Unum Type-IV: A Floating-Point Unit with Dynamically Varying Exponent and Mantissa Sizes

Micaela Moraes Serôdio
Electrical and Computer Engineering Department
Instituto Superior Técnico
Lisbon, Portugal
micaela.serodio@tecnico.ulisboa.pt

***Abstract*—This paper presents a new floating-point format, called Unum-IV, that replaces the unary numeral system used in the Regime bits of the Unum Type-III format, aka *Posit* numbers, with a regular binary system. The Unum formats (types I, II and III) introduced the concept of tapered precision: numbers having an absolute value close to 1 are given more significand bits and fewer exponent bits. This way, it is possible to extraordinarily increase the dynamic range or accuracy, depending on the application. The new format recovers and extends Unum Type-I ideas but, like Unum-III, it is meant to be hardware fiendly, is hence named Unum Type-IV.**

The main ideas of the new Unum-IV format are to use a dynamically variably-sized significand and exponent with a 1's complement representation of the exponent, and a 2's complement representation of the significand. The Unum-IV format uses one hidden bit both in the exponent and significand representations to prevent redundancy. However, unlike other formats, including Posits, that use a fixed hidden 1 bit only in the significand, Unum-IV uses dynamic but simply computed hidden bits for both the significand and the exponent. This format is aimed at small memory footprint and low energy consumption applications, such as those that can be found in the Internet of Things (IoT). In fact, in most practical cases, Unum-IV can replace IEEE 754 double precison numbers, which means half the memory footprint and 30% less hardware resources.

***Index Terms*—Unum Computing, Floating-Point Unit, Computer Arithmetic, High Precision Arithmetic.**

I. Introduction

We live in an era where the trade-off between performance, cost, resources, and power consumption has a huge impact in the computer science area. The increase of the complexity of the algorithms and the demand on Big Data (BD) analyses, the need for High-Performance Computing (HPC) and the never-ending need for energy-efficient computing in fields of Machine Learning (ML), Artificial Intelligence (AI) and the Internet of Things (IoT) are creating a new computing paradigm. The majority of the numerical computation algorithms require the capacity of a system to perform arithmetic operations and manipulation of real numbers. In digital electronics, the most extensively adopted approximation of the real numbers is given by the floating-point formats.

Throughout our history, some disasters happened due to floating-point issues, usually because of rounding errors [1], such as the Patriot Missile Failure on February 25, 1991, where an Americal Patriot Missile battery in Dharan, Suadi Arabia, failed to intercept an incoming Iraqi Scud missile. That happened due to a rounding error in the time calculation, which costed the life of 38 people.

Another disaster was the explosion of the Ariana-5 rocket launched by the European Space Agency in 1996. The rocket exploded just forty seconds after the lift-off due to a software error in the inertial reference system. One more example of a floating-point error disaster was the Sleipner Oil Platform that has collapsed to the ocean floor due to a floating error in the structural analysis, costing a nearly 1 billion dollar loss. There were other famous disasters in different areas caused by issues of the floating-point format design, like the overflow, underflow and rounding error problems [2].

During the 1960s and the 1970s, there was no ubiquity in the floating-point format, so each computer manufacture developed its own 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 it is, nowadays, the most common representation of real numbers on computers, including Intel-based PC's, Macs and most Unix platforms. This format has been reviewed and replaced two times, in 2008 [3] and 2019 [4] but there were only minor changes between them to maintain compatibility in existing implementations.

However, some drawbacks have been identified in the IEEE 754 Standard [5], [6], [7], such as:

- Overflow and Underflow: overflowing to $-\infty$ or $+\infty$ and underflowing to 0, increases the relative error by an infinite factor and leads to sign information loss, respectively.
- No Gradual Overflow and Fixed-Accuracy: accuracy is flat across a vast range, then "falls off a cliff".
- Wasted bit-patterns: there are too many NaN representations and two bit-patterns to represent 0, the "negative zero" (0^-) and the "positive zero" (0^+).
- There is no guarantee of identical results across systems.
- Exponents usually take too many bits.
- Equality verification test between two floating-point is complex due to the presence of redundant representations.

Throughout the years, different number systems and techniques have been proposed to overcome these challenges. In [8], Morris suggested a "tapered" system to solve the fixed accuracy problem of the floating-points [9].

In 2013, John L. Gustafson introduced a new binary and arithmetic way of representing real numbers, a number system called "Universal Numbers" [5], [10], [11]. They have so far three different types of representation. Type-I is a superset of the IEEE 754 Standard floating-point format, and it was introduced in [5]. 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 a "real" number (u=0) is an exact float or lies in the open interval between two consecutive exact (u=1). Type -II [12] enables a clean mathematical design based on the *projective reals* and relies on lookup tables. It is a direct map of signed integers to the projective real number line. The last version introduced in 2017 is Type-III or posits [13], [14], which is a hardware-friendly version with all the advantages from the previous types. Nowadays, even though many shortcomings have been pointed out upon the IEEE 754 Standard, it is still the most commonly implemented in microcontrollers and general-purpose microprocessors to perform floating-point arithmetic. However, the trade-off between the dynamic range and the precision of the floating-point formats can be explored to create a new number system that avoids those shortcomings.

II. Unum Type-IV

The Unum Type-IV format explores and combines ideas from both Unum Type-I and the Unum Type-III formats, specifically, the use of an exponent size field, the use of variable-sized significand and exponent, avoids redundant representations by introducing a hidden bit into the exponent and eliminates the use of NaN, $-\infty$ and $+\infty$ representations as well as the use of two different patterns to represent 0.

A. Unum Type-IV Generic Format

The Unum-IV generic format has three fields: the exponent, the fraction and the exponent size. In order to avoid redundant representations, a hidden bit is added to the exponent as well as to the significand. These hidden bits optimise the use of all bit patterns as representable real numbers. In both cases, exponent and significand, the hidden bit is the Most Significant Bit (MSB). The significand and exponent in this format are both signed, with the hidden bits providing the sign information. The exponent is in 1's complement format, so there is no need for a bias as in the IEEE 754 and the significand is in 2's complement format, which dispenses with the sign bit field used in other formats such as the IEEE-754 format.

