Texas Tech University
ECE 5310 - Introduction to VLSI Design
Professor: Dr. Tooraj Nikoubin

Architecture for a Convolutional Neural Network

A VLSI Multiply-Accumulate Architecture for CNN Convolution

Portfolio Report

Main Author: Khalil Blaine
Project Collaborators: Anthony Burks, Samuel Dunham, Tucker Hortman
Institution: Texas Tech University
Course: ECE 5310 Introduction to VLSI Design
Professor: Dr. Tooraj Nikoubin
Date: May 2022

Authorship and Acknowledgements

Khalil Blaine served as the main author for this portfolio report and led the written presentation of the design approach, architecture, simulation interpretation, and final engineering summary. The report is based on the original team project titled *Architecture for a Convolutional Neural Network*, completed for Texas Tech University's ECE 5310 Introduction to VLSI Design course in May 2022.

The original class project was completed with Anthony Burks, Samuel Dunham, Khalil Blaine, and Tucker Hortman. Anthony Burks, Samuel Dunham, and Tucker Hortman are acknowledged for their collaboration and contributions to the project effort. Appreciation is also extended to Dr. Tooraj Nikoubin, Professor for ECE 5310 Introduction to VLSI Design, for instruction and guidance throughout the course.

Abstract

This report presents a VLSI-oriented architecture for implementing the convolution operation used in convolutional neural networks. The design focuses on the core arithmetic activity behind convolution: repeated multiplication of an input matrix element with a kernel element, followed by accumulation into a running sum. The resulting hardware datapath uses two 8-bit inputs, an 8-bit multiplier, a 32-bit adder, and a register-based accumulator to preserve the partial result between computation steps.

The project was developed from elementary logic blocks through full-system simulation. XOR and NAND gates were used as foundational components for arithmetic and storage circuits. The adder was used both as a lower-level arithmetic block and as part of the 32-bit accumulation path. A flip-flop based register stores the running sum, allowing the convolution engine to support a variable number of multiply-accumulate iterations. Simulation results reported in the project include a 0.7 ns XOR gate delay, a 1.5 ns adder delay, a 33 ns delay for the 32-bit adder path, and an 11 ns delay for the 8-bit multiplier. The full system simulation demonstrates functional operation, while the timing results identify the multiplier and 32-bit adder as the primary areas for optimization.

Keywords: VLSI design, convolutional neural network, convolution, multiply-accumulate, XOR gate, NAND gate, adder, multiplier, register, timing simulation.

Contents

List of Figures

1 Introduction

Convolutional neural networks rely on convolution to extract features from structured data such as images. In a digital hardware context, the convolution operation can be reduced to a sequence of multiply-accumulate operations. Each local input value is multiplied by a corresponding kernel value, and the products are accumulated to produce an output value. This project implements that computational pattern as a VLSI datapath built from elementary logic and arithmetic blocks.

The objective of the original design was not to build a complete neural-network accelerator, but to design and verify the essential arithmetic hardware needed to perform a convolution step. The project therefore focuses on a practical hardware path: accept one cell from the input matrix, accept one cell from the kernel matrix, multiply the values, add the product to a running sum, and store the updated sum in a register. Repeating that sequence allows the same datapath to evaluate different kernel sizes, subject to accumulator width and overflow limitations.

This portfolio paper expands the original presentation into a formal engineering report. It preserves the core project content while adding context, clarifying design intent, and explaining how each circuit block contributes to the complete convolution architecture. The report also identifies the major timing bottlenecks and suggests professional next-step improvements that would strengthen the design for a higher-performance VLSI implementation.

Portfolio Context
This document is prepared as a professional portfolio version of a Texas Tech University ECE 5310 Introduction to VLSI Design class project. Khalil Blaine is identified as the main author of the report, with classmates acknowledged for their collaboration on the original project effort.

2 Project Scope and Design Goals

The project was organized around the hardware implementation of a convolution datapath. A convolution layer contains many operations, but the repeated mathematical core is a dot-product style calculation between a window of input values and a set of kernel coefficients. For hardware design purposes, the project reduced this larger operation into a reusable multiply-accumulate structure.

The major design goals were as follows:

- Define convolution as a hardware-friendly repeated dot product.
- Use 8-bit input operands for each pair of matrix and kernel cells.
- Multiply the two input values to produce a product term.
- Add the product term to a 32-bit running sum.
- Store the updated running sum in a register for the next accumulation cycle.
- Verify the behavior of the major gate-level and system-level blocks through simulation.
- Evaluate the delays of key blocks to understand timing limitations and optimization needs.

A key theme of the architecture is modularity. Rather than designing a single monolithic block, the system is assembled from simple circuit components: XOR gates, NAND gates, adders, flip-flops, a multiplier, an accumulator, and a final test bench. This bottom-up approach is appropriate for an introductory

VLSI design project because it demonstrates how basic transistor-level or gate-level structures can be integrated into a meaningful computation engine.

3 Convolution Operation

3.1 Mathematical Definition Used in the Project

In the scope of this project, convolution was represented as the dot product between two matrices: an input matrix window and a kernel matrix. For a 3-by-3 example, matrix X is convolved with matrix Y by multiplying corresponding entries and adding all products. The original project expression is shown below:

$$\begin{aligned} S = {} & X_{00}Y_{00} + X_{10}Y_{10} + X_{20}Y_{20} \\ & + X_{01}Y_{01} + X_{11}Y_{11} + X_{21}Y_{21} \\ & + X_{02}Y_{02} + X_{12}Y_{12} + X_{22}Y_{22} \end{aligned} \tag{1}$$

This equation maps naturally into hardware because each product can be computed by a multiplier and then added to an accumulator. The same arithmetic block can be reused for each term, which reduces circuit complexity compared with implementing every product and addition in parallel. The result is a sequential multiply-accumulate approach that is more compact and flexible.

