

An Ultra-Compact, BRAM-Free Streaming 2D Convolution Accelerator on Low-Power FPGAs

Amir Atef

24P0310@eng.asu.edu.eg

Ain Shams University

Yahia Sameh

Ain Shams University

Research Article

Keywords: 2D Convolution, FPGA Accelerator, Streaming Architecture, Line Buffer, Deep Learning Hardware, Edge AI, Fixed-Point Arithmetic, Shift Registers

Posted Date: October 14th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11344548/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

An Ultra-Compact, BRAM-Free Streaming 2D Convolution Accelerator on Low-Power FPGAs

Amir Atef Abdallah Yahia Sameh Zakaria

Faculty of Engineering, Ain Shams University, Cairo, Egypt

amiratefabdallah1122@gmail.com

yahiasameh645@gmail.com

Abstract

Two-dimensional (2D) spatial convolution represents the primary computational workload in edge computer vision and deep learning inference. However, conventional implementations on resource-constrained embedded platforms suffer from severe memory bandwidth bottlenecks and latency penalties when buffering full video frames in external memory or on-chip Block RAM (BRAM). In this paper, we present an ultra-compact, streaming 3×3 2D convolution hardware accelerator tailored for single-channel image streams on low-power Field-Programmable Gate Arrays (FPGAs). The proposed architecture ingests continuous raster-scan pixel streams at one pixel per clock cycle and reconstructs 3×3 spatial neighbourhoods on the fly using dual line buffers entirely mapped onto Look-Up Table (LUT)-based shift-register primitives (SRLC32E), eliminating Block RAM utilization entirely. The spatial neighbourhood is processed by a three-stage pipelined Multiply-Accumulate (MAC) datapath featuring nine parallel DSP48E1 multipliers, a balanced adder tree, and a mathematically derived 20-bit internal accumulator that guarantees exact, overflow-free accumulation across all valid input and weight combinations. An output narrowing stage incorporates monotonic 16-bit saturation clamping and optional Rectified Linear Unit (ReLU) activation to prevent signed overflow wrap-around artifacts. Control flow is orchestrated by a three-state Moore finite state machine (FSM) providing input stream gating and output-sample-based drain accounting, ensuring complete pipeline flushes without stall overhead. The architecture was synthesized and implemented for a Xilinx Zynq-7020 SoC (xc7z020c1g400-1) under a 10 ns (100 MHz) clock constraint. Post-synthesis reports confirm zero Block RAM usage, 231 Slice LUTs (0.43 %), 417 flip-flops (0.39 %), and 9 DSP48E1 slices (4.09 %), achieving a worst negative slack of +3.724 ns (estimated $f_{\max} \approx 159.3$ MHz) and an active compute power of only 9 mW (0.159 W total on-chip). The pipeline achieves an initial latency of $2W + 7$ cycles (71 cycles for 32×32 frames) and sustains a peak throughput of one output pixel per cycle (100 Mpixel/s, 900 MMAC/s). Exhaustive multi-tier verification confirms zero mismatches, including a 900/900 bit-exact match against an independent 32-bit software golden model on natural photography and an exact 1860-window geometric match on non-square images.

Keywords: 2D Convolution, FPGA Accelerator, Streaming Architecture, Line Buffer, Deep Learning Hardware, Edge AI, Fixed-Point Arithmetic, Shift Registers.

1 Introduction

Two-dimensional (2D) convolution constitutes the foundational computational primitive across modern computer vision algorithms and deep convolutional neural networks (CNNs) [1–3]. Across vision workloads, 2D convolution filters extract spatial representations ranging from low-level morphological features (such as edge gradients, sharpening, and Gaussian blurs) to high-dimensional hierarchical features in multi-layer deep networks. For a canonical 3×3 spatial kernel, computing each output pixel requires nine multiplications and eight additions over a localized spatial neighbourhood. When applied across high-frame-rate sensory streams, the sheer volume of multiply-accumulate (MAC) operations imposes substantial processing demands.

Executing spatial 2D convolutions on conventional general-purpose central processing units (CPUs) or microcontrollers expends most of the system energy budget on instruction fetch, decode, register spilling, and memory cache misses rather than on arithmetic computation [4]. While graphics processing units (GPUs) deliver high parallel throughput, their substantial power consumption (typically tens to hundreds of watts) prohibits

deployment in energy-constrained edge devices such as micro-aerial vehicles, wearable sensors, and embedded Internet-of-Things (IoT) vision platforms [5].

Field-Programmable Gate Arrays (FPGAs) provide fine-grained bit-level customizability, hard digital signal processing (DSP) slices, and high energy efficiency, making them well-suited for edge intelligence [6, 7]. However, implementing 2D spatial convolution on resource-constrained FPGAs introduces two major engineering challenges:

1. **Data Ingestion and Buffering Bottleneck:** Image sensors transmit video streams serially in raster-scan order (one pixel per clock cycle along horizontal scanlines). Conversely, a 3×3 spatial kernel requires simultaneous access to pixels across three vertically adjacent rows. Storing the entire image frame on-chip in Block RAM (BRAM) or off-chip in external DRAM incurs prohibitive memory footprints, high memory latency, and excessive dynamic power dissipation [8, 9].
2. **Precision and Arithmetic Overflow:** Fixed-point quantized CNN layers frequently operate on 8-bit integers [10]. When calculating the sum of multiple signed products, intermediate register overflow must be prevented. Furthermore, narrowing wider intermediate accumulators to the final output precision often introduces wrap-around artifacts (where bright positive activations wrap around to negative values), causing feature inversion and destabilizing downstream vision models.

To resolve these challenges, this paper presents an ultra-compact, streaming 3×3 convolution accelerator tailored for resource-constrained edge FPGAs. By employing dual chained shift-register line buffers, the architecture buffers only two scanlines of image data ($2W$ pixels), reconstructing full 3×3 spatial neighbourhoods on the fly with zero Block RAM utilization. The design features a fully unrolled, three-stage pipelined MAC datapath equipped with nine hard DSP48E1 slices, computing one complete 2D convolution per clock cycle in steady state.