The Unum-IV configuration and enconding is determined by the format size (DATA_W) and by the number of bits necessary to represent the exponent size (EXP_SZ_W). Therefore, the notation

Unum-IV<DATA_W,EXP_SZ_W>

is used here to denote a DATA_W-bit Unum-IV with EXP_SZ_W bits for the exponent size field. The generic Unum-IV floating-point format is encoded as shown in Figure 1, and the next subsections explain each of the fields.

Using this encoding, and processing the fields as explained below, one can obtain the significand s and the

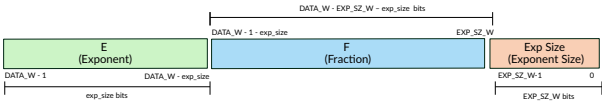

Fig. 1. Generic Unum-IV representation format.

exponent e of the real number r represented in the Unum-IV<DATA_W,EXP_SZ_W> format and given by

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

1) Exponent Size (Exp Size): The Exponent Size (Exp Size) is a small unsigned integer, which has a width defined by the parameter EXP_SZ_W, and can assume a value in the range from 0 to $2^{\text{EXP\_SZ\_W}} - 1$. The Exp Size field allows for tapered accuracy since it represents the number of explicit bits necessary to represent the exponent of the number in the 1's complement format. It establishes a higher accuracy for numbers that are close to 1, and a lower accuracy for very large or very small numbers.

Denoting the Exp Size bits by ExpSize_i, the explicit exponent size in bits $ExpSz$ is given by

$$ExpSz = \sum_{i=0}^{\text{EXP\_SZ\_W}-1} \text{ExpSize}_i 2^i \tag{2}$$

2) Exponent (E): The Exponent (E) field holds the explicit exponent, as indicated by its name. Unlike the IEEE 754, the exponent is not biased and is represented as a signed integer in the 1's complement format with a hidden (implicit) most significant bit. The exponent hidden bit is always the negation of the most significant bit of the E field. Moreover, the 1's complement representation breaks for the widest exponent and most negative 1's complement exponent: $(1)00...00$, where the hidden bit is shown in brackets followed by EXP_SZ_W zeros. This combination is reserved for subnormal representations to allow for a graceful underflow. Subnormal representations are also used in the the IEEE 754 standard, though with a different implementation.

The E field width is ExpSz, therefore it is variably-sized. If $ExpSz = 0$, the E field is not present. Following the logic of the 1's complement signed integers, if the hidden bit is zero the exponent is positive and negative otherwise.

When E is filled with zeros and Exp Size is filled with ones, indicating the widest possible exponent, the subnormal representation applies, as explained above. In this mode, the exponent is valued $-2^{ExpSz} + 2$, which is one unit more than the normal 1's complement valuation of $-2^{ExpSz} + 1$. Hence, the exponent's magnitude falls in the following range of values: $[-2^{ExpSz} + 2 : 2^{ExpSz} - 1]$.

Without the subnormal representation, there would be a noticeable gap between 0 and smallest negative and positive numbers. By filling this gap around 0, the logarithmic distance between the numbers when approaching zero increases but not as abruptly as with a simple flush to zero approach. This allows computation results to lose precision slowly when very small.

Hence, in the new Unum-IV format, the exponent value is given by

$$e = \begin{cases} 0, & \text{if } ExpSz = 0 \\ -2^{ExpSz} + 2, & \text{if } ExpSz = 2^{\text{EXP\_SZ\_W}} - 1 \wedge E = 0 \\ \overline{E_{ExpSz-1}}(-2^{ExpSz} + 1) + \sum_{i=0}^{ExpSz-1} E_i 2^i, & \text{otherwise} \end{cases} \quad (3)$$

The exponent hidden bit E_{HB} is introduced to avoid redundant representations. It does not need to be stored in memory, and is given by

$$E_{HB} = \begin{cases} absent, & \text{if } ExpSz = 0 \\ \overline{E_{ExpSz-1}}, & \text{otherwise} \end{cases} \quad (4)$$

3) Fraction (F): The Fraction (F) field represents the significant digits on the right side of the significand binary point. Like in the IEEE 754 format, in the Unum-IV format, the significand has an implicit leading bit. However, unike in the IEEE 754 format, this hidden bit is not always 1.

In the Unum-IV case, the significand is represented by a signed 2's complement number, and the hidden bit is always the complement of the fraction's MSB, except for the subnormal representation (see previous section), where the hidden bit equals the fraction's MSB. Thus, if the hidden bit is 0, the represented number is positive or zero; if the hidden bit is one, the number is negative. The F field bit-width $FracSize$ depends of course on the format width and exponent size and is given by

$$FracSize = \text{DATA\_W} - ExpSz - \text{EXP\_SZ\_W} \quad (5)$$

Hence, in the new Unum-IV format, the significand is given by

$$s = \begin{cases} -F_{-1} + \sum_{i=1}^{FracSize} F_{-i} 2^{-i}, \\ \qquad \text{if } ExpSz = 2^{\text{EXP\_SZ\_W}} - 1 \wedge E = 0 \\ -\overline{F_{-1}} + \sum_{i=1}^{FracSize} F_{-i} 2^{-i}, \quad \text{otherwise} \end{cases} \quad (6)$$

The significand's hidden bit F_{HB} in the new Unum-IV format dispenses with the sign bit as in the IEEE-754 format and is given by

$$F_{HB} = \begin{cases} F_{MSB}, & \text{if } ExpSz = 2^{\text{EXP\_SZ\_W}} - 1 \wedge E = 0 \\ \overline{F_{MSB}}, & \text{otherwise} \end{cases} \quad (7)$$

4) Tapered Precision: The tapered precision is easily verified after having explained the format's fields in the previous sections. The sum of the exponent and significand sizes is constant and given by

$$ExpSz + FracSize = DATA\_W - EXP\_SZ\_W \quad (8)$$

On the one hand, the very large or very small numbers require more exponent bits to represent, get fewer fraction bits and therefore less precision. On the other hand, the numbers closer to magnitude 1 require less exponent bits, get more fraction bits and therefore more precision. The numbers whose exponent equals 0 do not require any exponent bits and get all the available DATA_W - EXP_SZ_W bits for precision.

