated external interface does not sustain that assumption, so we do not use it as an external switching result or as a speedup over DPR.

The Processor Configuration Access Port (PCAP) and Internal Configuration Access Port (ICAP) provide partial-configuration paths in the cited controllers, with DMA supplying data for one ICAP configuration. Table 3 reports those DPR values at their own payload scopes beside this work's register transaction and external-loading lower bound: register configuration preserves the device configuration while DPR changes a hardware partition.

Table 3. Configuration costs with distinct timing scopes

Source/path	Payload or boundary	Time and evidence
PCAP [15]	five partial modules	49.5 ms, reported
ICAP + DMA [15]	five partial modules	15.5 ms, reported
VERSATILE [15]	five partial modules	4.435 ms, reported
PCAP DPR [16]	partition switch	9.1 ms, reported
This: registers	16 transactions	10.24 μs, simulated
This: SPI data	23,400 payload bytes	≥ 14.976 ms, bound
This: full switch	external supply to ready	unmeasured

Note: Times retain each source's payload and boundary. SPI data excludes address and control overhead.

5.6. Comparison with existing accelerators

We compare where the workload, precision, device, and reporting boundary are stated and comparable, and we position the remaining designs architecturally. Table 4 collects the cited designs that report resources and performance on a stated workload, spanning device, precision, LUT, DSP, latency, and rate. Works without

numerically comparable results, including fpgaConvNet [9] (normalized metrics only), Eyeriss v2 [6] (ASIC), and the shared-engine platforms [7, 8], stay in the architectural discussion below. Every number retains its source's own workload, device, and reporting basis; the table supports multi-dimensional reading, not a ranking.

Three differences follow from our own measurements. First, reconfiguration here is a register transaction that keeps intermediate tensors on chip: 16 writes take 256 SCLK (10.24 μs at 25 MHz) and the tested controls span convolution window, stride, channel masks, pooling operation, input dimensions, and FC output counts, whereas reported partial-reconfiguration flows load a hardware partition. Second, FC reuse is explicit and modeled: the neuron-group schedule and output-window model reproduce the measured frame counts of all nine routed points on one device and one clock. Third, parallelism is independent per layer, so the tuple $P_1/P_2/P_3$ (not a single global lane count) sets the latency-resource point, and the realized DSP count follows the output-count regime of Section VI.

Table 4. Multi-dimensional comparison: each row keeps its scource's workloda, device, and reporting basis

Design	Device	Precision	LUT	DSP	Latency	Rate
This work: 64/64/32	KCU116	8w8a	87,559	160	12.82 μs	34.57 eff. GOPS
This work: 128/128/64	KCU116	8w8a	119,119	319	9.91 μs	44.72 eff. GOPS
FPQNet [2]	VU37P	8-bit	53,292	2,614	9.32 μs	1,108 GOPS
XNORCONV [3]	7Z020	binary hybrid	36,798	214	18.97 μs	n/a
Ji et al. [4]	VU37P	1-bit	445k–1,318k	3,501–7,884	1.12–6.33 ms	n/a
FINN (SFC-max) [5]	Z7045	1w1a	91,131	n/a	0.31 μs	12.3M img/s
Zhang and Tang [13]	ZU3EG	8-bit	13,866	256	n/a	38.5 GOPS avg
Zhao et al. [29]	ZC702	8-bit	39,898	0	n/a	33.6 GMACS peak
Wu et al. [12]	VCU118	uint8	237k–444k	1,347–4,419	n/a	311.48 GOPS
Irmak et al. [16]	Zynq-7020	n/r	26,845	220	62 μs	45K img/s
NRUCA [18]	U250	8-bit	300k	2,381	0.03 ms	799.3 GOP/s
Shah et al. [17]	VX485T	18-bit	78,318	1,034	n/a	162.7 GOp/s
Prost-Boucle et al. [28]	VX690T	2-bit	170,555	95–99	0.135 ms	60.2K fps

Note: Ranges cover multiple configurations in one source. Eff. denotes our serial-processing effective rate at 200 MHz; external rates retain peak, aggregate, or average definitions. n/a denotes no comparable value; n/r denotes not reported. Devices differ by generation and capacity, so the columns support reading, not ranking.