1.1 Key Contributions

The main contributions of this work are as follows:

- **BRAM-Free Streaming Microarchitecture:** We design a streaming window generator that avoids dedicated Block RAM tiles by mapping dual line buffers onto 16 Look-Up Table (LUT)-based shift-register primitives (SRLC32E). This achieves a 512-bit on-chip storage footprint for 32×32 imagery, freeing scarce BRAM blocks for downstream neural network parameters.
- **Mathematically Exact Accumulation with Saturated Narrowing:** We establish a formal dynamic range derivation demonstrating that a 20-bit internal accumulator guarantees zero intermediate overflow across all signed 8-bit weight and unsigned 8-bit pixel extremes. A terminal monotonic 16-bit saturation stage and optional ReLU activation prevent wrap-around feature inversion.
- **Deterministic Control and Zero-Stall Pipeline:** We formulate a three-state Moore control FSM with input stream gating and output-sample drain accounting, ensuring complete pipeline flushes without bubble-insertion stalls, achieving an initial output latency of $L = 2W + 7$ clock cycles.
- **Rigorous Multi-Tier Verification:** We demonstrate 100% bit-exact equivalence (900/900 outputs) against an independent 32-bit Python/NumPy golden model on natural photography, validated alongside non-square geometric testing (64×32 , 1860 windows) and directed boundary stress cases.
- **Comprehensive Hardware Characterization:** We present post-synthesis implementation results on a Xilinx Zynq-7020 SoC, providing critical-path logic breakdowns, power dissipation profiles (distinguishing 9 mW fabric compute from I/O and static leakage), and an architectural trade-off analysis between shift registers and BRAMs.

2 Related Work

2.1 FPGA Accelerators for Convolutional Neural Networks

The deployment of deep neural networks on embedded FPGAs has received substantial attention due to the demand for energy-efficient edge intelligence. Zhang et al. [6] proposed an analytical design methodology based on the roofline model, optimizing computation-to-communication ratios via loop tiling and unrolling on high-performance FPGAs. Qiu et al. [5] introduced an embedded FPGA accelerator for large-scale VGG models on the Zynq ZC706 platform, demonstrating that convolutional layers are compute-centric while fully

connected layers are memory-bandwidth-bound. Podili et al. [7] explored fast 2D convolution algorithms on FPGAs, optimizing DSP usage through spatial-domain mappings. In contrast to these macroscopic systems that target large multi-layer CNNs backed by external DDR memory, our work investigates an ultra-compact, standalone 2D convolution IP core optimized for streaming edge sensors without DDR or BRAM dependencies.

2.2 Streaming 2D Convolution and Line Buffering Microarchitectures

Streaming window generation via line buffers is a classic architectural paradigm in hardware image processing [11]. Perri et al. [12] designed a reconfigurable 2D convolution processor on FPGAs using SIMD arithmetic modules and line buffers to minimize redundant pixel fetches. Shi et al. [13] developed D-SWIM, a line buffer microarchitecture capable of handling arbitrary image dimensions without run-time reconfiguration, focusing on BRAM bank allocation. Kang [9] proposed AoCStream, an all-on-chip CNN accelerator that replaces frame buffers with stream-based line buffers, eliminating off-chip DRAM accesses entirely for resource-constrained edge FPGAs.

While conventional line buffer architectures typically instantiate dual-port BRAM blocks, smaller feature maps or localized video front-ends do not require large kilobit-scale memories. In 7-series Xilinx FPGAs, standard Block RAM tiles provide 18 Kb or 36 Kb minimum granularity [14]; allocating a full BRAM tile to store a 256-bit or 512-bit scanline results in severe memory under-utilization. Our architecture resolves this inefficiency by exploiting distributed Look-Up Table shift registers (SRLC32E), providing a zero-BRAM footprint while maintaining single-cycle throughput.

2.3 Quantization and Integer-Arithmetic Inference

Quantization of neural network activations and weights from single-precision floating point (FP32) to 8-bit fixed-point integers (INT8) has become standard for edge deployment [4, 10]. Jacob et al. [10] formalized an integer-only quantization scheme that maps float operations to integer arithmetic with zero runtime floating-point conversions. In dedicated hardware datapaths, fixed-point integer arithmetic eliminates the large silicon area and multi-cycle pipeline latency associated with floating-point units. However, fixed-point datapaths require precise accumulator sizing and deliberate saturation policies to handle arithmetic overflows. Our design builds upon these principles by providing a mathematically closed proof of non-overflowing internal accumulation, coupled with monotonic saturation clamping and optional ReLU activation.

3 Mathematical Formulation and Dynamic Range Analysis

3.1 Two-Dimensional Convolution Arithmetic

The accelerator computes valid-mode 2D spatial cross-correlation, conventionally termed 2D convolution across deep learning frameworks:

$$\text{acc}[i, j] = \sum_{m=0}^2 \sum_{n=0}^2 x[i + m, j + n] \cdot w[m, n], \quad 0 \leq i \leq H - 3, \quad 0 \leq j \leq W - 3, \quad (1)$$

where x represents an input image matrix of dimensions $H \times W$, w is the 3×3 spatial kernel, and $\text{acc}[i, j]$ denotes the unquantized intermediate accumulator sum. Under valid boundary mode (no synthetic zero-padding applied to borders), the spatial dimensions of the resulting output feature map are given by:

$$H_{\text{out}} \times W_{\text{out}} = (H - 2) \times (W - 2). \quad (2)$$

For the baseline reference input of $H = 32$ and $W = 32$, the accelerator generates an output feature map of $30 \times 30 = 900$ samples. Equation 1 implements cross-correlation without spatial kernel rotation (180° reflection), matching standard software formulations in PyTorch and TensorFlow. If mathematical convolution with kernel reflection is desired, the host programs the kernel coefficients in spatially inverted order.

3.2 Precision, Word-Length Derivation, and Overflow Prevention

The input pixels are represented as 8-bit unsigned integers, $x[i, j] \in [0, 255]$, corresponding to the native digitizer output of standard CMOS grayscale image sensors and post-ReLU quantized neural network activations [10]. The kernel coefficients are represented as 8-bit two’s complement signed integers:

$$w[m, n] \in [-128, 127]. \quad (3)$$

To determine the required datapath word lengths, we analyze the arithmetic extremes. The maximum absolute product of a single tap occurs at the boundary of both dynamic ranges:

$$|x \cdot w|_{\max} = 255 \times 128 = 32\,640. \quad (4)$$

A signed 16-bit register in two’s complement notation represents integers in the range $[-32\,768, 32\,767]$. Because $32\,640 < 32\,768$, a product word length of `PRODUCT_WIDTH` = 16 bits prevents overflow for any individual tap multiplication.

Accumulating the nine products of a 3×3 kernel yields the theoretical maximum accumulator magnitude under uniform worst-case inputs and weights:

$$|\text{acc}|_{\max} = \sum_{k=1}^9 32\,640 = 9 \times 32\,640 = 293\,760. \quad (5)$$

Accommodating the sum of nine 16-bit products requires an additional bit growth headroom of:

$$\text{GROWTH} = \lceil \log_2 9 \rceil = 4 \text{ bits}. \quad (6)$$

Setting the internal accumulator register width to `ACC_WIDTH` = 16 + 4 = 20 bits provides a representable two’s complement range of:

$$\text{acc} \in [-524\,288, 524\,287]. \quad (7)$$

Because the maximum achievable magnitude satisfies $|\text{acc}|_{\max} = 293\,760 < 524\,287$, intermediate accumulator overflow is mathematically impossible. The nine-term addition is numerically exact across all valid image datasets and kernel configurations.

