AN ENERGY EFFICIENT RISC-V BASED HARDWARE ACCELERATOR FOR CNN INFERENCE ON LOW POWER EDGE DEVICES.

A dissertation submitted in partial fulfillment of the requirement for the degree of

Master of Technology (M.Tech)

Submitted by,

AILA TEJ RITHVIK

244102401

Under the Supervision of of

Prof. Roy Paily Palathinkal

Department of Electronics and Electrical Engineering

Indian Institute of Technology, Guwahati

November, 2025

ABSTRACT

Convolutional Neural Networks (CNNs) have become foundational in modern computer vision tasks due to their ability to extract and process hierarchical visual features. However, deploying CNNs on resource-constrained edge platforms presents significant challenges in terms of computational load and energy efficiency. This work presents a custom hardware accelerator built around the open-source RISC-V architecture, specifically optimized for CNN inference on low-power edge devices. The proposed design leverages Parallel Processing, Deep Pipelining, Streaming Architecture and Data Reuse Strategies to improve throughput and reduce External Memory Bandwidth. The Accelerator is accessable via a custom instruction set. The results demonstrate a significant improvement in performance with the use of these custom instructions during convolution operations, which typically account for up to 90 percent of the total computation workload in CNNs. Such enhancements make the design well-aligned with the performance and energy requirements of emerging edge AI workloads.

Contents

1 Introduction

1.1 Background and Motivation

The rapid advancement of deep learning has revolutionized computer vision, natural language processing, and other data-driven applications. Among these, Convolutional Neural Networks (CNNs) have emerged as a dominant architecture due to their ability to automatically learn hierarchical visual features from data. However, the computational and memory demands of CNNs pose significant challenges when deploying such models on edge devices with limited resources.

Edge platforms—such as IoT nodes, drones, and mobile devices—require efficient hardware solutions capable of maintaining acceptable inference accuracy and latency while minimizing power consumption. This motivates the exploration of custom accelerators that can deliver high performance per watt compared to traditional CPUs and GPUs.

1.2 Convolutional Neural Networks for Edge Inference

CNNs consist of multiple layers that perform operations such as convolution, pooling, and activation, progressively extracting higher-level representations from input data. While effective, these operations are computationally intensive, often requiring billions of multiply-accumulate (MAC) operations for even moderate-sized networks.

Deploying CNNs at the edge introduces constraints such as limited on-chip memory, restricted bandwidth, and low energy budgets. Therefore, hardware designs must exploit model-level optimizations—such as quantization, pruning, and reduced precision arithmetic—alongside architectural techniques like data reuse and parallelism to achieve real-time performance.

1.3 Need for Energy-Efficient Hardware Acceleration

Traditional processors are not optimized for the massive data movement and parallel computation required by CNNs, leading to inefficiencies in both speed and power usage. Hardware accelerators, on the other hand, can be tailored to the specific computation patterns of CNNs, offering significant improvements in energy efficiency.

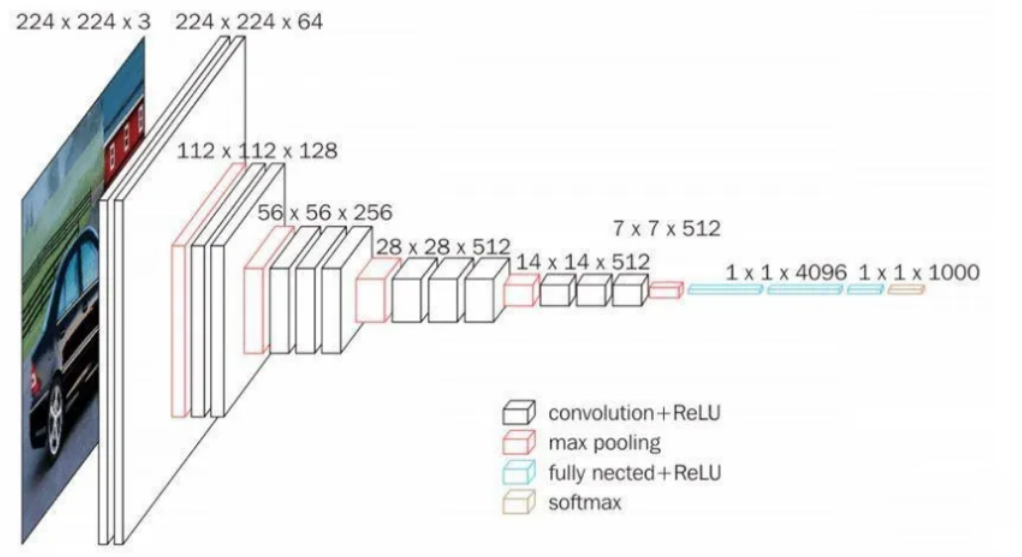

Figure 1: CNN Architecture (VGG-16) [1]

1.4 Computational Complexity of Convolutional Neural Networks

The computational cost of a convolutional layer depends on the number of input channels, output channels, filter size, and the spatial resolution of the feature maps. Let the input feature map be represented as:

$$I \in \mathbb{R}^{C \times H \times W},$$

where C is the number of input channels, and $H \times W$ denotes the height and width of the feature map.
A convolutional kernel is represented as:

$$K \in \mathbb{R}^{N \times C \times S \times S},$$

where N is the number of output channels (or filters) and $S \times S$ is the spatial filter size. The resulting output feature map is:

$$O \in \mathbb{R}^{N \times H \times W}.$$

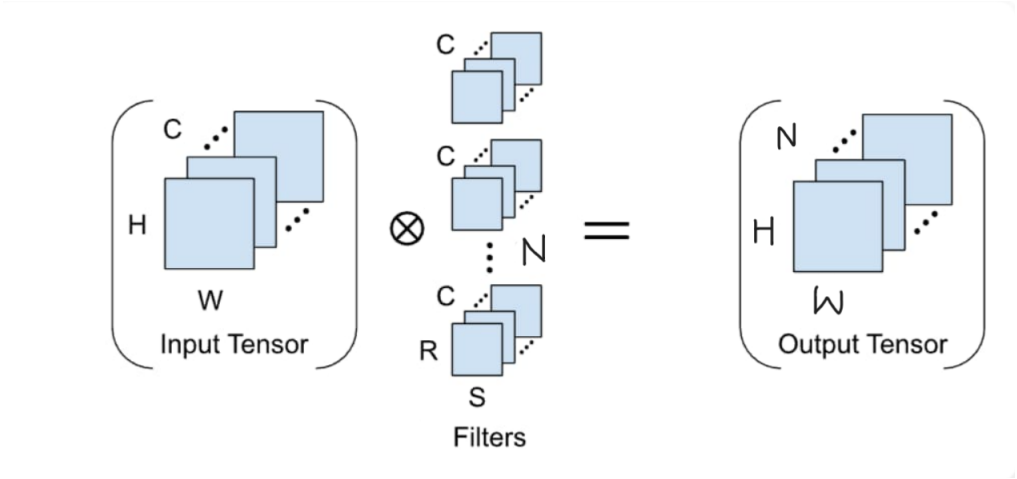

Figure 2: Tensor Convolution

Operation Count

Each filter performs a weighted sum across all C input channels over an $S \times S$ region. Therefore, the total number of multiplication–accumulation (MAC) operations for one output position is:

$$\text{MACs per position} = C \times S \times S.$$

Since this is repeated for every output pixel and every filter, the total computational complexity becomes:

$$\text{Total MACs} = H \times W \times C \times S^2 \times N.$$

This shows that convolutional layers scale linearly with the number of input channels, output channels, and spatial dimensions, but quadratically with the filter size.

