Final Report: FPGA-Based Systolic Array Accelerator for Vertex Transformation

Taegon Hibbitts, Manoj Franklin
Department of Electrical and Computer Engineering
University of Maryland

Abstract

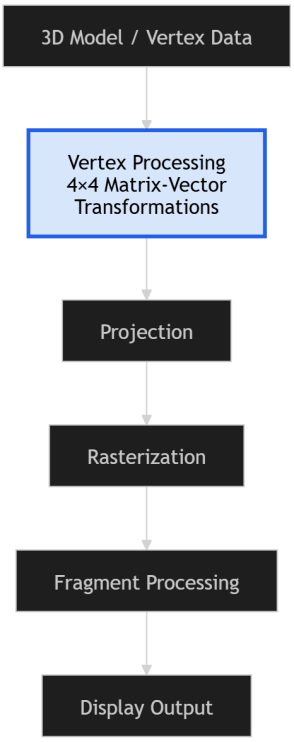
This work presents the design and evaluation of a systolic array–based hardware accelerator for 4×4 matrix–vector transformations on an FPGA, a fundamental operation in graphics vertex processing. The accelerator is implemented in Verilog using fixed-point arithmetic and synthesized for a Xilinx Artix-7 device. The design employs a hierarchical architecture composed of multiply-accumulate (MAC) units organized into processing elements and arranged in a pipelined systolic array. Functional correctness is verified through simulation, while performance is evaluated using cycle-accurate analysis.

Results show that the accelerator achieves a sustained throughput of one result per clock cycle after pipeline fill, corresponding to 100 million operations per second at 100 MHz. Post-implementation power analysis with switching activity estimation reports a dynamic compute power of approximately 9 mW, yielding an energy efficiency of 0.09 nJ per operation. These findings highlight the tradeoff between increased pipeline latency and improved throughput, while demonstrating that pipelined systolic dataflow can achieve high throughput with moderate FPGA resource utilization. Overall, the project shows that systolic architectures can effectively accelerate structured graphics computations while maintaining favorable performance, resource utilization, and energy efficiency.

---

1. Introduction

Matrix–vector transformations are a fundamental operation in computer graphics, forming the basis of vertex processing stages responsible for geometric operations such as translation, rotation, scaling, and projection. These computations are heavily used in real-time rendering pipelines and are traditionally executed on CPUs or GPUs using sequential or SIMD-based approaches. While such platforms provide flexibility and programmability, they do not always fully exploit the structured dataflow and regular computation patterns inherent in linear algebra workloads.



Systolic array architectures provide an alternative execution model that leverages spatial dataflow and localized communication between processing elements to achieve high computational throughput. Originally proposed for efficient matrix computations and more recently adopted in machine learning accelerators, systolic arrays are particularly well-suited for repetitive arithmetic operations such as matrix–vector multiplication. Their regular structure enables efficient pipelining, predictable data movement, and effective utilization of dedicated arithmetic hardware, making them strong candidates for accelerating graphics-oriented workloads.

This project investigates the use of a systolic array architecture to accelerate 4×4 matrix–vector transformations on an FPGA platform. A fixed-point systolic accelerator is implemented in Verilog and synthesized for a Xilinx Artix-7 device. The work evaluates architectural tradeoffs involving latency, throughput, resource utilization, and energy efficiency, while demonstrating how systolic architectures can be extended beyond machine learning applications to classical graphics computations.

***Figure 1**. Simplified graphics pipeline highlighting the vertex processing stage accelerated by the systolic array architecture.*

---

2. Methodology

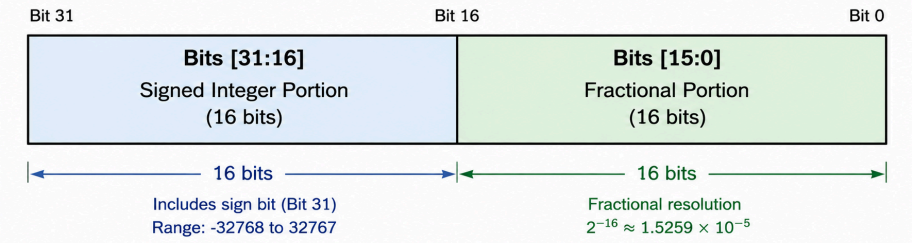
2.1 Fixed Point Arithmetic

To reduce hardware complexity and improve computational efficiency, the design uses fixed-point arithmetic rather than floating-point representation. Fixed-point arithmetic enables efficient mapping to FPGA DSP resources while avoiding the additional latency, area, and control overhead associated with floating-point units.

The implementation uses a 32-bit signed fixed-point format with 16 fractional bits (Q16.16), defined by:

- WIDTH = 32
- FRAC = 16

In this representation, the upper 16 bits correspond to the signed integer portion, while the lower 16 bits represent the fractional component.



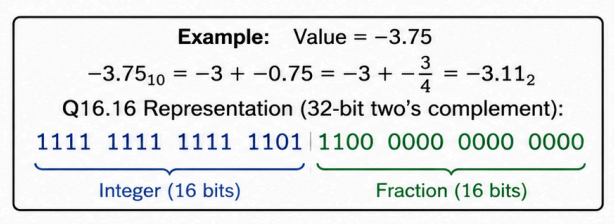


***Figure 2**. Q16.16 fixed-point representation used throughout the accelerator datapath.*

Arithmetic operations are performed using standard fixed-point scaling, where multiplication results are shifted appropriately to preserve numerical precision and maintain a consistent Q16.16 representation throughout the datapath. This format provides a practical balance between dynamic range and precision while remaining well-suited for implementation using FPGA DSP48 blocks.

