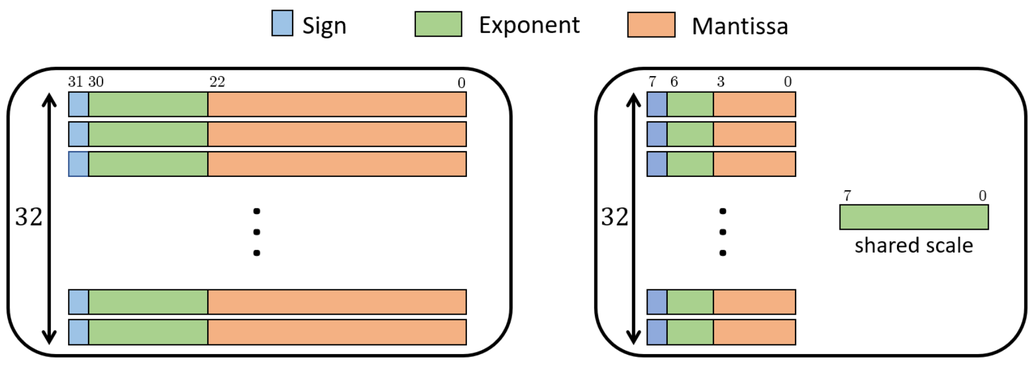
MAC Unit Design for MX Formats

Hong-seung Yu (2019-10519)
Advising professor : Jae-joon Kim

Introduction

With recent advances in AI, models have grown larger, demanding more memory and compute resources. To address this, the Microscaling (MX) format—proposed by major companies at the 2023 OCP Global Summit—has been gaining attention. MX formats such as MXFP8, MXFP6, and MXFP4 can efficiently represent a wide numeric range using only a small number of bits. In this work, a MAC unit supporting all three MX formats was designed in Verilog, with a focus on minimizing accuracy loss during computation. Its efficiency in terms of area, power, and other metrics was also evaluated and verified against an FP32 MAC.

Background



(a) FP32 repr. (b) MXFP_E4M3

Fig 1. Representation of K Numbers in FP32 and MXFP_E4M3 Formats

The MX format is a type of Block Floating-Point (BFP) format in which a block of numbers shares a common scale. Each element can take formats such as FP8, FP6, FP4, or INT8. As shown in Figure 1b, the MX format assigns one shared scale in E8M0 format per group of 32 numbers [1].

Many previously proposed BFP formats use fixed-point elements, which yield advantages in MAC unit hardware and latency [2,3]. In contrast, the MX format carries an individual exponent for each element, requiring a new and efficient MAC unit design that accounts for this characteristic.

