Design and Implementation of an 8x8 Systolic Array

Vishal Kevat
ECE, Purdue University
West Lafayette, IN, U.S.
vkevat@purdue.edu

Jagdish Kurdiya
ECE, Purdue University
West Lafayette, IN, U.S.
jkurdiya@purdue.edu

Niharika Tulugu
ECE, Purdue University
West Lafayette, IN, U.S.
ntulugu@purdue.edu

***Abstract*—The systolic array architecture is a powerful and scalable structure designed to accelerate matrix operations such as matrix multiplication, a key computation in many fields like machine learning and signal processing. In this project, we implement an 8×8 systolic array consisting of 64 processing elements (PEs) arranged in a regular 2D mesh. Each PE is responsible for performing multiply-accumulate (MAC) operations, receiving inputs from activation FIFOs (First-In-First-Out buffers) and weight FIFOs at the array's boundaries. A centralized control unit manages the flow of data into the FIFOs and orchestrates the overall operation. As data moves rhythmically across the array horizontally and vertically, each PE computes partial results and passes on the inputs to its neighboring PEs. This project covers the full digital design flow—RTL development, pre-synthesis simulation, synthesis, formal equivalence checking, place-and-route (PnR), and gate-level simulations, enabling a complete ASIC implementation and real workload analysis using convolutional layers from AI models like AlexNet and YOLO.**

***Index Terms*—Systolic Array, Multiply and Accumulate, Power, Delay, Area, CNN.**

I. Background and Motivation

A Systolic array is a homogeneous network of processing elements (PEs) tightly coupled to work together in a pipelined fashion. Each PE computes a partial result from the inputs passed from its neighbors from one direction and passes on either both the result and the inputs or just the inputs (based on the stationarity used). This rhythmic movement of data through the PE network in two dimensions earns its name, from 'systole', the regular pumping of blood by the heart [1]. Systolic arrays were first used in World War II, but were independently invented by H T Kung and Charles Leiserseon in the 1970s [2]. In recent times, systolic arrays have gained traction after Google's Tensor Processing Unit (TPU) implemented using systolic array surpassed other contemporary hardware like CPUs and GPUs in performance and energy for ML workloads [3].

The ever increasing popularity of ML increases the demand for efficient computing. Most modern ML models burn huge amounts of energy in both training and inference. This creates a need for energy-efficient hardware that can run ML training and inference without compromising performance. Google's TPU paper [3] presents an architecture based on systolic arrays with minimal control logic overhead, in contrast to GPUs. It exploits regularity in the matrix multiplication operation, the key primitive of all ML to simplify hardware. The TPU v1 paper reports a 15-30x speedup and 30-80x better TOPS (Tera operations)/Watt over its contemporary GPU and CPU.

In this project, we implemented an 8×8 8-bit systolic array and completed its physical design. The design was functionally verified and synthesized using Cadence Genus to generate a gate-level netlist, which was then used for place and route in Cadence Innovus. Post-layout analysis was performed to evaluate the design's performance in terms of timing, power, and area. Test cases with randomized input activations and filters were executed to validate functionality. Additionally, ScaleSim was employed to simulate and analyze the systolic array's behavior for real world workloads. The detailed results are presented in Section VI.

II. Proposed Architecture

The architecture mainly consists of three modules: the PE unit, Activation and Weight FIFOs, control logic, address selection logic and write out. Each of these each explained in detail in the following subsections.

A. Processing Element

The processing element (PE) in our design is a multiply-accumulate unit (MAC). It takes in two 8-bit inputs, does a scalar multiplication and adds it to the partial sum stored in the internal register. The result is then written back to the register whereas the inputs are passed to the downstream neighbor PEs. Figure 1 shows a functional block diagram of a PE.

