International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 15, Issue 11 [November 2026] PP: 164-168

Design and Synthesis of an Energy-Efficient Sparsity-Aware 32-Bit Pipelined RISC-V SoC with Integrated CNN Accelerator

Manova M[1], Jithin Rio R[2], Harini K S[3], Shanmathi S[4]
Sri Shakthi Institute of Engineering and Technology Coimbatore, India
Mentor: Mrs. Prema C (Assistant Professor)

Abstract*—Edge inference platforms must run control software and convolutional neural network (CNN) workloads inside a tight energy budget. This paper presents the design, verification and synthesis of a 32-bit pipelined RISC-V system-on-chip (soc_top) with an integrated CNN accelerator that exploits sparsity: [one sentence on the mechanism, e.g. zero-valued operands are detected and their multiply-accumulate operations are skipped]. The design was simulated in Cadence SimVision, [testbench / workload used], and code coverage was collected with Cadence IMC. It was synthesized with Cadence Genus 21.14 in a slow-corner standard-cell library against a 2 ns (500 MHz) clock, giving 924 cells, 6,699.3 µm² of cell area and 1.66 mW of power from simulation-derived activity. The worst setup path runs from the processor write-data input rv_wdata[1] to the CNN partial-sum register u_cnn_pe_psum_out_reg[12] and misses the 2 ns clock by 41 ps, which corresponds to a maximum clock frequency of about 490 MHz. Combinational logic accounts for 88.2% of the power and internal power for 62.9%. [Add the sparsity / energy-saving result once measured.]*

Index Terms*—RISC-V, pipelined processor, CNN accelerator, sparsity, energy efficiency, system-on-chip, logic synthesis, Cadence Genus.*

Date of Submission: 05-11-2026 Date of acceptance: 16-11-2026

I. INTRODUCTION

Convolutional neural networks are increasingly run at the edge, where area and energy matter as much as throughput. A general-purpose core alone is too slow and too energy-hungry for the multiply-accumulate (MAC) load of a CNN, while a stand-alone accelerator still needs a processor to configure it and move data. A system-on-chip that couples a small RISC-V core [1] with a CNN accelerator on one die addresses both needs.

CNN activations and weights contain many zeros, especially after ReLU and pruning, and a MAC with a zero operand does no useful work. A sparsity-aware accelerator detects such operands and avoids the associated switching activity. [Add 2–3 sentences of related work with references: sparse CNN accelerators, RISC-V + accelerator SoCs.]

The paper makes three contributions. (i) A 32-bit pipelined RISC-V core integrated with a CNN accelerator in one SoC, soc_top. (ii) A sparsity-aware processing-element (PE) datapath [describe in one line]. (iii) Simulation, coverage and ASIC-flow synthesis results (area, power and timing) from Cadence Genus [2], together with an analysis of what drives them.

II. ARCHITECTURE

The SoC is written in synthesizable Verilog. As shown in Fig. 1, it consists of a 32-bit pipelined RISC-V core, a system bus interconnect (AXI4-Lite / APB, memory-mapped I/O), a sparsity-aware CNN accelerator acting as a co-processor, and a memory subsystem made of an instruction RAM and a data RAM. Both data and address paths are 32 bits wide.