3.2 Hardware Interpretation

The hardware interpretation of the equation is straightforward. One element from matrix X and one element from matrix Y are selected and treated as the two operands for the multiplier. The product is then added to the current accumulated value. After the addition, the register updates and stores the new sum. When the next pair of matrix values is applied, the stored value becomes the starting point for the next accumulation step.

This project therefore implements a core multiply-accumulate path rather than a complete CNN processor. A full CNN accelerator would require additional control logic, memory hierarchy, data reuse scheduling, input buffering, and output feature-map storage. The project focuses on the arithmetic engine that would sit inside such a larger system.

4 System-Level Architecture

4.1 Datapath Overview

The architecture uses two inputs, labeled Input A and Input B. Input A represents a cell from matrix X, and Input B represents a corresponding cell from matrix Y. Each input is treated as an 8-bit value. The two 8-bit values are multiplied, producing a 16-bit product. That product is then added to a 32-bit running sum. The register captures the updated sum and feeds it back into the adder for the next iteration.

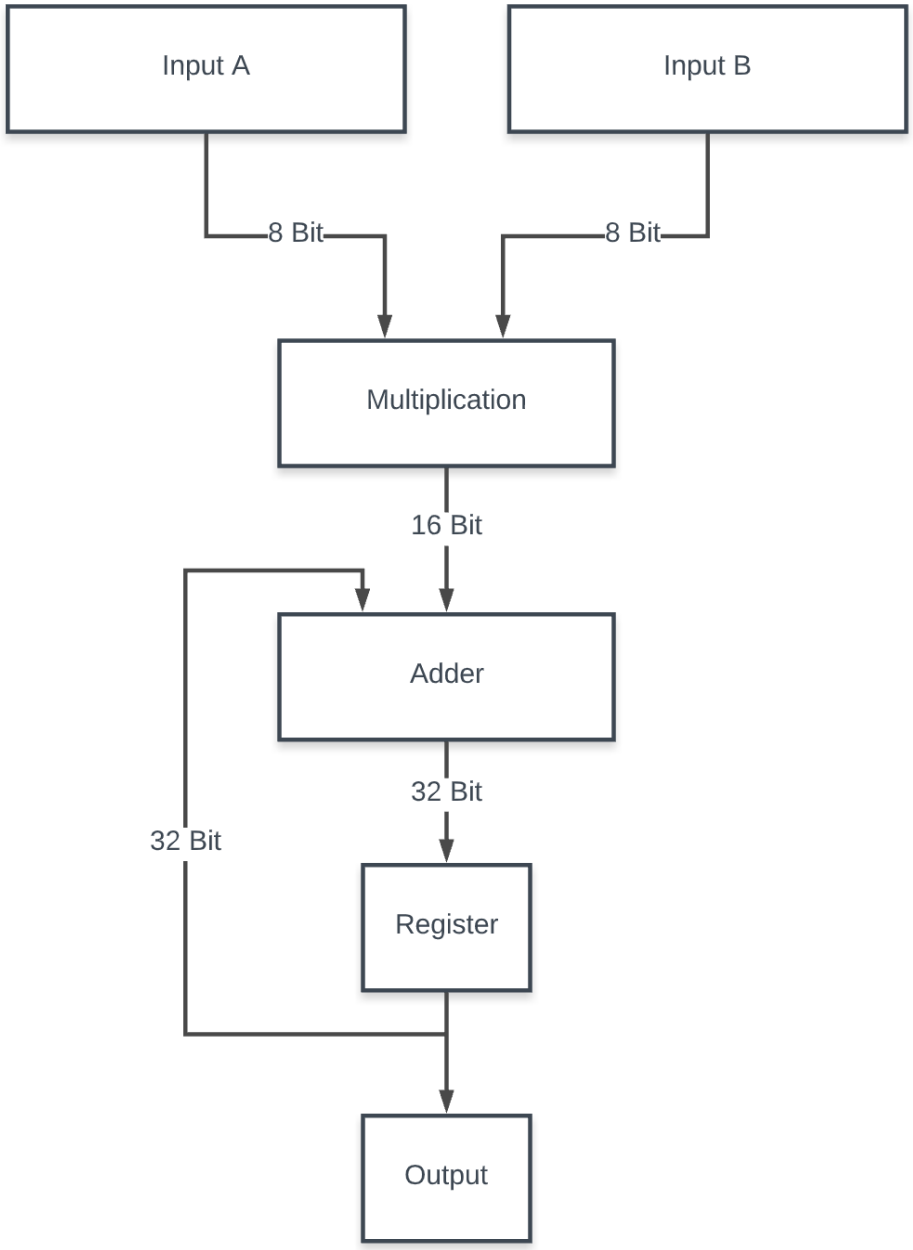

Figure 1: System-level multiply-accumulate flow for the CNN convolution architecture.

This feedback structure is important because it enables dynamic kernel size. The same datapath can process a 3-by-3 kernel, a larger kernel, or another repeated accumulation sequence, provided that the control logic supplies the correct number of input pairs and the register can represent the accumulated sum without overflow. In this way, the register is not only a storage element; it also defines part of the numeric range of the architecture.

4.2 Data Widths and Accumulator Behavior

The chosen data widths reflect the project-level arithmetic needs. Two 8-bit input operands produce a product that may require up to 16 bits. Because convolution requires multiple products to be added

together, the running sum must be wider than the product to avoid immediate overflow. The design uses a 32-bit adder and a 32-bit register for the accumulation path.

The 32-bit width provides a practical margin for repeated accumulation, but it does not eliminate overflow for every possible kernel size or input range. The maximum number of safe accumulation steps depends on input value range, signed or unsigned interpretation, product magnitude, and accumulator width. The original conclusion therefore correctly identifies register size as the upper limit on dynamic kernel size.

5 Design Methodology

The project used a bottom-up VLSI design methodology. The design begins with foundational gates, then uses those gates to support arithmetic and sequential storage, and finally integrates the resulting blocks into a full convolution computation path.