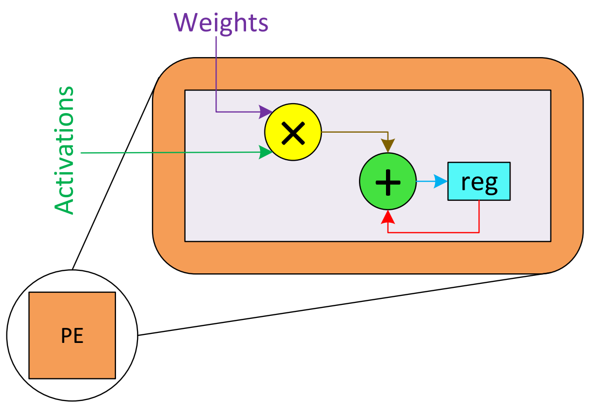

Fig. 1. A Processing Element

B. Systolic Array

There are a total of 64 PEs arranged in a 2-D 8x8 array. Each PE is connected to four of the PEs immediately adjacent to it. For any given PE, the activations are passed to it by the PE left to it (or from the Activation FIFOs in case of the left most set of PEs in the array), the weights are passed from the PE above it (or from the Weight FIFOs in case of the top most set of PEs). Similarly, the current PE passes the activations to the PE to its right and weights to the PE below it. The partial sum is held within the PE and is updated after every iteration. Figure 2 shows the 2D arrangement of the PEs, the input FIFOs and the control block.

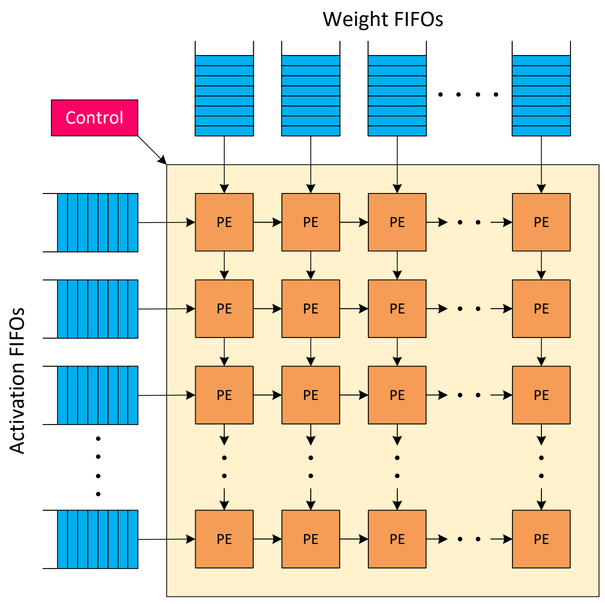

Fig. 2. Architecture of Systolic Array

C. Control Unit

The rhythmic movement of data through the array is achieved through a control module. The control follows a finite state machine (FSM) as shown in Figure 3 which conditionally updates the state of the system, say from IDLE to LOAD_DATA when the inputs are ready or from WAIT to ROLLING, once the inputs have been loaded into the PEs.

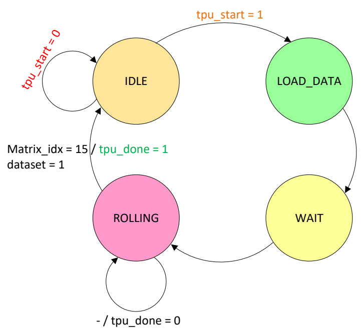

Fig. 3. Control Unit's Finite State Machine

III. RTL Verification and Testbench

Functional verification of the 8×8 systolic-array RTL is performed solely with the **Xilinx Vivado** Simulator. A self-checking Verilog testbench *test_tpu.v* instantiates the top-level design *tpu_top.v* and its submodules, applies a comprehensive set of matrix inputs via file-based stimulus, and compares DUT outputs against a behavioral reference model. Waveforms and coverage reports are generated within Vivado to confirm correct systolic dataflow, and a pass/fail summary is printed at the end of each run.

A. Testbench Architecture

1) DUT Instantiation: *test_tpu.v* instantiates *tpu_top* along with SRAM models $sram\_128 \times 32b.v$, $sram\_16 \times 128b.v$, address selector, quantizer, PE array, controller, and write-out modules.

