PT-Float: A Floating-Point Unit with Dynamically Varying Exponent and Fraction Sizes

J.T. de Sousa, J.D. Lopes, M. Serôdio, H.C. Neto, M. Véstias

INESC-ID
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
`https://www.inesc-id.pt/`

Arith 2024 - 31st IEEE International Symposium on Computer Arithmetic
Málaga, June 10-12 2024

Outline

Outline

Motivation

- IEEE 754 floats are too rigid with their 32, 64, and 128-bit sizes
- New algorithms, especially AI, are demanding more flexibility for trading off size and accuracy

$$\Downarrow$$

Research Proposal

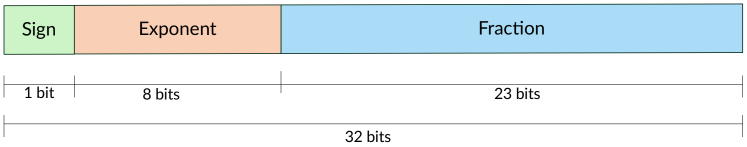
A new floating-point format that uses the concept of Tapered Precision (Morris 1971)

Objectives

- A more optimized tapered precision floating-point format compared with Posits
- Develop a software library and hardware modules
- Leverage the new format in real-world applications

Outline

IEEE-754 Floats

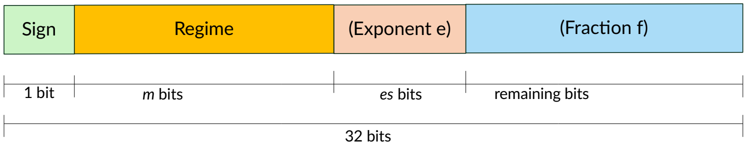


- One sign bit
- Fixed-size exponents (single: 8 bits, double: 11 bits)
- Fixed-size significands: one implicit integer bit and a fixed number of fraction bits
- Fixed dynamic range and precision
- Nice and predictable but inflexible

Tapered Precision

- In 1971, Morris hinted at adding a new field to indicate the exponent size and implicitly the fraction size, making them variable, introducing the idea of Tapered Precision
- Later, Gustafson proposed the Unum I, II, and III tapered precision formats
  - Unum I: has the Morris field but also has redundant representations, wasting a significant number of binary combinations
  - Unum II: uses a large look-up table, no redundant representations, but it is impractical to implement in hardware
  - Unum III (aka Posits): encodes the exponent without using the Morris field, solves both previous problems, and has become the most representative tapered precision format

The Posit Format



$$x = (-1)^s \times 2^{k2^{es}+e} \times 1.f$$

$$k = \begin{cases} -m, & \text{if the regime bits form a string of m zeros} \\ m-1, & \text{if the regime bits form a string of m ones} \end{cases}$$

- Posits solve the redundancy problem as each exponent has a single representation
- The significand, if present, has an implicit integer bit (*1*, like IEEE-754) and a variable number of fraction bits

Unary vs. Binary Exponents

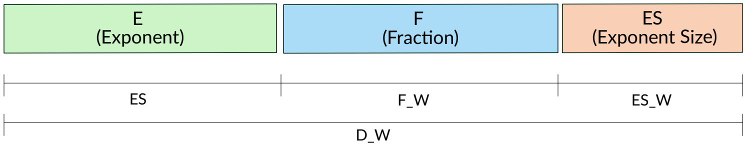
- The effective exponent grows linearly with k (or m)
- The Posit exponent uses a *unary* number system!
- The optimally compact exponent should use a *binary* number system
- We show that the exponent can be *binary*, and still avoid redundant representations

Figure from [Gustafson, "Posit Arithmetic", 2017]

"I've completely lost track. Is it 1 million or 2 million notches to A.D.?"

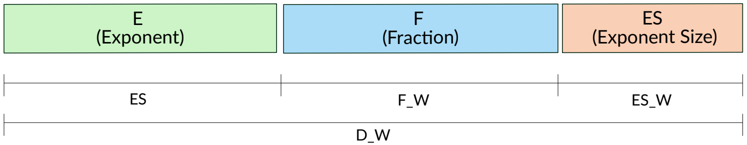
Outline

The PT-Float<D_W,ES_W>



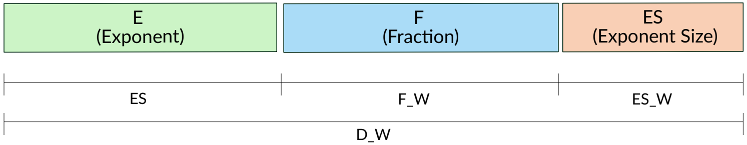
- The key idea is to use the original Morris approach and add an implicit leading bit to the exponent, normalizing its range
- The exponent is formed by the implicit bit, followed by field E, using the 1's complement signed representation (for symmetry)
- The number of bits in E is given in ES, a small unsigned integer
- The significand is formed by an implicit leading bit, binary point, and fraction, using the 2's complement signed representation (for hardware simplicity)

Why Does it Work?