3.3 Output Precision Narrowing and Saturation Policy

The accelerator interface produces a standard 16-bit signed output $y[i, j] \in [-32\,768, 32\,767]$. Narrowing the 20-bit exact accumulator down to 16 bits necessitates an overflow policy. Conventional truncation or modular wrapping discards the four most significant bits ($[19 : 16]$), which causes sign inversion: large positive values wrap to large negative values, turning high-intensity visual edges into dark troughs. To maintain monotonicity and visual signal fidelity, the datapath implements hard saturation clamping:

$$y[i, j] = \begin{cases} +32\,767, & \text{acc}[i, j] > 32\,767, \\ -32\,768, & \text{acc}[i, j] < -32\,768, \\ \text{acc}[i, j][15 : 0], & \text{otherwise.} \end{cases} \quad (8)$$

Saturation preserves directional gradient extrema and prevents activation wrapping.

3.4 Optional Rectified Linear Unit (ReLU) Activation

When the control signal `relu_en` is asserted, the saturated 16-bit output passes through a non-linear activation unit:

$$y_{\text{relu}}[i, j] = \max(0, y[i, j]) = \begin{cases} 0, & y[i, j] < 0, \\ y[i, j], & y[i, j] \geq 0. \end{cases} \quad (9)$$

This allows the accelerator to serve directly as a combined convolution-plus-activation layer for quantized edge neural networks without requiring an external processor.

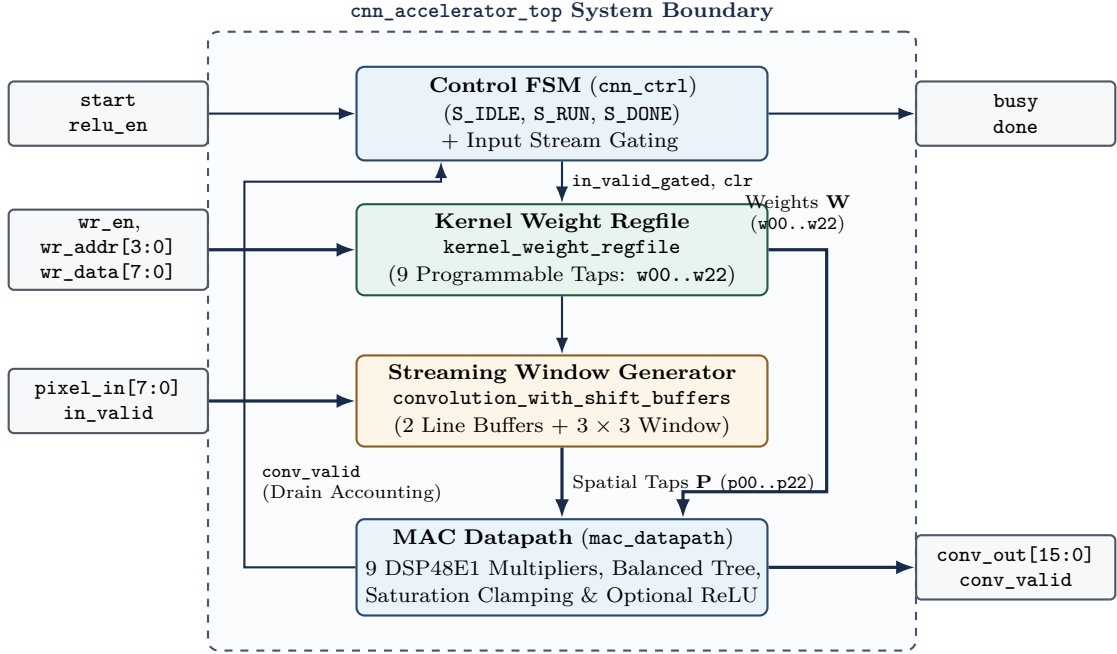


Figure 1: Top-level hardware block diagram of the streaming 2D convolution accelerator.

3.5 Sliding-Window Memory Storage Schemes

Computing Equation 1 requires simultaneous access to nine pixels spanning three consecutive image scanlines. Raster-scan sensors stream pixels sequentially row-by-row, separating vertically adjacent pixels by exactly W clock cycles. Three primary architectural strategies exist to resolve this temporal-spatial mismatch:

1. **Full Frame Buffering:** The entire image of $H \times W$ pixels is captured into on-chip SRAM or external DRAM before processing begins. This introduces a full-frame latency penalty ($\approx H \cdot W$ cycles) and requires large storage arrays (8192 bits for 32×32 , or 16.8 Mb for 1080p).
2. **Multi-Pass Sensor Re-Reading:** The processing element requests the image three times, traversing the image per kernel row. This eliminates large buffers but requires three times the sensor bandwidth.
3. **Streaming Line Buffering:** The architecture retains only the two most recent scanlines in circular delay buffers ($2W$ pixels). A 3×3 window register file samples the incoming stream alongside the delayed tap lines, producing a valid window every clock cycle.

The streaming line buffer approach minimizes storage to exactly $2W$ pixels (512 bits for $W = 32$). Crucially, 512 bits is small enough to map entirely into FPGA Look-Up Table shift registers, completely avoiding Block RAM instantiation.

4 Hardware Architecture and Microarchitecture

4.1 Top-Level System Organization

The complete accelerator is implemented in synthesizable Verilog as a modular hierarchy rooted at `cnn_accelerator_top`. As illustrated in Figure 1, the top level houses the master control finite state machine (FSM) and coordinates three primary submodules:

- **weights:** A 9×8 -bit programmable kernel register file.
- **window_gen:** The streaming convolution window generator, containing two cascaded instances of `line_buffer_shift` and a 3×3 register grid.
- **mac:** The three-stage pipelined arithmetic datapath executing multiplication, balanced accumulation, saturation, and ReLU.

The physical interface signals and functional descriptions are catalogued in Table 1. Parameterization supports

Table 1: Interface port list of `cnn_accelerator_top`.