2) Clock & Reset: A 200 MHz clock is generated via $always\ \#2.5\ clk\ = !clk$, which is 5 ns clock period, with reset asserted for the first 10 ns.

3) Stimulus Driver: Input matrices A and B are loaded using $\$readmemb$ tasks and streamed into the activation and weight FIFOs on the top and left edges of the array.

4) Scoreboard & Reference Model: A behavioral Verilog model computes $C = A \times B$. The testbench checker asserts equality $(if(dut\_out\ ! = ref\_out)\ \$error(...))$, halting simulation upon mismatch. This self-checking mechanism automates correctness checks without manual intervention.

B. Stimulus & Test Cases

The testbench drives four categories of inputs to ensure thorough verification of the design:

- **Random Matrices:** Uniform 8-bit signed-int values.
- **Identity / Diagonal:** Tests zero-padding and accumulation behavior.
- **All-Zeros / All-Ones:** Edge-case behavior for minimum and maximum patterns.
- **Max / Min Values:** Signed-extreme inputs to verify correct handling of overflow and quantization.

IV. RTL to GDS Flow

A. Synthesis Methodology

After completing functional verification, the RTL for the 8×8 systolic-array design was synthesized into a gate-level netlist using **Cadence Genus** tool. The synthesis was performed using the Cadence 45 nm PDK, which provided the standard-cell libraries and timing models. A single TCL driver script *syn.tcl* was used to automate the synthesis flow. The key steps of the flow are as follows:

- **Read Library:** `read_libs 45nm_typical.lib`
- **Read RTL:** `read_hdl -rtl ../src/*.v`
- **Link Design:** `link_design top_tpu`
- **Read Constraints:** `read_sdc top_tpu.sdc`
- **Synthesis:** `syn_generic -effort high`
- **Netlist Export:** `write_hdl > tpu_netlist.v`

- **QoR Reporting:** `report_qor > tpu_qor.rpt`

1) Post-Synthesis Area Report: The area estimation was obtained after logic synthesis. Table I summarizes the cell count and area for the top-level *tpu_top* module and its major submodules. The bulk of the area is consumed by the *systolic* array, which implements the core multiply–accumulate datapaths.

TABLE I
SYNTHESIZED AREA BREAKDOWN

Hierarchical Instance	Cell Count	Total Area (μm^2)
addr_sel	63	250.69
quantize	171	278.73
systolic	19,842	53,658.77
systolic_control	131	309.85
write_out	1,823	4,889.92
tpu_top	22,030	59,387.96

Overall, the synthesized design occupies approximately $59,388\ \mu$m^2. This breakdown confirms that the systolic-array datapath accounts for over 90 % of the total area, highlighting its dominance in both logic and routing resource requirements.

2) Post-Synthesis Timing Report: To verify that the 8×8 systolic-array meets the 200 MHz timing target (5 ns clock period), we extracted the most critical setup-check path from the Genus timing report. Table II lists the key parameters for the path running from one PE's data-queue register to another PE's matrix-multiply register.

TABLE II
CRITICAL PATH START/END AND CLOCK EDGES

Parameter	Value
Startpoint	systolic/data_queue_reg[7][1][1]/CK
Endpoint	systolic/matrix_mul_2D_reg[7][1][19]/D
Clock Domain	clk (5 ns period)
Launch Clock Edge	0 ps
Capture Clock Edge	+5000 ps

The slack is computed as:

$$\begin{aligned}\text{Slack} &= \text{Required Time} - \text{Arrival Time}\\ &= 4\,779 \text{ ps} - 4\,171 \text{ ps}\\ &= +\ 608 \text{ ps}\end{aligned}$$

Since the slack is positive, this path meets the timing requirement with a 608 ps margin, indicating potential headroom for performance tuning or additional pipelining. Detailed timing delays are shown in Table III.