- Example with positive exponents and *1* as the implicit leading bit
  - ES $= 2$, Exponent $=$ *(1)*00 represents the exponent 4
  - ES $= 3$, Exponent $=$ *(1)*000 represents the exponent 8
- The implicit leading bit distinguishes two same-value explicit exponents that have a different number of bits

Implementation with signed numbers



- The PT-Floats use signed representations for the exponent and significand
- This option contrasts with the IEEE-754 and Unum formats
    - It simplifies the adder hardware
    - The implicit leading bit implements the sign

The 2's Complement Significand

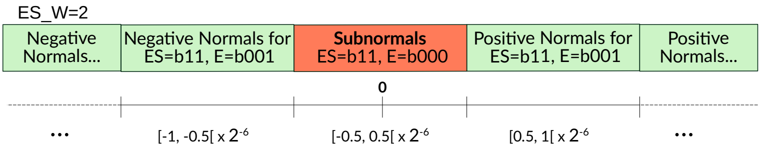
- The implicit leading bit is the fraction MSB negated, except for subnormals
  - Positive significands: implicit leading bit 0, fraction MSB 1
  - Negative significands: implicit leading bit 1, fraction MSB 0
- The significand covers the region [-1, -0.5[+ [0.5, 1[
- For example, for a 4-bit fraction,
  - (0.)1001 represents the positive significand 0.1001 (binary), or 0.5625 (decimal)
  - (1.)0000 represents the negative significand -1.0000

The 1's complement exponent

- The implicit leading bit is the explicit exponent MSB negated
  - Positive exponents: implicit leading bit 0, explicit MSB 1
  - Negative exponents: implicit leading bit 1, explicit MSB 0
- For example, for a 3-bit ES field
  - ES=100 and E=(0)1001 represents the positive exponent +1001 (binary), or +9 (decimal)
  - ES=100 and E=(1)0000 represents the negative exponent -1111 (binary), or -15 (decimal)

Subnormal numbers

- When ES is all ones and E is all zeros, a *subnormal number* is being represented (this combination is reserved)
- Subnormals have exponent $-(2^{2^{ES\_W}-1}-2)$
- The implicit leading bit of the significand becomes the fraction's MSB (no longer negated)
- This approach closes the interval [-0.5, 0.5[for this exponent, and includes 0



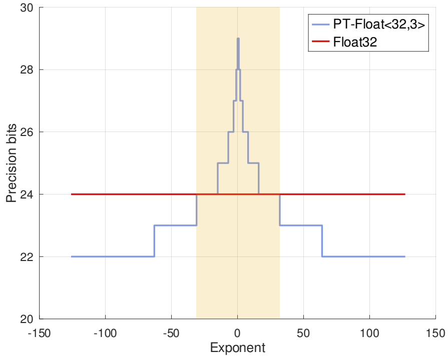
Features

- Optimally compact (binary) exponent representation
- No two binary combinations represent the same number
- All binary combinations represent a different number
- Subnormals are represented by all exponent size bits set to one and all exponent bits set to zero
- Zero is represented by all exponent size bits set to one and all exponent and fraction bits set to zero
- No exceptional patterns such as NaN, $-\infty$, $+\infty$, ∞, 0^+ and 0^-

Outline

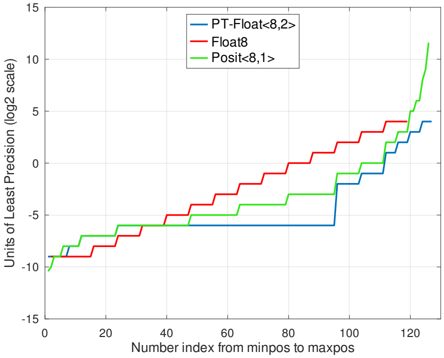
Precision Comparison Example: PT-Float<32,3>

- PT-Float<32,3> reaches a maximum of 8 exponent bits: Dynamic Range similar to IEEE-754 Float32
- The number of precision bits is greater or equal to the Float32 for exponents in the -31 to 31 range: the Golden Interval
- The number of precision bits is lower than the Float32 outside the Golden Interval



ULP Comparison Example with 8 bits

- IEEE-754: the ULP increases linearly with the exponent due to the fixed relative error
- Posit and PT-Float: the ULP increases rapidly near the limits of the range and slows down in the middle of the range
- PT-Float: the ULP is much flatter than the Posit in the middle of the range



