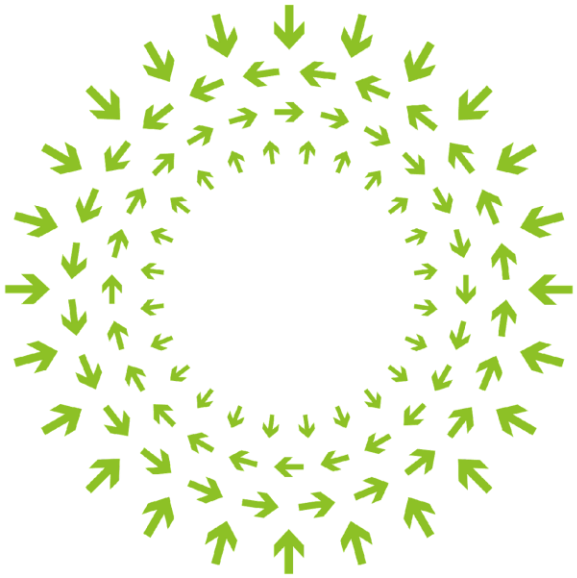


OPEN Compute Project

OCP Microscaling Formats (MX) Specification

Version 1.0

Author: Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Microsoft
Author: Michael Schulte, Ralph Wittig, AMD
Author: Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Arm
Author: Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Intel
Author: Summer Deng, Maxim Naumov, Meta
Author: Paulius Micikevicius, Michael Siu, NVIDIA
Author: Colin Verrilli, Qualcomm

Table of Contents

1. License

Contributions to this Specification are made under the terms and conditions set forth in Open Web Foundation Modified Contributor License Agreement ("OWF CLA 1.0") ("Contribution License") by:

AMD, Arm, Intel, Meta, Microsoft, NVIDIA, Qualcomm

Usage of this Specification is governed by the terms and conditions set forth in **Open Web Foundation Modified Final Specification Agreement ("OWFa 1.0") ("Specification License").**

You can review the applicable OWFa1.0 Specification License(s) referenced above by the contributors to this Specification on the OCP website at http://www.opencompute.org/participate/legal-documents/. For actual executed copies of either agreement, please contact OCP directly.

Notes:

1) The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Compliance with OCP Tenets

2.1. Openness

The Microscaling (MX) specification was developed in collaboration with a consortium of industry members, including AMD, Arm, Intel, Meta, Microsoft, NVIDIA, and Qualcomm. This spec represents an initial effort at standardizing an open, public, and inter-operable family of data formats with a shared, fine-grained block scale. The MX standard also embraces and builds on open data format standards such as OCP FP8.

The MX standard was the result of years of design space exploration, research effort, and production learnings. Ideas and results which led to it have been publicly released in papers such as [1,2,3]. Driving this standardization through OCP will enable us to open and further develop these advancements to the OCP community and others.

2.2 Efficiency

MX-compliant formats enable Artificial Intelligence (AI) training and inference with lower bit-width arithmetic operations and smaller memory footprints. This drives hardware performance and efficiency gains that can reduce overheads and operational costs. Additionally, standardizing new capabilities and formats that can be implemented across hardware and/or software—while enhancing existing standards such as OFP8 and INT8—will reduce software and infrastructure costs and any associated costs or overheads with customized solutions.

2.3 Impact

AI applications have had a transformative impact on the technology landscape. The MX consortium driving this specification includes key ecosystem players including the largest vendors of AI hardware accelerators, as well as major AI service providers who serve customers across the world. Experimental results show that the concrete MX-compliant formats introduced in this standard achieve robust model accuracy for AI training and inference using 8 bits or less. MX-compliant formats have the high potential to become mainstream AI data formats with widespread use and adoption in the technology industry.

2.4 Scale

This specification details the standard for MX-compliant data types. There are currently no sales or usage figures to report. Based on the members in the MX consortium who are willing to ratify the spec, there is strong potential for MX-compliant formats to address major segments of the global AI hardware and AI services market.

