**Pipelined MACs with Wallace Tree Architecture  
EE232 Coursework 2: Vector-Vector Multiplier**

Done By: Kieran Chew (Reg No. 240133326)

**Introduction**

Vector-vector multiplication is critical in modern AI accelerators, digital signal processors and edge compute platforms. These systems frequently require the computation of large dot products often exceeding 1024 elements. However, instantiating a massive number of multipliers in hardware is infeasible due to area and timing constraints on FPGAs.

This report presents a hardware-efficient approach to vector-vector multiplication using a scalable pipelined architecture. Multiple design iterations were explored to balance throughput, latency and logic utilization. The final architecture combines pipelined Multiply-Accumulate (MAC) units in parallel with a fully pipelined Wallace Tree. This setup delivered high throughput in testing, with minimal area overhead, and maintained good clock scalability and reliable accumulation.

*A diagram of a stage

AI-generated content may be incorrect.  
Final Architecture Diagram (i.e. 8 MACs)*

**Design Goals & Architectural Considerations**

The final architecture chosen was designed to meet the following criteria:

* *Supports worst-case vector multiplication scenarios*  
  Able to handle worst-case dot product of 1024 elements × 255 × 255. It would incur a value of 66,585,600 which requires at least a 27-bit register.
* *Optimized Multiplier Design*Uses a Pipelined Shift-and-Add for INT8 multiplication over multiple cycles
* *Configurable Parallelism*Number of MACs used is fully parameterizable *‘NUM\_MACS’*, allowing the system to scale with available FPGA resources while maintaining functional consistency
* *Final Accumulation*A pipelined accumulation scheme adds partial sums generated from compressed operands, supporting continuous throughput with minimal stalling.
* *Scalability and Timing Closure*Design was verified to meet timing up to 512 MACs reliably
* *Efficient Summation*Employs a pipelined Wallace Tree using Carry Save Adders (CSAs) and internal accumulation stages to compress and sum parallel MAC outputs into a single batch result.

**Implementation of Components (Final Architecture)**

The system implements a modular, parameterized, and pipelined architecture for vector multiplication. The top-level module accepts flattened INT8 input vectors of configurable length and computes the dot product using a scalable number of parallel MAC units, which are pipelined and unrolled to deliver high throughput. The results are passed to a multi-stage pipelined Wallace Tree, which compresses and accumulates the outputs into a batch result.

The design employs batch pipelining to support scalability across 1 to 256 MACs reliably by structurally separating computation into three distinct modules:

* *Pipelined MAC Unit [MAC\_pipelined.v]*  
  Implements an unrolled Shift-and-Add Multiplier across 2 pipelined stages. It accepts two 8-bit unsigned inputs and produces a 16-bit output result. Internally:
  + *Stage 1 –* Latches 8-bit inputs (*A*, *B)* and performs multiplication via 8 conditional Shift-and-Add operations. The 16-bit result is stored in *mult\_result*
  + *Stage 2 –* Latches *mult\_result* to *result* and asserts a *done* flag ‘

Internal control signals *stage1\_valid* and *stage2\_valid* manage the flow of data between stages. Once enabled, each MAC can process a new input pair every cycle.

**Design Justification:** A rolled sequential MAC was considered but rejected due to high latency and low throughput. The unrolled combinational design aligns with real-word application (e.g. DSPs, AI accelerators), providing one result per cycle per MAC. Although initial area cost is higher, the architecture scales well up to 256 MACs reliably.

* *Wallace Tree Reduction [WallaceTree.v]*Accepts N 16-bit MAC outputs as a flattened bus and processes them using a 4-stage pipelined architecture with handshake control (*in\_valid, in\_ready, out\_valid*).
  + Stage 0 – Extends all 16-bit MAC outputs to 32-bit for uniformity
  + Stage 1 – Groups inputs in triplets and compresses them using 3:2 Carry Save Adders (CSAs)
  + Stage 2 – Merges sum and carry values using left-shift and addition
  + Stage 3 – Accumulates all intermediate values to produce a final 32-bit batch sum

**Design Justification:** A Recursive Wallace Tree with multiple CSA layers was considered but dismissed to reduce timing and routing congestion. The chosen pipelined approach introduces controlled latency but improves throughput and timing closure. This design supports high MAC counts and scales efficiently with deeper pipelines.