Each closest neighbour fixes at compile time something we keep changeable at run time. FPQNet [2] specializes a quantized pipeline to one network, so its operator settings are fixed in the bitstream; we expose them as registers inside a fixed tensor-compatible structure. XNORCONV [3] specializes convolution to binary operands and keeps a separate conventional FC path; we keep INT8 operands end to end and still report a shorter one-frame latency on a LeNet-class workload (12.82 μs against 18.97 μs). fpgaConvNet [9] changes behaviour by reloading weights and configuring interconnections for a partition; our control path needs 16 register writes with intermediate tensors left on chip. FINN [5] generates a dataflow structure from a network description, so a different setting implies a different generated structure; our runtime fields change supported operator settings inside one existing structure.

Our minimum-latency point completes the 221,592-MAC frame in 9.91 μs (44.72 effective GOPS under serial frame processing at 200 MHz) and the balanced reference point in 12.82 μs (34.57 effective GOPS), both at two operations per MAC. The closest external latency with a comparable boundary is XNORCONV's

hybrid LeNet pipeline at 18.97 μs for one frame on an XC7Z020, so our reference point is shorter by about 32% on a LeNet-class workload and the fully unrolled point by about 48%. The comparison holds at each work's reported boundary.

Table 2 isolates FC parallelism under a common workload and clock, which is where our claim lives, and the quoted external latency places that operating region in context. Our rates derive from serial-processing frame cycles, so we compare them only against numbers with a comparable boundary; matched-network accuracy and complete switching measurements remain outside the current evidence.

6. Discussion

6.1. Why the numbers look as they do

Every number in Table 2 comes from one of three architectural parts. The streaming convolution-pooling front end overlaps layer execution and contributes a nearly constant cost, the main part of the 1,735-cycle baseline remainder that no FC setting removes. Flatten and the serial emission of FC results add a fixed serialization floor, which is why 247 output values still cost cycles at the fully unrolled endpoint. The folded FC backend supplies the part parallelism changes: each layer executes $R_i = \lceil N_i/P_i \rceil$ neuron groups, so its cycle term $W(L_i, N_i, P_i)$ shrinks with P_i while its lane count grows. The measured relation $C = 1,735 + \sum_i W(L_i, N_i, P_i)$ therefore states where the architecture spends time, not a fitted curve.

It also explains the two DSP regimes. Under a fixed output count, synthesis prunes lanes beyond the layer's output count, realizing $min(P_i, N_i)$ multipliers, so a fixed-output 128/128/64 netlist carries 247 DSPs. With runtime output counts, every lane is reachable at run time, so synthesis retains the nominal lane sum: the routed 128/128/64 point carries 319 DSP48s against 320 lanes. Independence per layer makes the tuple, not a single global parallelism, the design variable.

6.2. What the results support

The favorable reading is the ratio of what is gained to what is paid. Across the ladder, frame cycles fall 24.6-fold (48,760 to 1,982) while LUT use grows 2.03-fold (58,560 to 119,119) and DSPs stay within a sixth of the device. The reference point reaches 12.82 μs at 87,559 LUTs and 160 DSP48E2 and the minimum-latency point 9.91 μs, both from a LeNet-class workload on one device at one clock; all nine points close timing on the constrained core paths, so latency differences are cycle differences rather than frequency differences. Reconfiguration is also cheap where multi-task use matters: operator and shape settings change through 16 register writes (256 SCLK) with intermediate tensors left on chip, avoiding a hardware-partition reload.