To further enable scale, the concrete MX-compliant formats were designed to minimize vendor and end-user friction. The concrete formats were chosen to be compatible with existing hardware devices. Extensive experimental data shows that in many diverse real-world cases, the concrete

MX-compliant formats can be used as a robust alternative for existing training and inference data formats with low user friction.

2.5 Sustainability

AI training and inference is a major (and growing) workload for cloud service providers, contributing significantly to OpEx costs (power and cooling) and carbon emissions of datacenters. MX-compliant data formats are expected to improve the energy efficiency of AI at datacenter scale as well as on many AI endpoints, thus helping to reduce the environmental impact of AI technologies.

3. Version Table

Date	Version #	Author	Description
9/7/2023	1.0	Bita Darvish Rouhani	Defines MX-compliant data formats.

4. Scope

This specification defines Microscaling (MX)-compliant data formats, their binary interchange formats, and basic operations on them. It specifies concrete MX-compliant 8-bit, 6-bit, and 4-bit data formats. Additionally, it defines the binary encodings for 6-bit floating-point (FP6), 4-bit floating-point (FP4), and 8-bit integer (INT8) numbers.

Not in scope are:

- Binary encodings for 8-bit floating-point (FP8). The FP8 encodings in this specification are adopted from the OCP FP8 specification.
- Binary encodings for the element when the block scale is NaN.
- Detailed algorithms for computing the block scale.
- Details of applications' use of specified MX-compliant data formats in general.

This standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware.

4.1. Definitions

Data format: A data format is composed of all fields that are used to represent one or more real values, infinities (positive or negative), or NaN (quiet or signaling). Throughout this specification, the term "data type" is used interchangeably with "data format".

Encoding: A sequence of bits that stores data of a specific data format.

Float32: Binary32 format defined in the IEEE Standard for Floating-Point Arithmetic [4]. It is a scalar floating-point data format. $Vmax_{Float32}$ refers to the largest representable number in Float32.

Mantissa: The part of a finite floating-point number that contains the fraction bits of the significand (this is equivalent to the trailing significand field defined in the IEEE Standard for Floating-Point Arithmetic [4]). The term "mantissa bits" refers to the fraction bits of the significand.

Normal number: For a particular format, a finite non-zero floating-point number with magnitude greater than or equal to b^{emin} where *b* is the radix and *emin* is the minimum representable exponent. Such numbers can use the full precision of the mantissa.

Subnormal number: For a particular format, a non-zero floating-point number with magnitude smaller than the smallest normal number. Such numbers cannot use the full precision of the mantissa. Also known as *denormalized number*.

NaN: Not a number. A symbolic floating-point datum denoting the result of invalid or undefined operations. The IEEE floating-point specification defines two types of NaNs: quiet NaNs and signaling NaNs. This specification does not differentiate between quiet and signaling NaNs.

Inf: Infinity.

roundTiesToEven: A method for rounding to nearest, where ties are broken by rounding them to the value with an even least-significant digit. See IEEE Standard for Floating-Point Arithmetic [4], Section 4.3.1 for additional details.

E*x*M*y*: Notation for a scalar format with one sign bit, *x* exponent bits, and *y* mantissa bits. E.g., E4M3 refers to an FP8 format with one sign bit, four exponent bits, and three mantissa bits. In the case where *y* is zero (e.g., E8M0), the format does not include the sign bit.

4.2. Word Usage

In this specification, key words "*may*", "*should*", and "*must*" are used to specify and differentiate different levels of requirements. They are defined as follows:

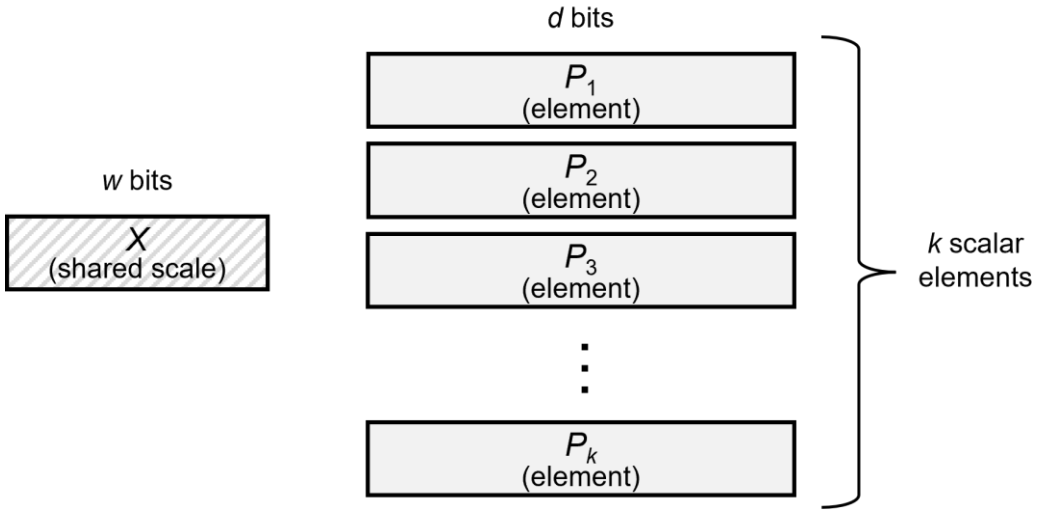
- "*must*" indicates an action is absolutely required to be compliant under this specification.
- "*should*" indicates an action is recommended, though it is not required, and other options are allowed.
- "*may*" indicates an action is optional. Implementations can choose whether to take the action or not.

5. Overview

5.1 Microscaling (MX)

An MX-compliant format is characterized by three components:

- Scale (X) data type / encoding
- Private elements (P_i) data type / encoding
- Scaling block size (k)



All k elements (P_i) have the same data type and, therefore, the same bit-width. The scale factor X is shared across all *k* elements. The data types of the elements and scale are chosen independently. In this sense, MX can be seen as a mechanism to build a vector data type from scalar data types.

The bit widths are represented by the following symbols:

- ***w***: number of bits used to encode the shared scale X
- ***d***: number of bits used to represent each element P_i

Therefore, each block of k elements can be encoded in $(w + kd)$ bits. The layout of the block in physical memory is not prescribed in this specification. If multiple blocks share the same scale factor, an implementation can compress or prune away the repeated scale factors. An implementation can store the scale factor X contiguously with or separately from the *k* elements.

The values $v_1, \ldots, v_k$ represented in an MX block are inferred from the constituent fields as follows:

- If $X = \text{NaN}$, then $v_i = \text{NaN}$ for $1 \leq i \leq k$ regardless of P_i
- If $X \neq \text{NaN}$:
  - If $P_i \in \{\text{Inf}, \text{NaN}\}$, then $v_i = P_i$
  - If $XP_i > Vmax_{Float32}$ or $XP_i < -Vmax_{Float32}$ then v_i is implementation-defined
  - Otherwise, $v_i = XP_i$

Here $Vmax_{Float32}$ is the largest representable number in the Float32 data format. MX-compliant format encodings are expected to be within Float32 range. When the absolute value of an encoded value is larger than $Vmax_{Float32}$, the behavior is implementation-defined. NaN encodings for the block scale can be found in Section 5.4. When the scale is NaN, all values in the MX block are NaNs and the element encodings are ignored.

5.2 Concrete MX-compliant Formats

A concrete MX-compliant format consists of a specific block size k and data types of X and P_i. The following concrete MX-compliant formats are part of this specification. The element and scale data types listed in this table are described in the next section.

Format Name	Element Data Type	Element Bits (d)	Scaling Block Size (k)	Scale Data Type	Scale Bits (w)
MXFP8	FP8 (E5M2)	8	32	E8M0	8
	FP8 (E4M3)				
MXFP6	FP6 (E3M2)	6	32	E8M0	8
	FP6 (E2M3)				
MXFP4	FP4 (E2M1)	4	32	E8M0	8
MXINT8	INT8	8	32	E8M0	8

Table 1. Format names and parameters of concrete MX-compliant formats.

To avoid confusion with existing scalar data types, we explicitly adopt the following naming convention: when referring to any of the MX-compliant data types, we prepend "MX" to the element data type name.