B. Features

1) Special Cases: The Unum Type-4 format does not have any "special" numbers such as "NaN", $-\infty$, $+\infty$, ∞, 0^+ and 0^-. Instead of having a range of representations locked for those "special" cases (losing space for real representable numbers), every combination of bits is used to represent a real number. The calculation is interrupted using flags when an overflow, underflow, divide by 0 or other types of exceptions occur. That simplifies the hardware and also extends the dynamic range.

2) Exceptions: There are three different types of exceptions in the Unum-IV number system: the "divide by zero" exception, the "overflow" exception and the "underflow" exception. The first exception occurs when a division operation between two Unum-IV numbers is performed, and the divisor is zero. The second happens when a two-argument operation like addition, subtraction, division or multiplication results in a number with a larger exponent than the exponents allowed by the Unum-IV parameters used or when the conversion between a decimal representation and this format falls outside of the Unum-IV dynamic range higher bound. The last exception occurs when the result has a smaller exponent than the smallest positive representable value or when the conversion falls outside the dynamic range lower bound.

All of those exceptions are handled using flags that control the propagation of the results. If any of the exceptions happen, the calculation or conversion is interrupted, and the error is reported by the flag, simplifying the hardware used.

3) Rounding: There are two rounding modes: "the round to the nearest, ties even" and the truncation mode. The rounding or truncation is necessary since the operations with floating-point can result in a non-representable number relative to the precision, so the result needs to be rounded to the nearest representable Unum-IV value or truncated to the last representable digit.

As in IEEE 754, the "round to the nearest, ties even" mode uses three extra bits of less significance than the significand bits, a *guard* bit, a *round* bit and a *sticky* bit. The most significant bit is the *guard* bit, and the least significant bit is the *sticky* bit.

In this mode, if the exact result can not be represented by an Unum-IV number, the result is rounded to the nearer of two possible values. If there is a tie between the two possibilities, then the even alternative is chosen.

C. Dynamic Range and Precision

The length of the fraction determines the precision of the representable floating-point number. The ration between

the smallest and the largest positive number determines the dynamic range of the number system in evaluation. Since the Unum-IV number system has a variable-sized significand and exponent, the Unum-IV dynamic range and precision depend on the value of the parameter EXP_SZ_W. The larger this parameter is, the greater the exponent contribution in the format, and therefore the dynamic range will increase.
The relation between the exponent size field and the dynamic range is: increasing the number of bits available to represent the exponent increases the dynamic range, and vice-versa. However, increasing the EXP_SZ_W means decreasing the maximum number of bits available to the fraction field since they are the remaining bits of the format. Therefore, if the maximum number of fraction bits decreases, so does the precision.
The principal advantage of the Unum-IV format is the ability to choose the parameters, DATA_W and EXP_SZ_W, to adjust the trade-off between the dynamic range and precision to meet the performance needs of an application. If the application needs more accurate answers with numbers of small magnitude, it will need more fraction bits and fewer exponent bits. On the other hand, if the application works within a range of extremely small or large values, it will need more exponent bits with less fraction bits available, increasing the dynamic range. This trade-off can make a huge difference in applications that process large amounts of data.

In the Unum-IV generic format, the smallest and largest positive representable number, *minpos* and *maxpos* respectively, are given by

$$\begin{cases} \textit{minpos} = 2^{\left[-2^{\left[2^{\text{EXP\_SZ\_W}}-1\right]}+2\right]} \\ \qquad\qquad \times 2^{-\left[\text{DATA\_W}-\left(2^{\text{EXP\_SZ\_W}}-1\right)-\text{EXP\_SZ\_W}\right]} \\ \textit{maxpos} = 2^{\left[2^{\left[2^{\text{EXP\_SZ\_W}}-1\right]}-1\right]} \\ \qquad\qquad \times \left(1-2^{-\left[\text{DATA\_W}-\left(2^{\text{EXP\_SZ\_W}}-1\right)-\text{EXP\_SZ\_W}\right]}\right) \end{cases} \tag{9}$$

Hence, the dynamic range, measured in decades, is given by the following formula

$$\textit{Dynamic Range} = log_{10}\left(\frac{maxpos}{minpos}\right) \tag{10}$$

Table I, resume and exemplifies the relation between the generic parameters of the Unum Type-IV number system and the dynamic ranges.

In terms of precision, the number of precision bits (equal to the number of fraction bits) for each representable Unum-IV value *p_bits* is given by

$$\textit{p\_bits} = \text{DATA\_W} - ExpSz - \text{EXP\_SZ\_W} \tag{11}$$

where

- $ExpSz$: number of exponent bits, excluding the implicit leading bit.

TABLE I
UNUM TYPE IV DYNAMIC RANGES.

Unum-IV$< \mathbf{n}, \mathbf{k} >$	*minpos* ($\approx$)	*maxpos* ($\approx$)	Dynamic Range
n=8, k=2	1.95×10^{-3}	1.12×10^{2}	4.76×10^{0}
n=16, k=2	7.63×10^{-6}	1.28×10^{2}	7.22×10^{0}
n=16, k=3	1.84×10^{-40}	1.67×10^{38}	7.80×10^{1}
n=32, k=3	2.80×10^{-45}	1.70×10^{38}	8.28×10^{1}
n=32, k=4	3.45×10^{-9868}	7.08×10^{9863}	1.97×10^{4}
n=64, k=4	8.03×10^{-9878}	7.08×10^{9863}	1.97×10^{4}
n=128, k=4	3.06×10^{-9883}	7.08×10^{9863}	1.97×10^{4}

Therefore, the precision is variable and depends on the Unum-IV number size and exponent. The tapered precision of this format implies that numbers having an absolute value close to 1 are given more significand bits and fewer exponent bits, and the other numbers trade-off significand bits with exponent bits.The precision bits can go from DATA_W-EXP_SZ_W to DATA_W-$2^{\text{EXP\_SZ\_W}}$+1-EXP_SZ_W bits.

Table II shows some examples of the relation between the Unum-IV parameters and the number of precision bits.

TABLE II
UNUM TYPE IV PRECISION.