Port Identifier	Direction	Width (bits)	Functional Description
<code>clk</code>	Input	1	Primary system clock (single synchronous domain)
<code>rst_n</code>	Input	1	Asynchronous active-low global reset
<code>start</code>	Input	1	Control strobe: initiates or restarts a convolution pass
<code>busy</code>	Output	1	High during active frame ingestion (<code>state == S_RUN</code>)
<code>done</code>	Output	1	Asserted when all output samples have been drained
<code>pixel_in</code>	Input	8	Raster-scan streaming pixel data (unsigned integer)
<code>in_valid</code>	Input	1	Input qualifier indicating valid data on <code>pixel_in</code>
<code>wr_en</code>	Input	1	Synchronous write strobe for kernel programming
<code>wr_addr</code>	Input	4	Tap index address ($0 \leq \text{wr_addr} \leq 8$)
<code>wr_data</code>	Input	8	Kernel coefficient value (two's complement signed)
<code>relu_en</code>	Input	1	Configuration flag enabling post-convolution ReLU
<code>conv_out</code>	Output	16	Pipelined filtered convolution output sample (signed)
<code>conv_valid</code>	Output	1	Output qualifier indicating valid result on <code>conv_out</code>

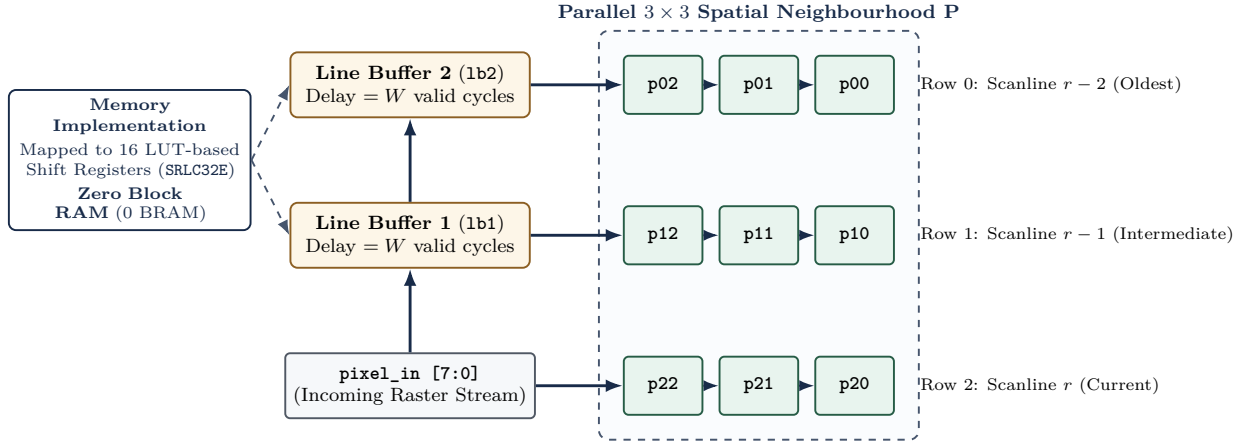


Figure 2: Microarchitecture of streaming window formation in `convolution_with_shift_buffers`.

reconfigurable spatial dimensions via `IMAGE_WIDTH` and `IMAGE_HEIGHT` (default 32), with configurable bit-widths `PIXEL_WIDTH` (8), `WEIGHT_WIDTH` (8), and `OUT_WIDTH` (16).

4.2 Streaming Window Generation and Shift-Register Line Buffers

The module `convolution_with_shift_buffers` reconstructs the 3×3 spatial kernel neighbourhood from the raster stream. As detailed in Figure 2, two line buffers (1b1 and 1b2) are chained in series. The incoming pixel stream `pixel_in` feeds the third row of the window (`p22`) directly while simultaneously entering 1b1. The output of 1b1 has experienced an exact delay of `IMAGE_WIDTH` valid cycles, corresponding to the preceding image scanline; it feeds the second row (`p12`) and enters 1b2. The output of 1b2 represents pixels delayed by $2 \times \text{IMAGE_WIDTH}$ cycles, feeding the first row (`p02`).

Each window row consists of three registers shifting from column 2 to column 0 on every accepted pixel. This forms a full spatial matrix $\mathbf{P} = [p_{mn}]$ available in parallel.

A critical architectural decision concerns how line buffer storage is implemented. In `line_buffer_shift`, storage comprises two parallel shift registers: an 8-bit-wide array `shift_reg` of depth W holding pixel data, and a 1-bit-wide array `valid_sr` holding valid qualification bits. To maximize FPGA synthesis efficiency, synchronous reset is applied *only* to `valid_sr`, while `shift_reg` remains unreset:

$$\text{clr} \implies \text{valid_sr} \leftarrow 0, \quad \text{shift_reg} \text{ unchanged.} \quad (10)$$

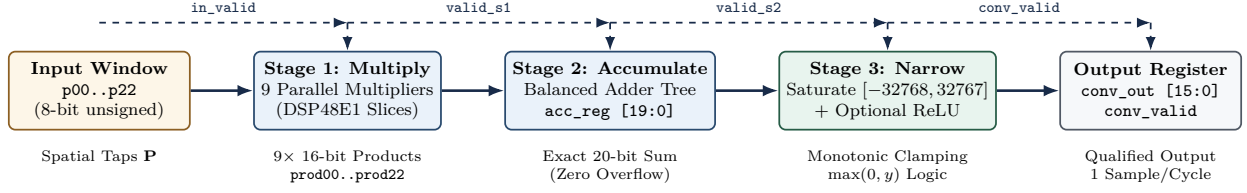


Figure 3: Three-stage arithmetic pipeline organization within `mac_datapath`.

Applying a synchronous reset to all data bits would force the synthesizer to map the storage onto discrete flip-flops and multiplexers. By omitting synchronous reset on the data array, the synthesis tool maps the $2 \times 32 \times 8 = 512$ bits of line buffer memory directly into 16 SRLC32E primitives within the FPGA’s Configurable Logic Blocks (CLBs) [14]. This achieves an exceptionally compact implementation consuming only 16 LUTs and exactly zero Block RAM blocks.

4.3 Pipelined Multiply-Accumulate (MAC) Datapath

To satisfy target timing constraints at high operating frequencies, the arithmetic pipeline in `mac_datapath` is structured into three registered stages, as diagrammed in Figure 3.

Stage 1: Multiplier Array

In Stage 1, all nine products are computed simultaneously in hard DSP48E1 slices. Because input pixels are unsigned $[0, 255]$ while kernel weights are signed $[-128, 127]$, direct signed interpretation of raw 8-bit pixel values would incorrectly treat pixel intensities greater than 127 as negative numbers. To prevent this, each 8-bit pixel is explicitly zero-extended by one bit before undergoing signed evaluation: `prod00 <= $signed({1'b0, p00}) * w00;`. The RTL enforces direct hardware mapping by attaching the synthesis directive (`* use_dsp = "yes"`) to the product registers, guaranteeing allocation of nine dedicated DSP48E1 multipliers [15].

Stage 2: Balanced Accumulation

Stage 2 sums the nine 16-bit signed product registers into a single 20-bit register `acc_reg`. The sum is expressed as a single balanced expression:

$$\text{acc_sum} = \sum_{m=0}^2 \sum_{n=0}^2 \text{prod}_{mn}, \quad (11)$$

allowing the logic synthesis engine to automatically optimize the carry-save adder tree and minimize routing propagation delays.

Stage 3: Saturation Clamping, ReLU, and Output Registration

Stage 3 applies non-linear transfer functions via combinational logic prior to the terminal flip-flops:

$$\text{sat_val} = \max(-32768, \min(32767, \text{acc_reg})), \quad (12)$$

$$\text{conv_out} \leftarrow \text{relu_en} ? \max(0, \text{sat_val}) : \text{sat_val}. \quad (13)$$

Valid qualifier signals (`valid_s1`, `valid_s2`) track data through the pipeline stages, asserting `conv_valid` synchronously alongside `conv_out`.

