An edited version of this work was accepted in IEEE Trans. on Computers, DOI 10.1109/TC.2015.2479623

New Formats for Computing with Real-Numbers under Round-to-Nearest

Javier Hormigo, and Julio Villalba, *Member, IEEE,*

Abstract—In this paper, a new family of formats to deal with real number for applications requiring round to nearest is proposed. They are based on shifting the set of exactly represented numbers which are used in conventional radix-β number systems. This technique allows performing radix complement and round to nearest without carry propagation with negligible time and hardware cost. Furthermore, the proposed formats have the same storage cost and precision as standard ones. Since conversion to conventional formats simply require appending one extra-digit to the operands, standard circuits may be used to perform arithmetic operations with operands under the new format. We also extend the features of the RN-representation system and carry out a thorough comparison between both representation systems. We conclude that the proposed representation system is generally more adequate to implement systems for computation with real number under round-to-nearest.

Index Terms—Real-number representation, round-to-nearest, radix complement, arithmetic operations

1 Introduction

Two basic operations that are required in many arithmetic data-paths are rounding circuits and radix complement circuits [1]. The rounding circuits are used when it is necessary to reduce the number of significant digits. Among the different rounding modes, the round-to-nearest is the default mode in most processors, compilers and standards (including the IEEE754-2008 [2]). The circuit that performs a radix complement function is used to change the sign of the number, which is required for subtraction or absolute value computation. Any improvement in the efficiency of these two circuits directly affects the efficiency of the majority of the functional units that include them, especially since they are usually in the critical path of these units.

Recently, a family of signed-digit formats, RN-Representation, the RN standing for round-to-nearest'', has been presented, which allows performing unbiased round-to-nearest by truncation [3] [4] [1]. Later, a particular encoding, canonical radix-2 encoding, which basically attaches a rounding bit to a conventional two's complement representation was introduced in [5]. Besides rounding by truncation, this encoding allows sign inversion and conversion from conventional radix complement representation in a constant time. According to the authors, small modifications on conventional circuits are required to perform arithmetic operations with the new encoding and both fixed-point and floating-point computation may be improved by using this encoding, taking into account the simplification of rounding and negation [5]. However, the extra-bit required by the new encoding, implies either larger memory storage (and wider bandwidth) or less precision.

On the other hand, in [6], we have presented the utilization of a new format for binary two's complement fixed-point representation to optimize the hardware implementation of digital filters. Similarly to the one in [5], this new representation system drastically simplifies the computation of round-to-nearest rounding and the two's complement operation. Here, we formalize the new proposed format [6] and generalize it for different radices and sign representation. These new formats are based on shifting the numbers that can be exactly represented under conventional formats by adding a bias which equals half unit-in-the-last-place (ULP). That is the reason to call them 'Half-Unit Biased' (HUB) formats. This shifting could be also interpreted as appending a hidden least significant digit set to one to the conventional number stream (similarly to the leading one in IEEE754 standard).

Moreover, in this paper HUB formats are also extended to floating point representation and the basic arithmetic operations, including unbiased rounding, negation, conversion, addition and multiplication are studied in detail. Besides that, we also provide some extensions to the canonical RN-representation and a thorough comparison between both representation systems. This comparison demonstrates that, considering the same precision, the hardware implementation of a system using HUB format is generally more efficient than using RN-representation. This study also shows that HUB formats may be well suited for general floating-point applications and application-specific fixed-point data-paths under round-to-nearest. Some suitable applications may be DSP, industrial control or physics simulation, but

• *J. Hormigo, and J. Villalba are based at the Department of Computer Architecture, Universidad de Malaga, Malaga, Spain, E-29071. E-mail: fjhormigo@uma.es*

they are not appropriated for cryptography or other applications in which exact computation is required.

This paper is organized as follow: Section 2 gives an overview of the binary canonical RN-representation presented in [5] [7]. Our contribution begins in Section 3 where we propose some extensions to RN-representation system. In Section 4 the fundamentals of the HUB formats are presented, along with the rounding, conversion and radix complement operations. Basic arithmetic operations under the proposed format are analyzed in Section 5. Then, an analysis and comparison between HUB formats and RN-representation are provided in Section 6. Finally, Section 7 gives the conclusions of this work.

