SWISS: Switch-augmented MXU Inferencing for Structured Sparsity

Aniket Chatterjee
Department of ECE
University of Illinois Urbana-Champaign
aniketc2@illinois.edu

Yu-Ting Kuo
Department of ECE
University of Illinois Urbana-Champaign
yutingk3@illinois.edu

Ben Blade
Department of ECE
University of Illinois Urbana-Champaign
bblade2@illinois.edu

Yun-Chi Chou
Department of ECE
University of Illinois Urbana-Champaign
yunchic2@illinois.edu

Ting-Yu Yang
Department of ECE
University of Illinois Urbana-Champaign
tingyuy4@illinois.edu

Bailey Faulk
Department of ECE
University of Illinois Urbana-Champaign
bfaulk2@illinois.edu

***Abstract*—Systolic Arrays have been adopted as a state-of-the-art method of accelerating AI workloads, used in devices such as Google's TPU [1], and featuring massive data reuse, high throughput, and a simple, regular topology with small local connections. In parallel, structured sparsity techniques have emerged as an effective approach to reduce model size and memory bandwidth with minimal accuracy loss, thereby improving efficiency in modern deep learning workloads. However, existing systolic designs, including those used in modern TPUs, remain optimized for dense computation and are not inherently adaptable to structured sparsity, resulting in unnecessary data movement and underutilized compute resources. To overcome this limitation, we propose a switch-augmented Matrix Multiply Unit (MXU) architecture to exploit 2:4 structured sparsity [2]. By inserting a lightweight switch network at the PE input stage, the design dynamically routes only nonzero weights to their corresponding PEs, recovering up to 2× memory bandwidth and improving computational throughput. A Vector Processing Unit (VPU) is integrated alongside the MXU to handle elementwise and nonlinear operations, enabling full end-to-end AI model inference on a single chip.**

I. Introduction

The rapid growth of large-scale deep learning has made memory bandwidth a critical bottleneck for hardware accelerators. NVIDIA addressed this with 2:4 structured sparsity [3], which enforces exactly 2 nonzero values per every 4 consecutive weights. By skipping zero-valued operands at the hardware level, this achieves up to a 2× reduction in memory bandwidth.

Unfortunately, this benefit does not extend to Google's Tensor Processing Units (TPUs). The Matrix Multiply Unit (MXU) follows a rigid systolic dataflow and treats all matrices as dense. As a result, the full weight matrix—including zeros—is fetched and propagated through the PE array, limiting both bandwidth and throughput gains.

To address this limitation, we propose a switch-augmented 8×8 MXU with a lightweight switch network integrated into the PE input stage. Each switch decodes 2:4 sparsity metadata and routes only necessary activation elements to the corresponding PEs, ensuring alignment with nonzero weights. This design preserves the bandwidth and throughput benefits of 2:4 sparsity while adding minimal overhead to dense computation.

II. Architecture

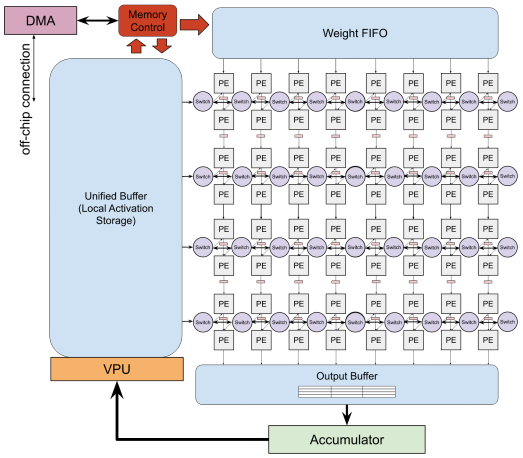

Fig. 1. Switch-augmented MXU architecture for 2:4 structured sparsity

A. Overall Architecture

Figure 1 illustrates the overall chip architecture. Note that the numbers below are based on our current understanding of the area characteristics of the TSMC 65nm process and may change once we gain access to the PDK. The architecture is comprised of:

Data Buffers There are 3 data buffers in our design, a unified buffer which stores activations, a weight buffer which stores weights, and an output buffer which stores the output of the systolic array temporarily for the purpose of accumulating the partial sums of the tiled array. We will use the leftover area of the chip after placement of logic elements for our SRAM.