Unum-IV $< n, k >$	*p_bits*
n=8, k=2	from 3 to 6
n=16, k=2	from 11 to 14
n=16, k=3	from 6 to 13
n=32, k=3	from 22 to 29
n=32, k=4	from 13 to 28
n=64, k=4	from 55 to 60
n=128, k=4	from 124 to 109

III. HARDWARE IMPLEMENTATION

In this paper, the design of a parameterized and pipelined Floating-Point Arithmetic Unit based on the new Unum Type-IV format was explored and implemented in Verilog. The hardware implementation supports four types of operations: addition, subtraction, division and multiplication, and it also supports two rounding modes: the truncation (ROUNDING=0) and the "round to the nearest, ties even" (ROUNDING=1).

The top-level module has six input and five output signals. These interface inputs and outputs size, direction and description are explained in Table III.

The hardware implementation of Unum-IV arithmetic is based on three hardware stages. The first stage is the unpacking module, where the Unum-IV exponent and significand are extracted from each Unum-IV operand. The intermediate stage is the processing module, where the four basic supported operations are performed. The remaining and final stage is the packing module, where the computed exponent and significand are packed in the Unum-IV format. The Unum-IV FPU has a hierarchical architecture, comprising three main stages. The hierarchy flow is controlled by the Control Logic (CL), preventing the propagation of errors and allowing exceptions handling.

TABLE III
FPU INTERFACE.

Signal	Size	Direction	Description
clk	1	Input	System Clock
rst	1	Input	Asynchronous active high reset
start	1	Input	Strobe to start calculation
op	2	Input	Operation selection
a	DATA_W	Input	Operand A
b	DATA_W	Input	Operand B
o	DATA_W	Output	Calculation Result
done	1	Output	Strobe to signal the end of calculation
div_by_zero	1	Output	Strobe to signal an invalid operation ($x/0$)
underflow	1	Output	Strobe to signal an underflow result
overflow	1	Output	Strobe to signal an overflow result

A. Functional Units

The proposed Unum-IV FPU has five principal parameterizable Functional Units (FUs) that make usage of five auxiliary units: a barrel shifter, an adder and subtractor, an exponent difference module, leading zeros/one's detector, a shift and subtract serial divider and a multiplier.

1) Unpack Unit: The Unpack Unit is responsible for taking an Unum-IV input and extracting the exponent, fraction and exponent size fields. The flow of this hardware floor is performed as shown in the block diagram presented in Figure 2.

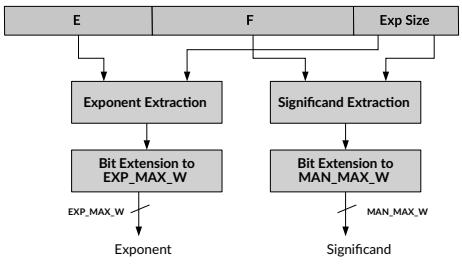

Fig. 2. Unum-IV Unpack Stage Block Diagram. Pipeline Stages: 3.

B. Processing Units

Two operation bits are used for this FPU, meaning that up to four operations are available. The processing stage can perfom additions, subtractions, divisions and multiplications. The addition and subtraction is performed by the same unit, while the division and multiplication are performed by different units.

1) Addition/Subtraction Unit: The Addition/Subtraction Unit is responsible for perfoming an addition (op=0) or subtraction (op=1) between two unpacked Unum-IV inputs. The blokc diagram of the Unum-IV proposed addition and subtraction unit is shown in Figure 3.

2) Multiplication Unit: The Multiplication Unit is responsible for perfoming a multiplication between two unpacked Unum-IV inputs. The block diagram of the Unum-IV proposed multiplication unit is shown in Figure 4.

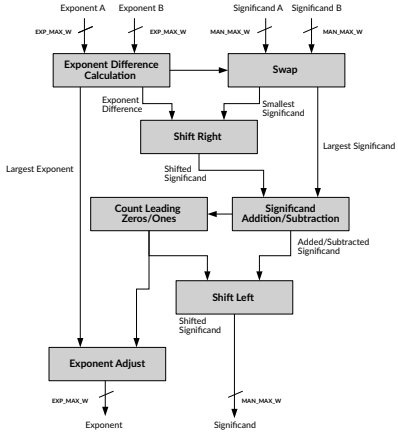

Fig. 3. Unum-IV Addition/Subtraction Unit. Pipeline Stages: 6.

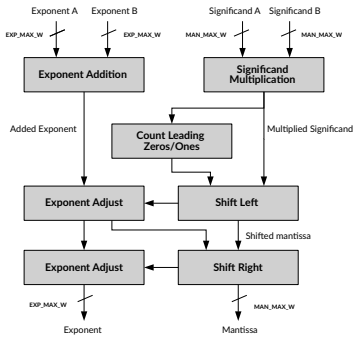

Fig. 4. Unum-IV Multiplication Unit. Pipeline Stages: 4.

3) Division Unit: The Division Unit is responsible for perfoming a division between two unpacked Unum-IV inputs. The block diagram of the Unum-IV proposed division unit is shown in Figure 4.

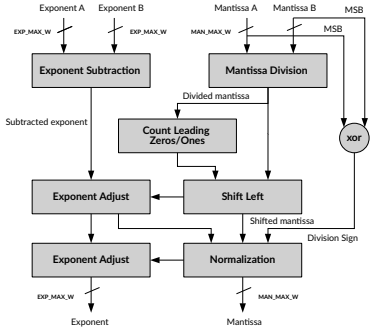

Fig. 5. Unum-IV Division Unit. Pipeline Stages: 5+MAN_MAX_W+1 EXTRA (0 or 3 extra bits).

C. Packing Unit

Two different modules can be generated to pack the FPU operation result into the Unum-IV format depending on the

rounding mode chosen using the ROUNDING parameter. The Pack Unit is responsible for packing the propagated result from the processing stage into a Unum Type-IV bit string and for outputing the interruption flags. The block diagram of the Unum-IV proposed packing unit is shown in Figure 6.

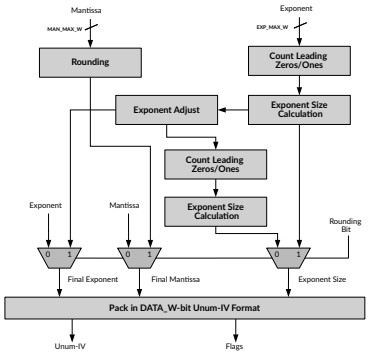

Fig. 6. Unum-IV Pack Unit. Pipeline Stages: 3 + ROUNDING (0 or 1).