2 Binary Canonical Round-to-Nearest Representation

In [5], basic arithmetic operations for binary canonical RN-representation are thoroughly studied along with conversions between this representation and conventional twos complement representation. This encoding has some characteristics similar to the ones of the formats presented here. For this reason, we summarize the main ideas of canonical radix-2 RN-representation here, to study the differences and similarities with our proposal later. At this point, we should note that all ideas presented in this section are directly extracted from [5], whereas our own analysis and some extensions will be provided later.

According to the authors, a canonical radix-2 RN-representation of a value is defined as a pair of a two's complement number (a) and a round-bit (r_a), such as this value is

$$\nu(a, r_a) = a + r_a u \tag{1}$$

where u is the weight of the least significant position of a and $\nu(x, r_x)$ denotes the value of an RN-represented number. Let us call "Exactly Represented Numbers" (ERNs) to those values which are represented without error in a given representation system. According to previous definition, all ERNs may be represented as two different RN-representations:

$$\forall a, \nu(a, 1) = \nu(a + u, 0) \tag{2}$$

However, this representation could also be seen as a kind of "carry-save" in the sense that it contains a bit not yet added in (after a rounding, as we will see later). Taking this into account, these two representations of the same ERN describe different intervals of non-exact values, such as the pair $(a, 1)$ represents the interval $[a+u/2; a+u]$ and the pair $(a+u, 0)$ represents the interval $[a+u; a+3u/2]$. In the following, we shall use $I(x, r_x)$ to denote said intervals, such as

$$I(x, r_x) = \left[x + r_x \frac{u}{2}; x + (1 + r_x)\frac{u}{2}\right] \tag{3}$$

Therefore, although as values $\nu(a, 1) = \nu(a + u, 0)$, when they are interpreted as intervals, these intervals

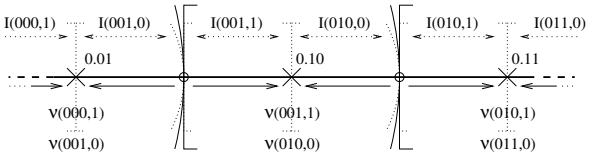


Fig. 1. Binary canonical RN-representations as both ERNs and intervals

are disjoint except for sharing the same ERN. In Figure 1, these ideas are shown through a graphical example. This figure shows the relation among intervals, ERN and rounding: ERNs are represented using a cross, circles represent the half point between two ERNs, dotted lines represent the bounds of the RN-represented number interpreted as interval whereas solid lines shows the range of values represented by the same ERN (values rounded to that ERN).

The canonical radix-2 RN-representation has very interesting properties. First, as RN-representation, it allows performing round-to-nearest rounding by a simple truncation. Considering an input value

$$(x, r_x) = (b_m b_{m-1} b_{m-2} \cdots b_l, r_x) \tag{4}$$

the rounding to nearest of this value is performed by truncating it, such as for $k > l$:

$$rn_k(x, r_x) = (b_m b_{m-1} b_{m-2} \cdots b_k, b_{k-1}). \tag{5}$$

Note that the new round-bit is equal to the MSB of the discarded bits.

Another interesting property is that negation of a radix-2 canonical RN-represented number is a constant time operation, since it is fulfilled

$$\forall x, \nu(\bar{x}, \bar{r_x}) = -\nu(x, r_x), \tag{6}$$

where $\bar{x}$ is the one's complement of x.

Conversion from two's complement representation is a trivial operation, which only requires to append a rounding-bit, set to zero, to the original two's complement number. However, conversion from canonical radix-2 RN-representation to two's complement representation requires at least logarithmic time, since it may need a non-zero round-bit to be added in (see equation(1)).

Regarding arithmetic operations, the authors recommend considering canonical RN-represented operands as twos complement values with a carry-in not yet absorbed to define them, and using interval interpretation to select the resulting round-bit. Using this methodology, addition and multiplication of canonical radix-2 RN-represented values are deeply studied. As a result, addition of two aligned numbers is defined as follows:

$$(a, r_a) + (b, r_b) = (a + b + (r_a \wedge r_b)u, r_a \vee r_b) \tag{7}$$

Subtraction may be easily performed by negating the adequate operand, i.e., inverting all its bits.