Memory Controller We borrow the concept of Processing-in-Memory (PIM) by placing the VPU immediately adjacent to the input of the activation SRAM. We want to analyze the benefits of this placement in order to verify whether the concepts learned in ECE 511 [4] can be applied to this chip as well.

Switch-Augmented MXU Weights are preloaded into the PE array, where each PE stores only nonzero weights under the 2:4 sparsity pattern. Activations enter column-wise and are fed to the input buffer. A switch network interleaved every two PE rows routes nonzero elements to their target PEs, bypassing zeros. Partial sums propagate downward through the systolic array, accumulate in the output buffer, and are forwarded to the accumulator. The systolic array itself has the ability to multiply a 2:4 structured sparse weight matrix W with an activation matrix A as output=WA. Due to the size of the systolic array, the dimensions for a single computation are 8×16 for the weights (in structured sparse format). The activations for a single computation are restricted to 16 rows, but can have an arbitrary number of columns as long as the resultant can be stored in the output buffer. In order to support general matrix multiplication with matrices of arbitrary dimension, we use a tiling strategy, explained in section III (inference). We plan on using 8-bit integers for the weights and activations, and 16 bits for the accumulations.

VPU and Accumulator The VPU applies elementwise operations (e.g., ReLU, bias addition) on the accumulated outputs to support end-to-end inference. When processing a tiled matrix multiplication that does not fit directly into the systolic array, partial sums are stored in the output buffer, and processed near-memory when new partials come in to be added. The accumulator processes the resultant activations near-memory, applying an activation function (such as RELU) and truncating the 16-bit accumulations into 8-bit activations before storing back the results.

Potential Extension: Temporal Coding Considering the relatively small area budget of our chip, a significant proportion of our chip power may be consumed by off-chip communication. We may consider implementing a temporal coding protocol as described in [5] to reduce the toggling of our data wires and decrease power consumption. This is not a fundamental component of our chip and is subject to time and feasibility constraints. If implemented, we can compare power usage between temporal coding and regular data transmission.

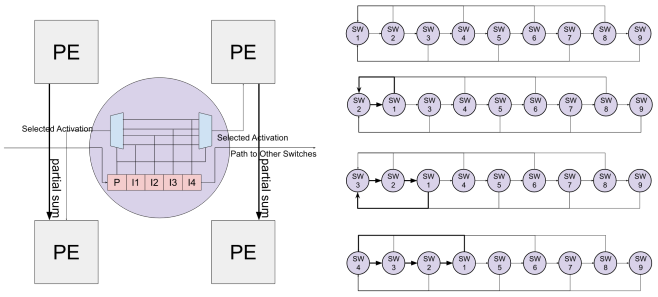

Fig. 2. Switches Network Design

B. The Switches Network

Figure 2 illustrates the switch microarchitecture. Each switch is placed between four PEs and serves two functions: operand selection and data forwarding. It contains a position register and four input registers forming a local FIFO buffer. The position bits encode the 2:4 sparsity metadata, directing the switch to select and forward only the corresponding activations to the PEs.

Because there is extra unused space when performing dense matrix multiplication in the switch registers, we aim to utilize this slack as a buffer. By leveraging the switch network, we can correctly reorder and route the preloaded data to ensure proper alignment and delivery to the PEs.

C. Input/Output

- 1 pin for VDD; 1 pin for GND
- 1 pin for clock; 1 pin for reset
- 4 pins for AXI Control
  - 1 pin for ready, 1 pin for valid, 1 pin for data strobe, 1 pin for bus direction
- 32 pins for data
  - 8-bit data at double data rate = bandwidth of 8 elements per clock
- DFT pins
  - 1 pin for scan enable, 4 pins for standard JTAG
- Remaining pins can be used for VPU control, DMA/memory controller, accelerator status as necessary

