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

José T. de Sousa*, João D. Lopes*, Micaela Serôdio*, Horácio C. Neto* and Mário P. Véstias†
* INESC-ID/IST/UL
Email: jose.t.de.sousa, joao.d.lopes, micaela.serodio, horacio.neto@tecnico.ulisboa.pt
† INESC-ID/ISEL/IPL
Email: mvestias@deetc.isel.ipl.pt

***Abstract*—This paper introduces Precision Trade-Off Floats (PT-Floats). This number system uses the concept of Tapered Precision, a concept introduced by R. Morris in 1971 and later exploited in the Unum formats, most notably by the Posit number system. The idea is to trade off exponent bits with fraction bits according to the application domain. In general, the numbers having an absolute value close to 1 are given more fraction bits and fewer exponent bits, and the numbers with very small or extensive magnitudes are given more exponent bits and fewer fraction bits. This way, near-unit values are kept at high precision while the other values span a vast dynamic range. This work proposes a solution to the problem of redundant representations in the Unum I system. This solution allows the exponent size to be stored as an unsigned binary number, using much less space than Posit's *es* field plus *regime bits*. Our experimental results indicate that the new format reduces hardware resources and energy consumption. We showcase examples where half-size PT-Float units have enough precision, accuracy, and dynamic range to replace full-size IEEE-754 units.**

***Index Terms*—Tapered Precision, Floating-Point, Unum Numbers, Posit, Computer Arithmetic.**

I. Introduction

Since most algorithms require arithmetic operations on real numbers, floating-point formats are the most extensively adopted approximation of real numbers. During the 1960s and the 1970s, there was no ubiquity in the floating-point format. Each computer manufacturer developed its floating-point system, resulting in floating-point inconsistency across platforms. In 1985, the IEEE Standard for Floating-point Arithmetic (IEEE-754) was established by the Institute of Electrical and Electronics Engineers (IEEE) and is nowadays the most common representation of real numbers on computers. This format has been reviewed in 2008 [1] and 2019 [2], but there were only minor changes between them to maintain compatibility with the existing implementations. The problems that have been identified in the IEEE-754 Standard [3], [4], [5] are the following:

- Overflow and Underflow: overflowing to $-\infty$ or $+\infty$ and underflowing to 0 increases the relative error by an infinite factor.
- No Gradual Overflow and Fixed-Precision: overflow happens suddenly after running out exponent bits; precision is flat across a vast range, then abruptly collapses.
- Wasted bit-patterns: there are too many NaN representations, two bit-patterns are used to represent 0, the "negative zero" (0^-) and the "positive zero" (0^+), and another two bit-patterns to represent infinity, $-\infty$, and $+\infty$.
- Exponents usually waste too many bits.

Over the years, several systems and techniques have been proposed to overcome these challenges. R. Morris, in his 1971 seminal 2-page paper [6], suggested a "tapered" system to allow trading-off exponent bits with fraction bits and vice-versa [7]. In 2013, John L. Gustafson introduced a new binary representation of real numbers, a number system called Universal Numbers or Unum [3], [8], [9]. So far, there are three different Unum types. Type-I [3] is a superset of the IEEE-754 Standard. This format uses a variable-length storage format for the exponent and fraction fields and a "ubit" at the end of the fraction that indicates if the number is an exact float (u=0) or lies in the open interval between two consecutive floats (u=1). Type-II [10] enables a clean mathematical design based on *projective reals* and relies on lookup tables. It directly maps signed integers onto a projective real number line. Type-III, or Posits [11], [12], was introduced in 2017 as a hardware-friendly version with all the advantages of the previous types. Nowadays, even though many shortcomings have been pointed out in the IEEE-754 Standard, it is still the most commonly implemented method in computing machines to perform floating-point arithmetic.

This paper introduces the Precision Trade-Off Floating-Point (PT-Float) number system. PT-Float implements an efficient variation of the tapered precision originally suggested by R. Morris and later pursued by the Unum systems. The PT-Floats go back to using a field to represent the variable size of the exponent like Unum I but solve its redundant representations problem by introducing an exponent normalization scheme. This solution provides the intended tapered precision without unnecessarily losing precious precision bits like the Posit regime bits. As shown in the following sections, the new PT-float number system can provide more precision, accuracy, and dynamic range than the Posit numbers with the same size

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020 (DOI:10.54499/UIDB/50021/2020). João Lopes acknowledges FCT for the support through grant 2021.05020.BD.

while still offering more dynamic range that IEEE-754 floats twice the size. Moreover, the hardware resources to implement PT-Float units are similar to those of Posits.

II. The PT-Float Format

The present PT-Float format explores and combines ideas from the Unum Type-I and Type-III formats, explicitly using an exponent width field and dynamically varying fraction and exponent widths. It solves the Type-I redundant representations problem using a complementary-valued hidden bit in the exponent. Like the Posits, it eliminates using NaN, $-\infty$, $+\infty$, $+0$ and -0 representations.

