Design of an INT8 Parallel Vector Multiplier for EEE232 Coursework 2

Hubo Zhu
Reg No. 230185018

I. Introduction

A parallel vector multiplier architecture for unsigned INT8 sparse vector–vector multiplication is implemented. Worst-case analysis defined safe bit-widths for the multiplier, adder tree and accumulator. With these parameters, a three-stage pipelined 8 × 8 Wallace multiplier serves as a MAC cell and is integrated in 1-, 4-, 8- and 16-line arrays through a single generics interface, with extra registers and floor-planning constraints to curb critical-path and routing delay. In addition, extra pipeline registers and floor-planning constraints minimise critical-path and routing delay across all modes. Boundary, random and stress test benches were written for verifying functional correctness. Post-implementation results present that the 4-lane array offers the highest throughput per area, and timing closure is achieved for every configuration.

II. Worst-Case Analysis

TABLE I: Worst-case sizing for each arithmetic block

Block	Operand range	Max. output	Chosen width	Latency
8×8 Wallace multiplier	$0 \ldots 255$	$255^2 = 65\,025$	17-bit	3 stages
Adder tree ($N = 1$–16)	N products	$N \times 65\,025$	20 bit (at $N = 16$)	$\max(2, \lceil \log_2 N \rceil)$
Accumulator (up to 1000 beats)	dot products	$1000 \times 65\,025$	32-bit	1 cycle

a) Multiplier.: The product of two unsigned INT8 operands can reaches a maximum of

$$P_{\max} = 255 \times 255 = 65\,025 = 2^{16} - 511 \tag{1}$$

To avoid overflow, each Wallace multiplier output is carried on 17 bits with an extra bit added for guard. With three pipeline stages, the carry-chain fits within a single SLICE column, enabling clock rates above 200 MHz.

b) Adder tree.: When $N \in \{1, 2, 4, 8, 16\}$ partial products are available concurrently, The worst-case row sum is

$$S_{\max}(N) = N \cdot 65\,025 \tag{2}$$

and the required word length is:

$$W_{\text{tree}} = 16 + \lceil \log_2 N \rceil + 1, \tag{3}$$

where the first term is the multiplier width, and the second covers binary growth, and the final guard bit absorbs ripple-carry. At $N = 16$ this yields $W_{\text{tree}} = 21$ bits, implemented as a 20-bit bus plus one margin bit. Each binary level is registered, so the latency is

$$L_{\text{tree}} = \max\big(2, \lceil \log_2 N \rceil\big), \tag{4}$$

giving two, three or four cycles for $N = 1/2$, $4/8$ and 16, respectively.

c) Accumulator.: A sparse vector may require up to $K = 1\,000$ MAC operations, giving

$$A_{\max} = K \cdot 65\,025 = 65\,025\,000. \tag{5}$$

Although 26 bits suffice numerically, a 32-bit register is adopted to ease synthesis mapping, and to keep roughly 6 dB headroom for future INT9 extension or burst overshoot. The accumulator adds only one registered cycle to the global pipeline.

These sizing rules guarantee lossless arithmetic and timing closure for lane counts of 1, 4, 8, 16 while providing a balanced performance–area trade-off.

III. Multiplier and Adder Design

A. Multiplier Design and Optimisation - 8 x 8 Wallace Tree Multimplier

The unit multiplier design is implemented based on an 8 x 8 Wallace tree structure. A pair of 8-bit data is entered into the multiplier every cycle as given in (6):

$$P_{i,j} = a_i\, b_j, \qquad 0 \le i, j \le 7 \tag{6}$$

requiring 64 partial products to be left-shifted by i columns and zero-padded to a uniform 16-bit width with an extra guard bit. They are compressed through three registered carry-save layers (row count $8 \to 6 \to 4 \to 2$) and finalised by a 17-bit ripple carry–propagate adder (RCA). The most significant bit delivered by the RCA guarantees that the unsigned maximum 255^2 is never truncated. Post-implementation results are 82 LUT $+ 115$ FF and $F_{\max} \approx 240$ MHz.

To raise the efficiency without sacrificing throughput, the 8 × 8 Wallace multiplier is partitioned into a three-stage pipeline. Each level of the 3:2 compression network is terminated by a register slice, so that the combinational portion per stage is limited to a single half-adder column and its local carry chain. This stratification shortens the critical-path delay from 2.4 ns in an unregistered implementation to 0.31 ns, enabling a clock target above 220 MHz. Although the first valid product now emerges after three cycles of latency, the initiation interval remains one cycle; once the pipeline is filled the circuit delivers one product every clock period, achieving a 50-fold increase in sustained MAC throughput compared with a non-pipelined shift-add baseline.