2.2 System Architecture

The accelerator is organized as a hierarchical dataflow architecture composed of multiply-accumulate (MAC) units, processing elements (PEs), and a 4×4 systolic array that performs matrix–vector transformations.

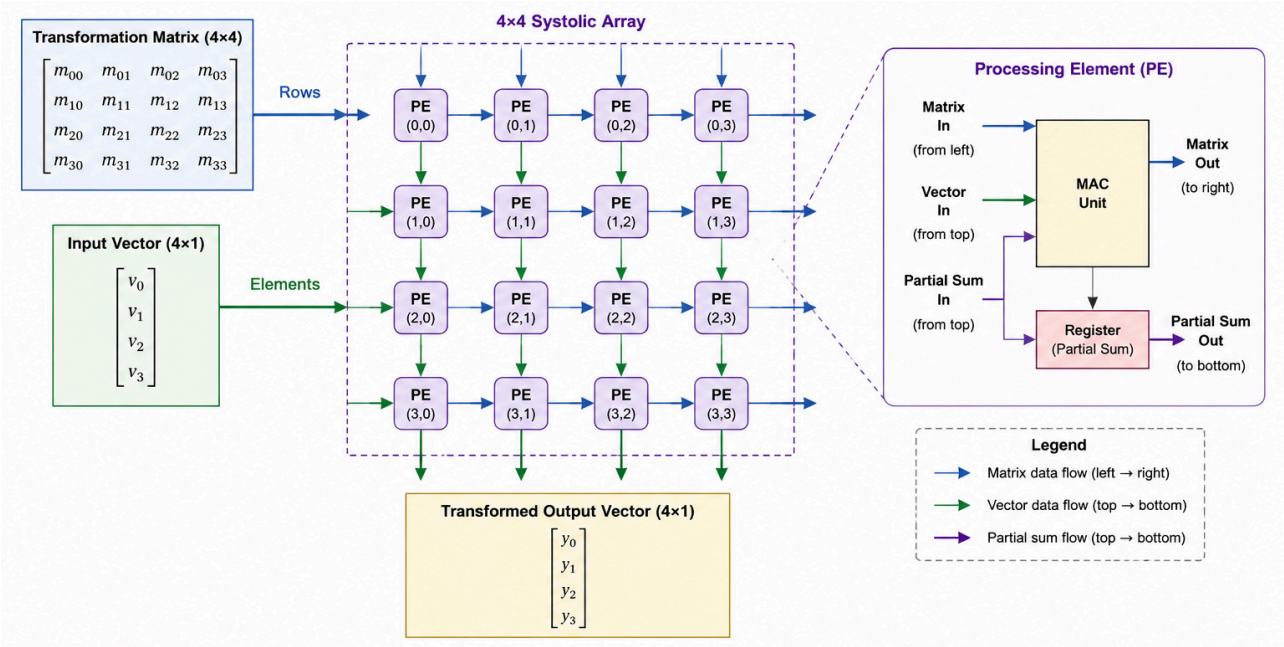


***Figure 3.** Hierarchical organization of the systolic accelerator, showing the relationship between MAC units, processing elements, and the 4×4 systolic array.*

At the lowest level, the MAC unit performs fixed-point multiplication followed by accumulation, forming the fundamental arithmetic operation of the design. Each MAC unit is encapsulated within a processing element, which incorporates pipeline registers and local control logic to support data propagation and intermediate result storage.

The processing elements are arranged into a 4×4 systolic array, where each PE computes a partial contribution to the final matrix–vector product. The overall computation follows the standard formulation:

$$\mathbf{v}_{out} = \mathbf{M} \cdot \mathbf{v}_{in}$$

where $\mathbf{M}$ is a 4×4 transformation matrix and $\mathbf{v}_{in}$ is a 4-component input vector.

Data propagates through the array in a structured and localized manner: matrix elements move horizontally across rows, while vector elements propagate vertically down columns. Each processing element multiplies its incoming operands and accumulates the result with a partial sum received from a neighboring element. After an initial pipeline fill phase, the array produces one output vector per clock cycle, enabling continuous streaming computation with localized interconnect between adjacent processing elements.

2.3 Multiply-Accumulate Unit

The multiply-accumulate (MAC) unit serves as the fundamental arithmetic building block of the accelerator. It performs multiplication followed by accumulation, forming the core operation required for matrix–vector computation within the systolic array.

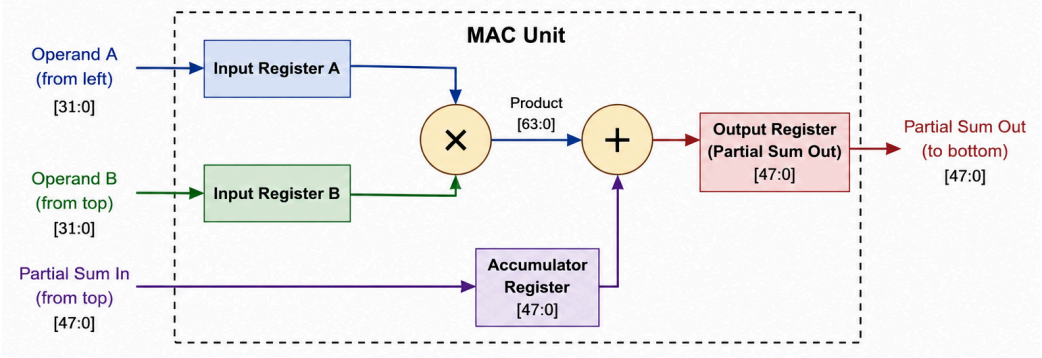


Figure 4. *Block diagram of the multiply-accumulate (MAC) unit used within each processing element of the systolic array.*

The MAC unit accepts two Q16.16 fixed-point inputs and computes their product, which is then added to an accumulated partial sum. To preserve numerical consistency within the fixed-point representation, multiplication results are scaled appropriately through arithmetic shifting before accumulation.