Similarly, multiplication is defined only for non-negative operands and considering $u \leq 1$ as follows:

$$(a, r_a) \times (b, r_b) = (ab + (br_a + ar_b)u, r_a r_b) \quad (8)$$

For general operands, a previous sign inversion of the negative operands is required and, accordingly, it may be required for the result.

Besides this detailed study of the fixed-point case, a floating-point format and some ideas about the implementation of a FP unit are also provided in [5] and extended in [7] [8]. This format comprises a significand encoded using a canonical two's complement representation with its round-bit and an exponent. The significand is normalized such as its value is between $[-1/2,\ 1/2]$ in [5], whereas its absolute value is between $[1, 2]$ with a hidden leading bit in [7] [8].

3 Some extensions to binary canonical RN-representation

In this section, we propose some extensions of the canonical RN-representation.

3.1 New formats

First, RN-representation is by definition a signed-digit format which, using canonical encoding, may be expressed by a two's complement number and a round-bit. However, using the interpretation of the round bit as a carry-bit not yet added in, new formats based on sign-and-magnitude or unsigned formats could be defined with similar characteristics to RN-representation.

Similarly to the definition of canonical binary RN-representation, we defined unsigned binary RN-representation as a pair of an unsigned number (a) and a round-bit (r_a), such as said value is

$$\nu(a, r_a) = a + r_a u \quad (9)$$

where u is the weight of the least significant position of a and $\nu(x, r_x)$ denotes the value represented by the RN-represented number.

In the same way, we defined sign-and-magnitude binary RN-representation as a triplet of a sign bit (s_a), an unsigned number (a) representing the magnitude and a round-bit (r_a), such as said value is

$$\nu(s_a, a, r_a) = (-1)^{s_a}(a + r_a u) \quad (10)$$

where u is the weight of the least significant position of a and $\nu(s_x, x, r_x)$ denotes the value represented by the RN-represented number.

Based on these definitions, the extension of the conversion and arithmetic operations defined in [5] for canonical binary RN-representation to the new defined formats is straightforward. Let us highlight that round-to-nearest is performed by truncation; conversion from the corresponding conventional representation is trivial whereas from the new formats to conventional takes at least logarithmic time; and addition and multiplication requires a reasonable cost increment compared to conventional numbers.

The presented fixed-point formats could be the base of new floating point-formats. For example using the definition of sign-and-magnitude binary RN-representation, it could be defined as an IEEE754 like format by using the RN-represented number for the significand.

3.2 Unbiased rounding

The rounding operation defined in [5] for RN-represented numbers is biased, since the tie case is always rounding up. When all discarded bits are zero, but the MSB, the value to be rounded is exactly halfway between two ERNs. These ERNs correspond to taking the MSBs of the value to be rounded and setting the round bit to either zero or one. However, according to Eq. (5), this bit is always set to one (the value of the MSB of the discarded bits). Nonetheless, it could be possible to define an unbiased rounding for RN-representation by selecting the value of the round bit in a more random way, i.e. based on a different bit or a logic equation.

First, the tie case should be detected. Similarly to conventional representations, the computation of a sticky bit could be utilized, along with the checking of the MSB of the discarded bits. Then, the value of the round bit could be selected according to the LSB of the remaining bits. For example, a tie-to-even rounding could be defined modifying Eq. (5), such as

$$rn'_k(x, r_x) = (b_m b_{m-1} b_{m-2} \cdots b_k, (T \vee b_k) \wedge b_{k-1}). \quad (11)$$

where T is the sticky bit ($T = b_{k-2} \vee b_{k-3} \vee \cdots \vee b_l$).

4 Half-Unit Biased formats

In this section, we formalize and extend the ideas used in [6] to optimize digital signal processing data-paths. A new representation system is defined, which allows simplifying the computation of round-to-nearest rounding and radix complement.