6.3. What the results do not support

Three limits bound that reading. First, the returns diminish sharply: from 32/32/32 only 1,146 cycles remain to the endpoint, and from 64/64/32 only 582, because the remainder is front-end and serialization cost that parallelism cannot remove. The last step from 64/64/64 to 128/128/64 buys 406 cycles for 23,606 LUTs and 127 additional DSP48s. Second, the energy ranking is weak evidence: the estimates are Low-confidence vectorless numbers, and the lowest two points (25.97 and 26.41 μJ) differ by 1.7%, so the sweep identifies a region rather than a precise optimum. Third, the rates assume serial, nonoverlapping frame processing rather than a measured steady-state initiation interval.

A further cost is specific to runtime configurability. When FC output counts become runtime-controlled, lanes beyond the compile-time output count can no longer be pruned. The routed ladder confirms this: against

a fixed-output netlist realizing 171 and 247 DSPs, the current RTL carries 192 and 319 DSP48E2 at 64/64/64 and 128/128/64, with BRAM rising from 171 to 182 and from 247 to 284.5 tiles, while 64/64/32 (where $P_3 = 32 < 43$) keeps 160 DSP48s and 160 tiles. Runtime flexibility therefore costs area exactly at the points where parallelism exceeds the class count, and that cost belongs to the configurability claim rather than to the folding model.

6.4. Choosing a parallelism tuple

No single tuple is best, and the ladder shows why. At 1/1/1, 58,560 LUTs and 3 DSPs buy 243.80 μs, which suits a tight area budget and no latency requirement. At 16/16/16 and 32/32/32, the curve is still steep: 32/32/32 reaches 15.64 μs at 74,430 LUTs, near the knee where additional lanes stop paying for themselves. At the reference point 64/64/32, $P_3 = 32$ stays below the 43-class output count, so every FC3 lane is reachable without oversubscription; the result is 12.82 μs and 87,559 LUTs, with estimated energy within 2% of the sweep minimum. At 64/64/64, $P_3 = 64$ exceeds 43 classes and buys 176 cycles over the reference at 7,954 additional LUTs. At 128/128/64, latency is minimal at 9.91 μs and the rate peaks at 44.72 effective GOPS, but the point costs 119,119 LUTs, 319 DSPs, and 27.03 μJ.

The practical rule follows the objective: 128/128/64 when per-frame latency dominates, 64/64/32 for balanced latency, area, and estimated energy, and 32/32/32 or below for a small device. When the target dimensions are fixed at synthesis time, choosing $P_i = N_i$ per layer avoids paying for lanes beyond the output count; runtime output counts require P_i to cover the largest supported dimension, and the oversubscribed lanes then remain in the netlist.

6.5. Scope limits

The fixed LeNet-style sequence, storage capacities, and tensor shapes bound every claim above; the dimension tests exercise core configuration ports rather than the complete SPI-to-core path or real-weight inference on the evaluated datasets, and the archived timing reports leave SPI and external I/O timing incomplete.

7. Future work

The configuration strategy can extend through explicit operator interfaces and tensor-layout contracts: wider channels require new capacities, deeper sequences additional stages or revised scheduling, and residual paths [33] new routing and buffering. Further tests should exercise SPI-driven real-weight task switching, complete external-loading time, paired floating-point/INT8 accuracy, and routed measurements of the runtime-dimension implementation discussed above.

8. Conclusion

FlexStream-CNN combines an on-chip streaming convolution-pooling front end with independently folded FC layers. Its neuron-group schedule and output-window model explain a nine-point trade-off: frame cycles fall from 48,760 to 1,982 under a common 200 MHz target. The minimum-latency configuration is 128/128/64; 64/64/32 is the balanced reference, with estimated energy within 2% of the minimum at 64/64/64. The current RTL links runtime input geometry, Flatten length, FC output counts, and Argmax termination within fixed capacities. Core-level tests verify selected resolution and class-count changes independently of the baseline's routed performance data. A 16-write configuration test takes 256 SCLK and excludes complete external weight replacement. Quantized weights and hidden activations reduce storage widths, while wider

accumulators and logits preserve the integer classification path. Together, FC scheduling analysis and tested runtime dimension linkage support configuration choices without conflating functional flexibility with implementation performance.