The design flow can be summarized in four phases:

1. **Primitive logic design:** XOR and NAND gates provide the basic logic functions needed for arithmetic and storage.
2. **Arithmetic block construction:** Adders and the 8-bit multiplier implement the numerical operations required for convolution.
3. **Sequential storage design:** Flip-flops are used to form the register that holds the running sum.
4. **System integration and simulation:** The multiplier, adder, register, and test bench are connected to verify full-system behavior.

This methodology is appropriate for VLSI coursework because it connects device-level and gate-level design choices to a recognizable computing workload. It also makes timing bottlenecks easier to identify. By measuring delay at the gate, arithmetic, and system levels, the designer can determine which blocks most strongly limit the achievable clock period.

6 Logic Gate Building Blocks

6.1 XOR Gate

The XOR gate is used to support half-adder functionality. In binary addition, the sum bit of a half adder is generated by an XOR operation between the two input bits. Because addition appears in both the multiplier and accumulation path, the XOR gate is an important primitive within the overall architecture.

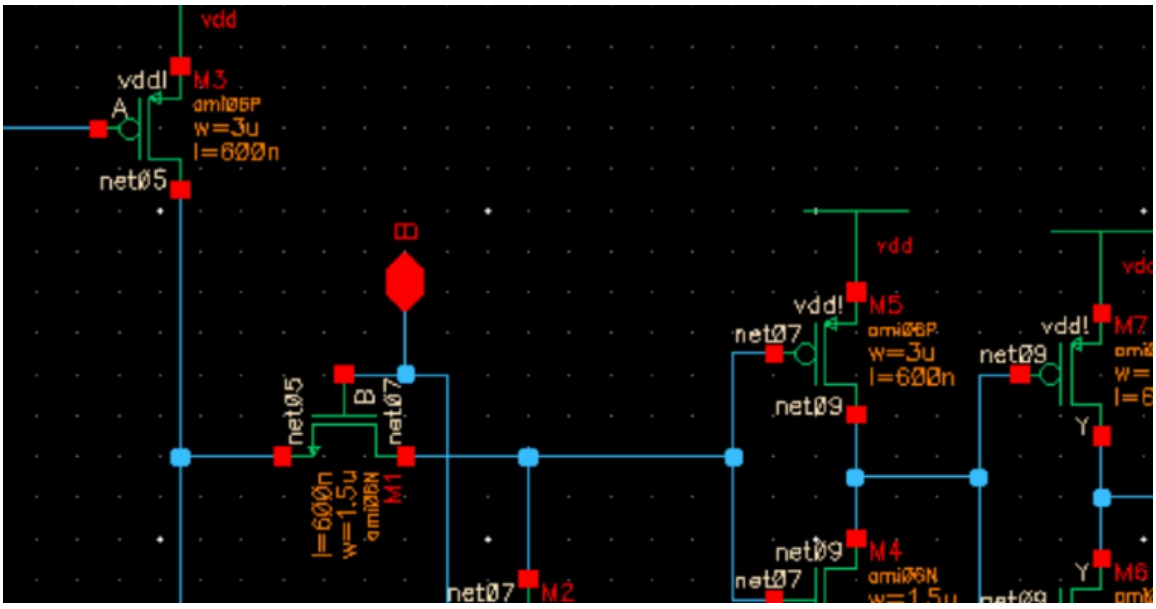

Figure 2: XOR gate schematic used as a foundational block for half-adder construction.

The simulated XOR delay reported in the presentation is 0.7 ns. This delay is relatively small compared with the larger arithmetic paths, but it still contributes to the total delay of compound circuits such as adders and multipliers. When gates are cascaded, even small delays become important because the total propagation delay grows across the critical path.

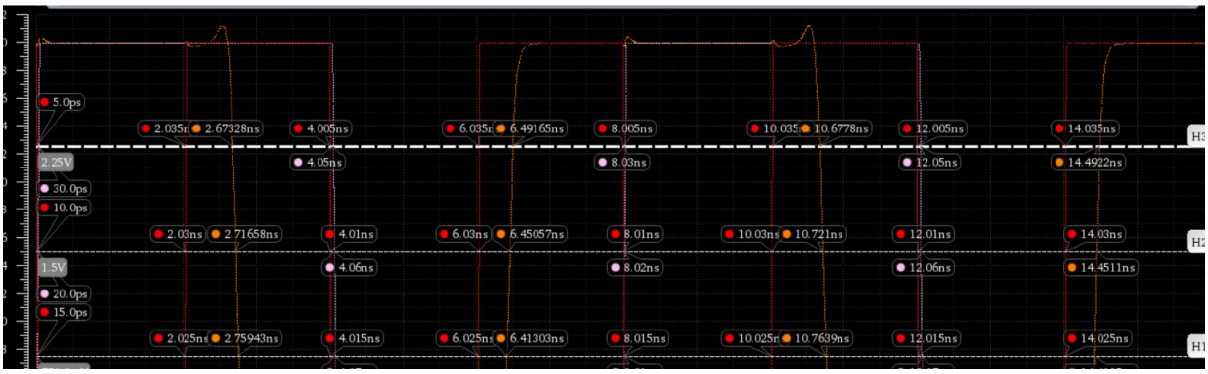

Figure 3: XOR gate simulation waveform showing a reported delay of 0.7 ns.

6.2 NAND Gate

The NAND gate is used in the flip-flop and half-adder related logic. NAND gates are valuable in digital design because they are functionally complete; larger Boolean functions can be constructed from NAND gates alone. This makes the NAND gate useful for building both combinational logic and sequential storage structures.