The unit is implemented as a pipelined structure to support high-frequency operation and continuous dataflow. Pipeline registers are inserted between computation stages to reduce critical path delay, improve timing closure, and enable sustained processing at one result per clock cycle after pipeline fill.

Functional correctness of the MAC unit was verified using a dedicated Verilog testbench. Simulation results confirm correct arithmetic behavior, proper fixed-point scaling, and expected pipeline latency characteristics under representative input conditions.

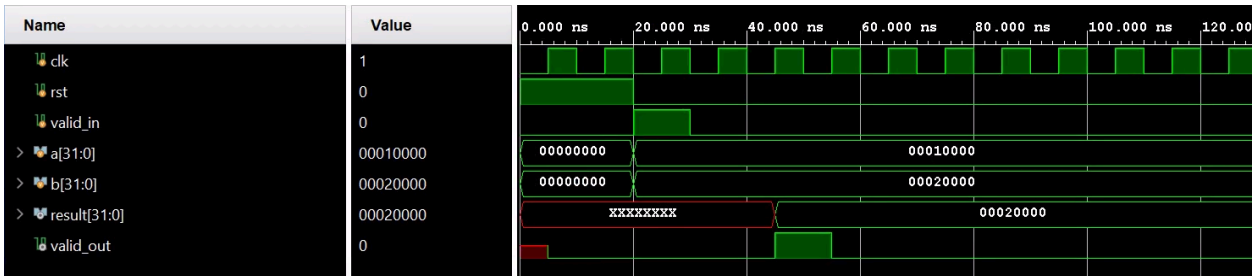


Figure 5: *Verilog simulation waveform of the MAC unit showing correct fixed-point multiplication and pipelined output behavior for representative input operands.*

2.4 Processing Element

Each processing element (PE) integrates a MAC unit with local storage and data forwarding logic, forming the fundamental computational unit of the systolic array. The PE computes a partial contribution to the overall matrix–vector product while maintaining continuous data movement throughout the array.

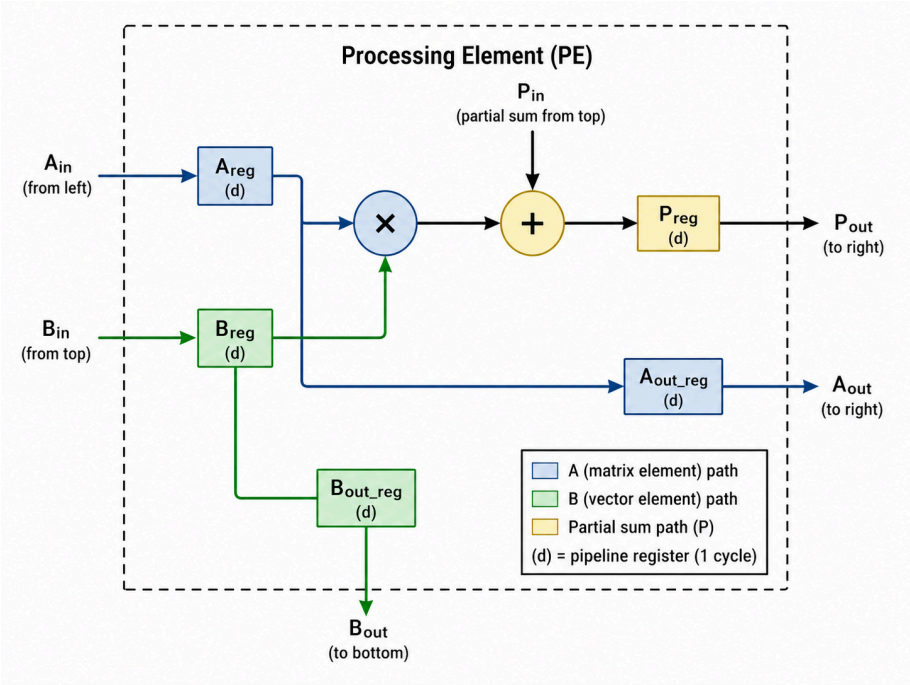


Figure 6: *PE Diagram*

At each clock cycle, the PE receives a matrix value and a vector value, multiplies them using the embedded MAC unit, and adds the result to an incoming partial sum. The updated partial sum is

then forwarded to the next processing element along the accumulation path. Simultaneously, matrix and vector values are propagated to neighboring PEs to support continued computation across the array.

This localized computation and communication pattern enables efficient pipelining and minimizes long-distance data movement. Each PE performs a small portion of the overall computation while forwarding intermediate values to adjacent elements, allowing the array to sustain continuous dataflow and achieve high throughput once the pipeline is filled.

2.5 Simulation and Verification

The design was functionally verified using Verilog testbenches that exercise the systolic array under a range of operating conditions. Multiple test cases were used to validate both arithmetic behavior and dataflow functionality, including identity transformations, nontrivial matrix transformations, signed arithmetic inputs, and saturation scenarios.