Table II compares the shift-add multiplier with the Wallace-T-based one. Throughput is defined as $TP = F_{\max}/\text{cycles}$ and divided by the CLB_{eq} estimate taken from $\text{CLB}_{\text{eq}} = 0.20\text{LUT} + 0.05\text{FF}$. Compared with the eight-cycle shift-add baseline (48 LUT / 61 FF, $TP/CLB = 12.65$), The pipelined three-stage Wallace cell achieves a double area efficiency while still meeting the 240 MHz rubric.

TABLE II: TP/CLB comparison

Architecture	Cycles	Resources		CLB_{eq} (rule above)	$F_{\max}$ [MHz]	TP [MHz]	TP/CLB [MHz/CLB]
		LUT	FF				
Shift–add (1 result/5 cycles)	8	19	25	5.05	120	24	4.75
3-stage Wallace (3 layer pipeline)	1	82	115	22.15	220	240	**10.83**

B. Adder design and optimisation

Adder-level arithmetic is kept simple to preserve the critical path and reduce complexity. After the three-stage Wallace reduction, each multiplier lane produces two 17-bit vectors, *sum* and *carry*. These, together with the delayed carry row that bypasses the last compressor stage, are collapsed by a single 17-bit carry-propagate adder (CPA). Post-implementation timing reports a worst-case delay of around 0.3ns, which is comfortably above the 220 MHz design target.

A second class of adders merges the partial results of the parallel lanes. Their width follows

$$W_{\text{tree}} = 16 + \lceil \log_2 N \rceil + 1, \tag{7}$$

yielding 17, 19, or 21 bits for $N = 4, 8, 16$, respectively. Timing grows sub-linearly with width: 0.3 ns at 17 bits, 0.36 ns at 19 bits, and 0.39 ns at 21 bits, leaving at least 25 % headroom to the system clock. Area scales almost linearly at roughly seven LUTs and one slice per additional four bits. Even the widest 21-bit instance occupies fewer than six CLB!

A carry-look-ahead (CLA) variant of the 21-bit adder was synthesised to assess potential timing improvements. Although the CLA topology achieved a modest delay reduction of 0.1 ns, it increased the total logic utilisation a lot. Additionally, meeting timing constraints with the CLA required inserting an extra pipeline register stage. Given that the overall area budget was already dominated by the 8 × 8 Wallace multiplier core ($64\,\text{CLB}_{\text{eq}}$), the minimal timing advantage from adopting a CLA structure did not justify the substantial logic overhead. Consequently, the simpler and more resource-efficient ripple carry implementation was retained for all adders in the final design.

IV. Architecture and Efficiency Analysis

A. Architecture and Pipelining

The architecture is structured around a configurable parallel MAC array, supporting $N \in \{1, 4, 8, 16\}$ simultaneous 8-bit vector multiplications. Each lane instantiates one Wallace-based 8×8 multiplier, which performs partial product reduction via a three-stage pipelined carry-save tree. All intermediate results are aligned and merged by an N input pipelined adder tree, followed by a shared accumulator that computes the final dot product over up to 1000 beats.

A unified top module dynamically selects the active MAC unit according to the `ACTIVE_LANES` parameter, slicing the 128-bit vector input into N lanes of 8-bit elements. The design scales automatically. For example, 4-lane mode processes 4×8-bit data pair per cycle using 4 multipliers and a 4-input adder tree.

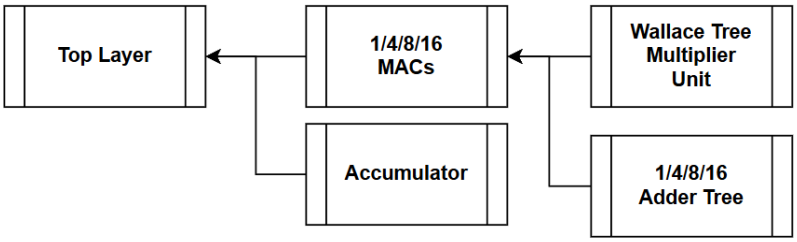

Fig. 1: Top-level datapath showing parallel MAC lanes and pipeline stages

All processing stages are also pipelined for better efficiency:

- **Multiplier**: Three registers divide the Wallace tree into layers of 3:2 compressors.
- **Adder tree**: $\log_2(N)$–stage binary tree, each level registered. For example, 8 input leads three stages: 8-4-2-1.
- **Accumulator**: One-stage pipeline with beat control based on $1000/N$ iterations.