III. Inference

TABLE I
MNIST CNN Architecture

Layer	Operation	Weight Shape	Output Shape
Input			$(B, 1, 28, 28)$
Conv1	Conv 3×3, stride 2, pad 1	$(32, 1, 3, 3)$	$(B, 32, 14, 14)$
ReLU	ReLU		$(B, 32, 14, 14)$
Conv2	Conv 3×3, stride 2, pad 1	$(64, 32, 3, 3)$	$(B, 64, 7, 7)$
ReLU	ReLU		$(B, 64, 7, 7)$
Flatten	Reshape		$(B, 3136)$
FC1	Linear	$(3136, 128)$	$(B, 128)$
ReLU	ReLU		$(B, 128)$
FC2	Linear	$(128, 10)$	$(B, 10)$

A. Model Inference

Tiling is a fundamental technique used to map large neural network computations onto hardware accelerators with fixed physical dimensions [6], such as an 8×8 systolic array. Because deep learning models contain weight and activation matrices of arbitrary and varying sizes, they cannot be computed in a single hardware cycle.

By partitioning these massive mathematical operations into smaller, uniform sub-matrices, tiling enables efficient model inference by providing three critical benefits:

- **Hardware Compatibility:** It bridges the gap between arbitrary software matrix dimensions and strict hardware limits.
- **Memory Hierarchy Optimization (Data Reuse):** Tiling ensures that blocks of data fit perfectly into fast local memory and remain stationary inside the PEs long enough to compute all necessary partial sums and reduce the memory bandwidth bottleneck.

TABLE II
MNIST CNN FORWARD PASS

Step	Operation	Input Shape	Output Shape
Input			$(1, 28, 28)$
Conv1 (stride 2, padding 1)			
1.1	im2col	$(1, 28, 28)$	$(196, 9)$
1.2	**matmul** @ $(9, 32)$	$(196, 9)$	$(196, 32)$
1.3	**+ bias** $(32,)$	$(196, 32)$	$(196, 32)$
1.4	reshape	$(196, 32)$	$(32, 14, 14)$
1.5	**ReLU**	$(32, 14, 14)$	$(32, 14, 14)$
Conv2 (stride 2, padding 1)			
2.1	im2col	$(32, 14, 14)$	$(49, 288)$
2.2	**matmul** @ $(288, 64)$	$(49, 288)$	$(49, 64)$
2.3	**+ bias** $(64,)$	$(49, 64)$	$(49, 64)$
2.4	reshape	$(49, 64)$	$(64, 7, 7)$
2.5	**ReLU**	$(64, 7, 7)$	$(64, 7, 7)$
Flatten			
3.1	reshape	$(64, 7, 7)$	$(1, 3136)$
FC1			
4.1	**matmul** @ $(3136, 128)$	$(1, 3136)$	$(1, 128)$
4.2	**+ bias** $(128,)$	$(1, 128)$	$(1, 128)$
4.3	**ReLU**	$(1, 128)$	$(1, 128)$
FC2			
5.1	**matmul** @ $(128, 10)$	$(1, 128)$	$(1, 10)$
5.2	**+ bias** $(10,)$	$(1, 10)$	$(1, 10)$
Output (logits)			$(1, 10)$

TABLE III
LAYER-BY-LAYER WORKLOAD MAPPING ON AN 8×8 SYSTOLIC ARRAY

Layer	Original GEMM Shape $(M \times K \times N)$	Tiled Shape $(M_t \times K_t \times N_t)$	Tile Ops (8×8)
Conv1	$196 \times 9 \times 32$	$25 \times 2 \times 4$	200
Conv2	$49 \times 288 \times 64$	$7 \times 36 \times 8$	2,016
FC1	$1 \times 3136 \times 128$	$1 \times 392 \times 16$	6,272
FC2	$1 \times 128 \times 10$	$1 \times 16 \times 2$	32

- **Pipelined Execution:** It breaks a monolithic matrix multiplication down into a stream of independent Multiply-Accumulate (MAC) operations: $C_{i,j} = \sum_k (A_{i,k} \times W_{k,j})$. This allows the hardware to continuously stream data tiles through the systolic array pipeline.