4.4 Kernel Weight Register File

The module `kernel_weight_regfile` stores the 3×3 kernel coefficients in an internal array of nine 8-bit signed registers. A single-clock write port (`wr_en`, `wr_addr`, `wr_data`) allows updating coefficients sequentially prior to assertion of `start`. Out-of-bounds addresses (`wr_addr` ≥ 9) are decoded as no-ops, providing interface protection. Global reset clears all nine registers to zero, preventing spurious arithmetic transients on startup.

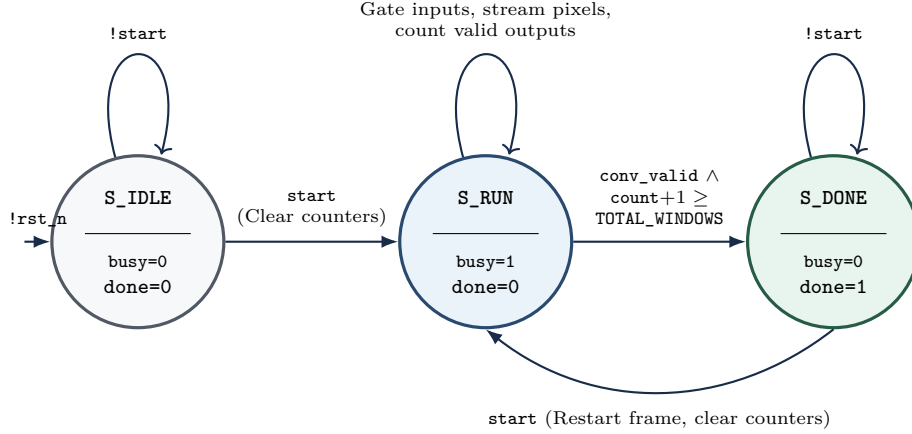


Figure 4: State transition diagram of the Moore control FSM.

4.5 Control Sequencing, Gating, and Latency Formulation

System execution is governed by a three-state Moore finite state machine (S_IDLE, S_RUN, S_DONE) situated in `cnn_accelerator_top`, shown in Figure 4.

The controller incorporates two key architectural mechanisms:

1. **Input Stream Gating:** The internal valid signal `in_valid_gated` qualifies external `in_valid` with the FSM state (`state == S_RUN`) and ensures total fed pixels do not exceed $H \times W$. Any pixel strobes arriving before `start` or after frame completion are discarded, preventing corruption of line buffer contents.
2. **Output-Sample Drain Accounting:** If the controller transitioned out of S_RUN immediately upon ingesting the final pixel, the remaining pixels in flight across the line buffers and MAC stages would be truncated. Instead, the controller monitors the actual output strobe: `conv_valid ^ (output_count + 1 ≥ TOTAL_WINDOWS) ⇒ state ← S_DONE`, where $TOTAL_WINDOWS = (W - 2)(H - 2)$. This ensures the pipeline drains fully without stall bubbles or auxiliary drain counters.

Latency and Inter-Line Throughput

Under continuous raster ingestion, the first complete 3×3 spatial neighbourhood is formed when pixel index $(2W + 2)$ is received (corresponding to spatial coordinates row 2, column 2). Registering `window_valid` introduces one cycle of delay, and the three MAC stages introduce three additional cycles. Consequently, measuring from the clock cycle the first pixel is ingested:

$$L = \underbrace{(2W + 3)}_{\text{Line buffer fill}} + \underbrace{1}_{\text{Window registration}} + \underbrace{3}_{\text{MAC pipeline}} = 2W + 7 \text{ clock cycles.} \quad (14)$$

For $W = 32$, $L = 71$ clock cycles (710 ns at 100 MHz).

Within any active image row, the datapath sustains an output rate of one pixel per clock cycle. However, when the window slides past the horizontal image boundary, the column counter resets. To prevent convolution across adjacent rows, `window_valid` is deasserted whenever `col_count < 2`, resulting in a deterministic two-cycle idle bubble at every scanline transition, as illustrated in Figure 5.

For a 32×32 frame, 30 bursts of 30 valid cycles are separated by 29 two-cycle gaps. The 900 valid outputs span $900 + (29 \times 2) = 958$ cycles, yielding a total frame processing time of $70 + 958 = 1028$ cycles.

5 Verification Methodology and Experimental Validation

5.1 Hierarchical Multi-Tier Verification Strategy

To ensure robust functional correctness, an exhaustive four-tier hierarchical verification strategy was executed, progressing from unit-level arithmetic stress testing to system-level image co-simulation. All testbenches

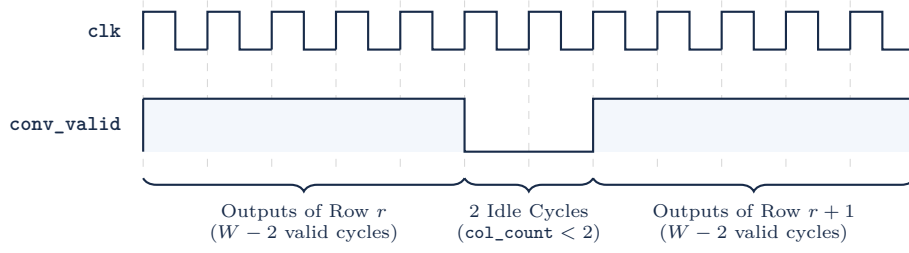


Figure 5: Timing waveform structure of `conv_valid` across scanline transitions.

Table 2: Comprehensive multi-tier simulation verification results.

Tier	Testbench	Evaluation Stimulus	Applied Kernel	Checks	Errors	Status
Unit (Math)	<code>tb_mac_datapath</code>	Signed range, positive saturation, negative saturation, ReLU	Edge, all +127, all -128	4 corner cases	0	PASS
Unit (Geom)	<code>tb_convolution</code>	Non-square (64×32) image, all 9 tap coordinates verified	Geometric grid	1860 windows	0	PASS
Integration	<code>tb_cnn_acc</code>	Full-chip 6×4 image, FSM <code>busy/done</code> handshake	Identity (centre 1)	8 windows	0	PASS
System	<code>tb_cnn_file_io</code>	Natural photography (32×32), cycle-by-cycle Python co-sim	Laplacian (3×3)	900 outputs	0	PASS

are automated and self-checking, evaluating cycle-accurate assertions against expected values. The complete verification outcomes across all tiers are catalogued in Table 2.