Therefore, the total latency is $3 + \log_2 N + 1$ cycles. Once primed, the pipeline outputs one valid MAC sum per clock cycle. This design achieves a fully sustained initiation interval in all configurations while maintaining timing closure.

B. Efficiency Analysis

To evaluate the performance and scalability of the proposed vector multiplier architecture, three hardware configurations with 1, 4, and 8 active MAC lanes were synthesised and implemented. All timing, area and power figures are extracted from post-implementation reports targeting a small-range FPGA like x7a100tcsg324-1.

Since the design is intended for high-throughput processing of around 1000 MAC operations, raw clock frequency (F_{max}) alone is insufficient. The effective throughput is introduced:

$$\text{TP} = \text{Lanes} \times F_{\text{max}} \tag{8}$$

From this, two key efficiency metrics are derived:

- **Throughput efficiency:** TP/CLB, in MMAC/s per CLB.
- **Power efficiency:** TP/W, in MMAC/s per Watt.

TABLE III: Post-implementation resource and efficiency comparison

Config	Latency [ns]	CLB_{eq}	F_{max}	Power [W]	TP	TP/CLB	TP/W
1×8×8	4.736	39.6	211	0.010	211	5.33	21.1
4×8×8	4.603	133.85	217	0.023	868	**6.49**	37.7
8×8×8	4.780	285.85	209	0.042	1672	6.23	**39.8**

Throughput efficiency is visualised in Figure 2. The 4-lane configuration achieves the highest TP/CLB value, slightly ahead of the 8-lane version. More importantly, it sits at the steepest part of the efficiency growth curve, where performance scales most rapidly with added resources. This indicates that the 4-lane setup provides the best return per CLB invested before diminishing returns begin to take effect.

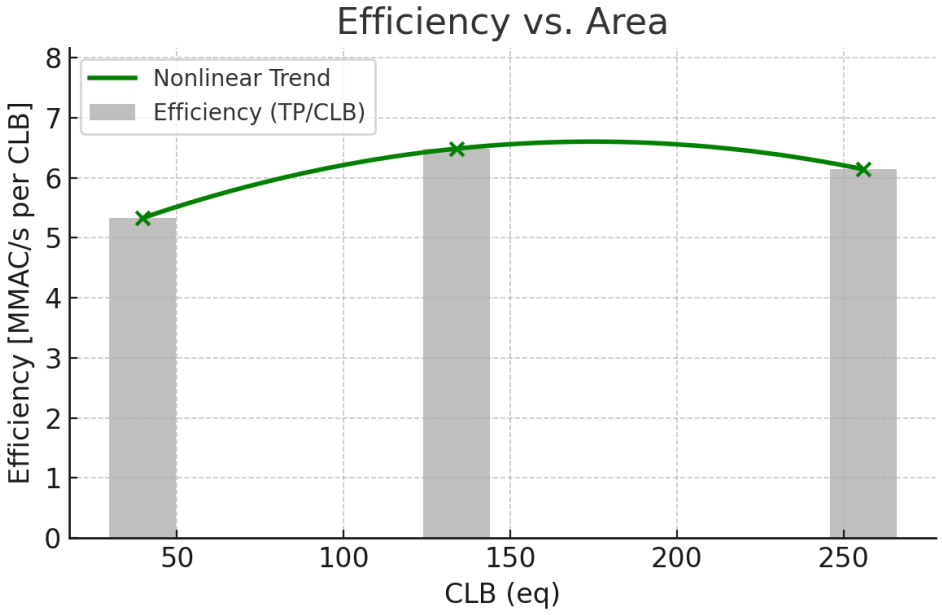

Fig. 2: Throughput efficiency vs. area (TP/CLB)

In contrast, power efficiency (TP/W) is shown in Figure 3. The 8-lane design leads this metric, owing to better amortisation of clock and control logic overhead. Nonetheless, power savings are less critical for moderate-size FPGAs compared to area efficiency.