Methods

I. Bitwidth Analysis

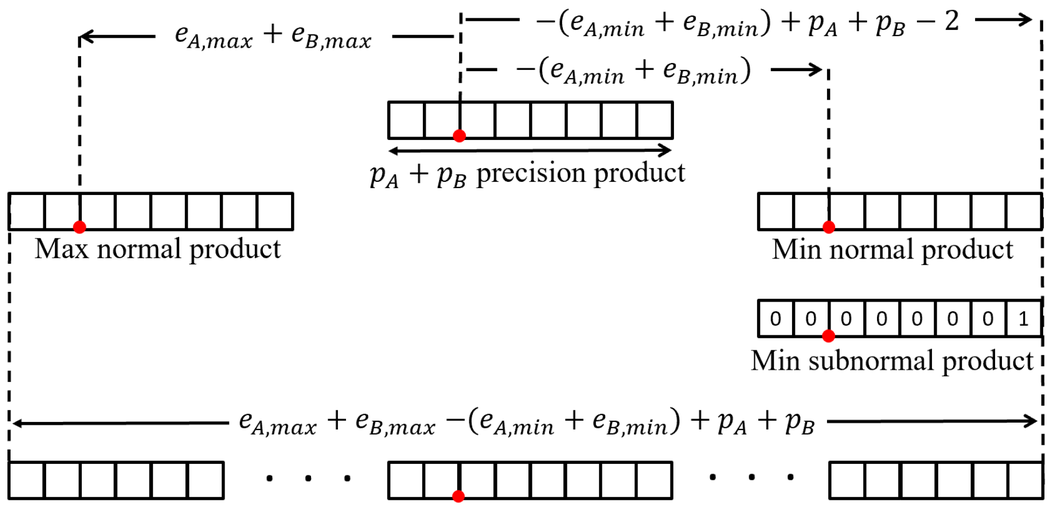


Fig 2. Bitwidth Derivation for Max Left/Right Shift in MX Format

Figure 2 illustrates the process of deriving the bitwidth required to represent the product of two MX-format elements as a fixed-point value. The values of max/min shift and precision (e_max, e_min, p) vary depending on the formats of the two input groups, and the longest bitwidth is required when both A and B use the E4M3 format. Substituting the values yields (8+8) – (-6 – 6) + 4 + 4 = 36 bits as the minimum required.

II. MXFP MAC Architecture

Figure 3 shows the MX MAC unit design, which incorporates an adder tree with a 36-bit datapath. When using FMA operations, adding fp_in while accounting for the shared scale introduces pipeline-stage imbalance, and an adder tree structure is needed to handle block-level computation [4].

In addition, a single unit supports mixed-precision operations. Since the design is based on E4M3 × E4M3, computations on lower-precision elements can also be performed using the same logic.

Finally, when accumulating with the previous result (fp_in), the output bitwidth of the adder tree is 42 bits, requiring a final adder with a 67-bit datapath. To minimize the delay of finding the leading zero during normalization, a barrel shifter is employed to prevent any additional cycles from being introduced.