3) Post-Synthesis Power Report: Power consumption was estimated by Cadence Genus after synthesis on the 45 nm standard-cell library. Table IV breaks down the total power into leakage, internal, and switching components for each cell category, in milli-Watts (mW).

Analysis of these results shows:

- **Dominant Internal Power:** Internal power is 1.60287 mW (87.65 % of total), indicating substantial short-circuit currents during gate switching.

TABLE III
CRITICAL PATH DELAYS AND SLACK

Parameter	Value
Capture Clock Edge	5000 ps
Setup Time	-121 ps
Uncertainty	-100 ps
Required Time	4779 ps
Launch Clock Edge	0 ps
Data Path Delay	4171 ps
Arrival Time	4171 ps
Slack	+608 ps

TABLE IV
POST-SYNTHESIS POWER BREAKDOWN BY CELL CATEGORY

Category	Leakage (mW)	Internal (mW)	Switching (mW)	Total (mW)
Register	0.00033	1.17050	0.07422	1.24505
Logic	0.00089	0.43237	0.15046	0.58372
Subtotal	0.00122	1.60287	0.22468	1.82877

- **Switching vs. Leakage:** Switching power contributes 0.22468 mW (12.29 %), while leakage is negligible at 0.00122 mW (0.07 %).
- **Category Breakdown:** Registers consume the majority of the power (1.24505 mW, 68.08 %), followed by combinational logic (0.58372 mW, 31.92 %).

These findings suggest that clock-gating and register-level optimizations would yield the most significant reductions in overall power consumption.

B. Logic Equivalence Checking

After synthesis, we performed formal Logic Equivalence Checking (LEC) between the original RTL and the Genus generated gate-level (synthesized) netlist to ensure functional correctness was preserved. The flow used **Cadence Conformal** tool and below is the verification report of the logic equivalence.

```
=============================================
            Verification Report
---------------------------------------------
Category                               Count
---------------------------------------------
1. Non-standard modeling options       0
2. Incomplete verification             0
3. User modifications to design        0
4. Clock-domain crossing warnings      0
5. Design ambiguities                  0
6. Compare Results                     PASS
=============================================
```

Since all compare points passed and no mismatches or warnings were reported, we confirm that the gate-level netlist is functionally equivalent to the original RTL. With functional equivalence established, the design is now ready to proceed to the physical implementation stage (Place and Route).

C. Place & Route

After confirming functional equivalence through Logic Equivalence Checking, the design was taken forward for physical implementation using the **Cadence Innovus** tool with a $45\ nm$ standard cell library. The Place and Route (P&R) stage maps the synthesized gate-level netlist onto physical silicon, optimizing for performance, area, and manufacturability.

The P&R flow consisted of the following major steps:

- **Floorplanning:** The chip outline, core area, and placement constraints were defined. Power and ground rings were inserted at this stage.
- **Placement:** Standard cells were placed in the designated core area while minimizing wirelength and avoiding routing congestion.
- **Clock Tree Synthesis (CTS):** A balanced clock tree was constructed to reduce clock skew and insertion delay, ensuring timing closure.
- **Routing:** Signal interconnections were routed using metal layers while complying with design rules. Design Rule Checks (DRC) and Layout vs. Schematic (LVS) checks were performed.
- **Post-Route Optimization:** Timing, area, and signal integrity were further optimized. Final reports for timing, power, and area were generated post-routing.