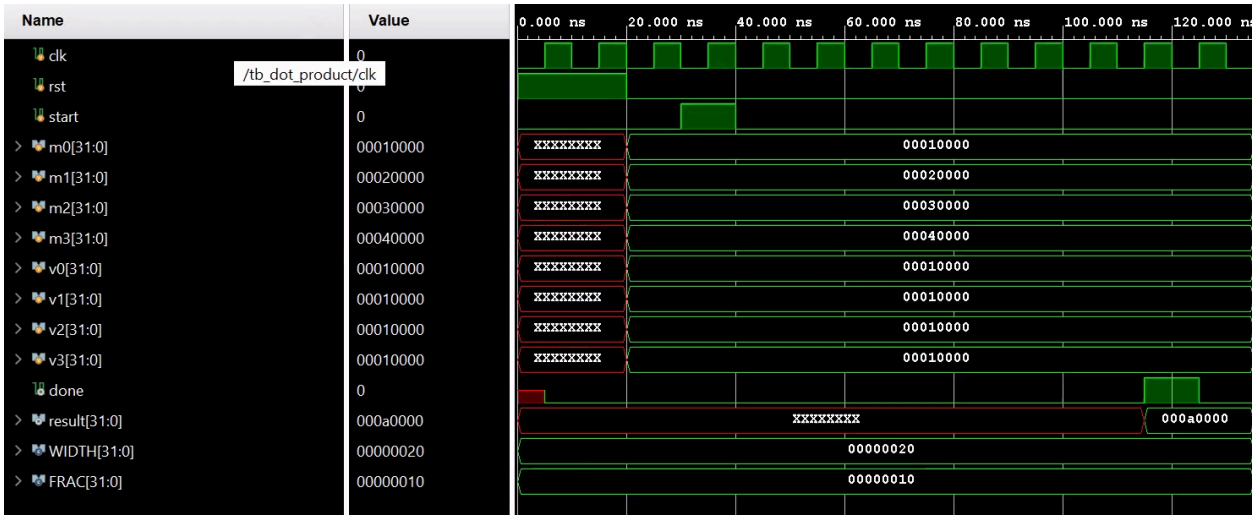


Figure 7: *Dot product simulation waveform*

At the unit level, simulation of individual processing elements and MAC units confirms proper fixed-point multiplication, accumulation, and propagation of intermediate values. At the system level, full-array simulations demonstrate that matrix and vector inputs propagate through the systolic structure as intended, with partial sums accumulating progressively across processing elements.

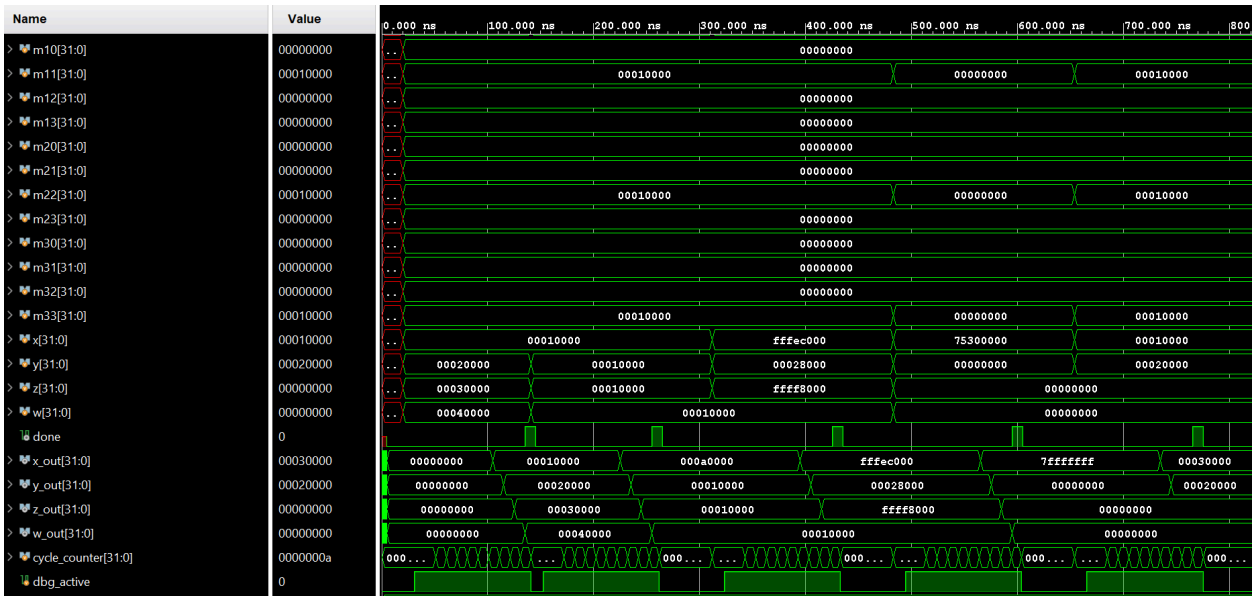


Figure 8: *Full systolic waveform*

Simulation results show that, after an initial pipeline fill phase, the array produces valid output vectors with a fixed latency corresponding to the pipeline depth. Output values match analytically expected results across all tested cases, confirming both functional correctness and expected timing behavior. These observations are consistent with the earlier simulation results presented in the progress report.

2.6 Performance Evaluation

Performance was evaluated using both cycle-accurate hardware simulation and a software-based sequential baseline implementation. The hardware design was analyzed using Verilog testbenches to measure pipeline latency, cycle counts, and steady-state throughput, while a C++ implementation of the same matrix–vector transformation served as a reference for sequential execution performance.