A. Generic Format

The PT-Float format is designed to represent a floating-point number x with significand s and binary exponent e:

$$x = s \times 2^e \tag{1}$$

The format uses three fields: the explicit exponent (E), the explicit fraction (F), and the exponent size in bits (ES), as shown in Figure 1. The notation PT-Float<D_W, ES_W> represents a PT-Float number with D_W bits, whose exponent size ES can be specified with ES_W bits. Hence, the fraction's number of bits F_W is given by the remaining free bits:

$$F\_W = D\_W - ES\_W - ES \tag{2}$$

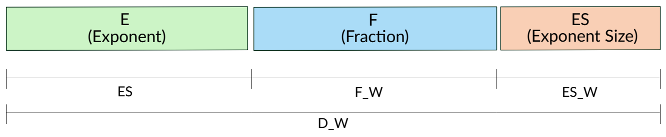

Fig. 1. Generic PT-Float representation format.

1) Exponent: The base-2 exponent e depends on the ES value according to the following expression:

$$e = \begin{cases} 0, \text{ if } ES = 0, \text{ zero} \\ \\ -2^{ES} + 2, \text{ if } ES = 2^{\text{ES\_W}} - 1 \wedge E = 0, \text{ subnormals} \\ \\ -\overline{E_{ES-1}}(2^{ES} - 1) + \sum_{i=0}^{ES-1} E_i \cdot 2^i, \text{ normals} \end{cases} \tag{3}$$

The above equation shows that, for the normal numbers, the exponent e is given in the one's complement binary format:

$$e = (E_H)E_{ES-1}E_{ES-1}...E_1E_0 \tag{4}$$

where E_H is the exponent hidden bit, obtained by negating the most significant explicit bit:

$$E_H = \overline{E_{ES-1}} \tag{5}$$

The one's complement notation is unusual, but the two's complement format would make positive and negative exponents asymmetric and leave the exponent -1 out.

The exponent hidden bit has two purposes: (1) it provides the exponent **sign** information, and (2) it normalizes the exponent by forcing its **magnitude** to be in the interval $[2^{ES-1}, 2^{ES} - 1]$. To have exponents outside this interval, one must change the value stored in the ES field.

2) Significand: The significand s is given by its two's complement binary notation S:

$$S = (S_H).F \tag{6}$$

where S_H is the significand's hidden bit, and, to the right of the binary point, F is the fraction binary representation:

$$F = F_{-1}F_{-2}...F_{-(F\_W-1)}F_{-F\_W} \tag{7}$$

The hidden bit S_H is given by:

$$S_H = \begin{cases} F_{-1} & \text{subnormals} \\ \\ \overline{F_{-1}} & \text{normals} \end{cases} \tag{8}$$

The significand hidden bit has two purposes: (1) it provides the significand sign information, and (2), when $S_H = \overline{F_{-1}}$, it normalizes the significand by forcing its magnitude to be in the interval $[2^{-1}, 1 - 2^{-F\_W}]$ for positive numbers or $[-1, -2^{-1} - 2^{-F\_W}]$ for negative numbers.

Table I presents numerical examples of the PT-Float<8, 2> format to fully illustrate the extraction of the exponent value (from the exponent field + hidden bit) and of the significand value (from the fraction field + hidden bit).

B. Features

The PT-Float format's main features are summarized in the following subsections.

1) Tapered precision with minimal overhead: The PT-Float format can trade off exponent and significand bits with the minimal overhead of log2(ES) bits.

2) No redundant representations: In the PT-Float format, no two binary combinations represent the same number.

3) No unused combinations: In the PT-Float format, all binary combinations represent a number.

4) No Exceptional Patterns: The PT-Float format does not have any patterns such as "NaN", $-\infty$, $+\infty$, ∞, 0^+ and 0^-. Instead, these particular cases are flagged in the hardware implementation with "overflow", "underflow" and "divide by 0" exceptions. This option simplifies the hardware.

III. Analysis

A. Qualitative Comparison

Table II compares the most relevant features of PT-Float, Unum Type-III (aka Posit), and IEEE-754. The IEEE-754 floats have a fixed exponent and fraction size, whereas the PT-Float and Posit formats have extra information to indicate the size of the exponent and, consequently, the fraction. These variable-precision formats are sometimes criticized for not having a constant relative error. The PT-Float offers a mid-ground between the IEEE-754 and the Posits, as seen in section III-C.

TABLE I
PT-Float<8, 2> numerical examples.