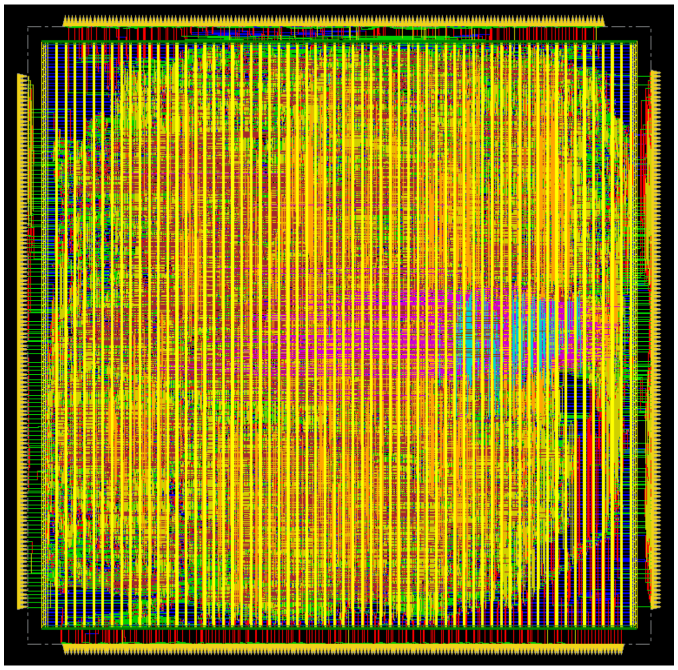

Fig. 4. Final Chip layout after Place and Route

The final layout met the defined timing constraints and passed all physical verification checks. It is expected that the power and area usage will be increased compared to post-synthesis estimates due to clock tree overhead and routing parasitics. Figure 4 shows the layout generated after the complete Place and Route process.

V. Power, Performance and Area Analysis (PPA)

A. Post-Layout Area Report

The area values reported here are obtained after the P&R stage using the Cadence Innovus tool. Compared to the synthesized area, the total area has increased due to additional buffers, fillers, and routing resources introduced during physical implementation.

TABLE V
POST-LAYOUT AREA BREAKDOWN

Hierarchical Instance	**Cell Count**	**Total Area** (μ**m**^{2})
addr_sel	71	266.08
quantize	439	685.37
systolic	22,142	57,266.87
systolic_control	145	336.87
write_out	2,670	6,351.28
tpu_top	25,664	65,156.13

As seen in Table V, the total area of the tpu_top module increased from the synthesized value of $59,388$ to $65,156$ μm^2. This increase is expected and typical during the physical design stage, primarily due to the insertion of additional cells such as:

- **Buffers and inverters:** Added for signal integrity and timing closure.
- **Filler cells:** Inserted to maintain DRC-compliant spacing between standard cells and ensure proper well tap connections.
- **Clock tree cells:** Introduced during clock tree synthesis to balance clock skews across flip-flops.
- **Routing congestion mitigation:** Additional whitespace or area used to ease global and detailed routing.

This post-layout area report reflects a more realistic estimation of the final silicon implementation and is used for further power, timing, and IR-drop analysis.

B. Post-Layout Timing Report

The following critical-path data are extracted from the post-layout timing analysis as shown in Table VI.

TABLE VI
POST-LAYOUT CRITICAL-PATH TIMING

Parameter	**Value**
Capture Clock Edge	5000 ps
Setup Time	-147 ps
Uncertainty	-100 ps
Required Time	4753 ps
Launch Clock Edge	0 ps
Data Path Delay	4726 ps
Arrival Time	4726 ps
Slack	+27 ps

Since the post-layout slack is positive ($+27\ ps$), this path meets the $5\ ns$ clock requirement with a small margin, reflecting the impact of routing parasitics and clock-tree insertion.

C. *Post-Layout Power Report*

Power consumption was estimated after Place & Route using parasitic-extracted netlist and typical activity factors. Table VII breaks down internal, switching, and leakage power by cell category.

TABLE VII
POST-LAYOUT POWER BREAKDOWN BY CELL CATEGORY

Category	**Leakage** (mW)	**Internal** (mW)	**Switching** (mW)	**Total** (mW)
Sequential	0.00033	1.273	0.1621	1.43543
Combinational	0.00114	1.495	1.479	2.97514
Subtotal	0.00147	2.768	1.6411	4.41057

Compared to the post-synthesis estimate of 1.83 mW total power, the post-layout total of 4.41 mW reflects the impact of parasitic capacitances, clock-tree buffers, and routing overhead introduced during P&R. The dominant contributors remain combinational logic (67.45 %) and sequential elements (32.55 %), suggesting that optimizations such as clock gating and register retiming could yield meaningful power reductions in the final silicon.