Fully Connected Layers as 1×1 Convolutions

A fully connected layer can be interpreted as a convolution with filter size 1×1. In this case:

$$K \in \mathbb{R}^{N \times C \times 1 \times 1}, \qquad O \in \mathbb{R}^{N \times H \times W}.$$

Thus, the total complexity reduces to:

$$\text{MACs} = C \times N \times H \times W,$$

highlighting that fully connected layers are a special case of convolution with no spatial aggregation.

Effect of Stride and Batch Size

When a stride of s is used, the spatial dimensions of the output reduce to:

$$H' = \frac{H}{s}, \quad W' = \frac{W}{s}.$$

Consequently, the total number of operations becomes:

$$\text{MACs} = H' \times W' \times C \times S^2 \times N.$$

For a batch size of B input images, the overall complexity is:

$$\text{Total Complexity} = B \times H' \times W' \times C \times S^2 \times N.$$

The computational burden of CNNs primarily arises from convolutional layers. The operation count depends on spatial resolution, filter size, number of channels, and number of filters.

1.5 RISC-V Instruction Set Architecture

RISC-V, an open-source Instruction Set Architecture (ISA), offers a promising platform for developing specialized and energy-efficient hardware accelerators. Its modularity, extensibility, and reduced licensing costs make it highly suitable for custom hardware design. By integrating application-specific instructions and hardware modules tailored for CNN computations, RISC-V-based accelerators can enhance performance while maintaining energy efficiency.

RV32I Instruction Format

The RV32I base instruction set defines a fixed 32-bit instruction length and supports multiple instruction formats. Each instruction is composed of specific fields that determine the operation to be performed, the registers involved, and any immediate values required. The commonly used formats include R-type, I-type, S-type, B-type, U-type, and J-type, each serving a particular class of operations.

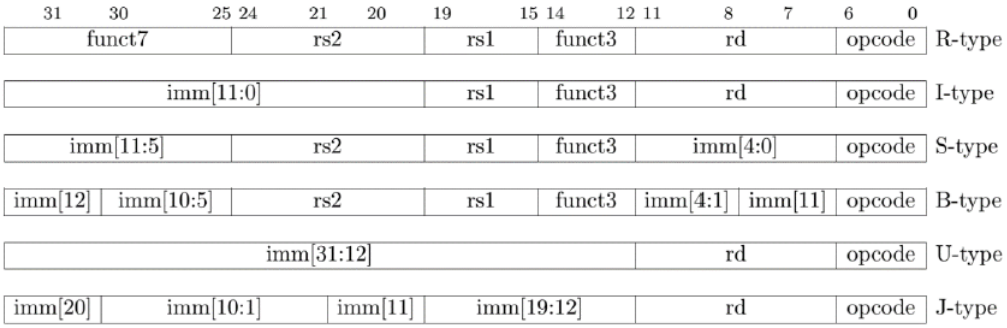

Figure 3: RISC-V ISA [2]

The key fields used in the RV32I instruction formats are:

- **opcode**: Indicates the operation category (e.g., arithmetic, load/store, branch).
- **rd**: Specifies the destination register where the result of the instruction is written.
- **rs1 and rs2**: Identify the source registers that provide input operands.
- **funct3**: Distinguishes variations of instructions sharing the same opcode.
- **funct7**: Provides additional encoding bits for specific arithmetic and logical instructions.
- **Immediate (imm)**: Represents constant values directly encoded within the instruction, used for arithmetic, addressing, or branch offset calculations.

2 Literature Survey

2.1 RISC-V Architecture in Edge AI

RISC-V has emerged as a popular choice for edge AI platforms due to its open-source nature, modular instruction set, and flexibility for hardware customization. Unlike proprietary architectures, RISC-V allows researchers to integrate custom instructions and co-processors tailored for AI workloads, particularly convolutional neural networks (CNNs). Studies have shown that lightweight RISC-V cores augmented with custom MAC (Multiply-Accumulate) units or vector extensions can significantly reduce computational latency while keeping power consumption within edge device constraints. Additionally, memory access patterns can be optimized using tightly coupled memory or scratchpad buffers to minimize DRAM usage, which is a major source of energy consumption. This adaptability makes RISC-V an ideal foundation for implementing domain-specific accelerators in low-power embedded systems.

2.2 CNN Acceleration Techniques

CNNs are computationally intensive due to the large number of convolution operations and frequent memory transfers. To address this, several hardware acceleration strategies have been proposed. Common techniques include data reuse through systolic arrays, loop unrolling, and tiling to enhance parallelism and reduce redundant memory access. Quantization is another widely adopted approach where floating-point values are converted to fixed-point representations, significantly decreasing both memory footprint and power consumption with minimal accuracy degradation. Pruning eliminates redundant weights and neurons, enabling smaller and more efficient networks. Other optimizations include pipelined architecture design, on-chip buffer usage, and sparsity exploitation to further speed up inference. Together, these methods contribute to developing CNN accelerators that are faster, energy-efficient, and suitable for real-time edge deployment.

- **Winograd Algorithm:**[3] Winograd minimal filtering can be applied to accelerate convolution operations in CNNs. For commonly used 3×3 kernels, this method reduces the number of arithmetic operations required. Analytical studies indicate that Winograd-based convolution can lower the computational complexity by approximately a factor of 2.25, resulting in faster inference with reduced hardware resource usage.

- **Layer Fusion:**[4] By merging the convolution and pooling operations within the accelerator, the intermediate write-back and subsequent memory-pooling read steps can be eliminated. Performing pooling directly after convolution inside the acceleration module reduces the overall processing latency and significantly lowers the memory access overhead, since intermediate feature maps no longer need to be stored and fetched from external memory.