IV. Evaluating and Comparing Unum Type-IV to Other Formats

A. Qualitative Comparison

Table IV presents a qualitative comparison between the most relevant features separating the Unum Type-IV, Unum Type-III and IEEE 754 number systems. The Unum-IV and Posit formats have some distinctive features which can not be found in the IEEE 754 number system.

TABLE IV
Qualitative Comparison between IEEE 754 Standard and Unum Type-IV.

Features	IEEE 754 [n-bit]	Posits [n-bit]	Unum-IV [n-bit]
Portability/Reproducibility	No	Yes	Yes
Redundant Representations	Many	None	None
NaNs Representations	$2^{n-e}-2$ w/ $e=\{5,8,11,15\}$	None	None
Infinity Representations	2 ($-\infty/+\infty$)	1 (∞)	None
Zero Representations	2 ($0^-;0^+$)	1 (0)	1 (0)
Real Number Representations	Exceptions: NaNs,$+\infty$,$-\infty$	Exceptions: ∞	2^n
Overflow	"Falls of a Cliff"	Gradual (tapered accuracy)	Never. Exceptions: $\frac{1}{0}=\infty$
Underflow	Gradual	Gradual	Gradual
Exponent	Fixed-Size; Biased; Unsigned; 0 Implicit Leading Bits	Variable-Size; Unsigned 0 Implicit Leading Bits	Variable-Size; Signed (1's Complement); 1 Implicit Leading Bit
Significand	Fixed-Size; Unsigned; 1 Implicit Leading Bit	Variable-Size; Signed (2's Complement) 1 Implicit Leading Bit	Variable-Size; Signed (2's Complement); 1 Implicit Leading Bit
Precision Bits	Fixed	Variable	Variable

B. Comparing Unum Type-IV Dynamic Range with other Formats

The Unum Type-IV number system can have greater or matching dynamic ranges with the IEEE 754 Standard formats depending on the configuration adopted. The half-precision IEEE 754 has a dynamic range of about 12 decades, single-precision has about 83 decades, double-precision has nearly 652 decades, and the quad-precision format has about 9882 decades. These four formats are the formats defined in the standard, having 5, 8, 11 and 15 exponent bits, respectively.

The Unum Type-III Standard draft includes the Posit<16,1>, Posit<32,2>, Posit<64,3> and Posit <128,4> configuration. The Posit<16,1> configuration has a dynamic range of about 16 decades, the Posit<32,2> has about 73 decades, the Posit<64,3> has nearly 299 decades, and, finally, the 128-bit Posit <128,4> has, approximately, 1214 decades.

Figure 7 shows the dynamic range, in decades, for the different IEEE 754 and Posit Standard formats and some of the most significant configurations of the Unum-IV, namely, the Unum-IV<16,3> with about 78 decades, the Unum-IV<32,3> with nearly 83 decades, the Unum-IV<32,4> with, approximately, 19731 decades, the Unum-IV<64,4> and the Unum-IV<128,4> with 19741 and 19746 decades, respectively.

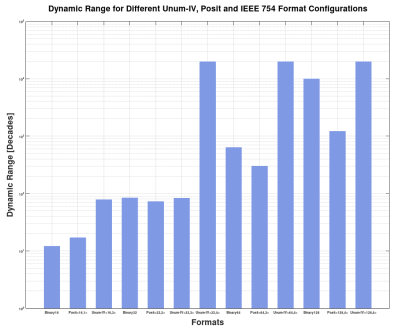

Fig. 7. Dynamic Range for Different Unum-IV and IEEE754 Format Configurations.

As can be seen in Figure 7, the Unum-IV$<16,3>$ configuration has almost the same dynamic range as the 32-bit IEEE 754 float, which is stored occupying twice of the computer memory. The Unum-IV$<32,4>$ has more than 237 times the dynamic range compared with the IEEE 754 bit string with the same width (single-precision), has more than 30 times the dynamic range of the 64-bit IEEE 754 format, which uses the double of the format memory, and it also has a slightly better dynamic range that the quad-precision floats, with 128 bits of memory. Figure 7 also shows that Unum Type-IV not only has a wider dynamic range for the same bit width as Posits but also with using half of it.

C. Comparing Unum Type-IV Precision with other Formats

In Figure 8 the number of significand bits is shown as a function of the exponent range of each format. The highlighted areas are the "golden zone" where the Unum-IV format in question has at least the same resolution as floats.

Since the IEEE 754 uses fixed-size exponent and fraction, the number of fraction or precision bits is constant. The Unum-IV format has variable resolution, having more precision bits for exponents near 0 and fewer precision bits for numbers with a higher magnitude. Therefore, the "precision" plot of the Unum-IV has a sine form as expected.

Figure 8 shows that for the same size, both Unum-IV<32,3> and Unum-IV<32,4> have exponent ranges where they have more fraction bits than the binary32 format.

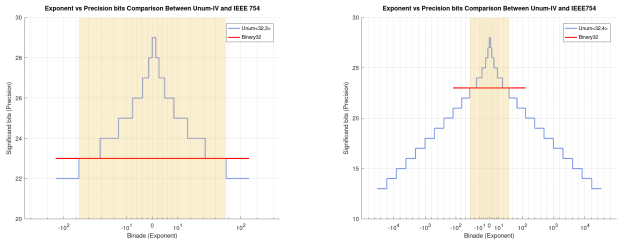

Fig. 8. Exponent vs Significand Bits Comparison Between Unum-IV<32,3> & 32-bit IEEE 754 Format. (LEFT) Exponent vs Significand Bits Comparison Between Unum-IV<32,4> & 32-bit IEEE 754 Format. (RIGHT)

The "golden area" occupied by Unum-IV<32,3> is wider than Unum-IV<32,4> because of the smaller dynamic range, having space for more precision bits. Near 0, the Unum-IV<32,3> gives a maximum of 29 precision bits, outperforming the 32-bit floats by 6 bits, while Unum-IV<32,4> has just more 5 bits. On the edges where the floats overflow and underflow, outside the golden area, Unum-IV<32,3> has 22 bits, having a smaller resolution by one bit and Unum-IV<32,4> has 21 bits of precision.