PT-Float<8, 2>	Fields			Extracted Values			
	ES	E	F	exponent	significand	value	comment
00000000	00	0	000000	no bits = 0	(1)000000 = -1.0	-1.0	
11000001	01	1	10000	(0)1 = 1	(0)10000 = 0.5	1.0	
11101010	10	11	1010	(0)11 = 3	(0).1010 = 0.625	5.0	
00100001	01	0	01000	(1)0 = -1	(1).01000 = -0.75	-0.375	
11111111	11	111	111	(0)111 = 7	(0).111 = 0.875	112.0	max
11100011	11	111	000	(0)111 = 7	(1).000 = -1.0	-128.0	min
00000011	11	000	000	subnormal = -6	(0).000 = 0.0	0.0	zero
00000111	11	000	001	subnormal = -6	(0).001 = 0.125	0.001953125	minp

TABLE II
Qualitative comparison of IEEE-754, Posit and PT-Float.

Features	IEEE-754 [n-bit]	Posits [n-bit]	PT-Float [n-bit]
Portability/Reproducibility	No	Yes	Yes
NaNs Representations	Many	None	None
Infinity Representations	2 $(-\infty; +\infty)$	1 (NaR)	None
Zero Representations	2 $(0^-; 0^+)$	1 (0)	1 (0)
Overflow	Sudden: after all exponent bits used	Graceful: after all bits used as exponent	Graceful: after max number of exponent bits used
Underflow	Gradual	Gradual	Gradual
Exponent	Fixed-Size; Signed (Biased); 0 Implicit Leading Bits	Varying-Size; Signed (Unary) 0 Implicit Leading Bits	Varying-Size; Signed (1's Complement); 1 Implicit Leading Bit
Significand	Fixed-Size; Unsigned; 1 Implicit Leading Bit	Varying-Size; Unsigned 1 Implicit Leading Bit	Varying-Size; Signed (2's Complement); 1 Implicit Leading Bit
Precision Bits	Fixed	Dynamically Varying	Dynamically Varying

B. Precision and Dynamic Range

The number of bits of the fraction F gives the precision. The Dynamic Range (DR) is the ratio between the largest and smallest positive representable numbers, *minp* (subnormal) and *maxp*, given by

$$\begin{cases} maxp = 2^{2^{2^{\text{ES\_W}}-1}-1} \times \left(1 - 2^{-\left[\text{D\_W}-\left(2^{\text{ES\_W}}-1\right)-\text{ES\_W}\right]}\right) \\ minp = 2^{-2^{2^{\text{ES\_W}}-1}+2} \times 2^{-\left[\text{D\_W}-\left(2^{\text{ES\_W}}-1\right)-\text{ES\_W}\right]} \end{cases} \tag{9}$$

Hence, the dynamic range is approximately calculated by

$$DR \approx 2^{2^{2^{\text{ES\_W}}}-3} \tag{10}$$

Tables III, IV and V compare the dynamic range and precision intervals for the PT-Float, IEEE-754, and Posit formats.

C. Golden Interval

A number format's Golden Interval (GI) is the positive number interval where the numbers have equal or greater precision than the IEEE-754 standard of the same data width. The Golden Interval Exponent (GIE) is the binary exponent

TABLE III
PT-Float dynamic ranges and precision intervals.

D_W, ES_W	*minp*	*maxp* $(\approx)$	DR (log10)	*F_W*
8, 2	2^{-9}	0.875×2^{7}	4.76	3 to 6
16, 2	2^{-17}	0.9995×2^{7}	7.22	11 to 14
16, 3	2^{-132}	0.984×2^{127}	77.96	6 to 13
32, 3	2^{-148}	2^{127}	82.78	22 to 29
32, 4	2^{-32779}	2^{32767}	19,731.31	13 to 28
64, 4	2^{-32811}	2^{32767}	19,740.95	55 to 60
128, 4	2^{-32875}	2^{32767}	19,760.21	109 to 124

that defines the golden interval such that, if GIE=x, then the golden interval is defined as the interval from 2^{-x} to 2^{x}. The Golden Interval Ratio (GIR) is the ratio between these two limits and is used to compare the magnitude of the GIs. Table VI shows GIE, GIR, and DR for selected IEEE-754, Posit, and PT-Float formats.

Table VI shows that the PT-Floats always have a larger GIR than the Posit numbers with the same size and almost always

TABLE IV
IEEE-754 dynamic ranges and precision intervals.

D_W	*minp*	*maxp* ($\approx$)	DR (log10)	*F_W*
8	2^{-9}	1.875×2^7	5.09	3
16	2^{-24}	1.999×2^{15}	12.04	10
32	2^{-149}	2^{128}	83.39	23
64	2^{-1074}	2^{1024}	631.56	52
128	2^{-16494}	2^{16384}	$9,897.26$	112

TABLE V
Posit dynamic ranges and precision intervals.