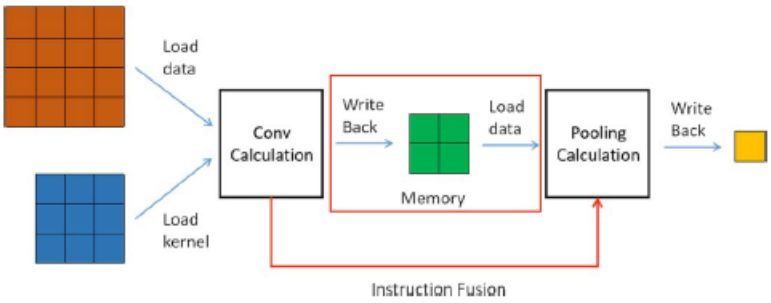

Figure 4: Layer fusion

- **Quantization** Fixed-point quantization, particularly 8-bit, is widely adopted in RISC-V accelerators due to its balance of accuracy and efficiency. Mixed-precision arithmetic (8/4/2-bit) further reduces power while maintaining accuracy on critical layers.

2.3 Existing Hardware Implementations

- **Eyeriss** [5]: Pioneered the Row-Stationary (RS) dataflow within a 12×14 spatial array to maximize on-chip data reuse (weights/inputs/partial sums) in Register Files, significantly reducing the energy overhead associated with global memory access (achieving up to 2.5× energy improvement).

- **GreenWaves GAP8/9**[6]: The GAP8 and GAP9 architectures integrate a RISC-V core with an 8/9-core cluster and a hardware convolution engine (HWCE). These platforms support 8-bit and 16-bit arithmetic, achieving over 20 GOPS while consuming tens of milliwatts, making them desirable for TinyML workloads such as keyword spotting and low-resolution vision tasks. The PULP-NN library provides optimized quantized kernels for enhanced performance.

- **Envision** [7]: Focused on extreme energy efficiency by implementing mechanisms for sparsity exploitation (handling compressed data) and supporting dynamic precision scaling (up to 16-bit fixed-point). This allows the system to compute only

non-zero values and adjust precision dynamically for optimal power use in edge applications.

- **Dustin**[8]: An ultra-low-power RISC-V accelerator that introduced the Vector Lockstep Execution Mode (VLEM). VLEM synchronizes the core cluster and enables power-gating the instruction fetch stage of follower cores, resulting in significant dynamic power reduction (up to $\approx 38\%$) during parallel DNN kernel execution.

- **Kendryte K210**[9]: The Kendryte K210 integrates dual RISC-V cores and a dedicated neural processing unit (KPU). It delivers approximately 0.5–0.8 TOPS on 8-bit quantized CNNs with a typical power budget below 300 mW. The nncase toolchain enables conversion of conventional CNN models to the device's supported kmodel format for fast deployment.

- **SamurAI** [10]: It introduces a unified processing platform built on the RISC-V ISA. Utilizes a heterogeneous architecture featuring an ultra-low-power Always-Responsive (AR) component for continuous event monitoring and an On-Demand (OD) ML accelerator (up to 36 GOPS). This partitioning drastically improves the system's idle power efficiency, extending battery life in IoT end-nodes.

- **XpulpNN**[11]: Integrated specialized SIMD Instruction Set Architecture (ISA) extensions into the RISC-V core pipeline to efficiently handle multi-precision arithmetic, specifically for heavily Quantized Neural Networks (QNNs) (e.g., 4-bit and 2-bit operations), balancing software flexibility with near-ASIC efficiency.

2.4 CNN Accelerator Architectures

- **Systolic Arrays** Systolic arrays are 2D grids of MAC units optimized for dense matrix operations. They achieve high throughput and local data reuse but may suffer from underutilization for sparse or irregular CNNs.

- **Spatial/Dataflow Architectures** Architectures like Eyeriss[5] adopt a row-stationary dataflow to maximize reuse of weights, activations, and partial sums. Eyeriss v2 introduces support for sparse data and achieves over 2500 inferences per joule for MobileNet workloads.

- **Sparsity-Aware Accelerators** Sparsity-aware designs (e.g., SPOTS[12]) exploit zero-skipping and compressed storage formats to reduce unnecessary computations. Such designs achieve up to 1.7× speedup and significant energy savings compared to dense accelerators.

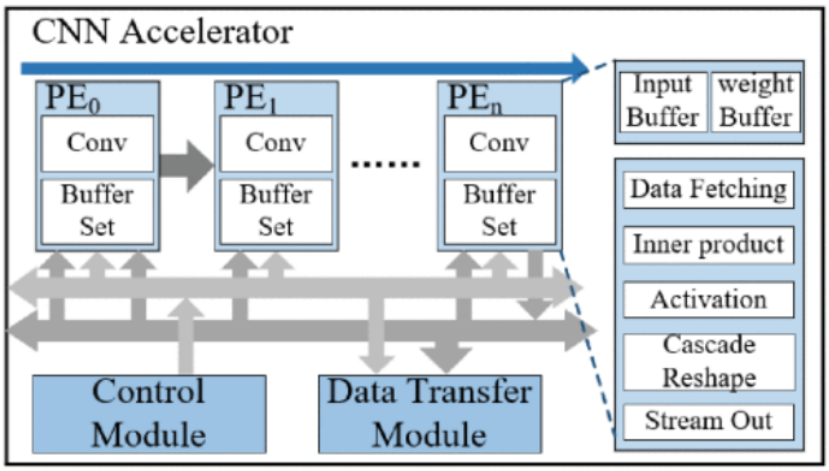

Figure 5: typical CNN accelerator architecture[13]

- **Inter-Macro Pipeline:**[13] The architecture establishes pipelining across multiple compute macros so that intermediate results can be forwarded directly, allowing different computation stages to proceed simultaneously without waiting for previous operations to finish.
- **Near-Memory Pipeline:**[13] Memory access operations are overlapped with computation. By preparing data closer to the compute units in advance, the design reduces stalls caused by memory latency and ensures a smoother data supply to the processing elements.
- **Tile-Level Nested Pipeline:**[13] Input feature maps are divided into tiles, and pipelining is applied across tile loading, computation, and output write-back. This hierarchical scheduling increases data reuse and enables higher throughput while avoiding unnecessary memory transfers.

2.5 Memory and Dataflow Optimization

Data movement is the primary source of energy in CNN accelerators. Thus, memory hierarchy optimization is critical. Typical solutions include multi-level scratchpad memories, direct memory access (DMA), tiling strategies, and multicast-based on-chip interconnects. RISC-V platforms like GAP8[6] provide compiler-level support for scheduling data transfers to minimize off-chip DRAM access.

2.6 RI5CY Processor

The RI5CY is an open source RISC-V processor that uses a four-stage pipeline design, consisting of four stages: Instruction Fetch (IF), Instruction Decode (ID), Instruction Execute (EX), and Write-Back (WB) stages.