First, let us remember that, under radix-β fixed-point conventional number systems, real numbers are represented by ordered n-tuples of digits, called digit-vectors, where each digit (which is lower than the radix β) is weighted by a power of β according to its position in the n-tuple [9]. Positive powers of the radix represent the integer part, whereas negative ones represent the fractional part. For instance, the n-tuple $(X_{n-1}, X_{n-2}, \ldots, X_1, X_0, X_{-1}, \ldots X_{-f})$, where $0 \leq X_i < \beta$, may exactly represent the positive real number X such as

$$X = \sum_{i=-f}^{n-1} X_i \cdot \beta^i \quad (12)$$

However, most of the actual values have to be approximated assuming an error by using one of the

ERNs, i.e. those numbers which fulfill Eq.(12). Said in a different way, they have to be rounded to one of these ERNs.

For simplicity, and without loss of generality, let us consider a fixed-point number X with n integer digits and f fractional digits, which has to be rounded to another number Y with g fractional digits, assuming $f > g$. If the new number is selected by truncation, which is simply discarding the $(f-g)$ Least Significant Digits (LSDs) of X, the rounding error $E_T = X - Y$ fulfills

$$0 \leq E_T < \beta^{-g} \tag{13}$$

The upper-bound of the error equals the weight of the LSD of Y. This weight is also referred as one Unit-in-the-Last-Place (ULP).

On the other side, if the ERN closest to X is selected, which is round-to-nearest, the rounding error E_R fulfills

$$-\frac{1}{2} \cdot \beta^{-g} \leq E_R < \frac{1}{2} \cdot \beta^{-g} \tag{14}$$

Thus, the bounds of this rounding error halve the ones due to truncation.

To define a new format which, by truncation, produces the same rounding error bound as the rounding-to-nearest, let us start from the following equation derived from eq.(13), corresponding to the rounding error due to truncation:

$$0 \leq X - Y < \beta^{-g} \tag{15}$$

If half ULP is subtracted from all terms of the inequality:

$$-\frac{1}{2} \cdot \beta^{-g} \leq X - Y - \frac{1}{2} \cdot \beta^{-g} < \beta^{-g} - \frac{1}{2} \cdot \beta^{-g} \tag{16}$$

If a new value Y' is defined as

$$Y' = Y + \frac{1}{2} \cdot \beta^{-g} \tag{17}$$

the new Eq.(16) corresponding to the rounding error will be

$$-\frac{1}{2} \cdot \beta^{-g} \leq X - Y' < \frac{1}{2} \cdot \beta^{-g} \tag{18}$$

It is clearly observed that eq(18), which corresponds to the rounding error due to truncation, is identical to eq(14) corresponding to round-to-nearest, but using Y' as target value instead of Y. Hence, to achieve our goal of simplifying rounding, we propose a new family of formats for even radices, such as the value represented by a digit-vector $X' = (X_{n-1}, X_{n-2}, \ldots, X_1, X_0, X_{-1}, \ldots X_{-g})$ is

$$X' = \left[\sum_{i=-g}^{n-1} X_i \cdot \beta^i \right] + \frac{\beta}{2} \cdot \beta^{-g-1} \tag{19}$$

Eq.(19) provides a similar definition as Radix-β conventional number systems (see Eq.12) but, in this case, the ERNs are half-ULP right shifted respect the ERNs of the conventional ones. For that reason, we call the new proposed formats as Half-Unit Biased (HUB) formats. For example, under the proposed representation system, a four-fractional-digit Radix-10 vector such as (4,3,8,2) represents the value $X' = 0.43825$ instead of the value $X = 0.4382$ corresponding to a conventional radix-10 representation. Similarly, if we want to round to the nearest the value 0.4963 using a two-fractional-digit Radix-10 number, $(5, 0)$ is the digit-vector under conventional representation, but $(4, 9)$ under the HUB one. In both cases the rounding error is below 0.005 (-0.0037 and 0.0013, for conventional and HUB representation, respectively).