When tiling, the weight matrix is broken up into 8×16-sized tiles which can fit in the systolic array. The activations are grouped into chunks of 16 rows each. Each chunk of activations is fed through each of its corresponding weight tile, producing a series of matrices of partial sums which are stored in the output buffer and combined using the accumulator. We use a double-buffering strategy to ensure that there is no downtime from weight propagation whenever the weight tile changes. As a row of weight tiles is finishing processing, the results are sent vector-wise to the VPU as the final vectors become available. Once a row of weight tiles has been processed we simply move to the next row of weight tiles and begin processing.

For matrices whose dimensions are not multiples of our base dimension, padding can be used, though this takes extra overhead and is not desirable.

B. SRAM Size Analysis

When performing the matrix multiplication, we want the unified activation SRAM to hold enough space for one row of tiles of input activations and one row of tiles of output activations.

Specifically, the matrix multiplication

$$A_{M\times K} \cdot W_{K\times N} + B_{M\times N} = Y_{M\times N}$$

requires storing at least $8 \times K$ input activation elements and $8 \times N$ output activation elements in the unified activation SRAM. The Weight SRAM only needs to be designed as a double buffer sized for two tiles of data, which amounts to 16×8 elements.

As illustrated in the naive model above for edge devices, approximately 2 kB of SRAM for unified activation storage is sufficient to meet our minimum requirements (covering one row of input tiles and one row of output tiles), while the weight FIFO needs only 128 B (a double buffer of two 8×8 weight tiles).

If additional on-chip area is available, we could store more than one row of tiles per chip and enlarge the SRAM accordingly. However, since our end goal is to have multiple customized chips cooperate, aggressively maximizing the SRAM capacity of a single chip is not a primary design objective.

C. 2:4 Structured Sparsity vs. Dense Matrix

The most direct benefit of employing structured sparsity is a $2\times$ improvement in throughput. Since we perform tiling for the matrix multiplication, the total number of tiling iterations is reduced to $K/(2 \times \text{tile size})$. Because each iteration fetches weights from off-chip memory, halving the iteration count correspondingly halves the off-chip memory traffic for weights. Meanwhile, the tiled activations are kept resident in the Unified Buffer and reused throughout the computation, so no additional overhead is introduced on the activation side.

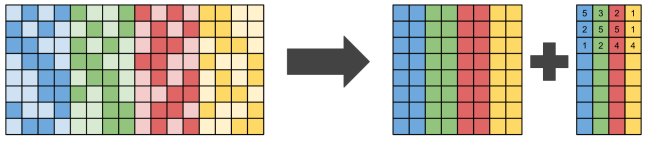

Fig. 3. 2:4 Sparsity Weight Example

IV. VERIFICATION

A. Methodology

We adopt a UVM-based verification flow [7], with a cycle-accurate software model of the 2:4 sparsity algorithm serving as the golden reference. All RTL simulation outputs are automatically cross-checked against this model. To

preserve design objectivity, DV tasks are assigned to team members who did not author the corresponding architectural specification or RTL, preventing shared logical blind spots between the design and its testbench. We further enforce a compositional testing policy: no block is integrated upward until it achieves complete functional and state coverage in isolation.

B. Unit-Level Testing

Unit tests target the custom routing switches and individual PEs. The primary goal is to confirm that the switch's 3-bit position registers correctly decode the 2:4 sparsity metadata and that the FIFO buffers route non-zero operands to the appropriate PEs without drops or misalignment. Constrained-random activation streams are used to drive these blocks to 100% functional coverage before integration.

C. Array-Level Verification

Validated units are composed into the 8×8 MXU grid and exercised against a mock memory interface. Verification here focuses on systolic data movement: correct per-cycle advancement of the active switch index, accurate partial-sum accumulation along the column, and absence of pipeline stalls introduced by the routing logic.

V. Roadmap

Our goal is to demonstrate end-to-end model inference on the proposed chip, either as a standalone accelerator or within a multi-chip system. We will target compact keyword spotting models for edge devices.

To enable inference, we will either develop a lightweight compiler to map model weights and activations onto the architecture, or preload fixed weight patterns onto an FPGA to validate functional correctness under controlled conditions. Both approaches verify that the switch-augmented MXU correctly supports 2:4 sparse operand routing in realistic workloads.