References

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

[2] Ji, M., Al-Ars, Z., Hofstee, H. P., Chang, Y., & Zhang, B. (2023). FPQNet: Fully pipelined and quantized CNN for ultra-low latency image classification on FPGAs using OpenCAPI. *Electronics, 12*(19), Article 4085. https://doi.org/10.3390/electronics12194085

[3] Zhang, L., Bu, X., & Li, B. (2020). XNORCONV: CNNs accelerator implemented on FPGA using a hybrid CNNs structure and an inter-layer pipeline method. *IET Image Processing, 14*(1), 105–113. https://doi.org/10.1049/iet-ipr.2019.0385

[4] Ji, M., Al-Ars, Z., Chang, Y., & Zhang, B. (2024). Fully pipelined FPGA acceleration of binary convolutional neural networks with neural architecture search. *Journal of Circuits, Systems and Computers, 33*, Article 2450170. https://doi.org/10.1142/S0218126624501706

[5] Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2017). FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)* (pp. 65–74). ACM. https://doi.org/10.1145/3020078.3021744

[6] Chen, Y.-H., Yang, T.-J., Emer, J. S., & Sze, V. (2019). Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9*(2), 292–308.

[7] Guo, C., Zhou, Y., Leng, J., Zhu, Y., Du, Z., Chen, Q., Li, C., Yao, B., & Guo, M. (2020). Balancing efficiency and flexibility for DNN acceleration via temporal GPU-systolic array integration. In *Proceedings of the Design Automation Conference (DAC)*. ACM.

[8] Yu, X., Wang, Y., Miao, J., Zhang, H., Meng, Y., Zhang, B., Min, B., Chen, D., Gao, J., & Wu, E. (2019). A data-center FPGA acceleration platform for convolutional neural networks. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. IEEE.

[9] Venieris, S. I., & Bouganis, C. S. (2019). fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. *IEEE Transactions on Neural Networks and Learning Systems, 30*(2), 326–342. https://doi.org/10.1109/TNNLS.2018.2844093

[10] Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., Blott, M., Lavagno, L., Vissers, K., Wawrzynek, J., & Keutzer, K. (2019). Synetgy: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)* (pp. 23–32). ACM. https://doi.org/10.1145/3289602.3293902

[11] Guo, K., Sui, L., Qiu, J., Yu, J., Wang, J., Yao, S., Han, S., Wang, Y., & Yang, H. (2018). Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37*(1), 35–47. https://doi.org/10.1109/TCAD.2017.2705069

[12] Wu, G., Lv, J., Chen, Y., Yin, S., & Liu, Y. (2026). Towards high-performance flexible FPGA-based accelerators for CNN inference. *ACM Transactions on Reconfigurable Technology and Systems, 19*(2), 1–31. https://doi.org/10.1145/3800704

[13] Zhang, M., & Tang, W. (2025). Design of a general-purpose CNN accelerator. In *Proceedings of the EEICE* (pp. 664–667). IEEE. https://doi.org/10.1109/EEICE65049.2025.11034038

[14] Dua, A., Li, Y., & Ren, F. (2020). Systolic-CNN: An OpenCL-defined scalable run-time-flexible FPGA accelerator architecture for accelerating convolutional neural network inference in cloud/edge computing. In *Proceedings of the IEEE International Conference on Field-Programmable Custom Computing Machines (FCCM)* (pp. 231). IEEE. https://doi.org/10.1109/FCCM48280.2020.00064

[15] Ibrahim, M., Pillement, S., Pinna, A., & Le Nours, S. (2025). VERSATILE: Very fast partial reconfiguration controller. *ACM Transactions on Reconfigurable Technology and Systems, 18*(3), 1–22. https://doi.org/10.1145/3748728