VI. RESULTS & ANALYSIS

A. *Real Workload Analysis*

We used ScaleSim to estimate the system level numbers for our macro. SCALE sim is a simulator for systolic array-based accelerators [4]. This simulator provides us the platform to run real workloads and estimate the number of cycles taken to run and percentage utilization of the PEs among other metrics.We have chosen Alexnet Conv1 layer and YOLO conv1 layer as our workloads. We took these workloads based on the difference in their input feature map sizes and number of channels.

1) Cycle count: We configured three different configurations of systolic array. The comparison chart in Figure 5 shows the cycle count for different configurations for real workloads. It can be seen that 8x8 takes the longest to complete for both the workloads, which is obvious from the fact that this configuration has the least number of PEs available when compared to 16x16 and 32x32.

2) PE Utilization: We also analyzed the percentage of PE utilization for different systolic array configurations as shown in Figure 6. We used the same workloads as mentioned above. Here the observation is completely opposite to the cycle count. 8x8 configurations have the best PE utilization for both the workloads when compared to 16x16, 32x32. In the case of Alexnet conv1, the difference is not much but when it comes to YOLO conv1, the 32x32 has severe underutilization. Percentage utilization gets better with 8x8 and 16x16 but there is still PE underutilization.

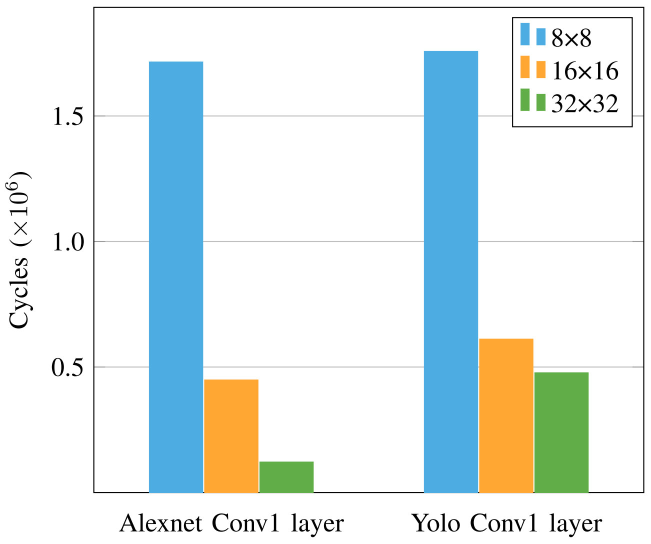

Fig. 5. Cycle count for different workloads

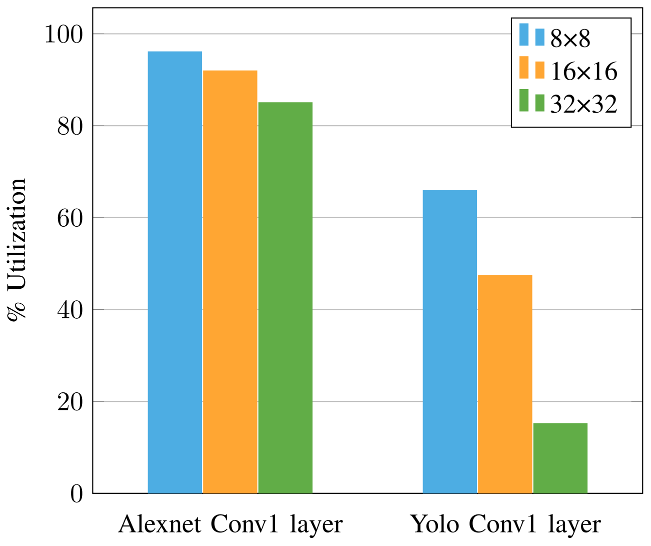

Fig. 6. PE Utilization for different workloads

3) Mapping Efficiency: We can also plot mapping efficiency too from the reports generated by ScaleSIM. For Alexnet conv1, the mapping efficiency is similar across all the configurations, and it almost reaches 100 %. In the case of YOLO conv1, the mapping efficiency significantly for 32x32 configurations and it complements the observation about 8x8 being the best choice for this workload.

