A.U.R.A. - Attention Unit for Resource-Constrained Architectures

Davis Sheppard, Jeffrey Yang, Artem Demenchuk, Dhruv Dighrasker, Andrew Ye, Yan Cheng Poon
(dwvshep, yangje, amenchuk, dhruvdi, yeandrew, ycpoon)@umich.edu
Department of Electrical and Computer Engineering, University of Michigan

***Abstract*—We present a novel fixed-point ASIC accelerator for edge platforms that implements an IO-aware algorithm that tiles Q, K, and V matrices and fuses softmax operations with surrounding matrix multiplications. We design the quantization scheme, datapath, and control such that all major components of the attention pipeline - including dot-product, softmax normalization, and value accumulation - operate in fixed point while preserving high numerical fidelity. The design operates at a 2.6ns clock period, or about 385 MHz. We implement memory management using a streaming, tile-based organization that minimizes off-chip bandwidth while keeping the PE Block fully utilized. Our experimental results show that our design significantly reduces power and area consumption relative to prior FlashAttention accelerators, making A.U.R.A. particularly well suited for edge deployments that require transformer-class models under tight energy and area budgets.**

***Index Terms*—Attention, Accelerator, Fixed-point**

I. Introduction

Transformers have become the backbone of modern large language and vision models, but scaling them to longer sequence lengths and deploying them under tight power and area budgets remains a major systems challenge. In a transformer, the attention layer is critically important, enabling models to weigh and assign importance to different information, but is the primary bottleneck in scaling to longer sequences: the standard attention mechanism incurs quadratic time and memory in sequence length and performs large matrix operations that are bandwidth-dominated.

FlashAttention [1] and FlashAttention-2 [2] address this by restructuring attention as an IO-aware algorithm that tiles Q, K, and V matrices and fuses softmax operations with surrounding matrix multiplications. This allows computation to be streamed on-chip with minimal memory transactions and linear efficiency (further detailed in Section 2). In addition, Alexandridis et. al [3] show that the expensive exponential and second matrix multiplication in the softmax attention algorithm can be simplified into an efficient fused "Expmul" primitive, converting a compute-intensive exponentiation and multiplication into a simple floating-point addition.

Most existing implementations of the FlashAttention kernel are designed around floating-point units and/or datacenter-class accelerators, limiting their applicability to relatively resource-constrained edge platforms [4], [5], [6]. A.U.R.A is tailored to this domain. It is an RTL implementation of a novel fixed-point ASIC accelerator for FlashAttention-style transformers that integrates an Expmul inspired fused exponential multiplier primitive directly into a tiled FlashAttention-2 dataflow.

Instead of relying on floating-point mantissa operations, our design adopts a carefully chosen family of Q-formats and implements exponential and scaling operations using integer adds and shifts. We co-design the quantization scheme, datapath, and control such that all major components of the attention pipeline - including dot-product, softmax normalization, and value accumulation - operate in fixed-point while preserving high numerical fidelity. Our experimental results show that we achieve accuracy sufficiently close to that of floating point units while reducing power and area constraints relative to prior FlashAttention accelerators, making A.U.R.A. particularly well suited for edge deployments that require transformer-class models under tight energy and area constraints.

II. Background

The attention mechanism of the transformer computes all pairwise attention scores between tokens in a sequence using the projections of the Query (Q), Key (K), and Value (V) matrices. Similarity scores are obtained through the dot product QK^T, scaled by $\frac{1}{\sqrt{d_k}}$, normalized with softmax, and applied to the values. Softmax requires subtracting the maximum score for stability, exponentiating each adjusted value, and dividing by their sum (operations that require careful handling of dynamic range). Although mathematically simple, this workflow accesses large tensors multiple times, and the intermediate states associated with normalization make data movement a dominant cost.

As sequence lengths grow, the number of token interactions increases quadratically, straining both on-chip memory capacity and off-chip bandwidth. Attention often becomes memory-bound due to repeated movement of Q, K, V, and partial softmax statistics, even with efficient matrix-multiply engines. This mismatch between the structure of softmax attention and typical hardware memory hierarchies has motivated IO-aware algorithmic approaches.