5.2 Unit Verification: MAC Datapath and Arithmetic Boundaries

The testbench `tb_mac_datapath` validates arithmetic precision, saturation boundaries, and ReLU behaviour across four targeted operational corner cases:

- Linear Signed Operation:** A directional edge kernel $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ applied to spatial inputs $1 \dots 9$ yields $(1 + 4 + 7) - (3 + 6 + 9) = -6$, confirming signed multiplication and adder tree balance in the absence of overflow.
- Positive Saturation Clamping:** Nine weights of +127 multiplied by nine pixels of 255 yield an exact mathematical sum of +291 465. The datapath clamps the result to +32 767, verifying positive saturation.
- Negative Saturation Clamping:** Nine weights of -128 multiplied by nine pixels of 255 yield -293 760 (the theoretical minimum of Eq. 5). The datapath clamps to -32 768, confirming negative saturation.
- ReLU Non-Linearity:** Repeating Case 3 with `relu_en` asserted clamps the negative saturated value to 0, validating ReLU functionality.

All test vectors passed with zero errors, exhibiting a deterministic two-cycle latency from `window_valid` to `conv_out`.

5.3 Unit Verification: Window Generator on Non-Square Dimensions

The testbench `tb_convolution` validates window generation on a non-square image of dimensions 64×32 . Using non-square dimensions is critical to exposing any latent row/column transposition bugs in hardware counters. Input pixels were assigned synthetic coordinates following $p(r, c) = (r \cdot W + c) \bmod 256$.

On every asserted `window_valid`, the testbench verifies all nine spatial taps (p00 through p22) against mathematical closed-form coordinates. All $1860 = (64 - 2) \times (32 - 2)$ windows were validated with zero mismatches, confirming that scanline transitions and delay buffers preserve strict geometric alignment.

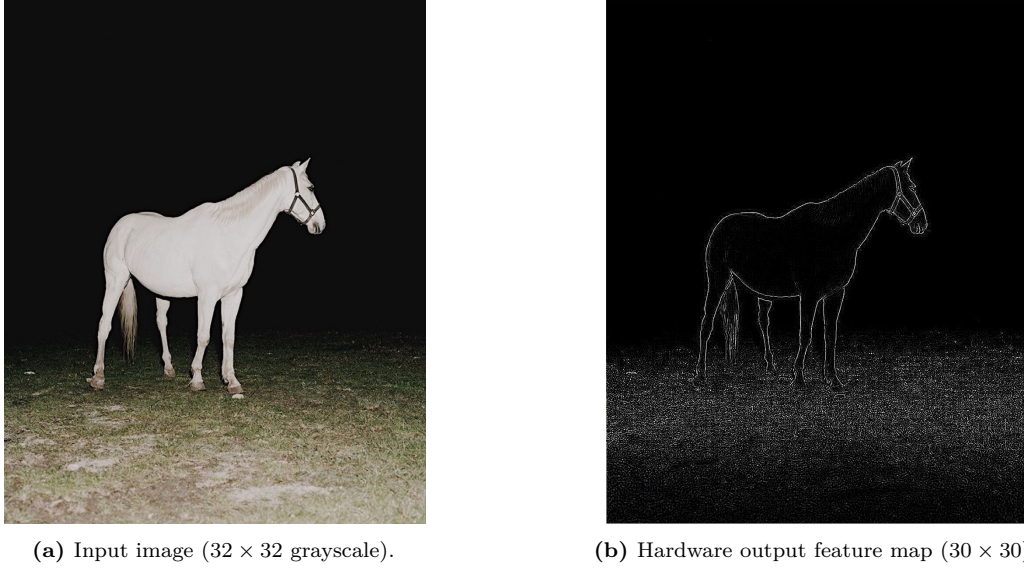


Figure 6: Experimental verification imagery: original photographic input and hardware Laplacian response.

5.4 Integration Verification: Full-Chip Handshake

The testbench `tb_cnn_accelerator_top` performs full-chip integration testing on a 6×4 image using an identity spatial kernel (centre tap 1, all other taps 0). Under an identity kernel, each output pixel must identically reproduce the corresponding input pixel at coordinates (r, c) . All eight output samples matched the test pattern exactly, with correct handshaking transitions on `busy` and `done`.

5.5 System Verification: Golden Model Image Co-Simulation

System-level verification was conducted using `tb_cnn_file_io`. A natural photographic image was converted to an 8-bit 32×32 grayscale input array (Figure 6a). A Python/NumPy golden model computed the 2D convolution using a discrete 3×3 Laplacian edge detection kernel:

$$\mathbf{K}_{\text{Laplacian}} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad (15)$$

evaluating exact 32-bit sums followed by 16-bit saturation. The testbench initialized the accelerator, streamed 1024 input pixels, and performed automated cycle-by-cycle comparisons against the software reference. The hardware accelerator achieved a 100 % bit-exact match (900/900 samples) with zero discrepancies. Figure 6b illustrates the reconstructed hardware output feature map, clearly depicting the extracted high-frequency structural contours.

6 FPGA Implementation Results

6.1 Implementation Flow and Synthesis Constraints

The hardware design was synthesized and implemented using the Xilinx Vivado Design Suite targeting the Xilinx Zynq-7020 All Programmable SoC (xc7z020c1g400-1). Synthesis was executed with default baseline parameters (32×32 frame, 8-bit unsigned pixels, 8-bit signed weights, 16-bit signed outputs). The design constraint file (`constraints.xdc`) specified a single synchronous 100 MHz clock:

$$\text{create_clock -period 10.000 -name clk [get_ports clk]} \implies T_{\text{clk}} = 10.000 \text{ ns}. \quad (16)$$

All 45 package I/O ports were mapped to the LVCMOS33 standard.

Table 3: Post-synthesis implementation, static timing, and power characterization on Xilinx Zynq-7020 (xc7z020c1g400-1) under a 100 MHz (10.0 ns) constraint.