5.2.1 Implementation Compliance

To be compliant with this specification, it is not required for all the concrete MX-compliant formats to be supported. An implementation *may* choose to support any subset of the formats. Different encodings (e.g., E5M2 and E4M3 for MXFP8) are considered different formats. For each supported format, an implementation *must* support the parameters listed in Table 1.

5.3 Element Data Types

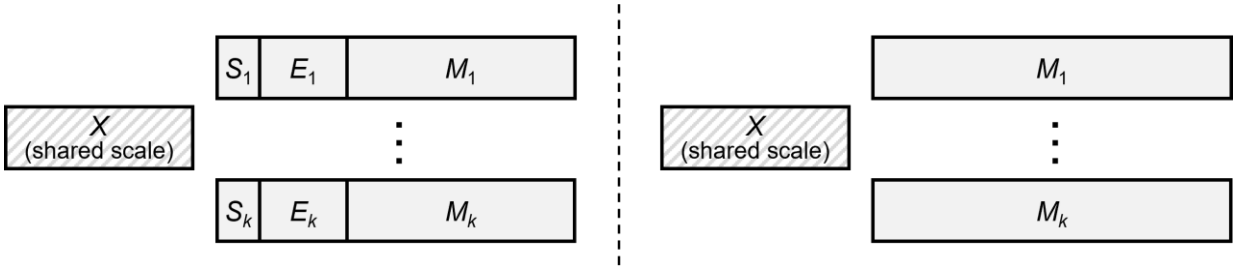
The following element data types are specified for MX-compliant formats. For the FP8, FP6, and FP4 formats, the value of an encoding (excluding Inf and NaN encodings for FP8 listed in Section 5.3.1) is inferred as follows:

a) if $E > 0$, then $v = (-1)^S \times 2^{E-bias} \times (1 + 2^{-m} \times M)$. This is a normal number.
b) if $E = 0$, then $v = (-1)^S \times 2^{1-bias} \times (0 + 2^{-m} \times M)$. This is a subnormal number.

Where:

- S, E, and M are the values of the Sign, Exponent, and Mantissa fields, respectively.
- $bias$ is the exponent bias.
- m is the number of mantissa bits.

The figure below shows the sign, exponent, and mantissa fields in an MX-compliant format with floating-point element data type (left) and an MX-compliant format with integer element data type (right).



5.3.1 FP8

FP8 implementations *must* adhere to OCP 8-bit Floating Point Specification. Table 2 and Table 3 below reproduce information from that specification. An implementation *must* support the saturate (SAT) and overflow (OVF) methods outlined in Table 3 for handling conversions of values from another format to FP8. Other methods *may* be supported, with a configurable overflow attribute to choose between the available methods.

	E4M3	**E5M2**
Exponent bias	7	15
Infinities	N/A	S 11111 00_2
NaN	S 1111 111_2	S 11111 $\{01, 10, 11\}_2$
Zeros	S 0000 000_2	S 00000 00_2
Max normal	S 1111 $110_2 = \pm 2^8 \times 1.75 = \pm 448$	S 11110 $11_2 = \pm 2^{15} \times 1.75 = \pm 57{,}344$
Min normal	S 0001 $000_2 = \pm 2^{-6}$	S 00001 $00_2 = \pm 2^{-14}$
Max subnorm	S 0000 $111_2 = \pm 2^{-6} \times 0.875$	S 00000 $11_2 = \pm 2^{-14} \times 0.75$
Min subnorm	S 0000 $001_2 = \pm 2^{-9}$	S 00000 $01_2 = \pm 2^{-16}$

Table 2. FP8 encoding details. This information is reproduced from Table 2 in the OCP FP8 specification [5].

Source Value (after rounding)	Destination Value			
	E4M3		E5M2	
	SAT	OVF	SAT	OVF
NaN	NaN	NaN	NaN	NaN
±Inf	±max_E4M3	NaN	±max_E5M2	±Inf
Greater than max FP8 magnitude	±max_E4M3	NaN	±max_E5M2	±Inf

Table 3. Special cases when converting to FP8, with the configurable overflow attribute set to OVERFLOW or SATURATE modes. This information is reproduced from Table 3 in the OCP FP8 specification [5].