D_W, ES	*minp*	*maxp*	DR (log10)	*F_W*
8, 1	2^{-12}	2^{12}	7.22	0 to 4
16, 2	2^{-56}	2^{56}	33.72	0 to 11
32, 2	2^{-120}	2^{120}	72.25	0 to 27
64, 2	2^{-248}	2^{248}	149.31	0 to 59
128, 2	2^{-504}	2^{504}	303.44	0 to 123

TABLE VI
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	8	7	5.09	5.09
Posit ES=1		2	1.19	7.22
PT-Float ES_W=2		3	2.68	4.76
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
IEEE-754	64	1023	631.56	631.56
Posit ES=2		32	19.27	149.31
PT-Float ES_W=4		255	154.43	19740.95
IEEE-754	128	16381	9897.26	9897.26
Posit ES=2		48	28.90	303.44
PT-Float ES_W=4		4095	2466.34	19760.21

have a much larger DR than the IEEE floats with the same size (except for the tiny 8-bit format). Moreover, the DR of the PT-Float16 is similar to the DR of the IEEE-float32, and the DRs of the PT-Float32(64) can be larger than the DRs of the IEEE-float64(128), which indicates that the PT-Floats can provide an adequate replacement for IEEE-floats with twice the data size.

Figure 2 shows the number of fraction bits as a function of the exponent range for the PT-Float<32,3> and PT-Float<32,4> formats (for normalized numbers). The golden interval is highlighted, and the horizontal line shows the IEEE-754 Float32 fixed precision (of normals). In the golden interval, the PT-Float formats have more fraction bits than the IEEE Float32 format. The precision of the PT-Floats peaks at the unit value and decreases when receding from this value, as expected from its tapered nature.

The golden interval of the PT-Float<32,3> is wider than PT-Float<32,4>, and conversely, its dynamic range is smaller, which illustrates the trade-off. Near the unit value, the PT-Float<32,3> and PT-Float<32,4> achieve maximums of 29 and 28 precision bits, outperforming the IEEE Float32 by 6 and 5 bits, respectively. While the IEEE-754 floats overflow sharply (right-end of the horizontal line, maximum exponent = 128) or underflow (left-end of the horizontal line, minimum exponent = -149), the PT-Floats gain or lose exponent bits to overflow and underflow, respectively, gracefully.

D. Accuracy

This section analyzes the accuracy of the PT-Float<8,2> format against the Posit<8,1> and the non-standard 8-bit IEEE-754 floats (Float8). The Float8 has one sign bit, four exponent bits, and three fraction bits. These 8-bit formats are selected because they have comparable dynamic ranges, and with only 8 bits, one can quickly analyze all the 256 combinations. The golden interval and dynamic range properties for these formats are given in Table VI.

Figure 3 compares the above three formats using the Decimals of Accuracy metric, given in the figure's vertical axis and explained in [11]. The metric is computed for the positive range of each number system, ordered from the minimum positive (minp) to the maximum positive (maxp) representable numbers of each format.

The PT-Float<8,2> is less accurate than the Float8 in the left and right extremities. The Float8 has tapered accuracy on the left as they use subnormals to obtain a gradual underflow. On the right side, the floats abruptly overflow, but all the same waste 14 NaN values. In contrast, since the PT-Float<8,2> format also has tapered accuracy on the right side, its accuracy gracefully degrades before the inevitable overflow.

The PT-Float<8,2> is also more accurate than the Posit<8,1> in the extremities and the vicinity of the unit value. Its wider GI shows that the PT-Float<8,2> numbers have more accuracy than the Posit<8,1> along the whole range. Both systems cover the 256 patterns and do not waste combinations, except for the combination that means NaR for the Posit.

Figure 4 compares the three formats using the Units of Least Precision (ULP) metric: the distance between two consecutive numbers. The ULP measures the resolution of a floating-point format. The number of fraction bits of the Float8 is fixed to 3; hence, ULP $= 2^{-3} \times 2^e$, depending exclusively on the

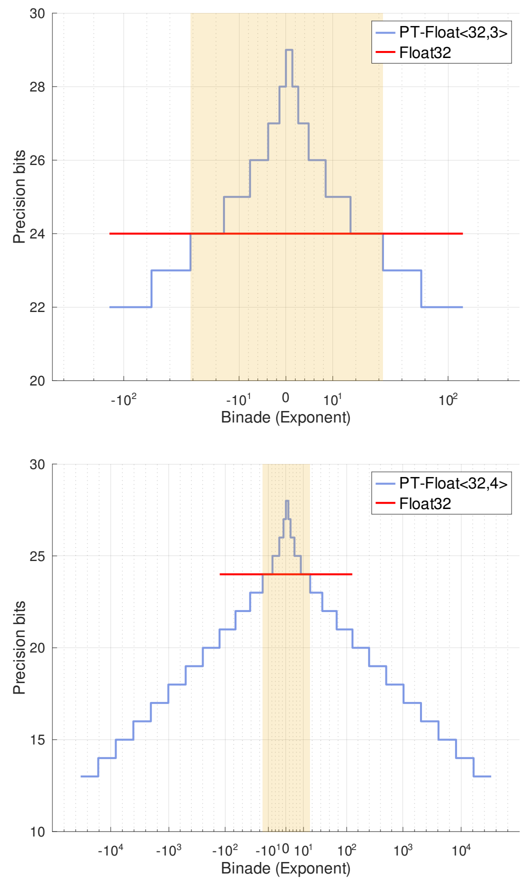

Fig. 2. Exponent vs. fraction bits for PT-Float<32,3> and 32-bit IEEE-754 format (top); exponent vs. fraction bits for PT-Float<32,4> and 32-bit IEEE-754 format (bottom)