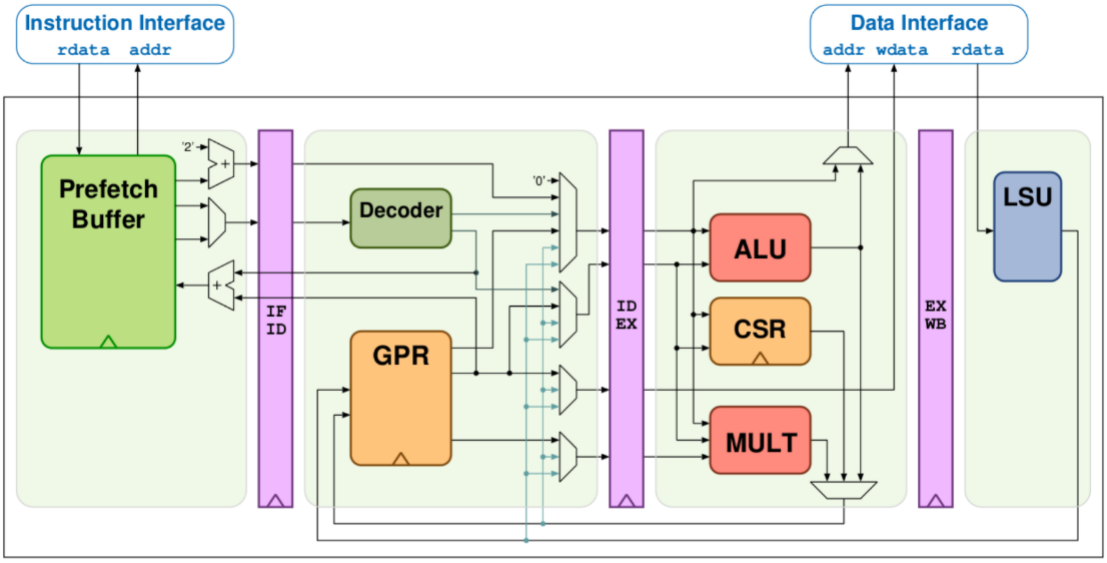

Figure 6: Micro Architecture of RI5CY processor[14]

The RI5CY core is a compact 32-bit RISC-V processor designed within the PULP (Parallel Ultra-Low Power) framework for energy-constrained computing. It follows an in-order, 4-stage pipeline and supports the RV32IMC instruction set with additional DSP-style extensions such as hardware loops and single-cycle MAC operations, which make it highly suitable for embedded CNN and signal processing tasks.

A key advantage of RI5CY is its low-power design philosophy. Techniques such as clock gating, small register files, and tightly-coupled instruction/data memory allow the core to operate in the milliwatt range under load and in the microwatt range during sleep modes. Its open-source availability enables researchers and industry designers to modify, extend, and integrate it alongside custom accelerators in edge-AI SoCs. Due to its minimal area, high efficiency, and customizability, RI5CY is widely used in edge devices where performance per watt is a critical metric.

3 Motivation and Problem Formulation

Memory Bottleneck and Need for Efficient Computing Platforms

Although the computational workload of CNNs is dominated by multiply-and-accumulate (MAC) operations, the overall system energy is often dictated by memory access rather than arithmetic. Accessing off-chip DRAM can consume up to two orders of magnitude more energy compared to an on-chip register or cache access. During convolution operations, the same feature map values and filter weights are reused hundreds of times across different spatial locations and channels. If these values are repeatedly fetched from external memory, it leads to severe memory bandwidth saturation, increased latency, and a drastic rise in energy consumption. This condition is commonly known as the *memory wall*.

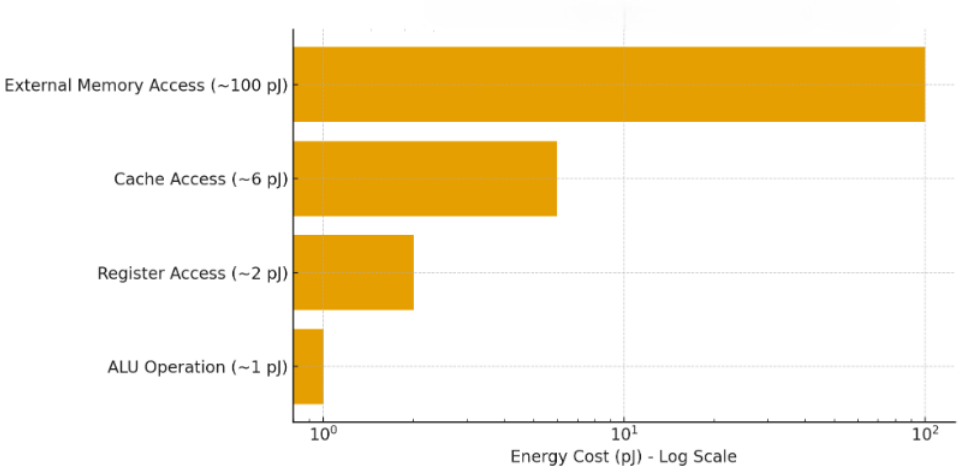

Figure 7: Approximate Energy Cost per Operation

To counter this, on-chip memory such as SRAM buffers or scratchpads is used to store frequently accessed data. By keeping feature maps and weights closer to the processing elements, redundant DRAM transfers are minimized. Techniques such as tiling, loop unrolling, and dataflow scheduling (e.g., weight-stationary or output-stationary) ensure that data is reused across multiple computations before being replaced. This improves both throughput and energy efficiency, since computation units remain active rather than stalling due to memory delays.

Limitations of Conventional CPU and GPU Platforms

General-purpose CPUs are not optimized for the highly parallel and repetitive nature of CNN workloads. They typically have limited parallel execution units and spend significant cycles fetching and decoding instructions, resulting in low throughput for deep learning

tasks. As a result, real-time CNN inference on low-power CPUs becomes inefficient and slow.

GPUs, on the other hand, offer massive parallelism with thousands of cores and high computational throughput, making them well-suited for training and inference in data centers. However, they consume a substantial amount of power, require high-bandwidth memory, and involve complex cooling mechanisms. Their large silicon area and general-purpose design make them unsuitable for deployment in energy-constrained environments such as IoT sensors, drones, and wearable devices.

This creates a clear motivation for designing specialized hardware accelerators. A RISC-V-based accelerator provides a balance between flexibility and efficiency by enabling custom instructions, optimized dataflows, and tightly coupled on-chip memory. Such architectures can significantly reduce memory traffic, exploit data reuse, and deliver high performance per watt, making them ideal for CNN inference on edge devices.

3.1 Problem Formulation

The key objective of this work is to design and develop an energy-efficient RISC-V-based hardware accelerator capable of performing CNN inference on low-power edge devices with minimal energy consumption and high throughput. The problem can be formulated as follows:

- **Input:** Pre-trained CNN models (with 3x3 filter size, which is common for most of the modern CNN models) and real-time input data streams (such as images or sensor data).
- **Goal:** Execute CNN inference efficiently with optimized latency, throughput, and power consumption while preserving prediction accuracy.
- **Constraints:**
  - Limited memory bandwidth and on-chip storage.
  - Restricted energy budget and thermal dissipation.
  - Hardware design complexity and compatibility with the RISC-V ISA.
  - The Critical path delay should not be more than the RISC-V Processor.
- **Optimization Problem:** Minimize the total energy consumption E and inference latency T while maximizing throughput τ, subject to hardware resource limitations and accuracy thresholds.