Metric / Resource Category	Utilized / Measured	Device Available	Utilization / Share
A. FPGA Fabric Resource Utilization			
Slice LUTs (Total)	231	53 200	0.43 %
LUT as Logic	215	53 200	0.40 %
LUT as Shift Register (SRLC32E)	16	17 400	0.09 %
Slice Registers (Flip-Flops)	417	106 400	0.39 %
Block RAM Tiles (RAMB36/RAMB18)	0	140	0.00 %
DSP48E1 Multiplier Slices	9	220	4.09 %
Bonded I/O Pins (IOB, LVCMOS33)	45	125	36.00 %
Global Clock Buffers (BUFG)	1	32	3.13 %
B. Static Timing Performance at 100 MHz ($T_{\text{target}} = 10.000 \text{ ns}$)			
Worst Negative Slack (WNS, Setup)	+3.724 ns	0 failing endpoints	901 total endpoints
Worst Hold Slack (WHS, Hold)	+0.105 ns	0 failing endpoints	901 total endpoints
Worst Pulse Width Slack (WPWS)	+4.020 ns	0 failing endpoints	443 total endpoints
Estimated Maximum Frequency (f_{max})	159.3 MHz	Target: 100 MHz	Margin: +59.3 %
Critical Path Delay (9 logic levels)	6.172 ns	3.414 ns logic	2.758 ns routing
C. Power Dissipation Breakdown at 100 MHz (Vivado Power Analysis, 25 °C)			
Clock Tree Distribution	0.004 W (4 mW)	—	7.5 % dynamic
Slice Logic (LUT, FF, SRL, CARRY4)	0.001 W (1 mW)	—	1.9 % dynamic
Interconnect Signals	0.002 W (2 mW)	—	3.8 % dynamic
DSP48E1 Arithmetic Slices	0.002 W (2 mW)	—	3.8 % dynamic
Core Computational Fabric Dynamic	0.009 W (9 mW)	—	17.0 % dynamic
I/O Pin Drivers (45 LVCMOS33 pads)	0.044 W (44 mW)	—	83.0 % dynamic
Total Dynamic Power	0.053 W (53 mW)	—	100.0 % dynamic
Device Static Power (Silicon Leakage)	0.107 W (107 mW)	—	67.3 % total
Total On-Chip Power Dissipation	0.159 W (159 mW)	—	—

6.2 Hardware Resource Utilization, Timing, and Power Profile

Table 3 synthesizes the complete post-synthesis physical implementation results, static timing metrics, and power dissipation profile on the target Xilinx Zynq-7020 SoC.

As demonstrated in Table 3A, the accelerator achieves an ultra-compact footprint consuming less than 0.5 % of the Zynq-7020 fabric. Crucially, the 512 bits of line buffer storage map entirely into 16 SRLC32E LUT shift registers, eliminating Block RAM usage (0.00 %). The nine dedicated DSP48E1 slices provide parallel single-cycle multiplication without consuming logic LUTs.

Timing analysis (Table 3B) confirms that the design easily meets timing at 100 MHz with +3.724 ns setup slack, enabling an estimated maximum operating frequency of:

$$f_{\text{max}} = \frac{1}{T_{\text{target}} - \text{WNS}} = \frac{1}{10.000 \text{ ns} - 3.724 \text{ ns}} = \frac{1}{6.276 \text{ ns}} \approx 159.3 \text{ MHz}. \quad (17)$$

The critical path originates at the multiplier product registers and terminates at the accumulator register `acc_reg`. The 6.172 ns total delay traverses nine logic levels (3.414 ns logic delay and 2.758 ns net routing delay), showing that fabric accumulation is the primary timing determinant while the hard DSP multipliers complete Stage 1 within 2.5 ns.

The power breakdown (Table 3C) indicates that the computational core consumes merely 9 mW of dynamic power. External 3.3 V I/O pin drivers account for 44 mW (83.0 % of dynamic power). When embedded within an SoC as an on-chip IP block communicating via internal AXI4-Stream buses, these external pad drivers are eliminated, confining active operational dissipation to under 10 mW.

6.3 Throughput, Latency, and Computational Efficiency

Table 4 synthesizes the operational performance metrics of the accelerator operating at 100 MHz. To evaluate the joint architectural efficiency across throughput, total power dissipation, and heterogeneous FPGA resource

Table 4: Latency, throughput, and performance summary for 32×32 imagery at 100 MHz.

Performance Metric	Measured Value	Analytical Basis
MAC Operations per Output Pixel	9 MACs	3×3 spatial kernel taps
MAC Operations per Frame	8100 MACs	900×9
Initial Output Latency	71 cycles (710 ns)	$2W + 7$ from Eq. 14
Output Feature Map Samples	900 pixels	$(W - 2)(H - 2)$ from Eq. 2
Total Frame Processing Cycles	1028 cycles (10.28 μ s)	Complete frame fill to pipeline drain
Peak Pixel Throughput	1.000 pixel/cycle	Continuous valid row streaming
Effective Frame Throughput	0.875 pixel/cycle	900 output pixels / 1028 total cycles
Peak Pixel Processing Rate	100.0 Mpixel/s	At 100 MHz system clock
Peak Arithmetic Compute Rate	900.0 MMAC/s (1.8 GOPS)	9×100 MHz
Effective Frame Rate	97.28 kframes/s	$1/(10.28 \mu\text{s})$
Energy per Image Frame	1.634 μ J	$0.159 \text{ W} \times 10.28 \mu\text{s}$
Energy per Output Pixel	1.816 nJ	$1.634 \mu\text{J}/900 \text{ pixels}$
Figure of Merit (Peak, FOM_{peak})	9.24×10^{-3}	From Eq. 18 ($1.000/[0.159 \times 681]$)
Figure of Merit (Avg., FOM_{avg})	8.09×10^{-3}	From Eq. 18 ($0.875/[0.159 \times 681]$)

footprint, we evaluate a composite hardware Figure of Merit (FOM) adapted from standard FPGA accelerator benchmarks [5, 6]:

$$\text{FOM} = \frac{\text{Throughput}}{\text{Power} \times (\text{LUTs} + 50 \cdot \text{DSPs} + 100 \cdot \text{BRAMs})} \quad (18)$$

where Throughput is evaluated in output pixels per cycle, Power is the total on-chip dissipation (0.159 W), and the denominator weights macroeconomic primitives by standard silicon area equivalence factors (1 DSP $\approx$ 50 LUTs, 1 BRAM $\approx$ 100 LUTs, yielding an equivalent resource footprint of $231 + 450 = 681$ units). Under peak continuous streaming (1.000 pixel/cycle), the architecture achieves $\text{FOM}_{\text{peak}} = 9.24 \times 10^{-3}$, while across full frame processing including boundary and fill overhead (0.875 pixel/cycle), the accelerator sustains an effective $\text{FOM}_{\text{avg}} = 8.09 \times 10^{-3}$.

7 Discussion and Architectural Trade-offs

7.1 Spatial Unrolling vs. Time-Multiplexed Compute Engines

The proposed datapath employs fully unrolled spatial parallelism, instantiating nine DSP48E1 slices to compute all tap products concurrently in a single clock cycle. An alternative architectural strategy is a time-multiplexed design, wherein a single physical MAC unit iteratively accumulates the nine taps over nine consecutive clock cycles. While time-multiplexing reduces DSP usage from nine slices to one, it drops throughput to 0.11 pixels/cycle and necessitates complex multi-bank window buffering, state machines, and pipeline stalls. Given that modern low-cost FPGAs (such as the Zynq-7020) contain over 200 DSP slices, dedicating nine slices (4.09%) to achieve single-cycle throughput represents a highly advantageous area-performance trade-off.