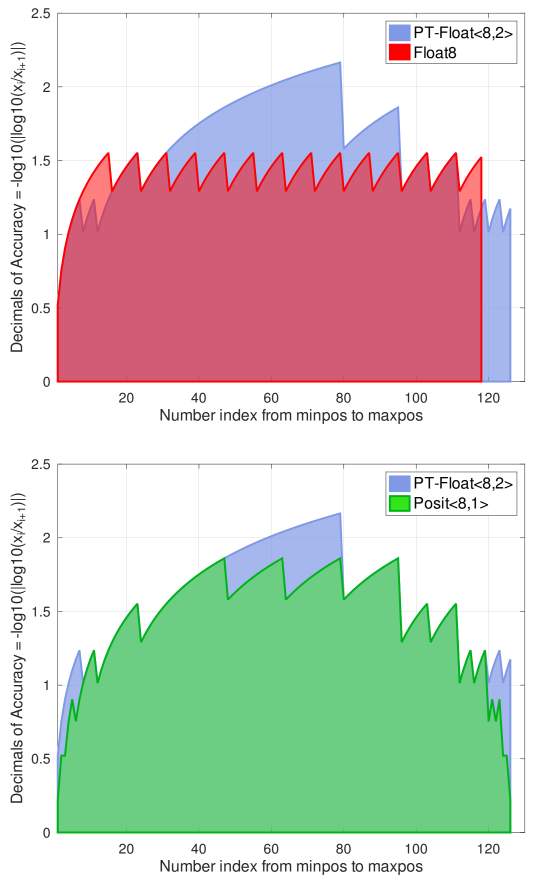

Fig. 3. Decimals of accuracy comparison between PT-Float<8,2> and Float8 (top), and between PT-Float<8,2> and Posit<8,1> (bottom).

fixed exponent bits. For the PT-Float<8,2> and Posit<8,1> formats, the ULP variation slows as the numbers get closer to one (more fraction bits) and accelerates as they recede from one (less fraction bits). The PT-Float<8,2> has a higher resolution for a considerable portion of the range than the other formats, as shown by its lower and flatter ULP. Note that the horizontal axis metric is not the numbers but their index from the minimum positive to the maximum positive. Hence, Figure 4 also shows the numbers' density peaks around the unit.

IV. Hardware Implementation

This section describes the implementation of a pipelined Floating-Point Arithmetic Unit based on the new PT-Float format, accepting D_W and ES_W as architectural parameters. The unit has three main Functional Units (FUs): Unpack, Process, and Pack. The Process unit supports four floating-point operations: addition, subtraction, division, and multiplication. Table VII characterizes the FUs regarding latency, throughput, and number of stages.

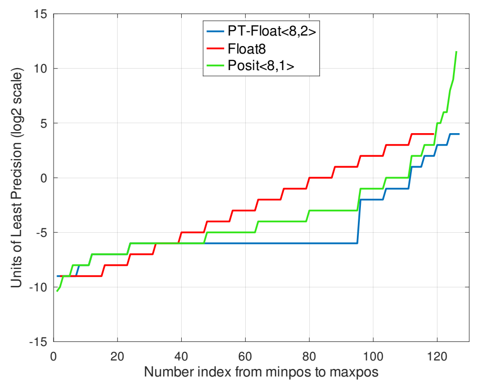

Fig. 4. ULP comparison of Float8, Posit<8,1> and PT-Float<8,2>.

A. Unpack Module

The Unpack Module takes a PT-Float input and extracts the exponent size, the exponent itself, and the fraction. This unit is illustrated in the block diagram in Figure 5.

B. Process Module

After unpacking the exponent and significand, standard floating point hardware can be used, provided they support

TABLE VII
PT-Float Functional Units.