4 Proposed Architecture and Work Progress

A 32-bit fixed-point number format is adopted in the design, as it offers a substantial reduction in hardware complexity and power consumption compared to floating-point representation, while introducing only a minimal loss in accuracy.

4.1 Acceleration Module

The proposed accelerator is designed to handle multi-channel and multi-filter parallelism efficiently.

The accelerator consists of the following major components:

- **Control Unit** – Manages instruction flow, synchronization, and overall operation of the accelerator.
- **Data Buffers** – Temporary on-chip memory used to store input feature maps, filter weights, and partial sums to enable data reuse.
- **Address Calculation Unit** – Generates memory addresses for input, weight, and output data, supporting efficient data movement.
- **Processing Element (PE) Array** – Performs parallel MAC (Multiply-Accumulate) operations across multiple channels and filters.

Micro-Architecture

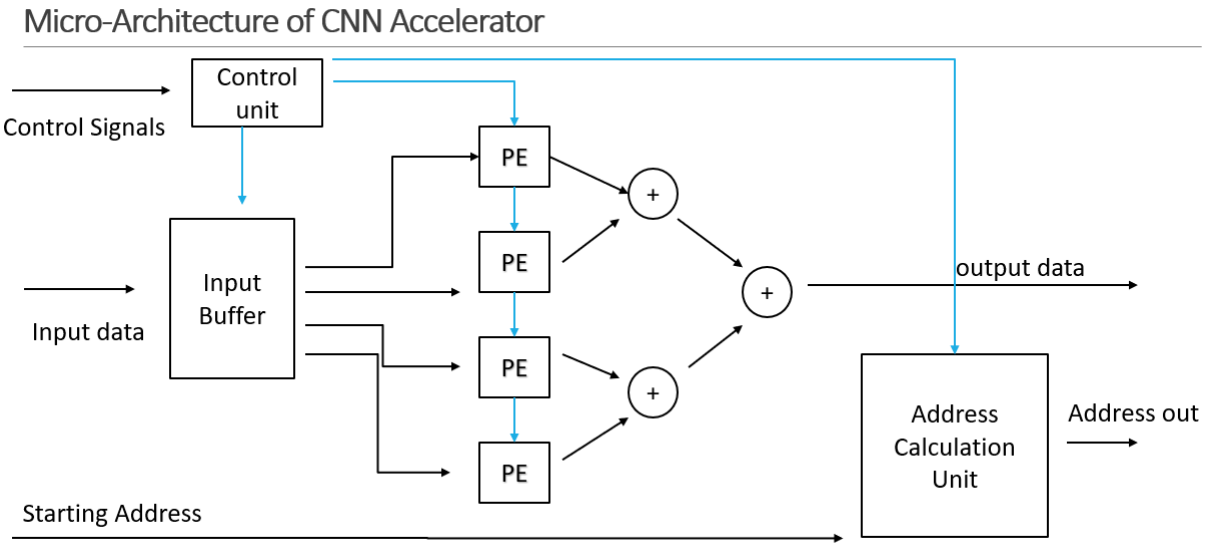

Figure 8: CNN Accelerator Micro-Architecture

4.1.1 Data Buffers

- Since only one read/write operation can occur at a time from the external memory, all Processing Elements (PEs) are synchronized by the control unit. As there are four PEs operating in parallel, the required input data is first fetched and stored in the data buffers. From these buffers, data is then synchronously supplied to each PE for computation.

- Likewise, the outputs from all PEs are generated simultaneously due to synchronization. Therefore, the results are temporarily stored in output data buffers. From these buffers, the data is sequentially written back to the external memory, ensuring no read/write conflicts and maintaining timing correctness.

4.1.2 Control Unit

- The Control Unit synchronizes the operation of all major modules, including the Address Calculation Unit (ACU), data buffers, and Processing Elements (PEs). It ensures that data read, computation, and write operations proceed in a well-defined sequence without conflicts.

- It manages the interleaving of read and write phases by issuing enable and stall signals at precise clock cycles. This prevents data hazards and ensures continuous data flow through the accelerator pipeline.

4.1.3 Address Calculation Unit (ACU)

- The ACU generates memory addresses for both input feature maps and weight tensors during read operations, as well as output feature map locations during write-back. Since tensors are stored linearly in memory, the ACU performs index arithmetic to traverse multidimensional data structures.

- It works in close coordination with the Control Unit by providing pre-computed or on-the-fly addresses, enabling the PEs to receive data without stalls. This ensures maximum utilization of the compute array while minimizing unnecessary memory access overhead.

4.1.4 Processing Element

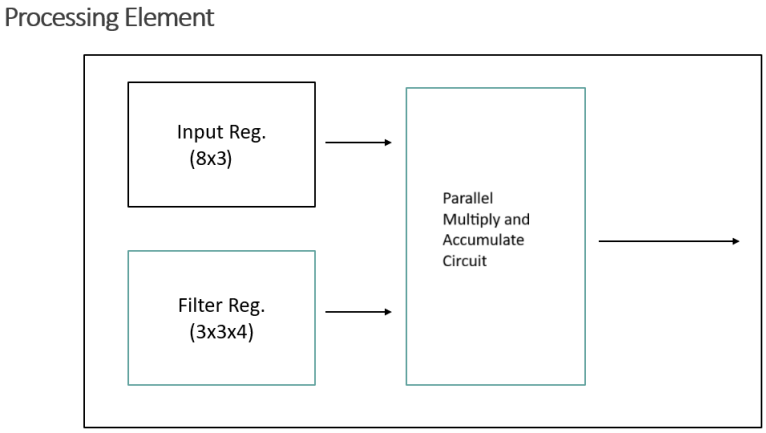

Figure 9: Processing Element Micro-Architecture

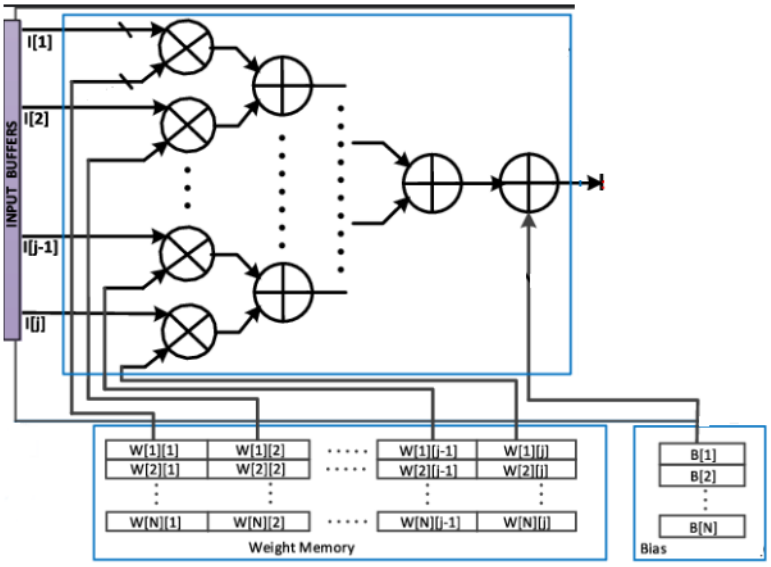

Figure 10: Mutliply-Accumulate Unit

Parallel Multiply and Accumulate operation on a 3x3 filter and 3x3 section of the input data. Each PE handles one channel. There are 4 PEs in total, so 4 channels are handled in parallel.