We plan to complete the RTL design by August, focusing on the routing network to reduce overhead, latency, and wiring complexity in the systolic dataflow. This will be the most time-intensive part and require multiple iterations.

Because hardware changes to systolic arrays are costly and slow to validate, optimizations are usually done at the compiler level. Our work instead emphasizes physical design, directly evaluating the cost of integrating the switch network.

A. April to May

This phase focuses on high-level design finalization and software preparation. Three primary milestones are targeted:

1) Finalize the full chip architecture and define block-level I/O interfaces for all major components.
2) Determine the SRAM sizing to optimally support the memory access patterns of both sparse and dense workloads.
3) Establish the model inference dataflow either by a self-developed lightweight compiler or by developing an FPGA-based controller to drive fixed weight patterns through the chip.

B. June to August

This phase focuses on RTL implementation and functional verification. SystemVerilog testbenches validate the switch-augmented MXU, control unit, and interface modules, using a golden model to provide reference outputs for comparison. Post-synthesis gate-level simulation is performed to identify timing violations at the target frequency.

By the end of July, the primary milestone is to demonstrate correct execution of matrix multiplication followed by ReLU activation, verified against the golden model. In August, the focus shifts to tiled execution of large matrix multiplication, validating correct partitioning and processing beyond the MXU array size.

C. Fall Semester

This phase targets physical design in the 65nm technology node, encompassing floorplanning, Design For Test(scan-chaing, MBIST, etc.), layout, and clock tree synthesis.

While established tooling exists for this process node, we dedicate particular attention to the routing of two non-standard structures: the PIM-style vector unit placed adjacent to the SRAM, and the switch-augmented systolic array. Custom scripts and refined EDA configurations are developed to minimize their area and interconnect overhead relative to a conventional systolic array, producing a clean, timing-closed layout that preserves the efficiency gains of the sparse dataflow without disproportionate routing cost.

D. Planned Enhancements

If hardware resources permit and the project progresses ahead of schedule, we plan to pursue the following enhancements, in order of priority, to make the project more complete:

1) Enable multiple chips to collaboratively perform inference on the same ML model (with corresponding compiler support).
2) If the SRAM budget can be further increased, explore batching to maximize the benefits of the weight-stationary systolic array.
3) Introduce a softmax or normalization unit to accelerate a broader range of ML models.

VI. Team Organization

A. Work Distribution

- **Phase 1: Compiler and Control Logic**
  - **Compiler Development:** Mapping DNN models into tiled matrix multiplication and defining DMA instruction sets.
  - **DMA Arch & RTL Design:** Implementing logic to process compiler instructions and managing communication between weight and unified SRAM.
  - *Responsibilities:*
    - Compiler: Aniket, Tim
    - DMA Arch & RTL Design: Aniket, Ben, Tim
- **Phase 2: Memory and Data Movement**

  - **Activation Data Movement & SRAM Bank:** Defining memory throughput, SRAM sizing, and ensuring compute unit saturation.
  - **Memory Controller Design:** Orchestrating weight/activation flow into the systolic array and input buffers.
  - **Switches Network Architecture:** Designing input buffers and interconnect topology for optimal data routing.
  - *Responsibilities:*
    * Data Movement, SRAM & Network Architecture: Bailey, Tim, Aniket
    * Memory Controller Design: Aniket, Bailey, Tim
- **Phase 3: Hardware Core & Implementation**
  - **Systolic Array & Switches:** Implementing weight-stationary systolic array RTL and data-forwarding logic for Processing Elements (PEs).
  - **Accumulator & Activation Function:** Designing the VPU, including normalization, softmax, and Processing-in-SRAM (PiS) activation.
  - *Responsibilities:*
    * Systolic Array & Switches RTL: Ben, Aniket, Christine
    * Accumulator & Activation Design: Tim, Ben, Daniel
- **Phase 4: Simulation and Verification**
  - **Simulator Construction:** Building a Python/C/Matlab-based simulator for golden reference generation.
  - **Testbench Development:** Developing comprehensive testbenches across hardware components (UVM).
  - *Responsibilities:*
    * Simulator: Christine, Aniket
    * Testbenches & UVM: Ben, Daniel, Bailey
- **Physical Design** Yun-Chi, Ting-Yu, Ben
- **Bringup** Aniket, Yun-Chi, Ting-Yu, Ben, Bailey