Operation	Latency (cycles)	Pipelined	#Stages	Throughput (op/cycle)
unpack	1	yes	1	1
add/sub	6	yes	6	1
mult	4	yes	4	1
div	6+F_W	no	1	1/latency
pack	3	yes	3	1

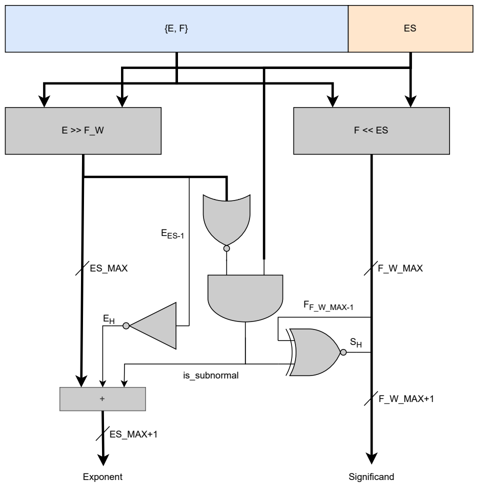

Fig. 5. 1-stage pipelined Unpack module.

the maximum exponent and significant sizes. A macro chooses the rounding mode: truncation or “round to the nearest, ties even”. Two bits select one out of the four available operations. The addition/subtraction and multiplication modules are pipelined. The divider is currently implemented as an unpipelined subtract-shift sequential unit.

C. Pack Module

The Pack Module takes the exponent and significand values produced by the Process module and outputs a PT-Float number and interrupt flags, as shown by the block diagram of Figure 6.

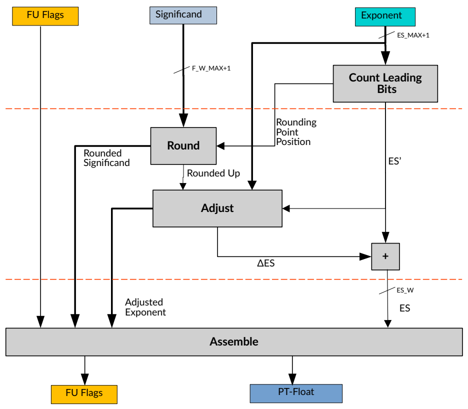

Fig. 6. 3-stage pipelined Pack module.

Although not shown before, the FUs produce underflow and overflow flags, which are inputs of the Pack unit. The packing algorithm can also issue such flags, so these two sources of flags are combined inside the module.

If the rounding mode is disabled (truncation mode), the packing algorithm is the unpacking algorithm reversed: detect subnormal mode, compute the hidden bits, shift the exponent left and the significand right, and merge them into a PT-Float word, appending the exponent size ES computed by the Count Leading Bits module. The leading bits may be zeros for positive or ones for negative exponents. The Assemble module executes the described packing algorithm, receiving ES and the unchanged exponent and significand from the FU.

If the rounding mode is enabled, the packing algorithm is slightly more complicated, but this additional complexity is also present in other formats. In this case, the significand must be rounded and the exponent and its size eventually adjusted. When the significand is rounded, if it is rounded up, the exponent might need to be incremented. In the PT-Float format, incrementing the exponent may also imply that the exponent size needs to be incremented or decremented. The Adjust module performs this function and outputs the adjusted (or not) exponent and its size increment ΔES, which may be -1, 0, or +1.

D. Hardware Implementation Results

This section compares the ASIC implementation results for the three formats, PT-Float, Posit, and IEEE-754 FPU, using the Cadence RTL Compiler for the UMC 130 nm process. The operating frequency limit was set to 200MHz, and no attempt was made to increase it further. At the current stage of development, frequency optimizations have not yet been considered but will be in future developments. When 200MHz has not been achieved, the maximum frequency achieved is given. Silicon area and power consumption results are also presented.

1) IEEE-754 vs. PT-Float: Table VIII compares the ASIC implementation results of the PT-Float FPU with the IEEE-754 FPU. Both FPUs are parameterizable, so they are compared with different configurations.

TABLE VIII
PT-Float vs. IEEE-754 silicon 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

These results show that the Float32 FPU is smaller than the PT-Float<32,3|4> FPUs. This difference is explained by the fact that the PT-Float FPU’s exponent and fraction must be internally extended to their maximum size ES_MAX and F_W_MAX. Another reason is the dynamically varying exponent and fraction sizes, which require more complex Pack and Unpack hardware.

However, replacing an IEEE-754 FPU with a smaller PT-Float FPU may be possible and advantageous. For example, if the PT-Float<16,3> replaces the Float32, it would be 1.36× smaller. Referring to Table VI, the Float32 has GIR=DR=83 decades. The PT-Float<16,3> can offer GIR=5.1 and DR=78 decades as a replacement. Another example is to replace a Float64 FPU (GIR=DR=632 decades) with a PT-Float<32,4>. The latter is 2.64× smaller than the former but can offer GIR=20 and DR=19731 decades as a replacement.