Data Reuse Concept

It can reduce the number of Load operations from External Memory by a factor of 9. This can be achieved by storing elements as shown below.

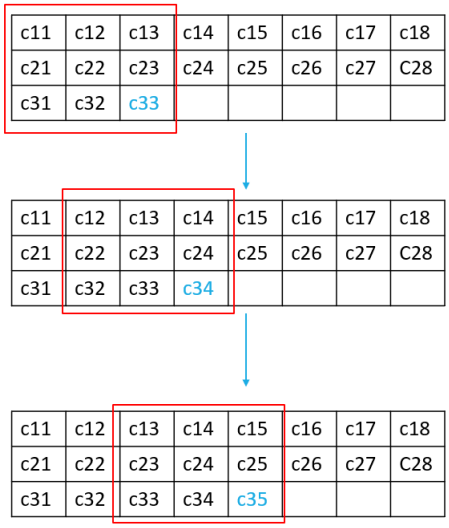

Figure 11: Data Reuse for Memory Bandwith reduction

Efficient Memory Utilization

Since the filter size is fixed to 3x3, maximum of 3 rows needed to be stored at a time. By efficiently organizing the incomming data, on-chip memory requirement can be reduced.

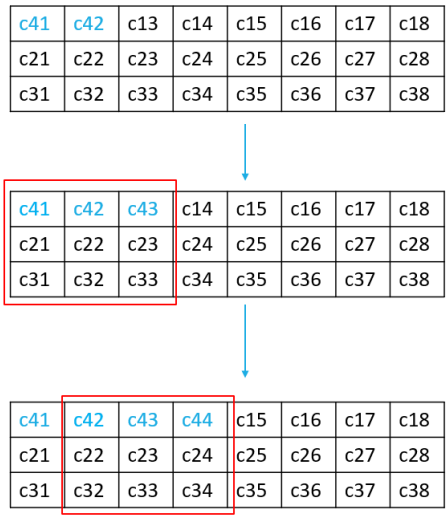

Figure 12: Efficient On-chip Memory utilization by replacing un-used elements

4.2 Integration with RISC-V

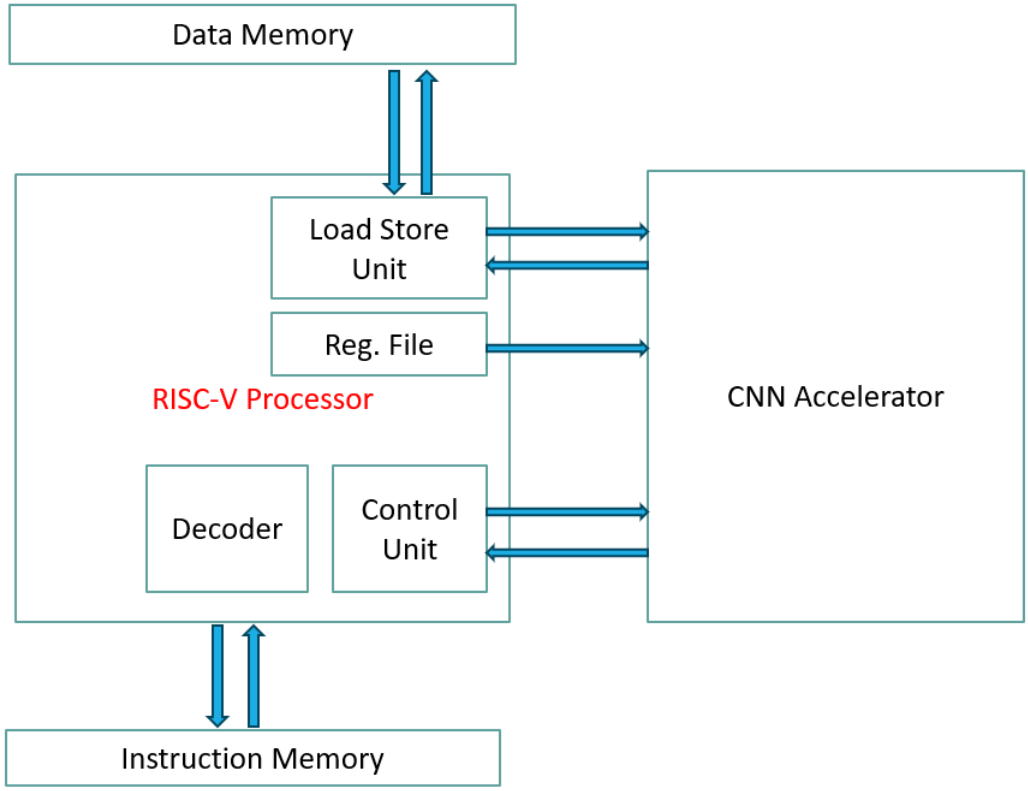

Figure 13: Integration with RISC-V

To interface the custom CNN Accelerator with the RISC-V core, minor modifications are required in the processor microarchitecture. The key modules affected are the Decoder, Control Unit, and Load-Store Unit, as summarized below:

- **Decoder Modification:** The instruction decoder is extended to recognize the newly added custom accelerator instructions. These instructions indicate when the processor should invoke the accelerator instead of executing the operation in the ALU datapath.
- **Control Unit Modification:** The control unit coordinates the execution between the processor and the accelerator. When an accelerator instruction is issued, the control unit temporarily stalls the processor pipeline to prevent register or memory hazards while the accelerator completes its computation.
- **Load-Store Unit Modification:** The load-store unit is updated to support data transfer between the register file, system memory, and the accelerator. This ensures that input feature maps, weights, and output feature maps are correctly fetched and written back in a synchronized manner.

4.3 Custom Instructions for CNN Accelerator

To efficiently interface the RISC-V core with the CNN accelerator, two custom instructions were developed. These instructions enable seamless transfer of filter weights, feature map data, and control parameters to the accelerator for hardware-accelerated convolution operations. The values related to the size of the image, number of input channels, number of output channels, and total filter size are preloaded into RISC-V general-purpose registers. The custom instructions make use of the `rs1` and `rs2` fields to pass base addresses or configuration data to the accelerator.

Instruction 1: Load Weights

This instruction is responsible for loading filter weights from memory into the accelerator's internal registers or buffer.