General Comparison

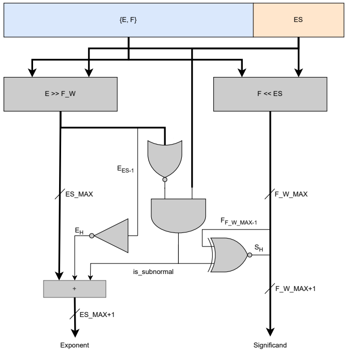
Table: Golden Interval Exponent (GIE), Golden Interval Ratio (GIR), and Dynamic Range (DR) for the different formats.

Format	**Width**	**GIE**	**GIR (log10)**	**DR (log10)**
IEEE-754	16	15	12.04	12.04
Posit ES=2		8	4.82	33.72
PT-Float ES_W=3		7	5.11	77.96
IEEE-754	32	127	83.39	83.39
Posit ES=2		20	12.04	72.25
PT-Float ES_W=3		63	38.83	82.78
PT-Float ES_W=4		31	19.57	19731.31

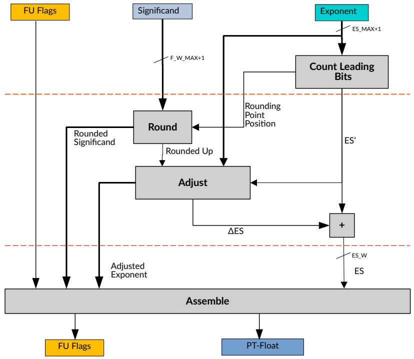
Outline

Hardware Implementation

Unpack Unit



Pack Unit



Outline

Hardware Implementation Results

Table: PT-Float vs. IEEE-754 silicon (130nm) implementation results.

FPU	Data Width	ExpSize Width	Rounding Mode	Area [mm^2]	Power [mW]	Frequency [MHz]
PT-Float	16	3	0	45.19	6.26	200
			1	49.89	7.01	200
	32	3	0	98.31	13.30	200
			1	104.58	14.08	200
		4	0	112.36	14.25	200
			1	123.95	15.38	200
	64	4	0	304.70	40.13	175.19
			1	344.93	38.94	190.25
IEEE-754	32	——	1	67.66	7.44	200
	64	——	1	267.34	27.97	169.15

- PT-Float is roughly 50% larger than IEEE-754 for the same data width
- Target frequency is 200MHz, except when timing closure is not possible

Hardware Implementation Results

Table: PT-Float vs. Posit silicon implementation results.

Format	**Configuration**	**ASIC Area $[mm^2]$**
Posit	<16,1>	31.77*
	<32,2>	97.79*
	<64,3>	327.5*
PT-Float	<16,3>	34.42
	<32,3>	74.14
	<64,4>	240.46

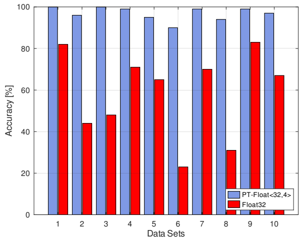
*Estimated

- Results suggest that PT-Float has slightly smaller area than the estimated Posit implementation for the same data width
- Posit results are extrapolated from FPGA results and need to be confirmed with silicon implementation
- PT-Float and Posit grow with the data width in a similar fashion

KNN Algorithm Results

KNN Random Dataset Parameters	Value
Number of datasets (benchmarks)	10
Spacial dimensions	2
Number of data points in each benchmark	100,000
Number of classification labels	4
Number of test points	100
Test point accuracy of reference	Float64

KNN Accuracy Results



- IEEE-754 64-bit floating-point: 100% accuracy. (baseline)
- Datapoints in narrow range within the Golden Interval

Dynamic Range Result

- Datapoints in the Float32 range
- PT-Float<32,4>: all test points classified with accuracy slightly lower than Float64
- Float32: many test point classifications failed due to square distance overflow

Outline

Conclusions

- We propose PT-Float, a new floating-point number system using tapered precision
- The new format solves the problem of redundant representations while maintaining a plain binary exponent representation
- The PT-Float format is simultaneously more precise and has a greater dynamic range than same-size Posits
- Smaller PT-Floats can potentially replace IEEE-754 floats in many applications

Future Work

- Simplify the PT-Float format even further to make it intuitive
- Map signed-integers to PT-Floats orderly to implement comparisons with integer hardware
- Develop a more efficient hardware implementation and integrate PT-Float FPUs in RISC-V processors and CGRAs
- Investigate the use of PT-Float in a comprehensive set of applications