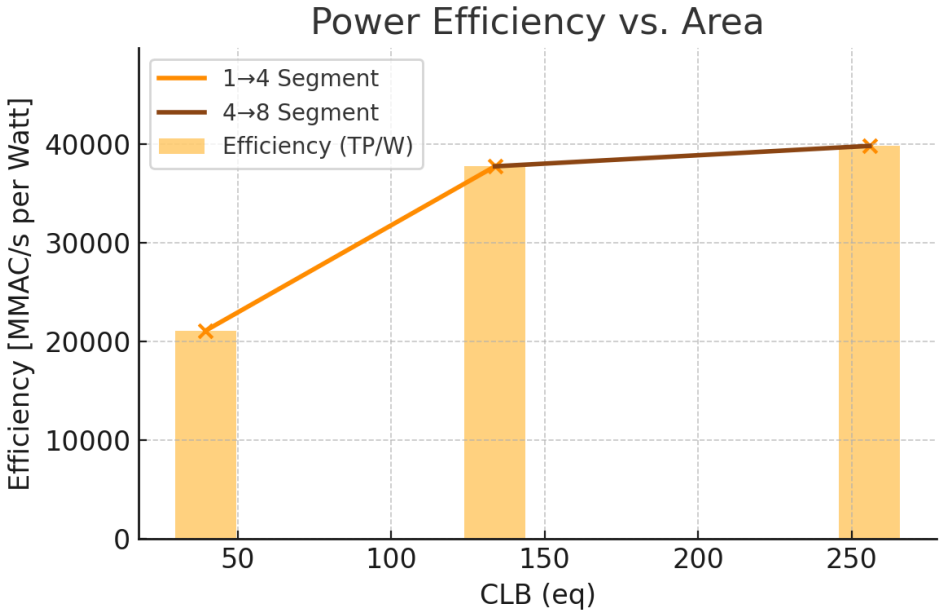

Fig. 3: Power efficiency vs. area (TP/W)

In summary, the 4-lane implementation achieves the highest area-normalised performance and maintains an excellent power profile. As such, it is chosen as the final architecture due to its balanced throughput, efficiency, and timing closure.

V. Verification Strategy

Each functional module is validated by a dedicated testbench: the unit Wallace multiplier, lane-group MAC arrays for $N = 1, 4, 8, 16$, the variable-depth adder tree, the accumulator, and a fully-parameterised top-level datapath. The master bench `tb_vector_mac_top_para.v` integrates all blocks and takes a compile-time parameter `LANE_P`, so exactly the same stimulus can be reused for different MAC configurations.

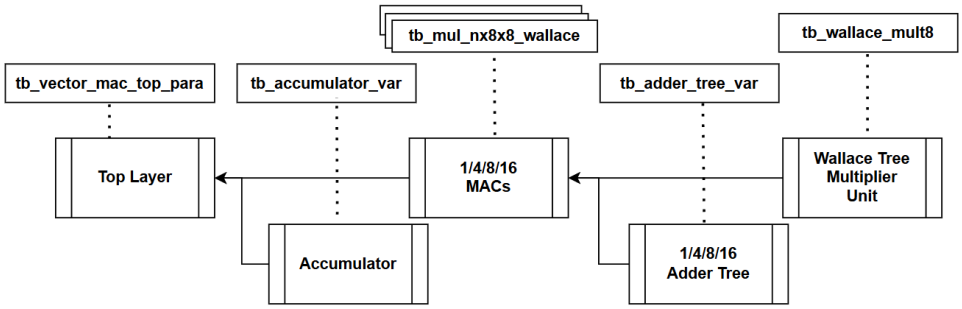

Fig. 4: Hierarchy of module-level testbenches.

Verification combines hand-crafted edge vectors (all-zero, all-255, 128×2 and 0×255) with thousands of pseudo-random INT8 pairs. Golden results are generated off-line and compared cycle-accurately. Stress tests run back-to-back transactions without pipeline bubbles, proving that the design sustains an initiation interval of one clock.

All simulations pass: every output matches its reference, no *X/Z* states propagate, and multi-stage pipelines remain phase-aligned across all lane counts. These results confirm functional correctness and timing robustness for both individual blocks and the integrated datapath.

VI. Conclusions and Future Work

In conclusion, the design delivers a parameterisable INT8 vector multiplier reaching up to 1.7 GMAC/s at 209–217 MHz, a three-stage Wallace multiplier and parameterised adder tree yields competitive TP/CLB¿. The 4-lane version offers the best area-normalised throughput. Future work includes on-board testing, INT 9–12 extensions, and on-chip buffering for continuous data streaming.

APPENDIX A
COMPLETE SOURCE REPOSITORY

All RTL source files and test benches are publicly available on GitHub:

https://github.com/0xCapy/int8_vecmac.git

The repository includes:

- Verilog modules for the Wallace multiplier, adder tree, accumulator and the parameterised top-level datapath.
- Self-checking test-benches with random and boundary-case generators.
- Vivado 2021.1 project scripts for out-of-the-box synthesis, implementation and power analysis.
- Markdown documents describing floor-planning constraints and timing-closure steps.

Readers can reproduce the results with:

```
git clone https://github.com/0xCapy/int8_vecmac.git
cd int8_vecmac
make vivado   # launches non-GUI batch flow
```