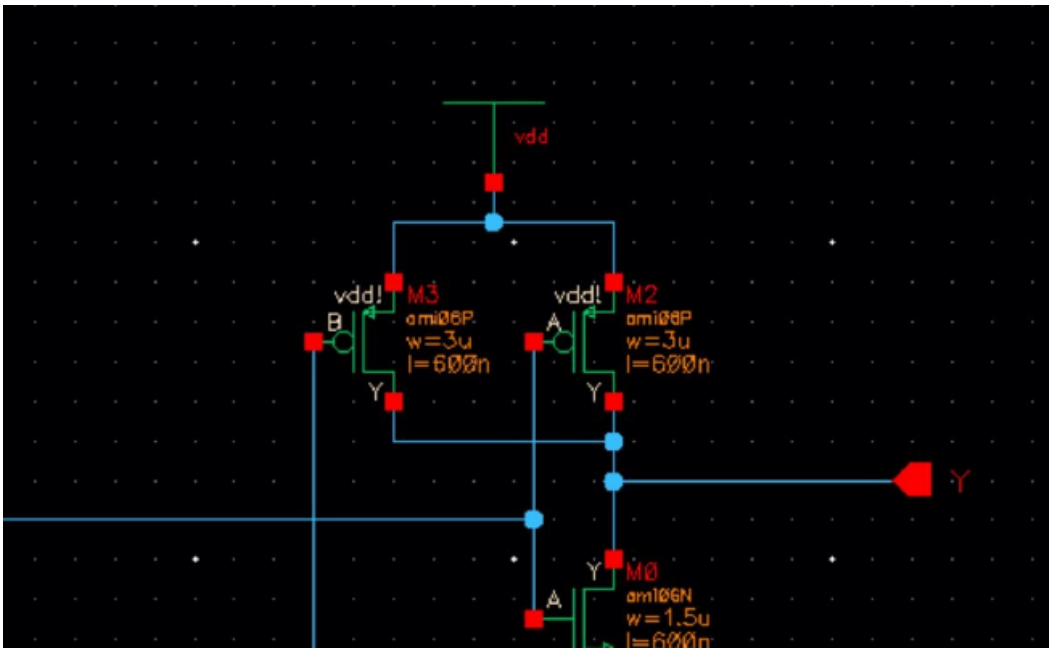

Figure 4: NAND gate schematic used in the flip-flop and half-adder support logic.

The NAND simulation confirms switching behavior for the gate. A specific numerical delay was not provided in the original slide for the NAND gate, but the waveform evidence supports its use as a verified building block in the design hierarchy.

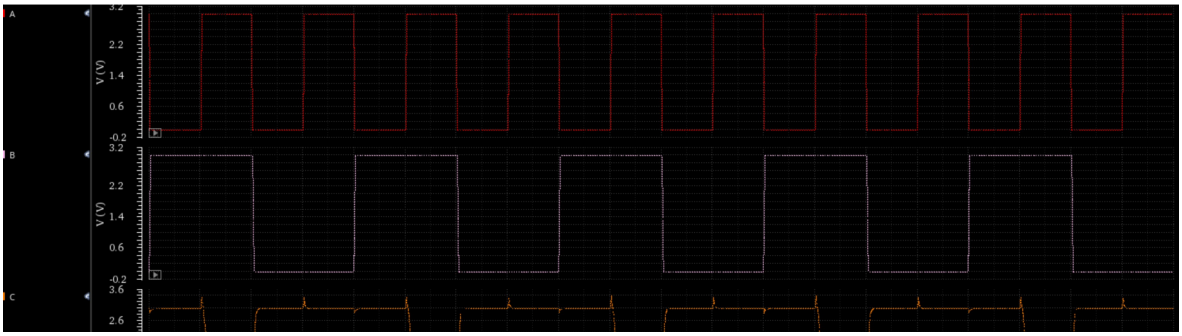

Figure 5: NAND gate simulation waveform used to verify logic behavior.

7 Adder Design and Timing

7.1 Adder Function in the Architecture

The adder is used in both the multiplier and the 32-bit accumulation path. In the multiplier, adders combine partial products. In the accumulation stage, the adder combines the current product with the running sum stored in the register. This makes the adder one of the most important blocks in the entire convolution datapath.

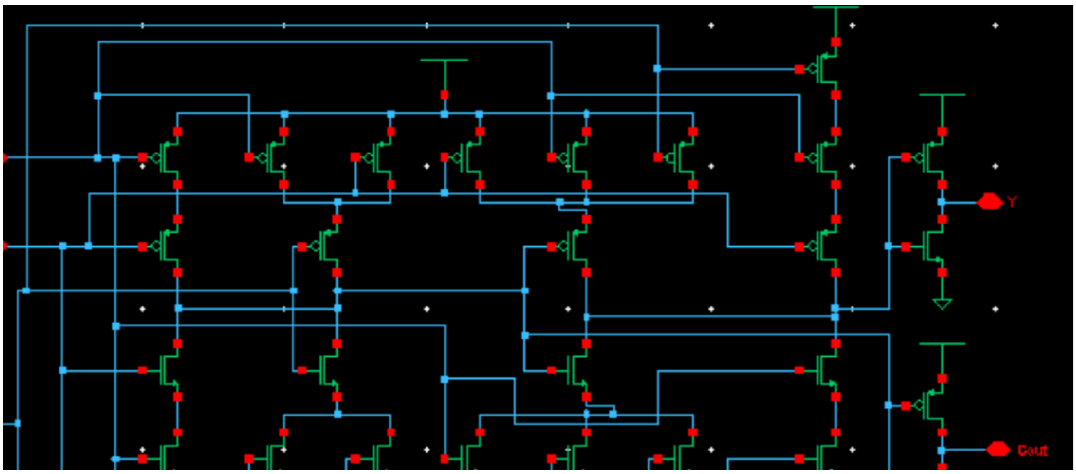

Figure 6: Adder schematic used as a repeated arithmetic component in the multiplier and 32-bit accumulator.