5.3.2 FP6

FP6 implementations *must* adhere to the table below, with support for subnormals. No encodings are reserved for Inf or NaN in FP6. An implementation *must* support the roundTiesToEven rounding mode for converting values to FP6. Other rounding modes *may* be supported.

During conversion to FP6, if a value exceeds the FP6 representable range after rounding, an implementation *must* support clamping (saturating) the value to the maximum FP6 magnitude, preserving the sign. Other methods *may* be supported, with a configurable overflow attribute to choose between available methods. Out-of-range values can be normal numbers or Infs in a wider data format.

During conversion to FP6, if a value has magnitude less than the minimum subnormal magnitude of FP6 after rounding, an implementation *must* convert the value to zero. Conversion from NaNs is implementation-defined.

	E2M3	E3M2
Exponent bias	1	3
Infinities	N/A	N/A
NaN	N/A	N/A
Zeros	S 00 000_2	S 000 00_2
Max normal	S 11 111_2 = ± 2^2 × 1.875 = ± 7.5	S 111 11_2 = ± 2^4 × 1.75 = ± 28.0
Min normal	S 01 000_2 = ± 2^0 × 1.0 = ± 1.0	S 001 00_2 = ± 2^{-2} × 1.0 = ± 0.25
Max subnorm	S 00 111_2 = ± 2^0 × 0.875 = ± 0.875	S 000 11_2 = ± 2^{-2} × 0.75 = ± 0.1875
Min subnorm	S 00 001_2 = ± 2^0 × 0.125 = ± 0.125	S 000 01_2 = ± 2^{-2} × 0.25 = ± 0.0625

Table 4. FP6 encoding details.

5.3.3 FP4

FP4 implementations *must* adhere to the table below, with support for subnormals. No encodings are reserved for Inf or NaN in FP4. An implementation *must* support the roundTiesToEven rounding mode for converting values to FP4. Other rounding modes *may* be supported.

During conversion to FP4, if a value exceeds the FP4 representable range after rounding, an implementation *must* support clamping (saturating) the value to the maximum FP4 magnitude, preserving the sign. Other methods *may* be supported, with a configurable overflow attribute to choose between available methods. Out-of-range values can be normal numbers or Infs in a wider data format.

During conversion to FP4, if a value has magnitude less than the minimum subnormal magnitude of FP4 after rounding, an implementation *must* convert the value to zero. Conversion from NaNs is implementation-defined.

	E2M1
Exponent bias	1
Infinities	N/A
NaN	N/A
Zeros	S 00 0_2
Max normal	S 11 1_2 = ± 2^2 × 1.5 = ± 6.0
Min normal	S 01 0_2 = ± 2^0 × 1.0 = ± 1.0
Max subnorm	S 00 1_2 = ± 2^0 × 0.5 = ± 0.5
Min subnorm	S 00 1_2 = ± 2^0 × 0.5 = ± 0.5

Table 5. FP4 encoding details.

5.3.4 INT8

INT8 implementations *must* adhere to the table below, with the sole exception of the maximum negative representation of −2 (see the note on 2's complement below). No encodings are reserved for Inf or NaN in INT8. An implementation *must* support the roundTiesToEven rounding mode for converting values to INT8. Other rounding modes *may* be supported.

During conversion to INT8, if a value exceeds the INT8 representable range after rounding, the implementation *must* support clamping (saturating) the value to the maximum INT8 magnitude, preserving the sign. Other methods *may* be supported, with a configurable overflow attribute to choose between available methods. Out-of-range values can be normal numbers or Infs in a wider data format. Conversion from NaNs is implementation-defined.

Integer data types use a 2's complement encoding, but the maximum negative representation (−2) *may* be left unused to maintain symmetry between the maximum positive and negative representations and avoid introducing a negative bias.

The INT8 encodings include an implicit scale factor so that there is one sign bit, one integer bit, and six fractional bits.