7.2 Shift-Register LUTs vs. Block RAM Crossover Analysis

The decision to implement line buffering via LUT shift registers (SRLC32E) rather than Block RAM is directly informed by memory scaling laws:

- For small image scanlines ($W = 32$), two line buffers require $2 \times 32 \times 8 = 512$ bits. Instantiating a 36 Kb Block RAM primitive to store 512 bits represents less than 1.5% memory capacity utilization, needlessly tying up a coarse-grained BRAM tile. In contrast, 16 LUTs provide the exact required capacity.
- As scanline width W scales to high-definition video (e.g., $W = 1920$), two line buffers require $2 \times 1920 \times 8 = 30720$ bits. Implementing 30.7 Kb in shift-register LUTs would require 960 LUTs, at which point migrating line buffers to dedicated Block RAM tiles becomes substantially more area-efficient.

Our microarchitecture thus occupies the optimal design niche for localized edge vision windows, patch processing, and compact feature maps.

7.3 Critical Path Bottlenecks and Frequency Optimization

Timing analysis indicates that the primary critical path lies in the nine-input adder tree within Stage 2, which requires nine levels of fabric logic delay (6.172 ns total). By comparison, the DSP48E1 multiplier slices finish within approximately 2.5 ns. To push clock frequencies beyond 200 MHz, two architectural optimizations can be deployed:

1. **DSP Pipelined Accumulation:** Leveraging the internal 48-bit adder within each DSP48E1 slice and cascading them via dedicated PCOUT→PCIN routing columns [15], bypassing general fabric routing entirely.
2. **Pipelined Tree Reduction:** Dividing the nine-operand adder into a two-level registered binary tree, inserting an intermediate pipeline register between the partial sums and final accumulator.

7.4 Impact of Monotonic Saturation Clamping

In edge computer vision systems, feature extraction layers feed directly into classification heads or edge detection thresholds. Modular arithmetic overflow causes sign-bit inversion: intense white edges ($\geq +32768$) wrap around to dark negative values, creating spurious gradient reversals that severely degrade downstream task accuracy. By enforcing monotonic saturation clamping (Eq. 8), extreme activations remain pinned to $+32767$, preserving edge continuity and gradient orientation.

7.5 I/O Dissipation vs. System-on-Chip (SoC) Integration

The power profile in Table 3C illustrates that off-chip pad drivers dominate dynamic power (83.0%). In a production deployment, this accelerator would not drive 45 external package pins; rather, it would be wrapped with standard AXI4-Stream slave and master interfaces, connecting directly to an on-chip DMA controller or video processing subsystem inside the Zynq Programmable Logic. In that configuration, I/O pad dissipation is completely eliminated, reducing total active system power to approximately 10 mW.

8 Limitations and Threats to Validity

While the accelerator demonstrates high performance and area efficiency, several structural limitations must be acknowledged:

1. **Kernel Topology Constraint:** The hardware datapath is specialized for 3×3 kernels. While image width, height, and data widths are parameterizable, modifying the kernel dimension (e.g., to 5×5 or 7×7) requires modifying the Verilog source to instantiate additional line buffers and multiplier stages.
2. **Single-Channel Processing:** The architecture is currently designed for single-channel (grayscale) 2D streams. Extending the core to multi-channel 3D tensor convolutions (typical of intermediate CNN feature maps where $C_{in} > 1$) would require replicating the 2D datapath across channels or implementing an inter-channel accumulation stage.
3. **Boundary Contraction (Valid Mode):** Convolution operates exclusively in valid mode, reducing the spatial dimension by two pixels per axis without boundary padding. Applications requiring same-dimension output feature maps would require an external hardware border-padding module.
4. **Open-Loop Stream Ingestion:** The pixel interface does not implement a back-pressure handshaking mechanism (e.g., an AXI `ready` signal). The accelerator assumes the input source streams data without stalling once `start` is triggered. If the producer pauses, valid qualification must be maintained explicitly.
5. **Post-Synthesis Estimation:** The timing and power metrics reported are based on post-synthesis static timing analysis and Vivado power models. Post-implementation place-and-route delays and real-world physical thermal measurements may introduce slight variations.

9 Conclusion and Future Work

In this paper, we presented an ultra-compact, BRAM-free streaming 2D convolution accelerator for low-power edge FPGAs. By cascading dual line buffers mapped onto 16 Look-Up Table shift-register primitives (SRLC32E), the architecture reconstructs 3×3 spatial neighbourhoods on the fly with zero Block RAM utilization. The datapath leverages nine parallel DSP48E1 multipliers and a mathematically verified 20-bit accumulator that guarantees zero intermediate overflow, coupled with monotonic 16-bit saturation clamping and optional ReLU activation.

Synthesized on a Xilinx Zynq-7020 SoC at 100 MHz, the design consumes only 231 Slice LUTs, 417 flip-flops, 9 DSP slices, and 0 Block RAM tiles, achieving an estimated $f_{\max}$ of 159.3 MHz and consuming only 9 mW of active compute power. The accelerator delivers an initial latency of $2W + 7$ cycles (71 cycles for 32×32 imagery) and sustains a peak throughput of one pixel per clock cycle (100 Mpixel/s, 900 MMAC/s). Rigorous multi-tier verification demonstrates 100% bit-exact equivalence (900/900 samples) against an independent 32-bit software golden model on natural photography.

Future work includes integrating standard AXI4-Stream interfaces for plug-and-play SoC integration, extending the datapath to multi-channel depthwise-separable convolutions, and pipelining the adder tree via DSP cascade chains to achieve operating frequencies exceeding 250 MHz.

References

- [1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations (ICLR)*, 2015, arXiv:1409.1556.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [5] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, X. Ning, C. Yuan, and Y. Wang, “Going deeper with embedded FPGA platform for convolutional neural network,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA ’16)*. ACM, 2016, pp. 26–35.
- [6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA ’15)*. ACM, 2015, pp. 161–170.
- [7] A. Podili, C. Zhang, and V. K. Prasanna, “Fast and efficient implementation of convolutional neural networks on FPGA,” in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 11–18.
- [8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
- [9] H.-J. Kang, “AoCStream: All-on-chip CNN accelerator with stream-based line-buffer architecture,” *arXiv preprint arXiv:2212.11438*, 2022.
- [10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.
- [11] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. John Wiley & Sons, 2011.
- [12] S. Perri, P. Corsonello, and G. Cocorullo, “A high-performance fully reconfigurable FPGA-based 2D convolution processor,” *Microprocessors and Microsystems*, vol. 29, no. 6, pp. 281–291, 2005.
- [13] R. Shi, J. S. J. Wong, and H. K.-H. So, “High-throughput line buffer microarchitecture for arbitrary sized streaming image processing,” *Journal of Imaging*, vol. 5, no. 3, p. 34, 2019.
- [14] Xilinx, Inc., *7 Series FPGAs Configurable Logic Block User Guide (UG474)*, Xilinx, Inc., 2016, v1.8.
- [15] —, *7 Series DSP48E1 Slice User Guide (UG479)*, Xilinx, Inc., 2018, v1.10.