The reported adder delay is 1.5 ns. This delay is useful as a local timing measure for the arithmetic cell, but the system timing depends on how many adder stages are cascaded and how carry information propagates through the final 32-bit accumulation structure.

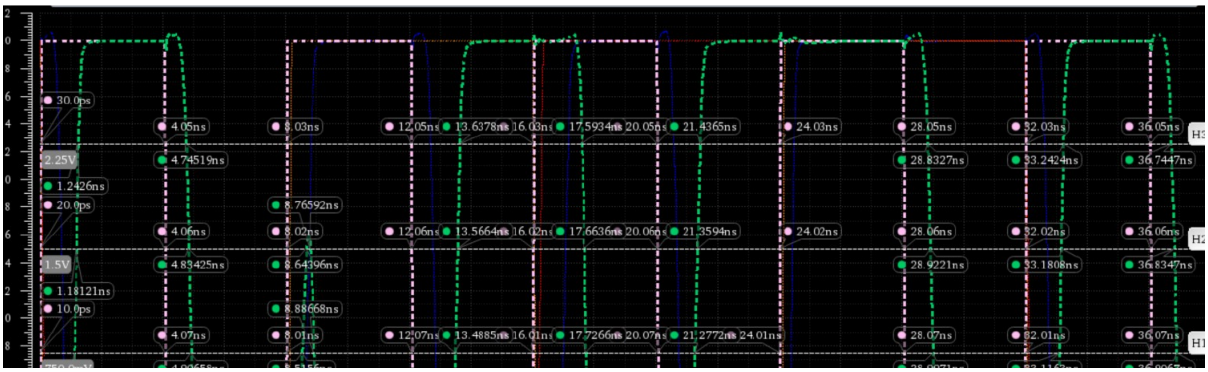

Figure 7: Adder simulation waveform showing a reported delay of 1.5 ns.

7.2 32-Bit Adder Timing

The 32-bit adder timing result is one of the most important measurements in the project because the accumulator path is central to convolution. The reported delay for the 32-bit adder timing simulation is 33 ns. This result indicates that the wider accumulation path is significantly slower than the individual adder cell.

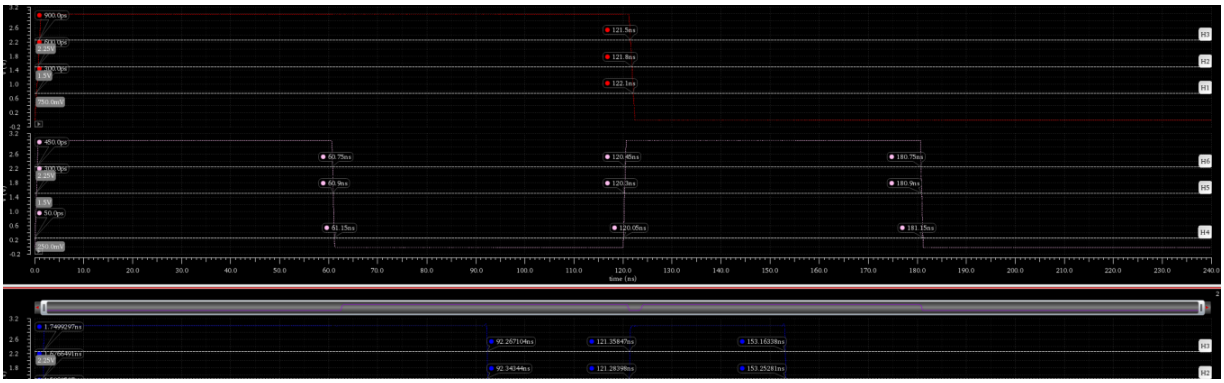

Figure 8: 32-bit adder timing simulation with a reported delay of 33 ns.

A 33 ns delay suggests that the 32-bit adder is a likely critical-path contributor. If this architecture were developed further, the adder would be a major candidate for optimization. Depending on the implementation, improvements could include replacing a ripple-carry style structure with carry-lookahead logic, carry-select logic, or a parallel-prefix adder. These optimizations generally trade additional area and wiring complexity for reduced propagation delay.

8 Register and Flip-Flop Design

8.1 Register Role

The flip-flop is used as the storage element for the running sum register. The register is essential because the convolution operation is sequential in this architecture. After each multiplication and addition, the updated result must be preserved so that the next product can be added to it.

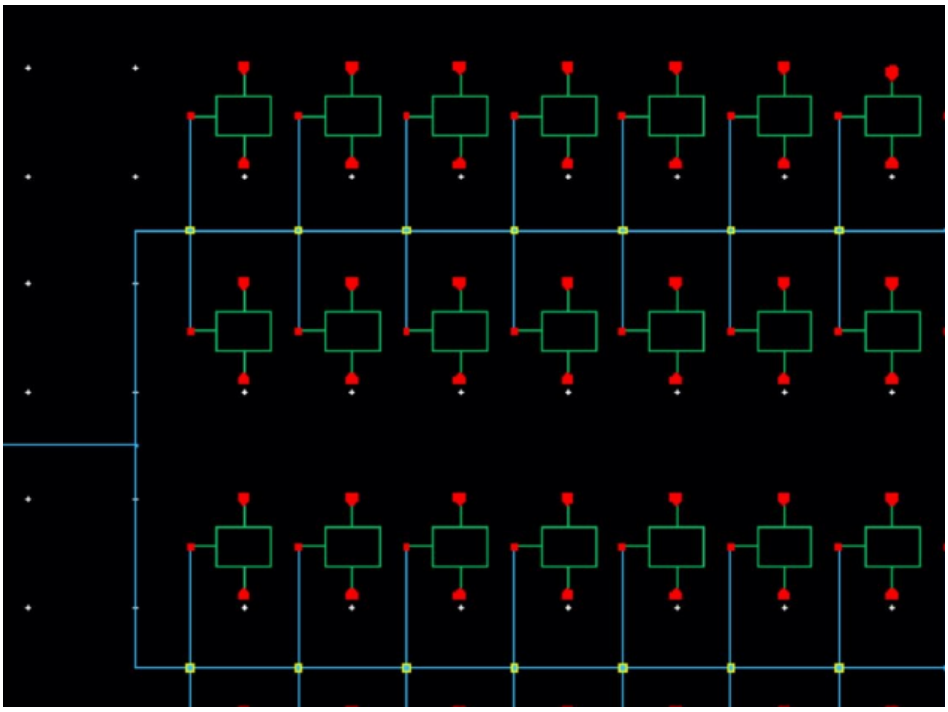

Figure 9: Flip-flop schematic used as the basis for the running-sum register.