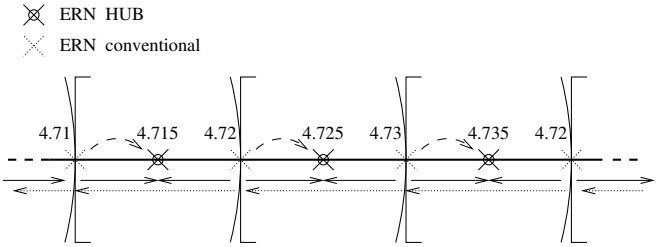


Fig. 2. HUB format vs. conventional one

Comparing conventional and HUB formats with the same number of digits, the number of ERNs under both formats is the same. However, the values exactly represented in each format are different. For example, under radix-2 fixed-point format with only two fractional digits, four values are represented exactly (0, 0.25, 0.5, 0.75), and in the corresponding HUB format, the same number of values are also represented exactly, but different ones (0.125, 0.375, 0.625, 0.875). Another example, under radix-10 fixed-point format with only one fractional digit, ten values are represented exactly$(0.0, 0.1, 0.2, \ldots, 0.8, 0.9)$, and in the corresponding HUB format, again the same number of values are also represented exactly, but different ones $(0.05, 0.15, 0.25, \ldots, 0.85, 0.95)$. More specifically, the ERNs under the HUB format appear exactly at the halfway points between the ERNs under the conventional format. This means that the accuracy will be equivalent in both formats, but conversions between them are not exact.

Figure 2 shows graphically the relation between a HUB format and its equivalent conventional one. It also shows how truncation produces a round-to-nearest under the HUB format. In that example, the selection of the three MSDs (truncation) produces that all numbers within the intervals $[4.72, 4.73)$ are rounded to 4.725 under the HUB format, whereas they will be rounded to 4.72 when targeting a conventional format. In the first case, a round-to-nearest is performed and the rounding error is bounded by ± 0.005. In the conventional case, a rounding down is always performed and the rounding error is bounded by 0.01.

Table 1 shows the codification of several selected real values using a 4-fractional-bit format for HUB

TABLE 1
Examples of HUB and conventional binary representation for real numbers

Real value	HUB	error$_{(10^{-3})}$	conventional	error$_{(10^{-3})}$
0.1	0.0001	6.25	0.0010	-25
0.2	0.0011	-18.75	0.0011	12.5
0.3	0.0100	18.25	0.0101	-12.5
0.4	0.0110	-6.25	0.0101	25
0.5	0.1000	-31.25	0.1000	0
1.00	1.0000	-31.25	1.0000	0
1.38	1.0110	-26.25	1.0110	5
1.72	1.1011	1.25	1.1100	-30
1.454	1.0111	-15.75	1.0111	15.50
1.65625	1.1010	0	1.1010	31.25

and conventional binary representation along with the associated rounding error. The bit vector may be either the same or different but the rounding error is always different (at least the sign). However the rounding error is always bounded by 0.5 ULP in all cases ($31.25e-3$ for 4 fractional bits).

In Table 1, it is also observed that HUB formats cannot exactly represent integer values. They are exactly represented in conventional fixed-point formats by setting all fractional bits to zero. However, under HUB formats, the ERNs that represents those values are shifted by half-ULP. As a consequence, integer values are always represented with an error, which decreases with the number of fractional digits used. This fact does not represent a problem in computation with real numbers, since in this context integer values do not have more importance than others. For example, in digital signal processing application, if a filter coefficient or a signal sample is an integer value, it is incidental.

4.1 Unbiased round-to-nearest

Returning to the rounding problem, when a real value is rounded to a HUB number simply by truncation, that rounding is performed to the nearest number, but using a biased version, i.e., if the value is exactly halfway between two ERNs, the greatest one is always selected (rounding up). For instance, if the radix-10 real value 0.3700 is rounded to two-fractional-digit HUB number, the value 0.375 is obtained (rounding up), but the value 0.365 is exactly the same distance. This may produce a bias in the rounding error which may be a problem for certain applications.

To overcome this bias problem, a solution which does not require carry propagation is proposed. First, the tie case is detected by testing when all discarded digits (including the MSD of the discarded ones) are zero. Then, the LSD of the ones that remains are checked, such as

- if it were even, nothing is done (rounding up).
- if it were odd, the value one is subtracted from that digit (rounding down).

In the last case, since the digit is odd, this subtraction is also carry-free. Thus, the main goal of getting a round-to-nearest without carry propagation remains. According to that, the value of the previous example, 0.3700, is rounded to 0.365 (rounding down), whereas 0.3800 is rounded to 0.385(rounding up).

In a practical implementation, where digits are implemented by conventional bit-vectors, both the checking and the subtraction of the LSD are simply implemented by zeroing the least significant bit (LSB) of the LSD:

- if the digit was even, the LSB was actually zero and nothing has been done.
- if the digit was odd, the LSB was one and subtraction has been done by setting it to zero.