FlashAttention and FlashAttention-2 address these challenges by adopting a tiled, streaming dataflow. Instead of materializing the full attention matrix, they process blocks of Q, K, and V that fit into fast SRAM and maintain softmax statistics incrementally across tiles using an online formulation. This eliminates large intermediate buffers and reduces DRAM traffic while preserving exact attention semantics.

FlashAttention-2 further refines the ordering and fusion of operations to improve locality and throughput. These ideas demonstrate that effective attention acceleration depends more on efficient memory organization than on raw compute.

The Expmul primitive tackles a different bottleneck. Traditional softmax requires computing exponentials and then applying a normalization multiplies. Expmul fuses these steps by operating in a domain where the combined effect of exponentiation and scaling can be approximated using basic additions, shifts, or small look-up tables. This significantly reduces arithmetic complexity and obviates the need for expensive floating-point exponential units.

Such algorithmic reorganizations interact strongly with the numerical format used. Floating-point arithmetic provides a wide dynamic range but incurs high area and power costs. Fixed-point Q-formats offer much cheaper computation using integer operations, but require careful scaling to avoid overflow in dot-product paths and precision loss in softmax. Since small perturbations in softmax inputs can meaningfully affect the outputs, fixed-point attention demands coordinated design of scaling, accumulation, and normalization steps.

Edge-based accelerators amplify these constraints. With limited SRAM, narrow memory interfaces, and strict energy budgets, they cannot support the wide floating-point pipelines or large intermediate buffers assumed in datacenter-oriented designs. While FlashAttention reduces memory traffic and Expmul simplifies exponentiation, deploying attention on edge hardware requires a fully integrated approach: a dataflow tailored to small on-chip buffers, numerical formats chosen for fixed-point stability, and datapaths designed to execute all major attention stages under tight resource limits.

III. Methodology: Fixed-Point

Alexandridis et. al show that the hardware-intensive exponential and multiplication operations in the FlashAttention-2 pipeline can be simplified to an atomic exponential-multiplication, implemented with a floating-point mantissa addition operation rather than a dedicated multiply and exponential. Our work builds on this by utilizing fixed-point Q-format representations and arithmetic, reducing floating-point overhead while retaining a high degree of accuracy.

Based on the work of Alexandridis et. al, we utilize the same approximation to calculate $e^x V$, that is, approximating $log_2 e \cdot X$ as $X + X >> 1 - X >> 4$ and calculating $2^{log_2 e} V$. The range of $X + X >> 1 - X >> 4$ is allowed to be between -21.64 and 0 in the original implementation. However, we limit it to an integer between -16 and 0, because the value of $2^{X+X>>1-X>>4}$ quickly closes to 0 when the exponent is more negative than -16. Thus, further increases in the magnitude of the exponent do not significantly change the result.

Additionally, because we round the exponent to an integer, this allows us to perform a right shift of V in place of multiplication. Further, limiting the minimum exponent to -16 allows us to save one stage in the barrel shifter we use to perform that shift.

Input Q, K, V	Q0.7
Multiplication Output	Q1.14
Dot Product Output	Q7.14
ExpMulFixed a and b	Q4.4
ExpMulFixed Internal Difference (X)	Q5.4
ExpMulFixed Exponent	Q4.0

TABLE I
FIXED-POINT FORMATS FOR KEY INTERMEDIATE VALUES.

Algorithm 1 Modified FlashAttention2

for each query $\overrightarrow{q}$ **do**
  for $i \leftarrow 1 : N$ **do**
    $s_i(Q7.12) = dot(\overrightarrow{q}, \overrightarrow{k_i})$
    $m_i(Q7.12) = max(m_{i-1}, s_i)$
    $o_i^* = ExpMulFixed(m_{i-1}, m_i, o_{i-1}^*) + ExpMulFixed(s_i, m_i, v^*)$
  end for
  $[l_N \ o_n] \leftarrow o_N^*$
  o_N / l_N
end for

Working backwards from the exponent, we determined how many fractional bits should be in the fixed-point Q-format for every intermediate value in the pipeline. Given that our final result is an integer and the result of 3 integer operations, we determined that we would need 4 fractional bits for the X value used in the calculation of that integer in order to maintain tolerable precision while trading off no loss in precision for some area savings. Using 4 bits for the fractional portion of X yields a 3% error rate in the calculation of the exponent compared to using full precision, which is 8 bits. These errors are usually off by one shift, a consequence of bits from the right-shift-by-4 not being captured properly in the result.