The register makes the architecture flexible. Instead of requiring a separate adder tree for every kernel size, the datapath can reuse the same multiplier and adder over multiple cycles. The tradeoff is that sequential reuse reduces hardware area but increases the number of cycles required to compute a full convolution output.

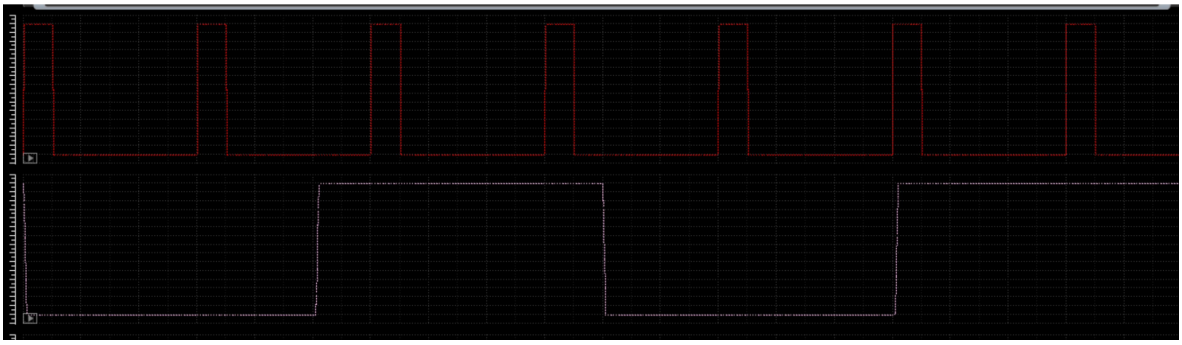

Figure 10: Flip-flop simulation waveform for sequential storage verification.

The original flip-flop simulation slide listed the delay field but did not include a numerical value. For that reason, this report describes the flip-flop functionally and does not assign a delay number that was not reported. In a more complete timing signoff, the clock-to-Q delay, setup time, hold time, and register loading would need to be measured and included in the final timing budget.

9 Multiplier Design

9.1 8-Bit Multiplier

The multiplier accepts the two 8-bit inputs and generates a product term for the convolution sum. Because multiplication is performed repeatedly for every matrix and kernel element pair, the multiplier has a direct effect on system throughput. The original project identified the multiplier as an 8-bit multiplier.

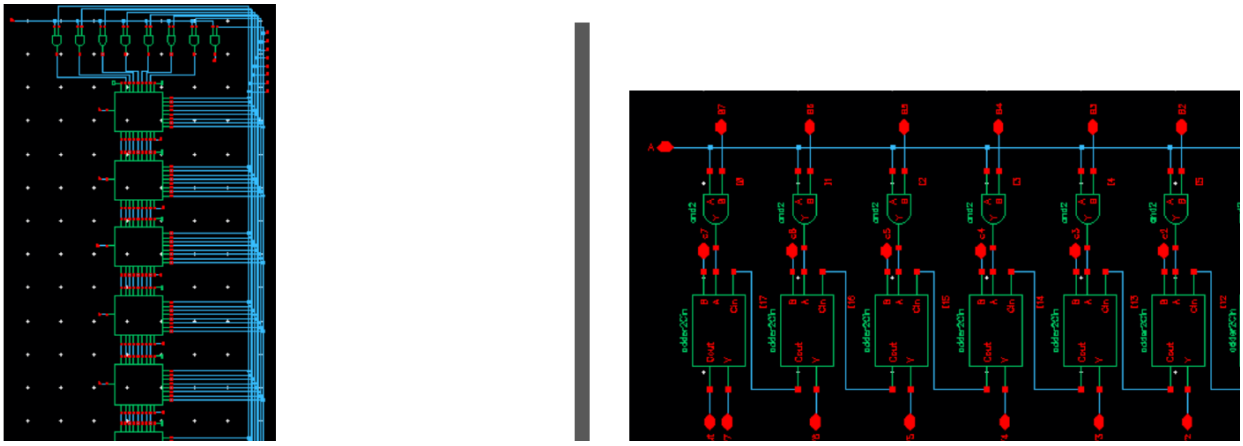

Figure 11: 8-bit multiplier architecture used to generate product terms for the convolution datapath.

The simulation result reported a multiplier delay of 11 ns. This makes the multiplier faster than the 32-bit adder path but still significant within the overall multiply-accumulate cycle. Because a product must be generated before accumulation can complete, multiplier delay contributes directly to the cycle time unless the design is pipelined.