Therefore, Unum-IV<32,3> has a similar dynamic range to the 32-bit floats and covers a larger area where numbers have at least the same resolution than floats compared with Unum-IV<32,4>. In terms of accuracy, it is a better option to replace 32-bit floats if the range is enough. The advantage of Unum-IV<32,4> is that besides having, on average, better resolution in the binary32 range than floats, it also has a massive difference in the dynamic range, covering the 64-bit and 128-bit floats ranges.

D. Unum-IV<8,2> vs. Quarter-Precision IEEE-Style floats vs. Posit<8,1>

In Figure 9, a quarter-precision IEEE-style float format (not standardized) is tested against Unum-IV<8,2> to compare both formats. These two low precision configurations are selected because they present comparable dynamic ranges and are practical to analyse the whole range since both sets have only 256 elements. The 8-bit IEEE-style format used in this comparison follows the IEEE 754 Standard rules even though this format does not make part of the standard formats. It has a sign bit, a 4-bit exponent and a 3-bit fraction field. It has a total of 14-bit patterns that represent NaN values and a dynamic range of about five decades, where the smallest positive value is $\frac{1}{512}$, and the largest is 240.

The Unum-IV configuration chosen, Unum-IV<8,2>, implies that the format has a total of 8 bits, the exponent ranges between 0 and 6 explicit bits, and the fraction is set between 3 and 6 bits, depending on the exponent size. Therefore, Unum-IV<8,2> has a dynamic range of about 4.8 decades, where the smallest positive value is also $\frac{1}{512}$ and the largest is 112.

Posit<8,1> has a total of 8 (n) bits, a maximum of 1 exponent (es) and of 4 explicit significand bits. Therefore, Posit<8,1> has a dynamic range of about 7.23 decades, where the smallest positive value is also $\frac{1}{4096}$ and the largest is 4096.

Figure 9 shows the application of this metric of study to analyse the accuracy between the positive range of the 8-bit IEEE-style float number system, Posit<8,1> and Unum-IV<8,2>. The xx axis represents the index i of a set of x_i representable values of each format, both ordered from the smallest (minpos) to the largest positive (maxpos) number.

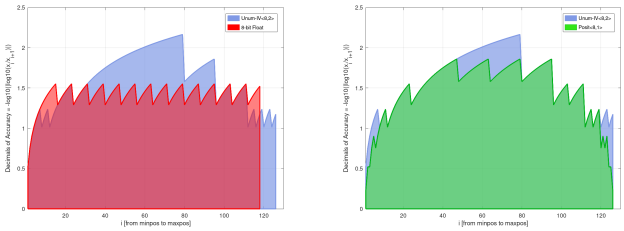

Fig. 9. Decimals of Accuracy Comparison between Unum-IV<8,2> & 8-bit Floats. (LEFT) Decimals of Accuracy Comparison between Unum-IV<8,2> & Posit<8,1> (RIGHT).

These graphs reveal that Unum-IV<8,2> is more accurate on average than the 8-bit floats. The results show that Unum-IV<8,2> has a minimum of 0.52, a maximum of 2.17 and an average of 1.58 decimals of accuracy. The 8-bit float format has a minimum of 0.52, a maximum of 1.55 and an average of 1.40 decimals. The decimal accuracy is at the highest in the centre of the graph for the Unum-IV, which is where the most common numbers used in the computations occur. As expected, due to the tapered precision of this format, the accuracy tends to decrease in both directions. At the centre of the graph are the numbers with smaller exponents, in terms of magnitude, requiring fewer exponent bits and using more fraction bits. As the numbers run from the centre, the exponent magnitude increases, demanding more exponent bits, which provides less accurate results. Figure 9 shows that the floats have tapered accuracy on the left as they use subnormals to obtain a gradual underflow. On the right side, the floats "fall of a cliff" to accommodate all the NaN values (14 in this case). On the other hand, the results show that the Unum-IV format has tapered accuracy on both sides, becoming closer to symmetrically tapered accuracy.

As can be seen in Figure 9, Unum-IV<8,2> is more accurate on average than the Posit<8,1> format due to the higher resolution of Unum-IV<8,2>. It is possible to take that Posit<8,1> has a minimum of 0.22, a maximum of 1.8605 and an average of 1.46 decimals of accuracy. However, Posit<8,1> has a greater dynamic range than the Unum-IV format, with a difference of nearly two decades.

Figure 10 shows the ULP variation of the 8-bit float, Posit<8,1> and Unum-IV<8,2> format from the smallest (minpos) to the largest positive representable value (maxpos). The ULP expresses the distance between two consecutive numbers and measures the resolution of a floating-point format.

Since the number of precision bits of the 8-bit floats is fixed and equal to 3, the ULP expression is given by $2^{-3} * 2^{e}$, depending exclusively on the exponent. Whereas Unum-

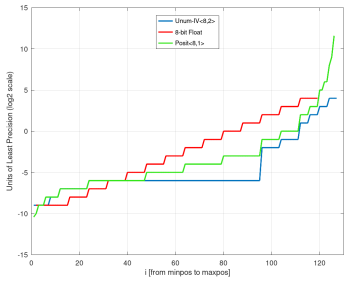

Fig. 10. ULP Comparison between Unum-IV<8,2> & 8-bit Floats.

IV<8,2> and Posit<8,1> ULP variation depends both on the exponent and precision. The Unum-IV and Posit ULP variation slows down as the numbers get closer to one because they receive more bits of precision.

From Figure 10, it is possible to infer that the Unum-IV<8,2> format has a higher resolution for a considerable portion of the format range because the ULP is smaller, meaning that they have smaller spacing between consecutive floating-point numbers.

V. Comparing Unum Type-IV Hardware Resources with other Formats

A. IEEE 754 and Unum Type-IV Comparison

Table V compares the ASIC implementation results of the Unum-IV FPU developed in this dissertation with the IEEE 754 FPU developed by a colleague in the IObundle company [15]. Both Unum-IV FPU and IEEE 754 FPU are parameterizable, so they are compared with different configurations in terms of the silicon area (Area), frequency (Clock Frequency) and power consumption (Power).

TABLE V
ASIC Implementation Results.