[16] Irmak, H., Ziener, D., & Alachiotis, N. (2021). Increasing flexibility of FPGA-based CNN accelerators with dynamic partial reconfiguration. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)* (pp. 306–311). IEEE. https://doi.org/10.1109/FPL53798.2021.00061

[17] Shah, N., Chaudhari, P., & Varghese, K. (2018). Runtime programmable and memory bandwidth optimized FPGA-based coprocessor for deep convolutional neural network. *IEEE Transactions on Neural Networks and Learning Systems, 29*(12), 5922–5934. https://doi.org/10.1109/TNNLS.2018.2815085

[18] Ye, W., Li, H., Zhou, X., Pi, H., & Li, K. (2026). A design of network reconfigurable universal CNN accelerator based on FPGA. *ACM Transactions on Embedded Computing Systems, 25*(5), 1–24. https://doi.org/10.1145/3779218

[19] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (Vol. 25)*, 1097–1105. Curran Associates.

[20] Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition.* arXiv. https://doi.org/10.48550/arXiv.1409.1556

[21] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). *DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients*. arXiv. https://doi.org/10.48550/arXiv.1606.06160

[22] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). *Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1.* arXiv. https://doi.org/10.48550/arXiv.1602.02830

[23] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2704–2713. IEEE. https://doi.org/10.1109/CVPR.2018.00286

[24] Venieris, S. I., Fernandez-Marques, J., & Lane, N. D. (2023). Mitigating memory wall effects in CNN engines with on-the-fly weights generation. *ACM Transactions on Design Automation of Electronic Systems, 28*(6), 1–31. https://doi.org/10.1145/3611673

[25] Chang, S.-E., Li, Y., Sun, M., Shi, R., So, H. K.-H., Qian, X., Wang, Y., & Lin, X. (2021). Mix and match: A novel FPGA-centric deep neural network quantization framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (pp. 208–220). IEEE. https://doi.org/10.1109/HPCA51647.2021.00027

[26] Gong, Y., Xu, Z., He, Z., Zhang, W., Tu, X., Liang, X., & Jiang, L. (2022). N3H-Core: Neuron-designed neural network accelerator with heterogeneous FPGA cores. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)* (pp. 112–122). ACM. https://doi.org/10.1145/3490422.3502367

[27] Cao, Y., Jiang, H., Li, H., Tang, Y., Shi, D., & Zhao, G. (2025). HMSA: High-performance heterogeneous mixed-precision CNN systolic array accelerator on FPGA. *ACM Transactions on Embedded Computing Systems, 24*(5s), 1–27. https://doi.org/10.1145/3759458

[28] Prost-Boucle, A., Bourge, A., & Pétrot, F. (2018). High-efficiency convolutional ternary neural networks with custom adder trees and weight compression. *ACM Transactions on Reconfigurable Technology and Systems, 11*(3), Article 15. https://doi.org/10.1145/3270764

[29] Zhao, M., Li, X., Zhu, S., & Zhou, L. (2019). A method for accelerating convolutional neural networks based on FPGA. In *Proceedings of the International Conference on Computer Information Science and Artificial Intelligence (ICCIS)* (pp. 241–246). IEEE. https://doi.org/10.1109/ICCIS49662.2019.00049

[30] Dai, K., Xie, Z., & Liu, S. (2025). DCP-CNN: Efficient acceleration of CNNs with dynamic computing parallelism on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 44*(2), 540–553. https://doi.org/10.1109/TCAD.2024.3435996

[31] Zeng, S., Dai, G., Zhang, N., Yang, X., Zhang, H., Zhu, Z., Yang, H., & Wang, Y. (2023). Serving multi-DNN workloads on FPGAs: A coordinated architecture, scheduling, and mapping perspective. *IEEE Transactions on Computers, 72*(5), 1314–1328. https://doi.org/10.1109/TC.2022.3214113

[32] Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms.* arXiv. https://doi.org/10.48550/arXiv.1708.07747

[33] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). IEEE. https://doi.org/10.1109/CVPR.2016.90