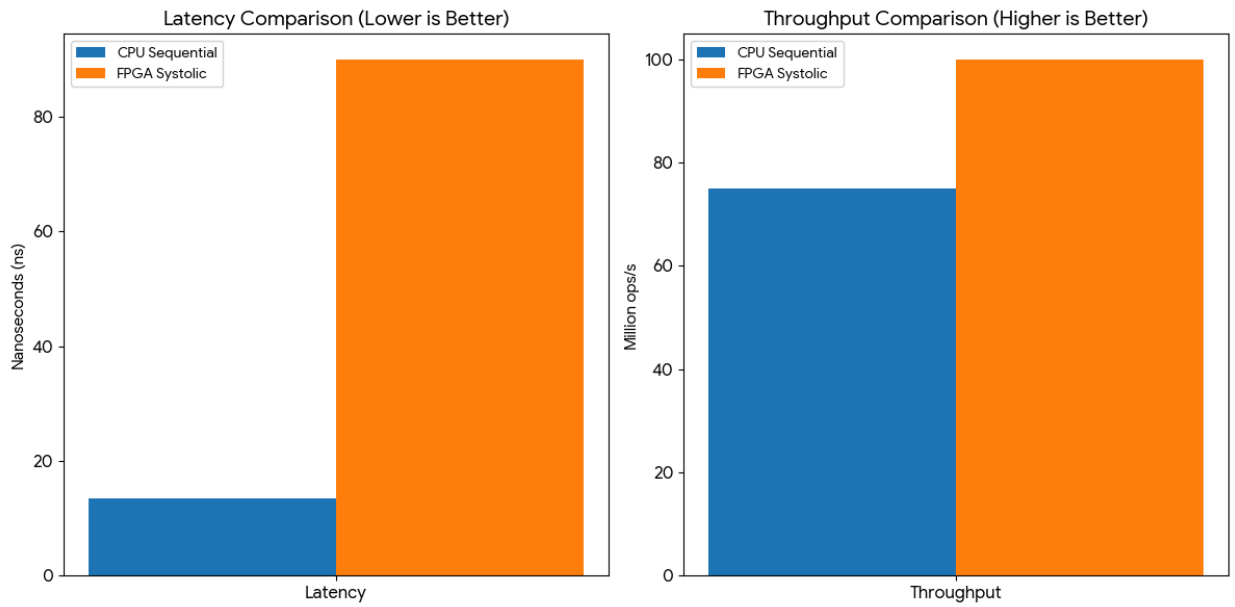


Figure 9: *Comparison of latency and throughput between the sequential CPU implementation and the FPGA-based systolic accelerator. The CPU achieves lower per-operation latency, while the pipelined systolic architecture sustains higher overall throughput after pipeline fill.*

Simulation results show that the systolic array exhibits a fixed latency corresponding to the depth of the computation pipeline, with valid outputs produced after an initial fill phase. Once the pipeline is fully occupied, the design sustains a throughput of one result per clock cycle. At an operating frequency of 100 MHz, this corresponds to a peak throughput of 100 million operations per second.

The C++ baseline executes operations sequentially and provides a reference point for evaluating latency and throughput tradeoffs between software and hardware execution models. This dual evaluation methodology enables a direct comparison between temporal (sequential) execution and spatially pipelined dataflow computation.

2.7 Power Analysis

Power consumption was estimated using Vivado's post-implementation power analysis tool with simulation-derived switching activity annotation (SAIF). SAIF-based activity estimation improves the accuracy of dynamic power analysis by capturing realistic signal transitions and switching behavior during operation.

To isolate the energy cost of computation, the design was evaluated under two operating conditions. In the idle configuration, the accelerator operated in a low-activity state with minimal datapath switching, representing the baseline static consumption of the FPGA fabric. In the active configuration, the systolic array was driven using continuously varying inputs to maintain full pipeline utilization and representative switching activity throughout the datapath.

Dynamic compute power was estimated as the difference between the total on-chip power measured in the active and idle configurations. This methodology separates the incremental energy cost of computation from the underlying static FPGA overhead, enabling a more meaningful evaluation of computational energy efficiency.

---

3. Results

3.1 Performance

Performance results demonstrate the fundamental tradeoff between latency and throughput in the systolic architecture. The FPGA implementation exhibits a fixed latency of 9 clock cycles, corresponding to approximately 90 ns at a 100 MHz operating frequency. This latency reflects the depth of the computation pipeline required to propagate data through the systolic array.

Once the pipeline is filled, the design produces one output vector per clock cycle, achieving a sustained throughput of 100 million operations per second. This steady-state behavior highlights the advantage of pipelined dataflow execution, where computation and data movement are overlapped across multiple processing stages.

In comparison, the C++ sequential baseline completes a single matrix–vector transformation in approximately 13.4 ns, corresponding to an effective throughput of roughly 75 million operations per second. Although the CPU implementation achieves lower latency per operation, it is constrained by sequential execution and therefore cannot sustain the same level of throughput as the pipelined hardware accelerator.

Overall, the FPGA implementation improves throughput by approximately 33% relative to the sequential baseline while trading off increased initial latency due to pipeline depth. These results demonstrate the effectiveness of systolic dataflow architectures for sustaining high computational throughput on structured linear algebra workloads.

Metric	CPU Sequential	FPGA Systolic
Execution Model	Sequential	Pipelined Systolic
Latency per Operation	13.4 ns	90 ns
Throughput	75M ops/sec	100M ops/sec
Pipeline Depth	N/A	9 cycles
Parallelism Style	Temporal	Spatial/Dataflow
DSP Utilization	N/A	16 DSP48E1

Dynamic Compute Power	N/A	9 mW
Energy per Operation	N/A	0.09 nJ/op

3.2 Resource Utilization

Post-synthesis resource utilization indicates that the design remains relatively lightweight in general-purpose logic while making efficient use of dedicated arithmetic hardware. The final implementation utilizes approximately 500 LUTs, 800 registers, and 16 DSP48E1 units, corresponding to roughly 18% of the available DSP resources on the target Artix-7 device.