- **rs1**: Base address of the weight (filter) matrix in main memory.
- **rs2**: Encodes configuration parameters such as:
  - Image size (height × width)(10 bits),
  - Total number of input channels(10 bits),
  - Total number of output channels(10 bits).
- **Opcode**: `x77`
- **funct3**: `100`
- **funct7**: `0000_001`

On execution, the accelerator fetches the filters from the address specified in `rs1` and stores them into its internal weight registers based on the configuration provided through `rs2`.

31	**25-24**	**20-19**	**15-14**	**12-11**	**7-6**	**0**
	funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
LD_FILTER	0000001	address	address	100	/	0x77

Table 1: Instruction Format: LD_FILTER

Instruction 2: Convolution Operation

This instruction initiates the convolution process by fetching input feature maps, performing convolution using preloaded filters, and writing results back to memory in an interleaved manner.

- **rs1**: Base address of the input feature map (activation matrix).
- **rs2**: Base address of the output feature map in memory.
- **Opcode**: `x77`
- **funct3**: `100`
- **funct7**: `0000_001`

Upon execution, the accelerator:

1. Fetches a block of input data from the address in `rs1`.
2. Performs convolution using the filters loaded by Instruction 1.
3. Writes the computed output feature map to the memory address specified in `rs2`.
4. This process is carried out in an interleaved load–compute–store fashion to maximize throughput and reduce idle cycles.

These custom instructions enable tight integration between the RISC-V processor and the CNN accelerator, reducing data movement overhead and improving overall performance during inference.

31	**25-24**	**20-19**	**15-14**	**12-11**	**7-6**	**0**
	funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
CONV_8x8x4x4	0000010	address	address	100	/	0x77

Table 2: Instruction Format: CONV_8x8x4x4

Additionally a special Reset instruction was also added which resets the Accelerator.

31	**25-24**	**20-19**	**15-14**	**12-11**	**7-6**	**0**
	funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
ACC_RST	0000011	/	/	100	/	0x77

Table 3: Instruction Format: ACC_RST

Custom Instruction Functionality

- **Weight Loading:** The accelerator begins by loading the weight data through LD_FILTER instruction. These instructions transfer 4 channels and 4 filters, each of size 3×3, and store them into dedicated internal registers for high-speed access during computation.

- **Convolution Execution:** Once the weights are loaded, CONV_8x8x4x4 instruction is issued. This instruction performs multi-channel and multi-filter convolution, processing an 8×8 image stride with 4 channels and 4 filters simultaneously. The operation employs interleaved read and write access to memory, thereby maintaining a continuous computation pipeline.

- **Processor Coordination:** Higher-level control tasks such as memory initialization and configuration are managed by the RISC-V processor. The processor issues the custom instructions, monitors completion flags, and orchestrates the sequence of accelerator operations.

4.4 Software and Simulation Tools Used

- **Verilator** is used for simulating the hardware design. It converts the Verilog RTL code into a cycle-accurate C++ simulation model, enabling high-speed functional verification and debugging of the accelerator without requiring commercial simulation tools.
- **RISC-V GNU Toolchain** is used to compile C programs into RISC-V compatible assembly and machine code. The toolchain ensures that the generated code follows the RV32IMC instruction set supported by the processor core used in the design.
- **Program Flow:**
  - Application code is written in C.
  - Custom Instructions are added using In-Line Assembly feature, written direclty in C code itself .
  - The RISC-V compiler converts the C program into assembly and then into executable machine code.
  - The generated machine code (or memory image) is loaded into the instruction memory of the design.
  - Verilator runs the simulation, where the RISC-V core fetches and executes instructions from the instruction memory.
  - Outputs and behavior are observed to validate functionality and performance.

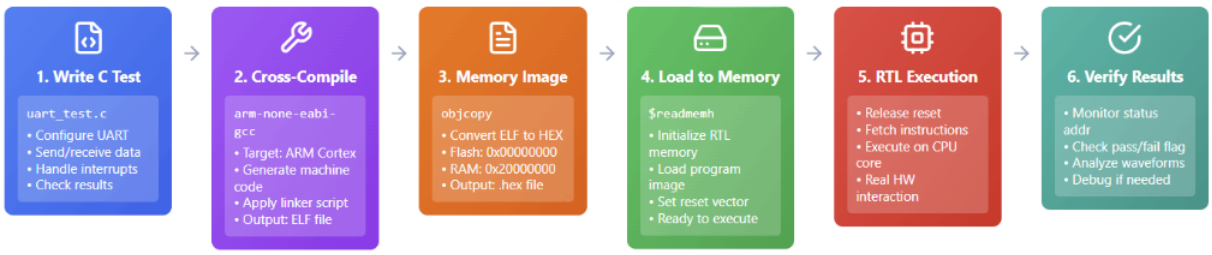

Figure 14: Software-Driven Verification Flow : C Code to RTL Execution [15]

4.5 Results and Performance Comparison

The following results are for integer (32-bit) number format. Hardware complexity is very similar to Fixed point(32-bit). Interger representation is chosen as it easier to verify the working of accelerator using C program as Fixed point representation is not implicitly declared in C language.

Note: Here accuracy refers to comparison between output values with and without custom instructions.

- **Input Image Size:** 8×8, 4 channels
- **Filter Size:** 3×3, 4 channels, 4 filters
- **Output Image Size:** 6×6, 4 channels
- **Speed-up:** 95.46×

```
Total Cycles with acelerator: 750
Total Cycles with acelerator only channael addition part : 8
Total Cycles without accelerator: 71597
Total matched cases: 144
Total mismatched cases: 0
Total accuracy: 100.000000 percent
Total speed-up: 95.462669 x
```

Figure 15: Results with minimal input sizes

- **Input Image Size:** 32×32, 32 channels
- **Filter Size:** 3×3, 32 channels, 32 filters
- **Output Image Size:** 30×30, 32 channels
- **Speed-up:** 25.55×

```
[tb_top_verilator] finished dumping memory
Total Cycles with acelerator: 3363533
Total Cycles with acelerator only channael addition part : 2198767
Total Cycles without accelerator: 85949300
Total matched cases: 28800
Total mismatched cases: 0
Total accuracy: 100.000000 percent
Total speed-up: 25.553278 x
0x2fefc0 , 0x2d9dc0 , 0x31efc0, 33587232:
TOP.tb_top_verilator @ 920004330: EXIT SUCCESS
- /root/core-v-verif/cv32e40p/tb/core/tb_top_verilator.sv:83: Verilog $finish
root@Laptop22:~/core-v-verif/cv32e40p/sim/core#
```

Figure 16: Results with Moderate input sizes and moderate number of filters and channels

- The performance improvement in throughput is evaluated by executing tensor convolution operations in two scenarios:
  - Without using custom instructions, using only RISC-V processor.
  - With the custom instruction set integrated into the RISC-V processor.

- The total number of clock cycles required to complete the tensor convolution is measured in both cases.

- The performance improvement (speed-up) is then calculated using:

$$\text{Speed-up} = \frac{\text{Latency (No Custom ISA)}}{\text{Latency (With Custom ISA)}}$$

- All measurements are obtained through simulation of the hardware accelerator design.