	INT8
Exponent bias	N/A
Implicit scale	2^{-6}
Infinities	N/A
NaN	N/A
Zeros	0 0.000000_2
Max (symmetric)	0 1.111111_2 = +1 63/64 1 0.000001_2 = −1 63/64
Min	0 0.000001_2 = +1/64 1 1.111111_2 = −1/64

Table 6. INT8 encoding details.

5.4 Scale Data Types

The following scale data types are specified for MX-compliant formats.

5.4.1 E8M0

E8M0 is an unsigned representation of a conventional biased Float32 exponent. Unlike Float32 exponents, there is no representation for Inf and only a single NaN encoding is reserved.

	E8M0
Exponent bias	127
Supported exponent range	-127 to 127
Infinities	N/A
NaN	11111111_2
Zeros	N/A

Table 7. Scale data type E8M0 encoding details.

6. Basic Operations

This section describes basic operations on MX-compliant formats.

6.1 Dot Product of Two MX-compliant Format Vectors

The dot product of two MX-compliant format vectors A: $\left\{X^{(A)}, \left[P_i^{(A)}\right]_{i=1}^{k}\right\}$ and B: $\left\{X^{(B)}, \left[P_i^{(B)}\right]_{i=1}^{k}\right\}$ of length k is a scalar number C. The following semantics *must* be minimally supported:

$$C = Dot(A, B) = X^{(A)}X^{(B)} \sum_{i=1}^{k} \left(P_i^{(A)} \times P_i^{(B)}\right)$$

Where:

- $X^{(A)}$, $X^{(B)}$ are the block scales of vectors A and B respectively.

- $P_i^{(A)}$, $P_i^{(B)}$ are the i'th element of vectors A and B respectively.

A and B *may* use different data types for either/both scale and element. The internal precision of the dot product and order of operations is implementation-defined. By factoring out the shared scales, the dot product reduction only computes on the elements.

6.2 General Dot Product

The general dot product of two vectors A and B *should* be a scalar Float32 number C. It is assumed that the vectors are padded to have length that is a multiple of scaling block size k. Let the padded vectors A and B have length $n \times k$ consisting of n MX-compliant vectors each of length k, then the general dot product of A and B is defined as:

$$C = DotGeneral(A, B) = \sum_{j=1}^{n} Dot(A_j, B_j)$$

Where $Dot()$ is defined in Section 6.1, and A_j and B_j are the j'th MX-compliant sub-vector of A and B.

6.3 Conversion from Vector of Scalar Elements to MX-compliant Format

A mechanism *must* be provided for converting a k-length vector $V: [V_i]_{i=1}^{k}$ of scalar elements to an MX-compliant format $\{X, [P_i]_{i=1}^{k}\}$ by producing the block scale X and the elements P_i. In particular, the following semantics *should* be minimally supported:

1. Set X to be the largest power-of-two[1] less than or equal to $\max_{V_i \in V}(|V_i|)$, divided by the largest power-of-two representable in the element data type.
2. Set P_i to be the scaled inputs V_i / X quantized to the element data type. For this quantization, normal numbers that exceed the max normal representation of the element data type should be clamped to the max normal, preserving the sign.

Other algorithms for conversion of a vector of scalar elements to MX-compliant formats *may* be supported. For quantization to the element data type, implementations *must* support roundTiesToEven as a rounding mode. Other rounding modes *may* be supported.

7. References

[1] Darvish Rouhani, Bita, et al., "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point", *Advances in Neural Information Processing Systems (NeurIPS),* link, Dec 2020

[1] A power-of-two is a number of the form 2^n where n is a positive integer, negative integer, or zero.

[2] Darvish Rouhani, Bita, et al., "With Shared Microexponents, A Little Shifting Goes a Long Way", *International Symposium on Computer Architecture (ISCA)*, link, June 2023

[3] Darvish Rouhani, Bita, et al., “OCP Microscaling Formats for Deep Learning”, link, Sep 2023

[4] IEEE Computer Society, “IEEE Standard for Floating-Point Arithmetic”, in *IEEE Std 754-2019 (Revision of IEEE 754-2008*), link, June 2019

[5] Micikevicius, Paulius, et al., “OCP 8-bit Floating-Point Specification”, link, June 2023