Resource	Used	Available	Utilization
Slice LUTs	556	20,800	2.67%
Slice Registers	1,339	41,600	3.22%
DSP48E1	16	90	17.78%
Block RAM Tiles	0	50	0.00%
BUFGCTRL	1	32	3.13%
Bonded IOB	7	106	6.60%

The final implementation utilizes 16 DSP48E1 units, matching the expected arithmetic structure of a fully spatial 4×4 systolic array. Each processing element maps closely to dedicated DSP hardware, enabling localized multiply-accumulate operations and continuous pipelined dataflow throughout the array.

Despite the fully spatial organization of the arithmetic datapath, overall FPGA resource utilization remains relatively modest, with only 17.8% of available DSP resources consumed on the target Artix-7 device. This demonstrates that systolic architectures can achieve high computational throughput while maintaining efficient hardware utilization on resource-constrained FPGA platforms.

Overall, the results highlight the effectiveness of localized communication and pipelined dataflow organization in sustaining high-performance matrix–vector computation. By distributing computation across dedicated processing elements, the accelerator achieves continuous streaming execution while minimizing global control overhead and long-distance data movement.

3.3 Power and Energy Efficiency

Power analysis results show a clear distinction between baseline FPGA power consumption and the incremental energy cost of computation. Total on-chip power was measured as 0.063 W in the idle configuration and 0.076 W during active operation.

The dynamic compute power, defined as the difference between active and idle power consumption (minus the static energy), is therefore:

$$WP_{compute} = 0.014 - 0.005 = 0.009 \text{ W}$$

Given a sustained throughput of 100 million operations per second, the corresponding energy per operation is:

$$E/op = \frac{0.009}{100 \times 10^6} = 9.0 \times 10^{-11} \text{ J/op} = 0.09 \text{ nJ/op}$$

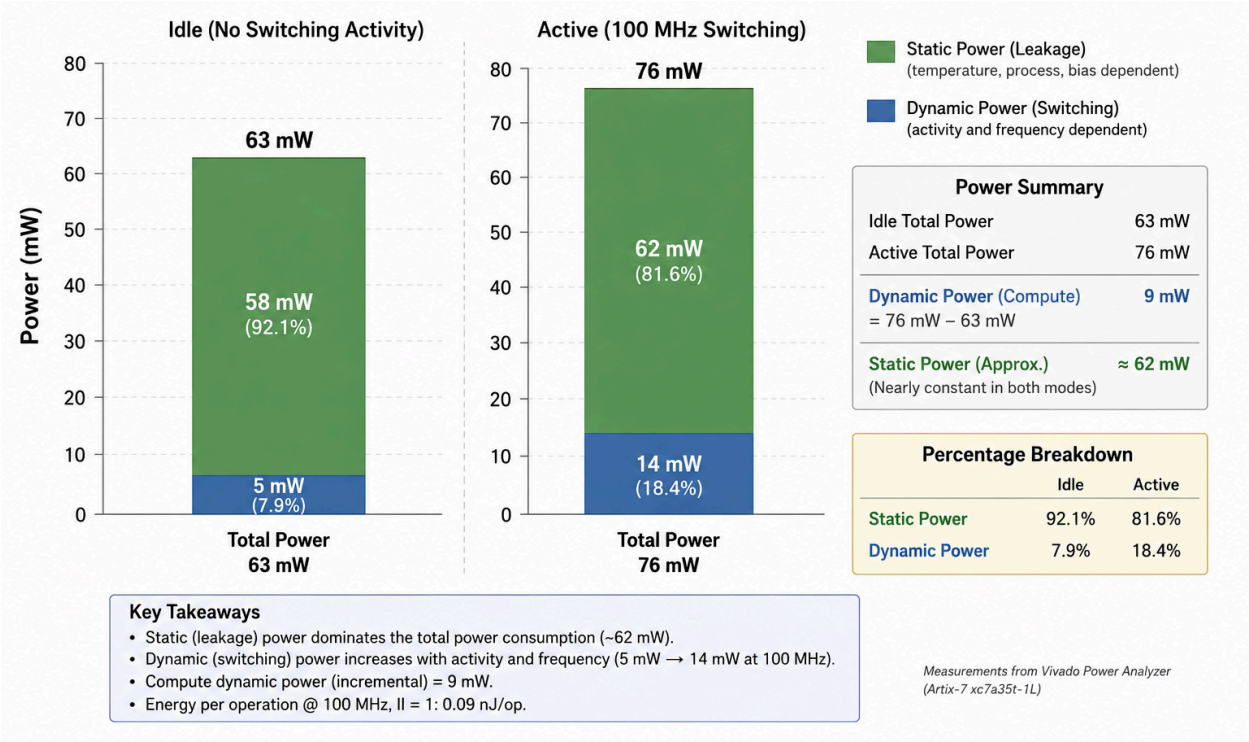


Figure 10: *Static vs dynamic power breakdown*

These results indicate that static FPGA overhead dominates total system power consumption. Approximately 77.5% of the active power corresponds to baseline device power, while only about 22.5% is associated with dynamic switching activity during computation. Relative to the idle configuration, continuous systolic computation increases total power consumption by approximately 23%.