We generalized this approach to the remainder of the pipeline, recursively calculating the amount of fractional bits in each intermediate value by considering the precision required immediately downstream. See Table 1 for the Q-format settings following an 8-bit input.

At any step where the number of fractional bits needs to be reduced for the next input, such as from the dot product output to ExpMulFixed a and b, the fractional bits which are no longer needed will be rounded and then removed in order to preserve as much precision as possible compared to simply truncating the extra bits.

For each line, the result on the left hand side is rounded to the fixed point format in the parentheses

IV. Description of Design

In this section, we discuss our RTL design, which is implemented and verified in SystemVerilog, and is publicly available at [7]. Simulation and synthesis are accomplished using Synopsys VCS and Design Compiler, which also provide estimations for power and area figures. The design operates

Algorithm 2 ExpMulFixed(a, b, V_{in})

Input: a and b where b $\geq$ a and an N-element vector of *Q0.7* values V_{in}

Output: Vector V_{out} where $V_{out}[i] \approx e^x V_{in}[i]$

1: **for** $i \leftarrow 1 : N$ **do**
2: $x(Q5.4) = a - b$
3: $(log_2 e \cdot X)(Q6.4) = X + X >> 1 - X >> 4$
4: $\hat{L}\ (Q4.0) = \lfloor log_2 e \cdot X \rceil$
5: $V_{out}[i](Q9.17) = V_{in}[i] >> \hat{L}$
6: **end for**

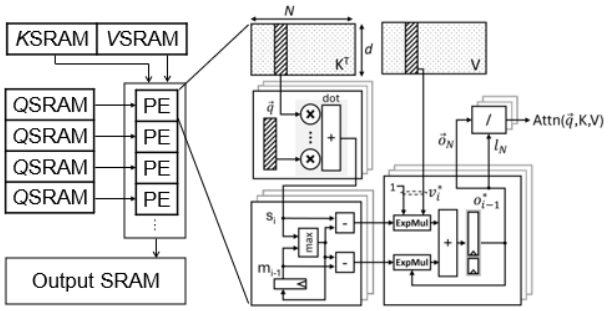

Fig. 1. Top-level architectural design with per-PE diagram (FlashAttention-2 kernel implementation) from Alexandridis et. al [1]

at a 2.6ns clock period, or about 385 MHz. The top-level architecture (Fig. 1) consists of a 1D-array of Processing Elements (PEs), a set of Input SRAM modules holding the Q, K, and V matrices, and an Output SRAM that stores the results of the computation.

We selected an optimal configuration of 4-8 Processing Elements, maximizing energy per token at roughly 47nJ. Each PE implements the FlashAttention-2 kernel and operates entirely in signed fixed-point Q-format arithmetic. A configuration of 4 PEs results in an area (excluding SRAM) of $0.049\mathrm{mm}^2$, a power consumption of 126mW, and a throughput of roughly 2.66 million tokens/sec. A configuration of 8 PEs results in area $0.084\mathrm{mm}^2$, power 220mW, and throughput 7.88 million tokens/sec. Thus, our area scales by $0.08\mathrm{mm}^2$ per Processing Element, with an additional $0.015\mathrm{mm}^2$ for other logic overhead.

We will first describe the PE Block microarchitecture, then detail the memory management scheme that keeps it fully utilized.

A. Processing Elements

Within each PE, computation is decomposed into four RTL blocks: vector dot-product units, maximum-calculator units, staged vector ExpMul units, and vector division units. The dot-product unit takes in a query vector and a key vector and computes a scalar attention score = $\mathbf{q} \cdot \mathbf{k}$, which is represented in an appropriately widened fixed-point format and forwarded downstream to both the max unit and the ExpMul unit.

The max unit maintains a running maximum over scores by making a numerical comparison between the previous max and the current score, outputting both the updated maximum and a delayed copy of the score and associated value vector for alignment in subsequent stages.