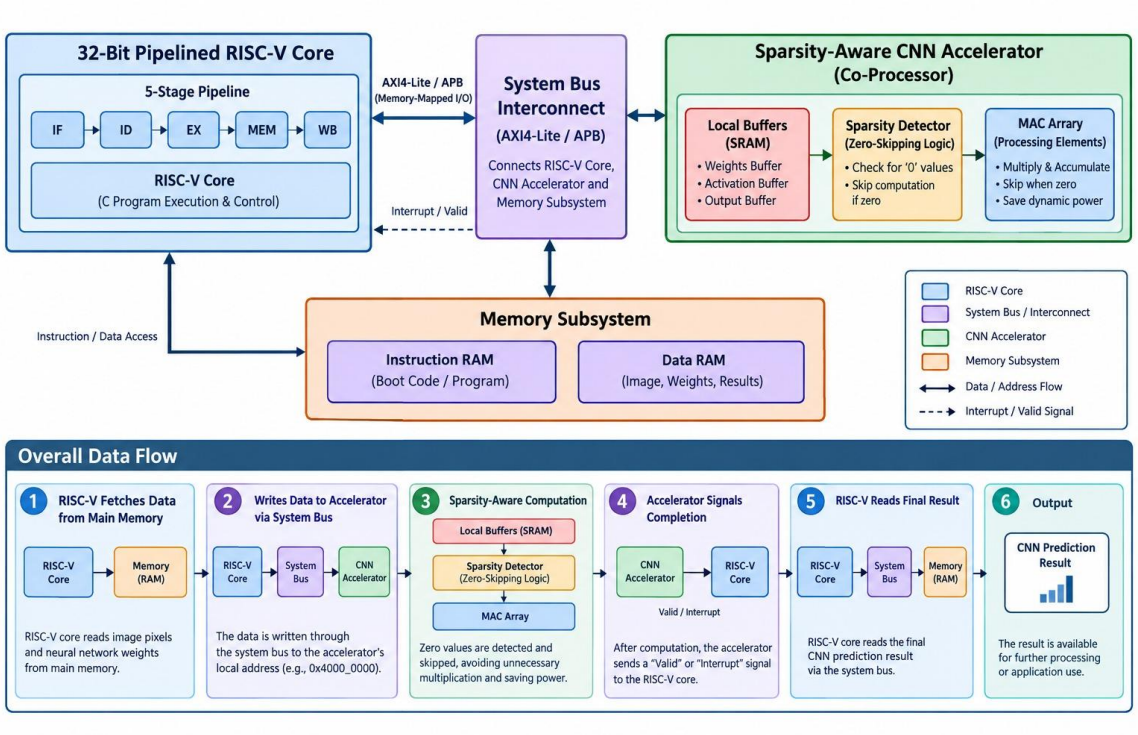


Fig. 1. Top-level architecture of the sparsity-aware RISC-V SoC with integrated CNN accelerator, with the overall data flow.

A. Pipelined RISC-V Core

The core is a five-stage pipeline with instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM) and write-back (WB) stages, and runs C programs that control the whole system. It fetches instructions and accesses data directly from the memory subsystem, and reaches the accelerator through the system bus. [Add ISA subset (RV32I?), forwarding unit, hazard detection and branch handling.]

B. CNN Accelerator

The accelerator is attached as a co-processor and has three parts: local SRAM buffers (weights, activation and output buffers), a sparsity detector, and a multiply-accumulate (MAC) array of processing elements. The processor writes data to the accelerator's local address range (for example 0x4000_0000) through the system bus, and the accelerator signals completion to the core with a valid or interrupt signal. [Add PE array size, dataflow, data precision and how partial sums are accumulated in u_cnn_pe_psum_out.]

C. Sparsity-Aware Operation

Before each multiplication the sparsity detector checks the operands for zero. When a zero is found, the multiply is skipped, which avoids unnecessary computation and saves dynamic power in the MAC array. [Add how skipping is implemented (operand isolation, clock gating or enable) and the area cost of the detector.]

D. Memory and Interface

The instruction RAM holds the boot code and program, and the data RAM holds the image, weights and results. The system bus interconnects the core, accelerator and memory subsystem. The overall data flow is: (1) the core reads image pixels and weights from main memory, (2) writes them to the accelerator through the system bus, (3) the accelerator computes with sparsity-aware skipping, (4) it signals completion with a valid or interrupt signal, (5) the core reads the final CNN prediction through the system bus, and (6) the result is available for further use. [Add the address map table.]