Despite this static overhead, the incremental cost of computation remains relatively low, reflecting the efficiency of the pipelined systolic dataflow architecture.

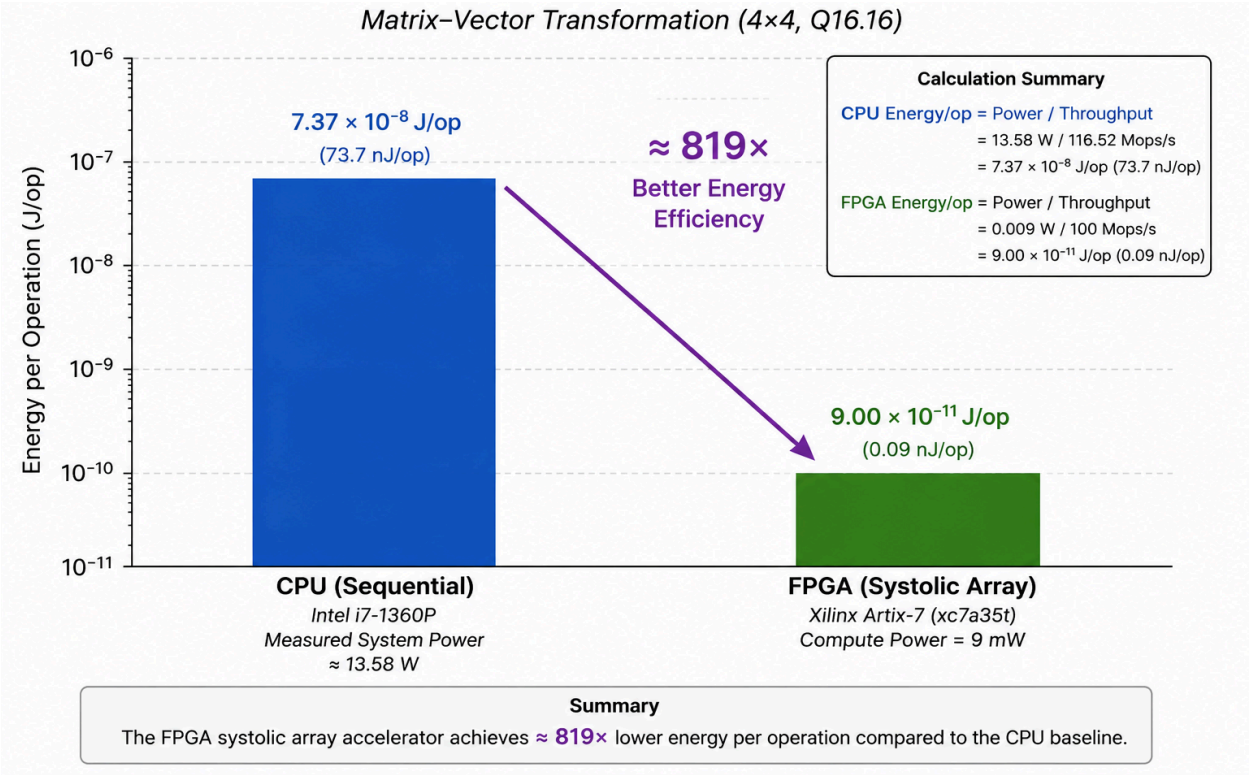


Figure 11: *Energy per Operation Comparison*

The measured energy efficiency demonstrates that the accelerator can sustain structured matrix–vector computation with a low incremental energy cost. This efficiency is enabled by the use of dedicated DSP hardware, localized communication between processing elements, and continuous pipeline utilization, which together minimize control overhead and unnecessary data movement.

---

4. Discussion

4.1 Latency vs Throughput

The performance results highlight a fundamental tradeoff between latency and throughput in different computational models. The CPU-based implementation achieves lower latency per operation, completing a matrix–vector transformation in approximately 13.4 ns through sequential execution with minimal pipeline overhead. In contrast, the systolic accelerator introduces additional latency due to pipeline depth, requiring 9 clock cycles (approximately 90 ns) before the first valid output is produced.

However, once the pipeline is filled, the systolic architecture sustains a throughput of one output vector per clock cycle, enabling a peak throughput of 100 million operations per second. By

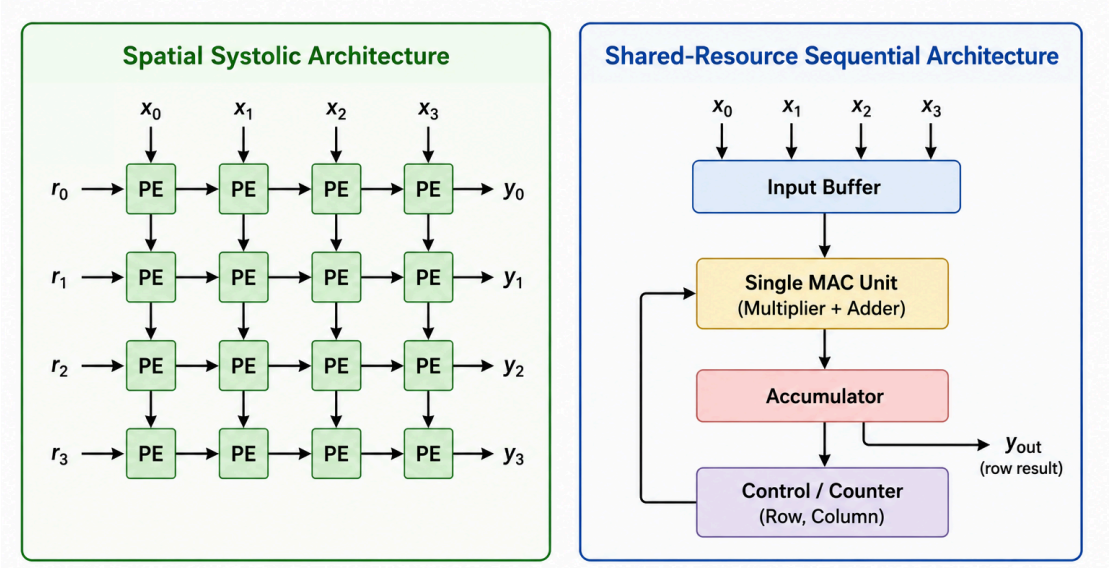
overlapping computation across multiple pipeline stages, the accelerator achieves approximately 33% higher throughput than the sequential baseline despite its higher initial latency.