The ExpMul unit then forms an augmented vector $\mathbf{v}_\star = [1, \mathbf{v}]^\mathsf{T}$ in a wider internal Q format and performs the fused operation described in Algorithm 2, accumulating partial numerator and denominator terms for the streaming softmax. The operations are divided into two stages: first, subtraction of x = a - b and calculation of $x + x >> 1 - x >> 4$, and second, shifting of V. The shifting is done via a barrel shifter with 5 steps. The first step always performs a right shift by 16, and subsequent steps perform left shifts; thus the total shift will be between 0 and 16 right shifts.

Finally, the vector division unit performs a normalization of the accumulated vector using fixed-point division, yielding the scaled output attention vector for that PE, which is rounded into the target output *Q*0.7 format before being written to the output SRAM. All stages are connected via valid/ready handshaking, enabling each PE in the PE Block to operate as a pipelined datapath that processes shared K/V streams while independently operating on its assigned Q vector. This enables parallel attention computation with high utilization and bounded numerical error under the fixed-point arithmetic scheme.

B. Memory Management

We implement memory management using a streaming, tile-based organization that minimizes off-chip bandwidth while keeping the PE Block fully utilized. The memory controller manages transfers between external main memory (DRAM) and four on-chip SRAM structures: a dual-banked QSRAM (Queries), FIFO-style KSRAM (Keys) and VSRAM (Values), and a dual-banked OSRAM (Output).

Initially, the controller loads keys and values, streaming entire K and V matrices into the KSRAM and VSRAM modules. Once filled, these modules are reused across all subsequent attention tiles without the need to re-access DRAM, which precludes the necessity of constant and expensive data transfers to and from main memory, and thus avoids the latency and energy overhead associated with repeatedly fetching K and V. Instead, the design pays a one-time cost to stage K and V on-chip and then amortizes that cost over all queries in the sequence.

Next, during the compute phase, the controller repeatedly loads Q tiles and drains O tiles. Each Q tile consists of consecutive query vectors stored as rows in QSRAM; the controller assembles each Q vector from a parametrized number of memory blocks into an internal buffer and, upon completion, pushes the vector into the current QSRAM fill bank. QSRAM operates as a "ping-pong" buffer: while one bank is being filled from main memory, the other bank presents a stable array of Q vectors to the PE Block. This double-buffered organization allows memory transfers and computation to overlap so that

the PEs can consume one Q tile while the next tile is being prepared, thereby improving effective PE utilization.

In parallel, KSRAM and VSRAM act as streaming, FIFO-like buffers that present the K and V sequences to the PE Block in a regular, sequential pattern. Because the entire K and V matrices have been staged on-chip, the PEs can sweep over these streams multiple times as needed by the FlashAttention-2 algorithm without incurring additional off-chip traffic. On the output side, an Output SRAM (OSRAM) is organized in a similar tile-based, dual-banked fashion to QSRAM: one bank collects tiles of O vectors produced by the PEs, while the other bank is flushed back to DRAM. This decouples the rate at which the PEs generate results from the rate at which the memory system can accept them, preventing short-term backpressure in the DRAM interface from stalling the compute pipeline.

Overall, this streaming and tiling strategy — staging K and V once, double-buffering Q and O tiles, and treating on-chip SRAMs as structured streams rather than random-access scratchpads — maximizes data reuse, reduces off-chip bandwidth requirements, and contributes directly to the energy-per-token efficiency of the accelerator.

V. Results

Testing and characterization of the system aims to determine the impact of quantization and integer approximation on the accuracy of the attention output. Some amount of error is expected compared to a purely floating point model, mainly due to the approximation of $e^x V$, but it should minimally affect normal operation and is the tradeoff made for reduced area and power consumption.

When running on a randomized set of Q, K, V matrices containing values between -1 and 1 in Q0.7 format, the end-to-end system had a maximum absolute error of 4 LSB difference when compared to a model that uses floating point computation. That translates to a maximum real difference in attention weight of 0.03. The root mean square error is around 1.02 LSB, or 0.008 in the real value. Prior works have shown that accuracy on downstream tasks can be maintained even with some precision loss due to quantization [8].