III. FUNCTIONAL VERIFICATION

A. Test Environment

[Describe the testbench: simulator used, stimulus (assembly program, test vectors, CNN layer used), clock frequency and reset sequence.] Waveforms were viewed in Cadence SimVision (Fig. 2) and code coverage was collected and analysed with Cadence Integrated Metrics Center (IMC).

B. Functional Results

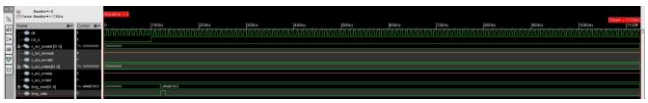


Fig. 2. SimVision waveform of the simulation of the SoC. [Say what the signals show: reset release, start of computation, result valid.]

[Report what was checked and the result, e.g. the processor program ran correctly, accelerator output matched the software reference, with a table of test cases if available.]

C. Code Coverage

Table I gives the coverage collected by IMC. [Fill the table from the IMC run on soc_top. Block, expression, toggle and FSM coverage are the usual metrics.]

TABLE I
CODE COVERAGE OF SOC_TOP (IMC)

Metric	Covered	Total	Grade
Block	[ ]	[ ]	[ ]
Expression	[ ]	[ ]	[ ]
Toggle	[ ]	[ ]	[ ]
FSM	[ ]	[ ]	[ ]
Overall	[ ]	[ ]	[ ]

IV. LOGIC SYNTHESIS

A. Setup

The RTL was synthesized with Cadence Genus Synthesis Solution 21.14 [2] against a slow-corner standard-cell library in enclosed wireload mode, with area taken from the timing library. The clock period was 2 ns (500 MHz) with 0.15 ns clock uncertainty, and input delays of 0.4 ns were applied to the primary inputs. The clock is ideal, so all results are post-synthesis and pre-layout; clock-tree and routing effects are not included. Power was computed from simulation-derived activity (stimulus frame stim#0/frame#0).

B. Area, Power and Timing

Table II summarizes the results. The design maps to 924 leaf cells and 976 nets, with 101 terminals, and 6,699.3 µm² of cell area, or about 7.25 µm² per cell on average. Total power is 1.659 mW. The worst setup path misses timing by 41 ps at the 2 ns clock. Adding the violation to the target period gives a minimum period of about 2.041 ns, an estimated maximum frequency near 490 MHz. This figure is extrapolated from the 500 MHz run, not obtained by re-synthesizing at a lower clock rate. Fig. 3 shows the synthesized netlist in the Genus schematic viewer; the flattened netlist is too dense to read at this scale and is shown for completeness.

TABLE II
SYNTHESIS SUMMARY AT 500 MHZ

Metric	Value
Clock period (target)	2 ns (500 MHz)
Total leaf cells	924
Nets	976
Cell area	6,699.3 µm²
Total power	1.659 mW
Worst setup slack	−41 ps (violated)
Estimated maximum frequency	≈ 490 MHz

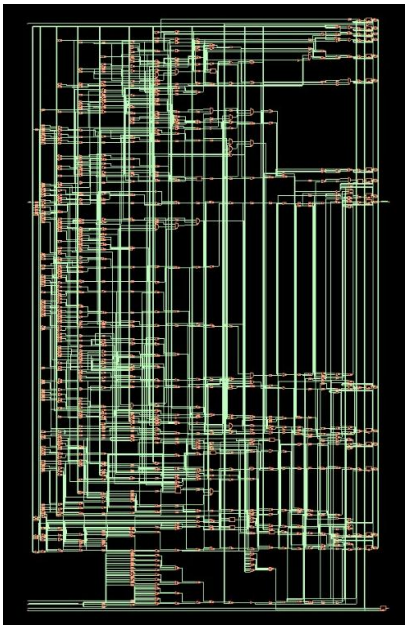


Fig. 3. Genus schematic view of the synthesized, flattened netlist of soc_top (924 leaf cells, 976 nets).