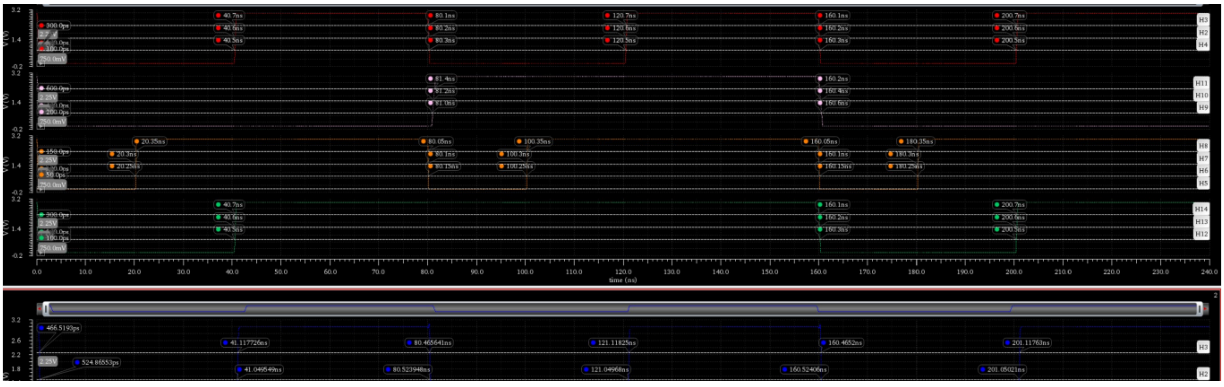

Figure 12: Multiplier simulation waveform showing a reported delay of 11 ns.

9.2 Multiplier Optimization Considerations

The original conclusion noted that both the multiplier and adder need optimization to reduce delay. For the multiplier, possible improvements include using a more efficient partial-product reduction tree, evaluating Booth encoding for signed multiplication, or introducing pipeline registers between partial-product generation and final addition. These techniques would increase design complexity but could improve the maximum operating frequency.

In a portfolio context, this observation is valuable because it demonstrates that the project did not

stop at functional verification. It also considered performance limitations and identified where the next engineering effort should be focused.

10 Full-System Integration

10.1 Final Test Bench

The final test bench integrates the major circuit blocks so that the complete datapath can be exercised. The test bench is used to apply input values, observe intermediate behavior, and verify that the multiplier, adder, and register operate together as a full convolution accumulation system.

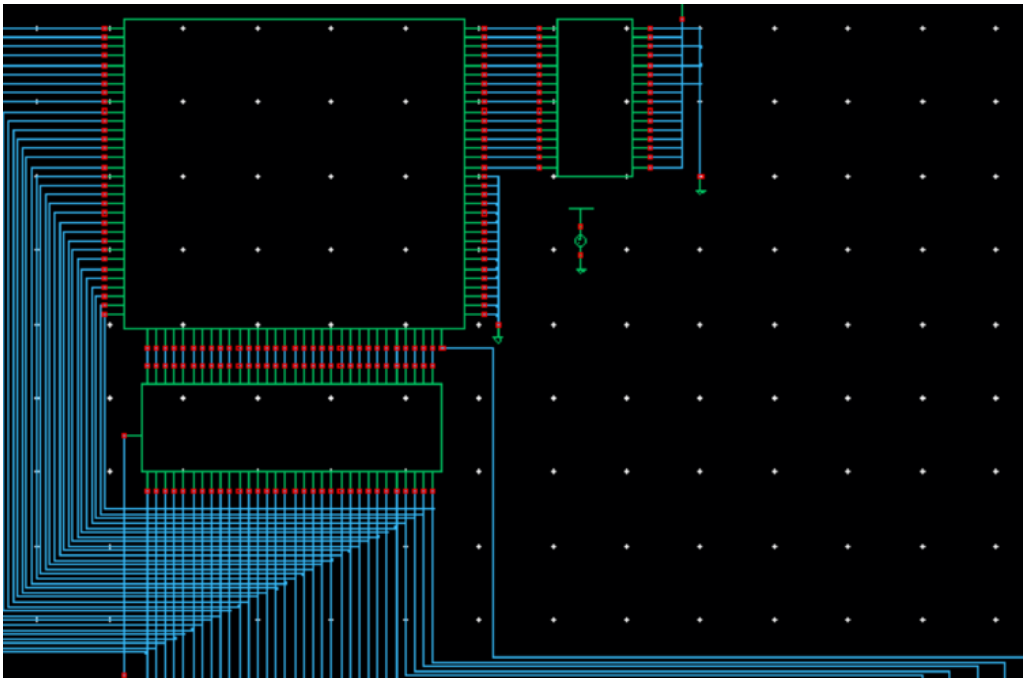

Figure 13: Final test bench used to integrate and verify the multiply-accumulate datapath.

The purpose of the test bench is broader than simply proving that a schematic is connected. It provides a controlled verification environment that can reveal timing issues, incorrect logic behavior, improper register updates, and unexpected signal dependencies. For a VLSI design, this kind of structured simulation is necessary before layout, timing closure, or further optimization.

10.2 Full-System Simulation

The full-system simulation demonstrates that the complete design operates as intended. The original conclusion states that the full system works. In this context, that means the architecture successfully performs the intended sequence: receive input values, multiply them, add the product to the running sum, and store the updated sum for continued accumulation.

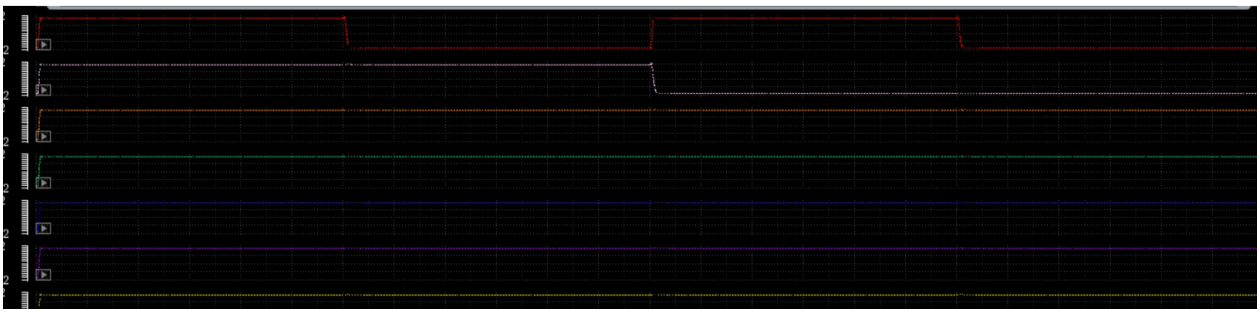

Figure 14: Full-system simulation waveform used to verify integrated datapath operation.