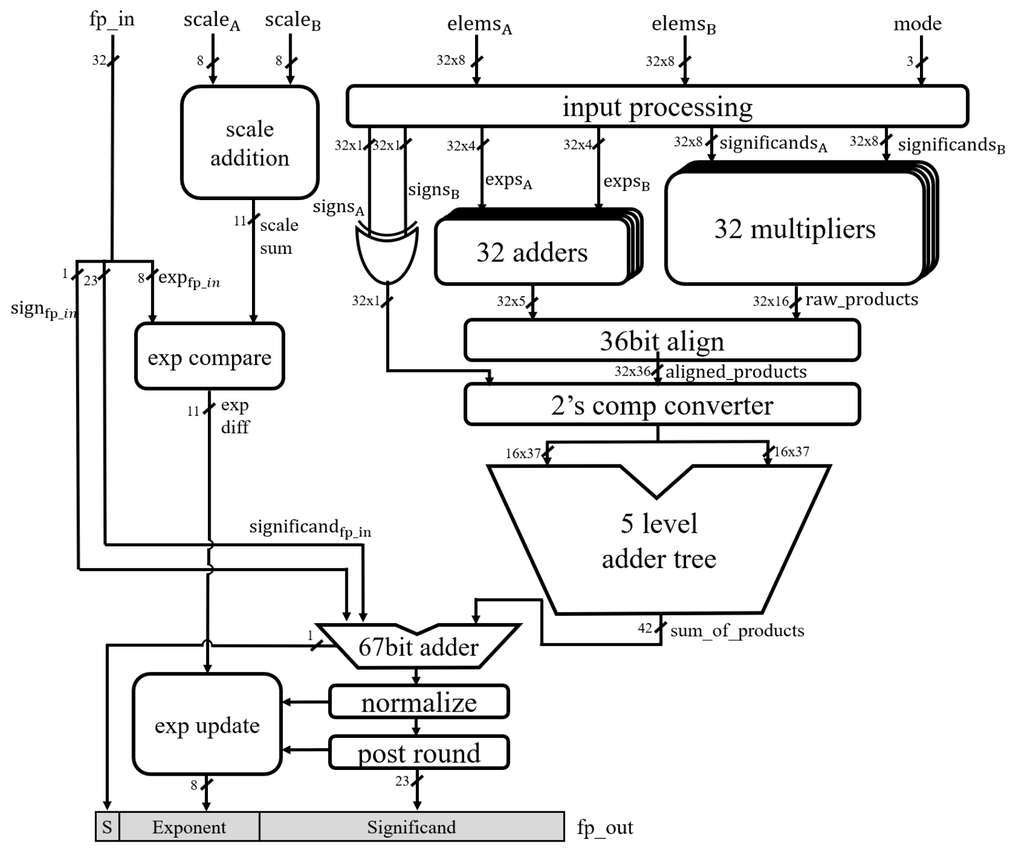


Fig 3. Architecture Overview of MX MAC Unit

Experiments and Results

Table 1. Comparison of FP32 and MXFP MAC

MAC HW	FP32 Baseline	MXFP
Area (μm^2)	45343.93	**24585.79**
Combinational	40464.92	**18727.37**
Noncombinational	**4879.01**	5858.42
Pipeline Stages	6 stages	**4 stages**
Power (μW)	4.75	**2.77**

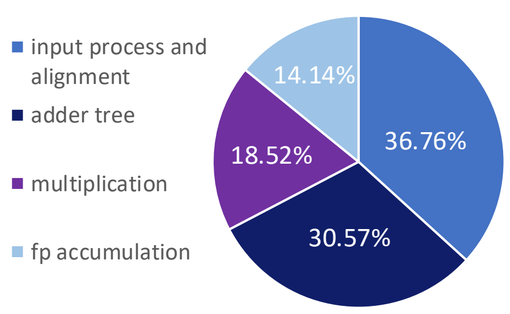


Fig 4. MXFP MAC area breakdown

To verify the performance of the implemented MAC, it was compared against an FP32 baseline MAC built with FP32 multipliers and adders that share the same input and output ports. Synthesis was performed using Synopsys Design Compiler on Samsung 28nm CMOS technology, with the clock period set to 2 ns based on the operation time required by the FP32 adder.

The measurements show that area was reduced by 46%, with power correspondingly reduced by about 42%, and latency was also significantly decreased—cutting the number of pipeline stages from 6 to 4 (Table 1). These improvements can be attributed to the fact that the FP32 baseline requires align logic during addition and performs normalization on the output at every level of the adder tree.

In the final architecture, the largest portion of the logic is occupied by input processing and alignment, followed by the adder tree and multiplication. Shift logic for handling 32 pairs of floating-point elements in parallel takes up a large share of the total, while multiplication, the adder tree, and arithmetic logic together account for only about 50–60% (Figure 4).

Discussion

In the level 1 adder of Figure 3, a 36-bit datapath adder was used. However, for BFP formats with fixed-point elements—including MXINT8—the shifting logic becomes unnecessary, and the corresponding adder datapath becomes significantly shorter. In other words, the logic overhead incurred by representing elements in floating-point rather than fixed-point form appears to be unavoidable.

Several additional optimization strategies are also possible. First, although a long datapath was required to prevent any accuracy drop, resource usage could be reduced by leveraging the trade-off between accuracy and logic. In addition, instead of using a barrel shifter in the fp_accumulator, a sequential shifter could be employed to reduce area and power.

Rather than focusing on a specific application, this study presents a MAC unit design methodology for the MX format that incurs no accuracy drop during computation while jointly considering timing, area, and power. Its significance lies in proposing a general design methodology that can be applied to a wide range of future applications where the MX format may be used.

Reference

[1] Open Compute Project, "OCP Microscaling (MX) Specification," 2023. [Online]. Available: https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf
[2] B. D. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, *et al.*, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," in *NeurIPS*, 2020.
[3] B. D. Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, *et al.*, "With Shared Microexponents, A Little Shifting Goes a Long Way," in *ISCA*, 2023
[4] S. Q. Zhang, B. McDanel, H. T. Kung, *et al.*, "FAST: DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding," in *HPCA*, 2022.