2) Posit vs. PT-Float: Since we do not yet have a Posit FPU implementation, the FPGA results published in [13] were used to estimate its silicon area (using the scale proposed in [14]). We used 10 and 7 equivalent NAND2 gates for each FPGA LUT6 and FF, respectively. The Posit implementation is a pipelined FPU with an adder and a multiplier in the following standard configurations: Posit<16,1>, Posit<32,2> and Posit<64,3>. The comparable PT-Float FPUs used similar configurations and rounding modes: PT-Float<16,3>, PT-Float<32,3> and PT-Float<64,4>. The results are presented in Table IX.

TABLE IX
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

These results show that the PT-Float's silicon area is similar to or smaller than the Posit's. The PT-Float<16,3> and the Posit<16,1> have a similar silicon area. However, the PT-Float<16,3> has GIR=5.1 and DR=78 decades, whereas the Posit<16,1> has GIR=3.6 and DR=17 decades, only. The PT-Float<32,3> uses 25% less area than Posit<32,2> but the former has GIR=39 and DR=83 decades, whereas the latter has GIR=12 and DR=72, only. Finally, the PT-Float<64,4> uses nearly 36% less area than the Posit<64,3>, with GIR=154 and DR=19740 decades, compared to GIR=34 and DR=299 decades for the Posit<64,3>.

V. Case Study: the KNN Algorithm

The KNN algorithm is a supervised machine learning algorithm that uses a labeled data set to classify test data points. Given a test data point, the algorithm finds the K closest labeled data points and classifies the test point with the most used label among its K neighbors. The number of neighbors K is an essential user-defined parameter: for each application, there's an optimal K value that ensures the best stability and accuracy of the classification. A low K leads to overfitting, and a large K produces underfitting.

A KNN application was implemented using a PT-Float C library, using the following data types: the IEEE-754 double-precision format (Float64), the IEEE-754 single-precision format (Float32), and the PT-Float<32,4> format. With 52 fraction bits, the Float64 format is used as a reference to compare the Float32 with the PT-Float<32,4>. Two experiments were devised to compare these formats for accuracy and dynamic range. The datasets were randomly generated using the parameters given in Table X.

TABLE X
KNN parameters.

Parameter	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

1) Accuracy Experiment: In the first experiment, the value of the dataset point coordinates ranges between 0.99999 and 1, and those of the test points range between 0.9 and 1. The idea is to compare the accuracy of the PT-Float<32,4> with that of the Float32 when the data spans a small interval of near-unit values; the results are summarized in Figure 7.

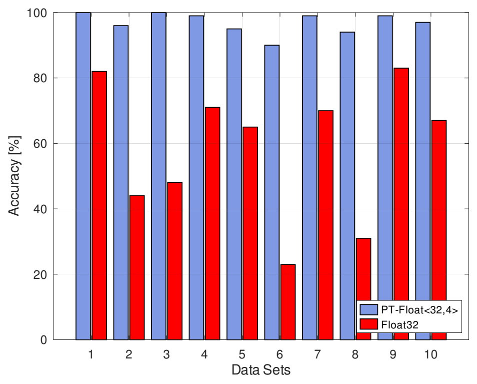

Fig. 7. Accuracy of classification for PT-Float<32,4> and 32-bit Floats.

The results show that the PT-Float<32,4> consistently has a higher percentage of correct classifications than the Float32 numbers. The accuracy of the PT-Float is always above 90%, whereas the accuracy of the Float32 is significantly poorer. The results are unsurprising since the PT-Float<32,4> numbers can have a maximum of 28 fraction bits for numbers near 1, while the Float32 numbers only have 23. Hence, the Float32 lacks enough resolution to give accurate classifications to some test points. Thus, the PT-Float<32,4> outperforms the Float32 in KNN problems where the data spans a small range. In this application, the 32-bit PT-Float<32,4> can be a compelling replacement for the 64-bit IEEE-754 format.

2) Dynamic Range Experiment: In the second experiment, the dataset points and test points range between 0 and 10^{22}. The idea is to compare the dynamic range of the PT-Float<32,4> with that of the Float32. All data points fall within the range of these two formats. However, for the Float32 format, the square distance between the labeled and unlabeled points often falls outside their range. For this format, most computed distances overflowed, resulting in an inability to perform. Moreover, the overflow operations cause the computed distance to be assigned to $+\infty$, with consequent mathematical incongruities if these exceptions are not trapped.

The PT-Float<32,4> format performed correctly in all the benchmarks because its dynamic range is greater than the Float64, using half of the computer memory for the format. The PT-Float<32,4> can satisfy the application requirements while guaranteeing computational efficiency and low power.

VI. Conclusions

This paper proposes PT-Float, a new floating-point number system with tapered precision. The new system has been implemented in software and hardware, tested, and compared to the IEEE-754 and Posit systems.

Like the Posit, the new system tackles the IEEE-754 floats' fixed number of exponent and fraction bits. This inflexibility creates shortcomings in dynamic range, accuracy, and precision. This work and previous works use dynamic exponent and fraction sizes to solve this problem and increase precision and dynamic range when needed.