Given the data shown in Figure 2, the energy efficiency of A.U.R.A. is a massive improvement over GPUs as our design achieves a token throughput on the order of several million tokens per second which rivals that of the H100 Nvidia GPU, yet requires less than half a watt while H100's typically run on the order of 100's of watts. This design also requires less space than the original Expmul ASIC implementation with an area of $0.05mm^2$ on a 45nm process node compared to $0.2mm^2$ on a 28nm process node. We captured power estimates from our synthesized RTL as well, and we should note that these estimates come from an assumed switching probability of 100 percent as well as using a much larger process node of 250nm compared to the Expmul ASIC implementation. Our estimates were on the order of a few hundred milliwatts while the Expmul paper claims numbers around 10 times less than this. We believe our metrics are actually in line with theirs since we used a process node that was almost 10 times as large and obtained a generous upper bound on power consumption given the configurations of our synthesis.

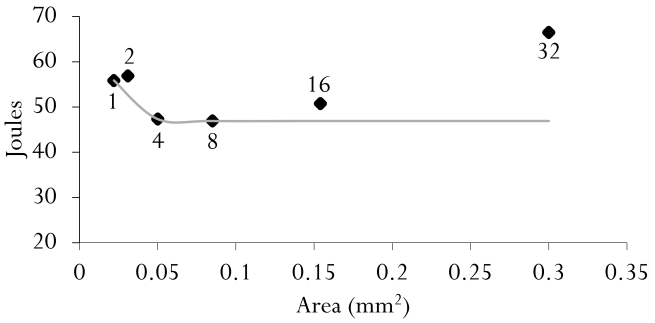

Fig. 2. Energy Per Token vs Area, #PEs

VI. Conclusion

Our design demonstrates the viability of fixed-point dataflow accelerators as an alternative to more costly alternatives such as floating-point processors or GPUs. There is already precedent for using quantized floating point models for edge applications, and this accelerator could lower the energy and area costs for running transformers effectively on small embedded devices. We made our design open source with the intention of future research building on the implementation to add even more flexibility and features common to transformer models such as masking and multi-head support. This project speaks to the broader research area of software-hardware co-design, and we believe our work demonstrates the tremendous potential of custom hardware when deployed in the proper settings.

VII. Work Distribution

Davis: Developed and verified the front end of the RTL including the memory controller and dot-product modules, and led system level integration in the top level module.

Jeffrey: Completed the RTL for the ExpMulFixed operator used and helped in debugging of top level module. Collected data on how changing different fixed-point intermediate values affect final accuracy.

Artem: Led synthesis efforts by writing tcl scripts and setting up cell libraries to characterize the physical implementation of our design.

Andrew: Helped develop and implement RTL for division module and wrote testbench for division verification.

Dhruv: Helped research various division algorithms and fixed point implementations and wrote division module RTL to develop a suitable division module for our ASIC.

Yan Cheng: Led the development of Python and C++ scripts to perform system level verification and accuracy measurements.

REFERENCES

[1] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," arXiv preprint arXiv:2205.14135, 2022.

[2] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," arXiv preprint arXiv:2307.08691, 2023.

[3] K. Alexandridis, V. Titopoulos, and G. Dimitrakopoulos, "Low-Cost FlashAttention with Fused Exponential and Multiplication Hardware Operators," arXiv preprint arXiv:2505.14314, 2025.

[4] E. Kabir, M. A. Kabir, A. R. J. Downey, J. D. Bakos, D. Andrews, and M. Huang, "FAMOUS: Flexible Accelerator for the Attention Mechanism of Transformer on UltraScale+ FPGAs," arXiv preprint arXiv:2409.14023, 2025.

[5] H. Wu, C. Xiao, J. Nie, X. Guo, B. Lou, J. T. H. Wong, Z. Mo, C. Zhang, P. Forys, W. Luk, H. Fan, J. Cheng, T. M. Jones, R. Antonova, R. Mullins, and A. Zhao, "Combating the Memory Walls: Optimization Pathways for Long-Context Agentic LLM Inference," arXiv preprint arXiv:2509.09505, 2025.

[6] NVIDIA Corporation, "NVIDIA H100 Tensor Core GPU Architecture," [Online]. Available: https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c

[7] https://github.com/dwvshep/A.U.R.A.—FlashAttention-ASIC-Accelerator/tree/main

[8] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT Quantization," arXiv preprint arXiv:2101.01321, 2021.