Input Size	**Filters**	**Channels**	**Output Size**	**Latency (No ISA)**	**Latency(With ISA)**	**Speed-up**
8×8	4	4	6×6×4	71,597 (cycles)	750 (cycles)	95.46×
16×16	8	8	14×14×8	1,109,314	25,048	44.28×
16×16	16	16	14×14×16	3,865,955	89,607	43.14×
32×32	16	16	30×30×16	24,182,660	553,427	43.96×
32×32	32	32	30×30×32	85,949,300	3,363,533	25.55×

Table 4: Latency Comparison for CNN Operations With and Without Custom Instructions

4.6 Estimated Performance per Watt

The accelerator is evaluated at a fixed precision of 32-bit fixed-point arithmetic. At this configuration, the design achieves an estimated throughput of approximately 5 Giga MAC operations per second (GMAC/s) while consuming 34.9 mW of power during active computation. The corresponding performance-per-watt metric is calculated as:

$$\text{Peak Performance per Watt} = \frac{2.1 \times 10^9 \text{ MAC/s}}{34.9 \times 10^{-3} \text{ W}} \approx 60.7 \text{ GMAC/s/W}.$$

This high efficiency demonstrates the suitability of the proposed accelerator for deployment in low-power edge inference scenarios, where maximizing computational capability under strict power budgets is essential.

4.7 Synthesis Results

The following are the synthesis results for UMC 65nm Technology node using Synopsys (Design Compiler) Tool.

Area

```
Combinational area:            345305.879959
Buf/Inv area:                   11591.280405
Noncombinational area:         110464.921296
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:               455770.801255
```

Figure 17: Area report

Power

```
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
Conv_Acc                              0.574   34.299 3.22e+07   34.906 100.0
```

Figure 18: Power Report

Timing

```
--------------------------------------------------------------------------------
data required time                                                       3.23
data arrival time                                                       -3.20
--------------------------------------------------------------------------------
slack (MET)                                                              0.04
```

Figure 19: Timing Report

	Area (μm^2)	**Power (mW)**	**Critical Path Delay (ns)**
RI5CY	70,224	5.99	3.25
Accelerator	455,770	34.9	3.20

Table 5: Area, Power, and Timing Metrics

5 Conclusion and Future Work

Conclusion

In this phase of the project, the CNN accelerator was successfully designed and integrated with the RISC-V processor. The proposed architecture demonstrates a significant improvement in throughput and reduced memory bandwidth usage during integer tensor convolutions across varying input size, channels and filters. This was achieved through parallel processing, efficient data reuse, streaming dataflow, Direct Memory Access (DMA) support, and a weight-stationary computation strategy. While these enhancements contribute to higher performance and improved energy efficiency, they also introduce additional hardware complexity and area overhead. Overall, the implementation establishes a scalable foundation for further optimization and deployment in edge-based AI acceleration systems.

Limitations

- Throughput reduction was observed for a higher number of channels.
- Channel addition stage is becoming the bottleneck, as it is software-based and consumes more than 70% of the total clock cycles (for 32 channels).
- Adding another accelerator block dedicated to multi-channel tensor addition can improve overall performance.

Future Directions

- More channels processed in parallel result in fewer store operations. More filters processed in parallel lead to fewer load operations.
- Increasing parallelism (more Processing Elements) and using more on-chip memory improves energy efficiency, as it reduces frequent load-store operations from external memory.
- Use SRAM-based cache blocks instead of registers or flip-flops to storing intermediate data.
- Techniques such as layer fusion (e.g., Convolution + Max-Pooling) can further reduce memory access and improve performance.
- Use zero skip/bypass mechanism to minimize dynamic power consumption.

References

[1] R. G, "Everything you need to know about VGG16," Medium (Great Learning), Sep. 23, 2021. [Online]. Available: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. [Accessed: Nov. 07, 2025].

[2] RISC-V Foundation, "RISC-V Specifications (Ratified)," [Online]. Available: https://riscv.org/specifications/ratified/. [Accessed: Nov. 07, 2025].

[3] S. Wang, Y. Li, H. Chen, Q. Zhang, and J. Liu, "Optimizing CNN Computation Using RISC-V Custom Instruction Sets for Edge Platforms," *IEEE Transactions on Computers*, vol. 73, no. 5, pp. 1371–1384, May 2024, doi: 10.1109/TC.2024.3362060.

[4] X. Wang, C. Feng, X. Kang, Q. Wang, Y. Huang and T. T. Ye, "RV-SCNN: A RISC-V Processor With Customized Instruction Set for SNN and CNN Inference Acceleration on Edge Platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1567–1580, Apr. 2025, doi: 10.1109/TCAD.2024.3472293.

[5] Y. H. Chen *et al.*, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proc. ISCA*, 2016.

[6] F. Conti, D. Rossi, A. Pullini, I. Loi and L. Benini, "PULP: A Ultra-Low Power Parallel Accelerator for Ultra-Efficient Embedded Computing," *IEEE Design & Test*, 2016.

[7] M. Bojnordi and E. Ipek, "Envision: A 16×16 SIMD MAC Accelerator for Convolutional Neural Networks," in *IEEE Micro*, vol. 37, no. 3, pp. 24–32, 2017.

[8] L. Bartolini, G. Tagliavini, and A. Marongiu, "Dustin: A RISC-V Based Compute Cluster for Mixed-Precision Deep Learning," in *IEEE Trans. Computers*, vol. 71, no. 12, pp. 3000–3013, 2022.

[9] Kendryte K210 Datasheet. Available: https://kendryte.com/downloads/ (Accessed: Jan. 2025).

[10] S. U. Khan, Y. Hara, and H. Tomiyama, "SamurAI: A RISC-V Based Processing Platform for Deep Learning Acceleration," in *Proc. SAMOS*, 2020.

[11] A. Garofalo *et al.*, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Cores through ISA Extensions," in *Proc. DATE*, 2020.

[12] A. Gondimalla and V. Sze, "SPOTS: A Dataflow Approach for Sparse Convolutions," in *Proc. MICRO*, 2019.

[13] T. Chen *et al.*, "PipeCIM: A High-Throughput Computing-In-Memory Microprocessor With Nested Pipeline and RISC-V Extended Instructions," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 7, pp. 3214–3227, July 2024. doi: 10.1109/TCSI.2024.3384271.

[14] A. Traber, M. Gautschi, P. D. Schiavone, *et al.*, "RI5CY: User Manual," Rev. 4.0, ETH Zürich & University of Bologna, April 2019. [Online]. Available: https://www.pulp-platform.org/docs/ri5cy_user_manual.pdf. [Accessed: Nov. 07, 2025].

[15] ChipVerify, "Software Driven Verification," Available online: https://www.chipverify.com/soc/soc-software-driven-verification, Accessed on: 07 November 2025