Final observation: We can conclude that there is always trade off between the performance and hardware utilization when we compare different systolic array dimensions. In the case of Alexnet conv1, the 32x32 performs better than the other 2 and PE utilization is not very bad either hence it would be a better choice. However, for YOLO conv1, though 32x32 performs better but it suffers from severe PE underutilization and 8x8 has much better utilization, hence it would be a better

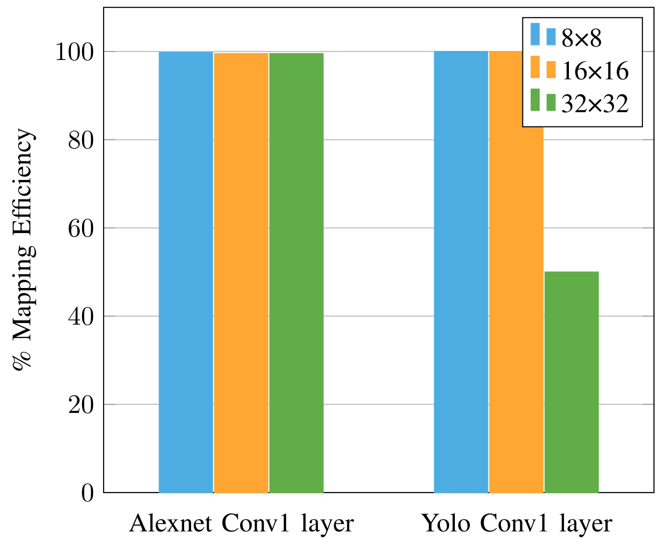

Fig. 7. Mapping Efficiency for different workloads

choice for this workload.

B. Optional Fixed-Point 8 Implementation

In addition to our primary signed 8-bit integer design, we implemented an alternative fixed-point 8-bit version by extending each MAC unit to support an 8-bit signed Fixed-Point-8 format [4.4], where first 4 bits represents integer part and last 4 bits represents fractional part of the inputs. This optional variant introduces scaling and rounding logic inside the accumulator. Table VIII compares the key post-layout metrics for both implementations.

TABLE VIII
POST-LAYOUT COMPARISON: SIGNED INT VS. FIXED-POINT 8-BIT

Implementation	Area (μm^2)	Worst Slack (ps)	Power (mW)
Signed Integer 8-bit	65 156.13	+27	4.41057
Signed Fixed-Point 8-bit	67 801.50	+33	4.85300
Change (% over Int-8)	+4.06%	+22.22%	+10.03%

As expected, the fixed-point version incurs a slightly increased area and power overhead (4 % and 10% respectively) and a slight increase in timing margin due to the added scaling logic. This trade-off may be acceptable in applications requiring fractional precision.

VII. Challenges & Learning Outcomes

A. Challenges

- It was difficult to ensure that the output-stationary dataflow across an 8x8 grid maintained correct timing alignment; multiple clock-skew analysis and retiming iterations were needed to remove hold violations.
- During P&R, the dense inter-PE routing created congestion hotspots. In order to solve this, we added routing blockages and changed the floorplan-pin assignments. We then re-tuned placement constraints to spread out critical nets.
- To meet our 200 MHz frequency target while maintaining power under 5 mW required experimenting with gate-sizing, buffer insertion, and clock-gating strategies. It was a continuous iteration effort to balance area, power, and slack.
- One of the biggest challenges was figuring out which tool would be best for estimating system-level results for our macro. Although TimeLoop is a widely recognized option, it had a challenging integration process and a steep learning curve. As a result, we explored alternative solutions and ultimately selected ScaleSim for its relative ease of use and suitability for our evaluation needs.
- Configuring ScaleSim to match our specific systolic array implementation posed considerable challenges. Additionally, running real-world workloads such as AlexNet and YOLO on the 8×8 array was a time-consuming process.