* *Top Level Module [Parallel\_Vector.v]*Orchestrates vector batching and accumulation across parallel MACs. It supports configurable vector length (*VECTOR\_*SIZE) and parallelism (*NUM\_MACS*). It proceeds as follows:
  + Inputs are 8-bit × No. Vector Elements flattened vectors that are unpacked internally into 8-bit elements using generate loops
  + Six-State FSM that manages the Datapath:
    - IDLE – Waits for *start* signal, resets *idx*, *result* and *cycle\_count*
    - LOAD – Enables MACs for one cycle
    - WAIT – Monitors *done\_flags* from all MACs
    - LATCH – Latches results into *mac\_results\_latched*
    - RUN – Feeds latched data into Wallace Tree, receives 32-bit *batch\_sum* and updates running result
    - DONE – Asserts *done* when all computation is complete
  + Registers used:
    - *idx [9:0]* – Tracks current vector offset
    - *cycle\_count [9:0]* – Tracks number of completed batches
    - *result [31:0]* – Final dot product result
    - *mac\_results\_flat* = Flattened 16-bit outputs to Wallace Tree

**Design Justification:** The controller implements batch pipelining through parameterized parallel MAC instantiation, enabling segmented processing of arbitrary vector sizes using the available MACs. Flattened input unpacking adds fixed logic overhead but amortizes as vector size increases. The FSM ensures safe pipelining and reliable control flow. This approach was verified through simulation and synthesis to scale across 1 to 256 MACs.

**Module Testing and Verification**

To validate the functionality, correctness and scalability of the final architecture, a comprehensive testbench *tb\_Parallel\_Vector.v* was developed. Prior to system integration, each individual module was independently verified to ensure correct functional behaviour and that all internal signals performed as expected when assembled into the top-level module.

The final system-level testbench is fully parameterized and supports vector lengths of up to 1024 elements and MAC counts ranging from 1 to 256 reliably, following each respective FMAX.

Key verification features:

* *Functional Correctness* – Compares the computed dot product with a pre-calculated  
  expected result and reports PASS/FAIL via console output
* *Cycle Counting and Latency Measurement* – A cycle counter measures the number of clock cycles from *start*/*done* signals, allowing precise latency evaluation
* *Configurable Parameter Testing* – Tests were performed across various configurations of *NUM\_MACS = 1 to 128* and *VECTOR\_SIZE = 1024*  to ensure scalability and correctness
* *Timing Behaviour and FSM coverage* – All FSM states were exercised and verified through simulation, ensuring correct state transitions and reset behaviour
* *Console Output and Traceability* – Expected result, performance metrics, intermediate and final results are printed using *$display* to aid traceability and debugging

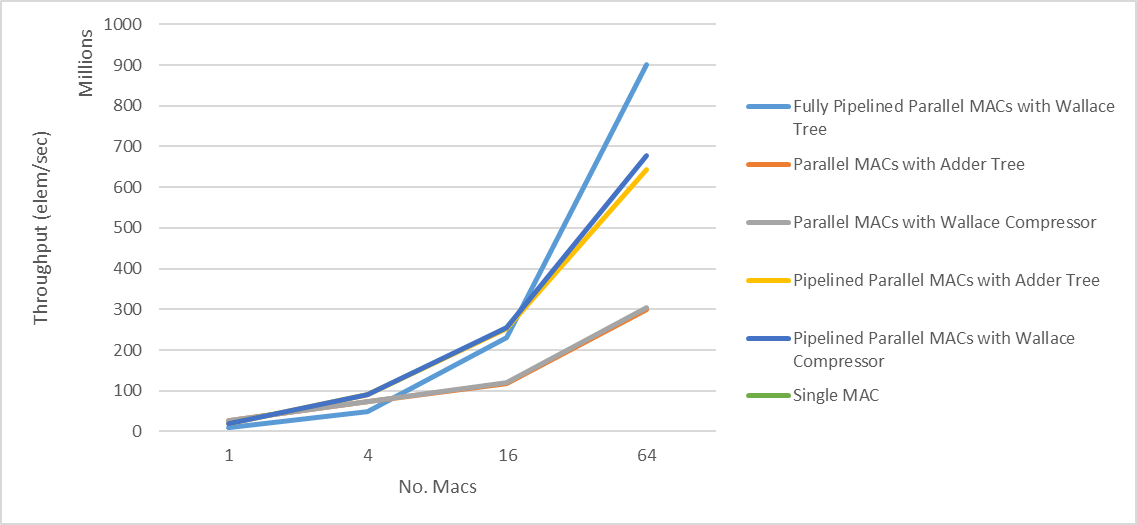
**Analysis of Design Iterations**

To determine the most effective architecture, six different implementations were designed, synthesized, and simulated. Each iteration explored a different balance of throughput, area, and latency. The modules ranged from simple single-MAC systems to fully parallel architectures using either Adder Trees or Wallace Trees, with varying levels of pipelining and compression depth.