B. Team Member Qualifications

- **Aniket Chatterjee** 2nd Year MS Student
  - **Relevant Coursework:** Computer Architecture, SoC Design, VLSI Design, Computer Design and Prototyping, ASIC Design Lab, Compilers, Artificial Intelligence
  - **Experience:** NoC Topology Optimization, Nam Sung Kim lab research (temporal coding, KV cache quantization), multicore cache-coherent CPU design, HLS LeNet accelerator
  - **Technical Strength:** Architecture Design, Compiler Mapping, Dataflow Optimization
- **Yu-Ting Kuo** 2nd Year MEng Student
  - **Relevant Coursework:** ECE 511, ECE 411, ECE 425, ECE 482, ECE 408
  - **Experience:** SRAM Circuit Design Intern@TSMC, 1 year RTL Design@Ranictek, Bank-Balanced SpMDV Multiplication in RTL, CNN Image Classification in RTL, CUDA programmed GPT2 kernels
  - **Technical Strength:** RTL Design, Microarchitecture
- **Ben Blade** 2nd Year MEng Student
  - **Relevant Coursework:** ECE 511, ECE 411, ECE 425, ECE 444, ECE 408
  - **Experience:** CUDA programmed CNN kernels, Kernel optimizations on Tenstorrent Blackhole
  - **Technical Strength:** RTL Design, Physical Design, System Integration
- **Yun-Chi Chou** 2nd Year MEng Student
  - **Relevant Coursework:** ECE 411, ECE 527, ECE 425, ECE 482
  - **Experience:** Systolic Array-based Native Sparse Attention Accelerator, Digital Low Power Multiplier Layout Design(free PDK 45nm).
  - **Technical Strength:** Physical Design, DFT, Layout Optimization
- **Ting-Yu Yang** 2nd Year MEng Student
  - **Relevant Coursework:** ECE 411, ECE 425, ECE 482
  - **Experience:** Digital Low Power Multiplier Layout Design(free PDK 45nm), Digital LDO FSM based Layout Design(TSMC 180nm)
  - **Technical Strength:** Physical Design, Layout Optimization
- **Bailey Faulk** 2nd Year MS Student
  - **Relevant Coursework:** ECE 411
  - **Experience:** 6 months Design Verification Intern @AMD
  - **Technical Strength:** Verification, Testbench, RTL design

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[2] D. Haziza *et al.*, "Accelerating transformer inference and training with 2:4 activation sparsity," *arXiv preprint arXiv:2503.16672*, 2025.

[3] A. Mishra, J. Albericio, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[4] University of Illinois Urbana-Champaign, "Ece 511: Computer architecture (spring 2026)," 2026, accessed: 2026-04-22. [Online]. Available: https://courses.grainger.illinois.edu/ece511/sp2026/

[5] M. Mishkin, N. Kim, and M. Lipasti, "Temporal codes in on-chip interconnects," in *ISLPED 2017 - IEEE/ACM International Symposium on Low Power Electronics and Design*, ser. Proceedings of the International Symposium on Low Power Electronics and Design. United States: Institute of Electrical and Electronics Engineers Inc., Aug. 2017, aCKNOWLEDGEMENTS This work was supported in part by NSF (CCF-1615014, CCF-1628384, CCF-1600896 and CNS-1600669).; 22nd IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2017 ; Conference date: 24-07-2017 Through 26-07-2017.

[6] J. Farley and A. Gerstlauer, "Memory-aware fusing and tiling of neural networks for accelerated edge inference," *arXiv preprint arXiv:2107.06960*, 2021.

[7] S. Nagaraj, "Role of systemverilog-uvm in modern hardware verification," *Journal of Information Systems Engineering and Management*, vol. 10, no. 1, 2025. [Online]. Available: https://jisem-journal.com/index.php/journal/article/view/13812