B. Learning Outcomes

- The 8×8 systolic array's design helped us better understand how processing-element (PE) reuse patterns maximize performance and data locality in convolution workloads, as well as weight-stationary versus output-stationary dataflows.
- Comparing signed-integer and fixed-point implementations highlighted how lower-precision arithmetic affects area, power, and timing.
- Gaining hands-on experience with Genus, Conformal, Innovus, and Vivado tools taught us the end-to-end steps required to implement, verify, and physically realize systolic array in hardware.
- During our project presentation, we were asked how our work on the 8x8 systolic array relates to the AI Hardware course. To answer that, we evaluated real-world AI workloads specifically the convolutional layers of AlexNet and YOLO on multiple array configurations (8x8, 16x16, and 32x32). By measuring cycle counts and resource utilization, we gained valuable insigts on how these systolic array will scale up for AI workloads.

VIII. Conclusion & Future Work

In this project, we explored the architecture, data flow and synchronization strategies needed for a systolic array. We also saw the effect of using different datatypes namely integer and fixed point, on power and area of the design. We observed that fixed point incurs higher area and power costs, as expected. We simulated Alexnet Conv1 and YOLO Conv1 layers as workloads and analyzed the utilization, cycle count and mapping efficiency for them.

As an optional part, we chose to do a comparison between different datatypes instead of FPGA prototyping. The future scope of the project could include FPGA prototyping and implementing a weight- stationary system. Further, clock gating and power gating can be done for increasing the power efficiency of the system.

IX. Team Contributions

Each team member contributed significantly across various stages of the project. The tasks ranged from RTL design and verification to synthesis, physical implementation, and report preparation. Table IX summarizes the division of responsibilities among the members.

TABLE IX
Detailed Task Distribution Among Team Members

Task	**Team Member(s)**
Literature Survey	All
RTL Architecture Planning	All
RTL Module Implementation	Jagdish
Systolic Array Datapath Design	Niharika
Control Logic Design	Jagdish
Testbench Development	Vishal
Functional Verification	All
Gate-Level Synthesis (Cadence Genus)	Niharika
Logic Equivalence Checking	Niharika
Static Timing Analysis (Pre-layout)	Vishal
Place and Route (Cadence Innovus)	Vishal
Post-Layout Timing Analysis	Jagdish
Post-Layout Power Analysis	Niharika
Post-Layout Area Analysis	Vishal
System-Level Numbers	Jagdish
Final Report Writing	All
Presentation and Video Preparation	All

Acknowledgements

We express our heartfelt gratitude to Dr. Kaushik Roy and Tanvi Sharma, the course instructors, for their exceptional teaching and constant encouragement. The concepts taught during the course provided the foundation and skills necessary for the successful completion of this project. We also sincerely thank Deepika Sharma and Soumyadeep Chandra, the teaching assistants, for their guidance in resolving doubts and clarifying concepts, which greatly aided us in this work.

Appendix

Software and Libraries Used

- **Xilinx Vivado 2023.2** – Pre-Synthesis Simulation
- **Cadence Genus 21.1** – Logic Synthesis
- **Cadence Conformal 24.10** – Equivalence Checking
- **Cadence Innovus 21.1** – Physical Design & Routing
- **Cadence 45nm PDK** – Including standard cell libraries, technology files, and parasitic extraction models.

Project Repository

The complete design and implementation flow, including RTL sources, simulation testbenches, synthesis and P&R scripts, and analysis reports, is available at the following GitHub repository: https://github.com/vishalkevat007/Design-and-Implementation-of-8x8-Systolic-Array

References

[1] Wikipedia contributors, "Systolic array," 2024, accessed: 2025-05-08. [Online]. Available: https://en.wikipedia.org/wiki/Systolic_array

[2] H. T. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," in *Sparse Matrix Proceedings 1978*, vol. 1. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979, pp. 256–282.

[3] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[4] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.