As a result, the CPU implementation is better suited for isolated or latency-sensitive computations, whereas the systolic architecture is more effective for workloads involving continuous streams of structured data and repeated linear algebra operations.

This distinction reflects a broader architectural principle: sequential systems are typically optimized for low-latency execution, while dataflow-oriented hardware architectures such as systolic arrays prioritize throughput through pipelining, parallelism, and continuous data movement.

4.2 Resource Efficiency

The accelerator achieves full throughput while utilizing 16 DSP48E1 units, corresponding to approximately 18% of the available DSP resources on the target FPGA. This utilization aligns with the expected arithmetic structure of a fully spatial 4×4 systolic array, where each conceptual processing element maps closely to dedicated DSP hardware. Despite this fully spatial organization, the overall FPGA resource utilization remains relatively modest, demonstrating that systolic dataflow architectures can achieve high computational throughput while maintaining efficient hardware utilization on resource-constrained devices.



Characteristic	Spatial Systolic	Shared Resource
Multipliers	16	1
Throughput	High	Lower

Area Usage	Higher	Lower
Dataflow	Parallel	Sequential
Latency Behavior	Pipelined	Iterative

Instead, the synthesized design closely maps the conceptual systolic array structure onto dedicated DSP hardware while leveraging pipelining and localized communication to sustain continuous dataflow across the accelerator. Each processing element performs a portion of the matrix–vector computation in parallel, allowing the architecture to maintain a throughput of one output vector per clock cycle after pipeline fill.

This observation highlights an important principle in FPGA-based accelerator design: high computational throughput can be achieved through structured dataflow organization and efficient pipelining without requiring excessive overall FPGA resource utilization. By distributing computation spatially across dedicated processing elements, systolic architectures enable scalable accelerator designs that balance performance, area efficiency, and hardware constraints on resource-limited FPGA platforms.

4.3 Energy Efficiency

Power analysis reveals that static FPGA overhead dominates total system energy consumption. During active operation, the accelerator consumes 0.080 W total on-chip power, of which approximately 0.062 W (77.5%) corresponds to baseline device power. The remaining 0.018 W is associated with dynamic switching activity, with approximately 0.009 W attributable to computation after subtracting idle-state consumption.

As a result, enabling continuous systolic computation increases total device power by only approximately 23% relative to the idle configuration. This indicates that the incremental energy cost of computation is relatively small compared to the baseline infrastructure overhead of the FPGA platform.

The accelerator achieves an estimated energy efficiency of approximately 0.09 nJ per operation while sustaining a throughput of 100 million operations per second. This efficiency is enabled by pipelined dataflow execution, localized communication between processing elements, and the use of dedicated DSP hardware, which together minimize control overhead and unnecessary data movement.

These results also highlight an important characteristic of FPGA-based acceleration: the majority of measured energy consumption is associated with the underlying programmable fabric rather than the computation itself. In practical accelerator deployments, such baseline costs are typically amortized across large workloads or integrated within larger heterogeneous compute systems. Consequently, dynamic compute power and energy per operation provide a more meaningful indicator of architectural efficiency than total device power alone.

5. Conclusion

This project demonstrates the viability of using systolic array architectures to accelerate matrix–vector transformations for graphics-oriented workloads on FPGA platforms. A fixed-point, pipelined systolic accelerator was designed in Verilog, implemented on a Xilinx Artix-7 FPGA, and evaluated in terms of performance, resource utilization, and energy efficiency.

The final implementation achieves a sustained throughput of 100 million operations per second while consuming approximately 9 mW of dynamic compute power, corresponding to an energy efficiency of roughly 0.09 nJ per operation. Despite utilizing only 16 DSP48 units, the accelerator maintains full throughput through efficient pipelined dataflow and localized communication between processing elements.

The results highlight several important architectural tradeoffs between sequential and systolic execution models, including the balance between latency, throughput, hardware utilization, and energy efficiency. Although the systolic architecture introduces additional pipeline latency, it achieves substantially higher sustained throughput through parallel dataflow execution and localized communication between processing elements.

Overall, this work demonstrates that systolic architectures can effectively accelerate structured graphics computations in addition to machine learning workloads. The project also illustrates how careful dataflow organization and pipeline scheduling can achieve strong computational performance with moderate hardware resources. Future work may include scaling the array size, exploring ASIC-oriented implementations, and integrating memory hierarchy optimizations to support larger and more complex workloads.

---

6. References

1. Hennessy, J. L., & Patterson, D. A. *Computer Architecture: A Quantitative Approach*
2. Kung, H. T. “Why Systolic Architectures?” IEEE Computer, 1982
3. Xilinx Vivado Design Suite User Guide
4. FPGA DSP48E1 Slice Documentation
5. UMD ENEE 499L course materials