This paper revisits the idea of using an extra field to represent the size of the exponent. This idea has the problem of redundant representations, as the same number can be represented with exponents of different widths. This paper proposes that the exponent has an implicit leading bit forced to the opposite value of the explicit MSB.

The new implicit bit enforces normalization, provides sign information and solves the redundant representations problem. Unlike other formats that use a fixed-value leading implicit bit to normalize the significand, using a dynamically varying leading implicit bit for the exponent is a new and powerful tool.

The PT-Float number system is a tapered precision system that is more precise, accurate, and has a more dynamic range than the Posit system. The exponent leading bit technique proved significantly more efficient than the Posit's regime bits, which waste precious bits in an after-all unary exponent representation.

A theoretical analysis of the new PT-Float format is presented, which shows that sensibly configured PT-Floats can replace IEEE-754 floats as they can flexibly be more precise near the unit or reach a farther range than IEEE-754 floats of the same size.

Our studies generally indicate that a PT-Float FPU of half the data size can replace a full-size IEEE-754 FPU if precision or dynamic range can be traded off. However, this paper does not claim that variable-relative-error formats can replace fixed-relative-error formats like the IEEE-754.

Configurable hardware implementations of PT-Float and IEEE-754 have been developed in Verilog and implemented for the UMC130nm ASIC technology. The FPUs include the four basic two-operand operations: addition, subtraction, division, and multiplication. For the same data width, the results show that the PT-Float FPU is roughly 50% larger and consumes twice the energy. However, since an IEEE-754 FPU can often be replaced with a half-data-size PT-Float FPU, replacing a 32-bit IEEE-754 FPU with a 16-bit-PT-Float FPU would save approximately 36% and 5%, in area and power, respectively; replacing a 64-bit IEEE-754 FPU with a 32-bit PT-Float FPU would yield area and power savings of roughly 156% and 99%, respectively.

The KNN application was used as a case study to compare the PT-Float<32,4> performance against the IEEE Float32, using the IEEE Float64 results as a golden reference. The results showed that the PT-Float outperforms the 32-bit floats and produces more similar results to the 64-bit floats.

References

[1] I. of Electrical, E. E. C. S. S. Committee, and D. Stevenson, *IEEE standard for binary floating-point arithmetic*. IEEE, 1985.

[2] M. S. Committee *et al.*, "IEEE standard for floating-point arithmetic. IEEE Std 754-2019 (Revision of IEEE 754-2008)," *IEEE Computer Society, New York, NY, USA*, 2019.

[3] J. Gustafson, *The End of Error: Unum Computing*. CRC Press, 02 2015.

[4] W. Kahan and J. Darcy, "How Java's floating-point hurts everyone everywhere," *... 1998 Workshop on Java ...*, pp. 1–81, 1998. [Online]. Available: https://people.eecs.berkeley.edu/∼wkahan/JAVAhurt.pdf

[5] E. Ternovoy, M. G. Popov, D. V. Kaleev, Y. V. Savchenko, and A. L. Pereverzev, "Comparative Analysis of Floating-point Accuracy of IEEE 754 and Posit Standards," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 2020, pp. 1883–186.

[6] R. Morris, "Tapered Floating Point: A New Floating-Point Representation," *IEEE Transactions on Computers*, vol. C-20, no. 12, pp. 1578–1579, 1971.

[7] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating Scientific Computations with Mixed Precision Algorithms," *Computer Physics Communications*, vol. 180, no. 12, p. 2526–2533, Dec 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2008.11.005

[8] P. Lindstrom, S. Lloyd, and J. Hittinger, "Universal Coding of the Reals: Alternatives to IEEE Floating Point," in *Proceedings of the Conference for Next Generation Arithmetic*. Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190339.3190344

[9] J. L. Gustafson, "A Radical Approach to Computation with Real Numbers," *Supercomputing Frontiers and Innovations*, vol. 3, no. 2, 2016. [Online]. Available: https://superfri.org/superfri/article/view/94

[10] W. Tichy, "Unums 2.0: An Interview with John L. Gustafson," *Ubiquity*, vol. 2016, pp. 1–16, 10 2016.

[11] J. Gustafson and I. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, pp. 71–86, 01 2017.

[12] J. L. Gustafson, "Posit arithmetic, [Online]," 2017. [Online]. Available: https://posithub.org/docs/Posits4.pdf

[13] L. Forget, Y. Uguen, and F. de Dinechin, "Comparing posit and IEEE-754 hardware cost," Apr. 2021, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03195756

[14] M. Posner, "How many asic gates does it take to fill an fpga?, [Online]," 2015, accessed: 2023-05-08. [Online]. Available: https://blogs.synopsys.com/breakingthethreelaws/2015/02/how-many-asic-gates-does-it-take-to-fill-an-fpga