Although functional verification was achieved, the timing results show that the design would benefit from optimization before being treated as a high-performance accelerator component. The 32-bit adder delay and multiplier delay are the most important reported timing values for future improvement.

11 Timing Summary

Table 1 summarizes the timing values and functions reported in the original project presentation. These values provide a concise view of the design's performance profile.

Table 1: Reported component roles and timing results.

Block	Role in the CNN Convolution Architecture	Reported Delay
XOR gate	Supports half-adder sum generation and arithmetic logic.	0.7 ns
NAND gate	Supports flip-flop and half-adder related logic.	Not specified
Adder	Used in the multiplier and as a lower-level arithmetic component.	1.5 ns
32-bit adder	Adds product terms to the running accumulated sum.	33 ns
Flip-flop / register	Stores the running sum between accumulation cycles.	Not specified
8-bit multiplier	Multiplies one input matrix cell by one kernel matrix cell.	11 ns
Full system	Integrates multiplier, adder, register, and test bench.	Functional operation verified

The timing profile indicates that the 32-bit adder is the largest reported delay contributor. The multiplier is also a meaningful timing component, while the smaller gate-level blocks contribute to delay indirectly through larger structures. A practical performance-improvement effort would therefore begin with the accumulator and multiplier architecture.

12 Engineering Assessment

12.1 Strengths of the Design

The main strength of the architecture is its clarity. The design directly maps the convolution equation into a hardware multiply-accumulate structure. This makes the system easy to understand, verify, and modify. The use of a register to store the running sum also gives the architecture flexibility because the number of accumulation cycles can be adjusted for different kernel sizes.

Another strength is the bottom-up verification approach. Each major building block has simulation evidence, and the full system is tested after integration. This reduces the risk of treating the system as a black box and helps connect circuit-level design choices to system-level behavior.

12.2 Limitations

The largest limitation is delay through the arithmetic path. The reported 33 ns delay for the 32-bit adder indicates that the accumulator path may restrict system speed. The multiplier also has a reported 11 ns delay and should be optimized if the architecture is expected to operate at a higher clock frequency.

A second limitation is accumulator overflow. The register allows dynamic kernel size, but the register width determines the maximum representable accumulated value. For larger kernels, larger input magnitudes, or signed arithmetic, overflow analysis becomes essential. A production-quality version would need overflow detection, saturation logic, scaling, or a wider accumulator.

A third limitation is that the project presentation focused on functional and timing simulations, not complete physical implementation metrics. A deeper portfolio version could include area, power, transistor count, post-layout parasitics, or timing under process-voltage-temperature corners. Those metrics would make the design more comparable to a full VLSI implementation.

12.3 Recommended Future Improvements

The following improvements would make the architecture more professional and performance-oriented:

- Replace or improve the 32-bit adder to reduce accumulation delay.
- Evaluate a faster multiplier architecture or introduce pipelining.
- Add overflow detection or a wider accumulator for larger kernels.
- Define signed versus unsigned arithmetic behavior explicitly.
- Add control logic documentation for kernel iteration, register reset, and output valid timing.
- Perform post-layout simulation to estimate parasitic effects on delay.
- Add area and power estimates to support a complete VLSI design tradeoff analysis.

These improvements would not change the fundamental design objective. They would strengthen the implementation and make it more suitable for a higher-speed or lower-power CNN hardware accelerator.

13 Conclusion

This project successfully demonstrates a VLSI-oriented architecture for the convolution operation used in convolutional neural networks. The design maps convolution into a repeated multiply-accumulate datapath using 8-bit inputs, an 8-bit multiplier, a 32-bit adder, and a register-based running sum. The

supporting logic blocks include XOR gates, NAND gates, adders, and flip-flops, all of which contribute to the integrated system.

The full system simulation confirms that the architecture works. The use of a register to track the running sum allows the design to support dynamic kernel sizes, because the same datapath can be reused across multiple accumulation steps. The main constraint on kernel size is the register width, since larger numbers of accumulated products increase the risk of overflow.

The project also identifies clear opportunities for future improvement. The multiplier and adder should be optimized to reduce delay, with particular attention to the 32-bit accumulation path. Overall, the design is a strong demonstration of how fundamental VLSI building blocks can be composed into a meaningful CNN computation engine suitable for portfolio presentation.

A Original Presentation Coverage Map

This appendix summarizes how the original presentation topics were incorporated into the portfolio report. The purpose of the table is to document that the core technical content was preserved while the writing was expanded and professionalized.

Table 2: Mapping of original slide topics to report sections.

Slide	Original Topic	Report Location
1	Project title and team members	Cover page and acknowledgements
2	What is convolution	Section 3, Convolution Operation
3	The plan	Section 4, System-Level Architecture
4	XOR gate	Section 6.1, XOR Gate
5	XOR gate simulation, 0.7 ns delay	Section 6.1 and Timing Summary
6	NAND gate	Section 6.2, NAND Gate
7	NAND gate simulation	Section 6.2
8	Adder	Section 7.1, Adder Function
9	Adder simulation, 1.5 ns delay	Section 7.1 and Timing Summary
10	32-bit adder timing, 33 ns delay	Section 7.2 and Timing Summary
11	Flip-flop register	Section 8, Register and Flip-Flop Design
12	Flip-flop simulation	Section 8
13	8-bit multiplier	Section 9.1, 8-Bit Multiplier
14	Multiplier simulation, 11 ns delay	Section 9.1 and Timing Summary
15	Final test bench	Section 10.1, Final Test Bench
16	Full system simulation	Section 10.2, Full-System Simulation
17	Conclusion points	Section 13, Conclusion