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteration** | **Architecture** | **Characteristics** | **Outcome** |
| *1* | Single MAC | Sequential MAC that computes over 8 cycles | Baseline design; Low area and throughput |
| *2* | Parallel MACs + Adder Tree | Parallel Sequential MACs with Adder Tree | Improved throughput but long critical path |
| *3* | Parallel MACs + Wallace Compressor | Parallel Sequential MACs with Wallace-style reduction | Reduced adder depth; Better FMAX |
| *4* | Pipelined MACs + Adder Tree | Pipelined Parallel MACs with Adder Tree | Higher throughput per MAC; Limited by Adder Tree |
| *5* | Pipelined MACs + Wallace Compressor | Pipelined Parallel MACs with Wallace-style reduction | Good throughput and area efficiency |
| *6* | Fully Pipelined Wallace Tree | Pipelined MACs with handshake-enabled multi-stage Wallace Tree | Best overall performance and FMAX; highly scalable |

*.  
Architecture Comparison Table*

* *Throughput Scalability*



*Throughput Scaling as Maximum FMAX*

From the chart above and accompanying data table, pipelined designs scale significantly better than unpipelined ones as the number of MACs increases. At 64 MACs, the Fully Pipelined Parallel MACs with Wallace Tree design reaches over 900 million elements/sec, achieving the highest throughput overall. In contrast, unpipelined designs plateau early due to dataflow bottlenecks. Additionally, designs using Wallace Tree summation consistently outperform those using Adder Trees, whether pipelined or not. The introduction of a fully pipelined Wallace Tree further improves performance by enabling handshake-driven reduction and sustained batch processing.

In conclusion, pipelining combined with Wallace Tree compression enables the most scalable high-throughput performance among all tested architecture

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MACs** | **1** | **4** | **16** | **64** |
| **Single MAC** | 12 | - | - | - |
| **Parallel MACs + Adder Tree** | 26 | 74 | 117 | 300 |
| **Parallel MACs + Wallace Compression** | 26 | 74 | 121 | 304 |
| **Pipelined Parallel MACs + Adder Tree** | 19 | 90 | 253 | 643 |
| **Pipelined Parallel MACs + Wallace Compression** | 19 | 91 | 255 | 678 |
| **Fully Pipelined with Wallace Tree** | 10 | 50 | 231 | 900 |

*.  
Throughput (Million elements/second) Scaling at Maximum FMAX*

**Final Evaluation**

After testing six different architecture iterations, I selected the final design, Pipelined MAC units with a fully Pipelined Wallace Tree, based on consistent improvements in throughput and timing closure. A total of six architectural variants were designed, synthesized, and simulated, each exploring different trade-offs between throughput, latency, logic utilization, and scalability.

To determine the optimal design, two key metrics were evaluated:

* Throughput (elements/second): How fast the design can compute dot products
* Area usage (LUTs): The FPGA logic resources consumed

As shown in the chart above, throughput scales nearly linearly with the number of MACs, while LUT usage rises steadily. Among all tested architectures, the Fully Pipelined Wallace Tree offers the best balance between speed and resource efficiency. It achieves the highest FMAX and throughput per MAC, while maintaining predictable scaling characteristics.

However, selecting the ideal number of MAC units for deployment requires careful consideration of the throughput-to-area ratio. While performance continues to increase beyond 128 MACs, the marginal gains diminish due to area constraints and tighter timing closure. Therefore, design decisions must balance resource availability with desired performance levels for practical FPGA deployment.

|  |  |  |  |
| --- | --- | --- | --- |
| **MACs** | **LUTs** | **Throughput (Million elements/sec)** | **Throughput per LUT (Thousand elements/sec)** |
| 32 | 7426 | 503.65 | 67.82 |
| 64 | 9615 | 900.54 | 93.66 |
| 128 | 14864 | 1264.59 | 85.08 |
| 256 | 25424 | 2133.333 | 83.91 |

*.  
Final Architecture Throughput Efficiency Table*

A graph with a line and a line

AI-generated content may be incorrect.

*Throughput vs Area Trade-off Across Varying MAC Configurations (Final Architecture)*

From the chart above, both throughput and LUT usage increase as more MAC units are added. However, the throughput gain per LUT begins to diminish beyond 64 MACs. For instance:

* From 1 to 64 MACs – Throughput increases significantly with relatively modest linear area growth
* From 64 to 256 MACs – Throughput continues to improve but begins to plateau due to rising area usage and tighter timing constraints, as deeper pipelined logic increases critical path management overhead

The 64-MAC configuration marks the optimal balance between performance and area efficiency. At this point, the design reaches approximately 900 million elements per second with the highest throughput per LUT. Beyond this, throughput gains diminish relative to resource usage, indicating a reduced return on area investment. Under real-world FPGA constraints such as LUT budgets, power consumption, and timing closure, 64 MACs offer the most practical and scalable design point for high-throughput embedded applications like DSP pipelines and AI inference workloads.

A key contributor to this performance is the fully pipelined Wallace Tree, which retains the compression benefits of Wallace-style reduction while enabling scalable, handshake-driven pipelining across multiple stages. Its modular, latency-friendly structure integrates cleanly with the data path and controller, reflecting modern DSP and accelerator designs. Architectures found in platforms such as AMD Versal AI Engines and Intel Agilex DSP blocks adopt similar principles, using deep pipelined multiplier-accumulator chains and staged compression to sustain high throughput under tight timing constraints. By mirroring these practices, the implemented design achieves production-grade scalability and timing closure across a wide range of real-world embedded applications where resource efficiency and deterministic latency are critical.