C. What Drives Power

Table III gives the power breakdown. Unlike a storage-dominated design, power here is dominated by combinational logic, which takes 88.2% of the total, while the registers take 11.1%. By type, internal (cell) power is 62.9% and switching power is 34.8%. Leakage is 2.3%. Because the clock is ideal and no clock tree is built, the 0.7% clock share is understated and will grow after clock-tree synthesis. [Link this to the sparsity result: logic switching is what skipping zero operands is meant to reduce. Add measured with/without-sparsity power if available.]

TABLE III

POWER BREAKDOWN AT 500 MHZ

Group	**Power (µW)**	**Share**
By category		
Register	184.3	11.1%
Logic	1,463.3	88.2%
Clock	11.5	0.7%
By type		
Leakage	38.3	2.3%
Internal	1,044.2	62.9%
Switching	576.6	34.8%
Total	1,659.1	100%

D. Critical Path

Table IV lists the worst path. It starts at the primary input rv_wdata[1], which has a fan-out of 22 and drives a load of 297.5 fF, and ends at the D pin of the partial-sum register u_cnn_pe_psum_out_reg[12]. After the 0.4 ns input delay the data path is 1.371 ns long and passes through 18 combinational cells, an average of about 76 ps per cell. It begins with a NAND4X8 gate (118 ps), and the slowest later stages are an AOI21X2 (108 ps), a MXI2X1 (105 ps) and an inverter stage (100 ps). The path therefore crosses from the processor write data through the accelerator's arithmetic logic to the partial-sum register, so the accelerator datapath, not the clock network, sets the frequency limit. Registering the write data at the accelerator input, pipelining the partial-sum computation, or relaxing the input delay would shorten it.

TABLE IV

WORST SETUP PATH (SYS_CLK)

Item	**Value**
Startpoint	rv_wdata[1]
Endpoint	u_cnn_pe_psum_out_reg[12]/D

Capture clock edge	2000 ps
Setup time	120 ps
Clock uncertainty	150 ps
Required time	1730 ps
Input delay	400 ps
Data path delay	1371 ps
Arrival time	1771 ps
Slack	−41 ps

V. DISCUSSION AND LIMITATIONS

The following limitations should be read together with the results above.

1) Timing. The design does not meet the 500 MHz target in this run; the worst path is 41 ps short. The path is input-to-register and depends on the 0.4 ns input delay assumed in the constraints.

2) Synthesis scope. Results are pre-layout with an ideal clock and an enclosed wireload model, and area, power and timing will change after clock-tree synthesis and place-and-route.

3) Power activity. Power was computed from one simulation frame, so it reflects that workload only. [State the workload.]

4) Verification scope. [State what was tested and what was not, e.g. directed tests only, no randomized or formal checks, and the coverage gaps seen in IMC, such as low toggle coverage.]

5) Sparsity evaluation. [State whether energy savings were measured, and for which sparsity levels.]

VI. CONCLUSION

This paper presented a 32-bit pipelined RISC-V SoC with an integrated sparsity-aware CNN accelerator. [One sentence on simulation / coverage outcome.] Genus synthesis against a 2 ns clock gave 924 cells, 6,699.3 μm² and 1.66 mW, with the worst path 41 ps short of the target, an estimated maximum frequency of about 490 MHz. Combinational logic, not storage, dominates power, and the critical path lies in the path from the processor write data into the accelerator's partial-sum register.

Future work includes pipelining the critical path, [quantifying the energy saved by sparsity], improving coverage and adding randomized and assertion-based verification, and place-and-route to obtain post-layout figures.

REFERENCES

[1] A. Waterman and K. Asanović, eds., The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA, RISC-V International.

[2] Cadence Design Systems, Inc., Genus Synthesis Solution, release 21.14, product documentation.

[3] [Add: Cadence SimVision and IMC documentation, sparse CNN accelerator papers, other related work.]