FPU	Data Width	ExpSize Width	Rounding Mode	Area [mm^2]	Power [mW]	Clock Frequency [MHz]
Unum-IV	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

In terms of silicon area, the results show that if the data width is the same between the FPUs, then the IEEE 754 FPUs are smaller than the Unum-IV FPUs. These area differences are explained by the fact that the Unum-IV FPUs exponents and significands are extended to their maximum size allowed by the configuration (EXP_MAX_W and MAN_MAX_W). Thus, all the modules like the barrel shifters and adders will be as large as the configuration parameters. The other reason is that the field extractions require more logic since the Unum-IV has variably-sized exponent and significand. Thus, the unpacking and packing require more hardware.

Despite that, the main focus of this dissertation was to develop an FPU with a configurable exponent and significand sizes. It is also relevant to compare the FPUs based on their number system configuration and dynamic range to verify if it was possible and advantageous to replace a format with another using less memory storage.

Let us take Unum-IV<16,3> into consideration. This format has the same exponent range as the single precision IEEE 754 (binary32), having a similar dynamic range. In this scenario, Unum-IV$< 16, 3 >$ FPU is 1,36x smaller than the binary32. Unum-IV<32,4> has a higher dynamic range compared with the double-precision IEEE 754 (binary64), and its FPU is 2,16x smaller.

B. Posits and Unum Type-IV Comparison

In the comparison between the Unum-IV and Posit hardware resources utilization, the paper [16] results are used to estimate the ASIC area of a Posit FPU. In this article, a pipelined FPU with an adder and multiplier was implemented in an FPGA using the Posit Standard configurations.

Table VI compares the silicon area of the Unum-IV FPU to the estimated silicon area of the Posits FPU for the following standard configurations: Posit<16,1>, Posit<32,2> and Posit<64,3>. For the Unum-IV, the ASIC results are obtained by directly implementing the Unum-IV<16,3>, Unum-IV<32,3> and Unum-IV<64,4> FPU configurations, using a UMC 130nm process. It could be shown that the silicon area estimated from the Unum-IV FPU FPGA results and its actual silicon area are similar. However, the actual results are shown since they are available.

TABLE VI
Silicon Area Comparisons between Different Unum-IV and Posits Configurations.

Format	Configuration	Maximum Precision [Bits]	Dynamic Range [Decades]	Estimated ASIC Area [mm^2]	ASIC Area [mm^2]
Posit	<16,1>	12	16.86	31.77	——
	<32,2>	27	72.25	97.79	——
	<64,3>	58	298.62	327.5	——
Unum-IV	<16,3>	13	77.96	——	34.42
	<32,3>	29	82.78	——	74.14
	<64,4>	60	19740.9	——	240.46

The results in Table VI show that the Unum-IV area is similar or smaller than the Posits area. Given that the accuracy and dynamic range of the Unum-IV has been shown superior to those of the Posits, one concludes that the Unum-IV format is a better replacement for the IEEE 754 format than the Posits.

For the same bit width, Unum-IV<16,3> has a similar area, a wider dynamic range by almost 30 decades and a greater maximum number of precision bits compared with Posit<16,1>. Unum-IV<32,3> uses 25% less area than Posit<32,2> while having a similar dynamic range and supporting more precision bits. Finally, Unum-IV<64,4> uses nearly 36% less area, has $66x$ more decades of dynamic range than the Posit<64,3> configuration, and also supports more precision bits.

VI. Proof of Concept: KNN Application

A KNN application was implemented using three different data types: the IEEE 754 double-precision format, which is

used as a reference, the IEEE 754 single-precision format, and the Unum-IV<32,4> format. The Unum-IV<32,4> is suitable to be used in ML application such as the KNN and tested against the 32-bit IEEE 754 floats because these type of applications use a high amount of floating-point operations and have a level of accuracy tolerance that others do not. It is expected that Unum-IV<32,4> exceed both 32-bit and 64-bit floats in terms of dynamic range. In terms of accuracy, the variable-precision format should also exceed the 32-bit floats for numbers near 1. Therefore, two different tests are made to compare the accuracy of the classification results for benchmarks that cover those features, using the 64-bit float results as a reference.

The algorithm finds the K closest labelled data to the data point to be classified using the distance as criteria, and then it predicts the test point classification by the majority class voted by its K neighbours. Thus, the class with the most votes is held as the predicted class of the unlabeled data point.

A. Experimental Results

The proof of concept application was run for two different sets of benchmarks. The number of neighbours is defined as 10 to provide a more accurate classification. To compare the performance between Unum-IV<32,4> and the IEEE single-precision floating-point format, we use the IEEE double-precision format results as a reference.

1) Experiment 1: In the first test, ten different benchmarks are randomly generated. The dataset points range between 0.99999 and 1, and the test points range between 0.9 and 1. This setup provides sparsely dispersed datasets. These benchmarks are suitable to compare and verify if Unum-IV and 32-bit floats have enough resolution to give accurate answers for numbers near 1.

Since the 32-bit floats have a 23-bit fraction, the effective resolution of the format lies between six and seven decimal fractional digits. Hence, it is expected that the 32-bit floats might not have enough resolution to give accurate classifications to the test points. On the other hand, for numbers near 1, Unum-IV<32,4> can have a maximum of 28 bits of fraction. Therefore, the Unum-IV format can lead to more accurate classifications than 32-bit floats because it supports 6 to 9 decimal fractional digits of resolution.

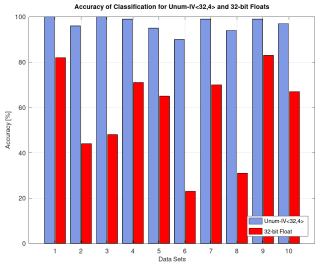

Fig. 11. Accuracy of Classification for Unum-IV<32,4> and 32-bit Floats.

The results of the first test show that Unum-IV<32,4> have a higher percentage of correct classifications than the 32-bit floats for all the ten benchmarks. The accuracy of Unum-IV is set between 90 to 100 per cent, while the 32-bit floats can only afford results between 23 and 83 per cent of accuracy compared with the 64-bit floats classifications. As expected, Unum-IV<32,4> outperforms the 32-bit floats for numbers with small magnitude.

2) Experiment 2: In the second test, another ten different benchmarks are randomly generated. However, the dataset points and test points generation use a wider range than the previous test, ranging between 0 and 10^{22}. The main focus of this test is to compare the Unum-IV<32,4> classifications with the results of the 64-bit float. It is expected that in this test, both formats provide similar classifications, considering their dynamic ranges.

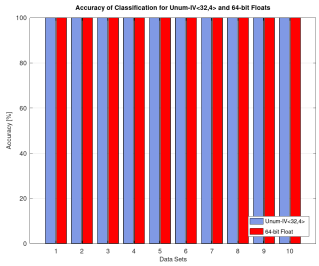

Fig. 12. Accuracy of Classification for Unum-IV<32,4> and 64-bit Floats.

All the data used in the set of benchmarks are supported by the 32-bit floats, which, nonetheless, may not apply to the square distance between the labelled and unlabeled points, as the computed distance might fall outside the range of the 32-bit floats. For this reason the 32-bit floats have a poor performance, as most of the computed distances overflow, resulting in less accurate classifications. On the other side, we have the Unum-IV format performs correctly in all the benchmarks, which is easily explained by the fact that Unum-IV$< 32, 4 >$ has a greater dynamic range than 32-bit and 64-bit floats. Another note that can be added to these results is that the behaviour of floats towards the overflow of assigning it to the $+\infty$ can cause mathematical incongruities, as the operations are not interrupted.

Finally, both experimental results show that Unum-IV<32,4> can indeed produce more accurate results than the IEEE 754 single-precision floating-point format for numbers with a small magnitude but also cover all the IEEE 754 double-precision dynamic range with very accurate results. This means that for these type of applications that tolerate some accuracy loss, 32-bit Unum-IV<32,4> can be a compelling replacement for the 64-bit IEEE 754 format. Unum-IV<32,4> has a wider dynamic range than the 64-bit IEEE 754 using half of the computer memory for the format and can provide near-one results with higher accuracy than the IEEE 754 format with the same word size.

With the addition of the conclusions obtained with these two experiments in terms of accuracy, it is possible to conclude that the Unum-IV<32,4> can satisfy the application requirements

while guaranteeing computational efficiency and lowering the power.

VII. Conclusions

In this work, we introduced a new floating-point number system, which can replace the IEEE 754 format offering significantly more precision and dynamic range, and beating the previous Unum-III proposal in terms of precision bits and dynamic range decades per logic gate.

The first step to achieve this goal involved studying the state-of-the-art of the IEEE 754 Standard and the different Unum formats to understand their drawbacks and how the new proposal could add value.

Then, the Unum Type-IV format is proposed, which introduces variably-sized exponent and significand scheme that effectively increases the dynamic range and accuracy compared to Unum-III. The new scheme is based on a dynamic hidden bit for the 1's complement exponent and another dynamic hidden bit for the 2's complement significand, that dispenses with the use of a unary representation for the super exponent called Regime bits in Unum-III (Posits). Compared to Posits, accuracy and dynamic range is increased by using a binary representation for both the exponent and the significand.

A parametrizable Unum-IV Floating-Point Unit (FPU) is developed in Verilog and implemented in FPGA and ASIC technology and tested in different format configurations. The new FPU is compared with IEEE 754 and Posits FPUs in terms of the used silicon. The new FPU has three hardware levels, unpacking, processing and packing, and includes four basic two-argument operations: addition, subtraction, division and multiplication.

A corresponding parametrizable IEEE 754 FPU is also implemented in FPGA and ASIC technology, and the synthesis results show that, for same bit width configurations, the area and power consumption of the Unum-IV FPU is lower. Published results on a Posits FPU have been used for comparison. For equivalent hardware size, Unum-IV has much more maximum precision than IEEE 754 and more maximum precision than Posits. For equivalent hardware size, Unum-IV has much more dynamic range decades than IEEE 754 and more dynamic range decades than Posits. This shows that Unum-IV has a better performance in terms of power consumption and silicon area than IEEE 754 and Posits.

For example, the 32-bit Unum-IV<32,4> configuration has 30x the dynamic range in decades compared with 64-bit floats while using 2.1569x less silicon area with 1.8189x less power consumption with a higher maximum clock frequency (200 MHz against 169.15 MHz of the 64-bit floats), using half of the computer memory for the format. This means Unum-IV<32,4> can produce similar results as when using 64-bit floats using half the memory and about 30% less silicon. For the same bit width, the Unum-IV<32,4> has a 237x larger dynamic range compared with the 32-bit floats. In terms of accuracy, it can outperform the 32-bit floats as the Unum-IV fraction bits float between 28 and 21 bits, whereas the 32-bit floats have a fixed fraction with 23 bits. Thus, in the best scenario, the Unum-IV<32,4> produces answers with five additional accuracy bits, and in the worst case, with fewer two bits of accuracy.

For Posits, the 64-bit Unum-IV<64,4> configuration has a dynamic range with more 19442 decades and a significand with 2 extra precision bits compared with Posit<64,3> while using 1.32x less silicon area, for example. This means the Unum-IV<64,4> can produce better results in terms of the dynamic range and precision than the 64-bit Posit<64,3>, while using about 32% less silicon. For equivalent hardware size, Unum-IV<32,3> has more 2 precision bits and about 10 more dynamic range decades than Posit<32,2> format with the same bit width.

It is concluded that the initial goals have been achieved: the proposed floating-point system can be a suitable replacement for the IEEE 754, in particular for the area of HPC and low precision applications, and beats its main competitor, Unum-III (Posits).

References

[1] D. Goldberg, "What Every Computer Scientist Should Know about Floating-Point Arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, p. 5–48, Mar. 1991. [Online]. Available: https://doi.org/10.1145/103162.103163

[2] A. Di Franco, H. Guo, and C. Rubio-González, "A Comprehensive Study of Real-World Numerical Bug Characteristics," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 509–519.

[3] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[4] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

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

[6] 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

[7] 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.

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

[9] 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

[10] 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

[11] 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

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

[13] 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.

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

[15] IObundle, "Iob-fpu: Parameterized Floating Point Unit." [Online]. Available: https://github.com/IObundle/iob-fpu

[16] 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