We should note that, although this unbiased rounding prevents the carry propagation, it may require testing for all zeroes of the discarded digits, which is also at best a logarithmic time operation. However, the utilization of this unbiased mode is not required when the initial value to be rounded is a HUB number. For instance, that happens after a multiplication or unaligned addition of two HUBs numbers. Due to the definition of HUB numbers, these values could never be exactly in the middle point between two ERNs. Thus, the simple truncation of a HUB number always produces an unbiased rounding when targeting another HUB number with fewer digits.

4.2 Double rounding error

The double rounding error is a well-known problem, which may happen when a number is rounded twice in a row using an unbiased round-to-nearest mode, first to a format with $n1$ digits, and then to a format with $n2$ digits, being $n2 < n1$ [10]. It occurs when the first round goes to a number which is in the middle of two ERNs of the second format, and the second rounding cause a round to the same direction, due to the unbiased approach. Then, the rounding error of the last number obtained is greater than $1/2$ ULPs, which means that it is not the nearest ERN to the initial value. For example, if the value 3.174962 is rounded to a conventional number with 4 fractional digits first and then to a number with two fractional digits using round-to-nearest-even, 3.1750 (round up) is obtained first and then, the number 3.18 (round up) is obtained at the end, but the closest one is 3.17.

Another advantage of using HUB formats is that the double rounding error is overcome. Since the round-to-nearest is performed by truncation, the rounding error of the last number obtained, even after several intermediate rounding operations, is always bounded by $1/2$ ULPs.

4.3 Radix-complement operation

Besides allowing the round-to-nearest by only truncation, a side effect of using a HUB format is that the radix complement operation ($RC(x)$) is also performed without the final addition. The diminished radix complement ($DRC(x)$) of a HUB fixed-point number X' is

$$\begin{aligned} DRC(X') = \\ &= \left[\sum_{i=-f}^{n-1}(\beta-1-X_i)\right] + \left(\beta-1-\frac{\beta}{2}\right)\cdot\beta^{-f-1} \\ &= DRC(X) + \left(\frac{\beta}{2}-1\right)\cdot\beta^{-f-1} \end{aligned} \quad (20)$$

Adding one ULP to obtain the radix complement

$$\begin{aligned} RC(X') &= DRC(X) + \left(\left(\frac{\beta}{2}-1\right)+1\right)\cdot\beta^{-f-1} \\ &= DRC(X) + \frac{\beta}{2}\cdot\beta^{-f-1} \\ &= (DRC(X))' \end{aligned} \quad (21)$$

Therefore, the addition of one ULP does not produce carry propagation towards the rest of the digits and the radix complement of a HUB number is obtained by just computing the digit-wise diminished-radix-complement of the HUB number. Thus, no final addition is required. This simplification is very beneficial for operations like absolute value, addition/subtraction for Sign-and-magnitude numbers, or conversions.

4.4 Conversion from or to conventional representation

The conversion from a conventional number to a HUB number may be performed simply by truncation. In this way, the original value is rounded to the nearest HUB number. For example, considering a conventional four-fractional-digit number $(2,3,6,8)$, which represents the value 0.2368, its conversion to a HUB would be $(2,3)$ (truncation), which represents the value 0.235 (the closest HUB number). This conversion (or rounding) could also be unbiased using the technique explained in Section 4.1. This last option is highly recommended when both formats have the same numbers of digits, since in this case the conventional number is always halfway between two HUB numbers (see Figure 2). For example, the conventional two-fractional-digit number $(2,3)$ is exactly halfway between the HUB numbers $(2,3)$ and $(2,2)$, but the latter is selected if the unbiased rounded defined in Section 4.1 is used. However, the conversion between two formats with the same number of digits should be prevented, since it introduces an unnecessary rounding error. Since the operations involving both conventional and HUB numbers are easy to implement (as we will see later), then it will be better to delay the conversion until a rounding is really required. For example, let suppose a system working with HUB numbers as internal representation which firstly requires the multiplication of two n-digit inputs values to obtain another n-digit value. If the input numbers are under conventional representation (or only one of them is under HUB representation), it is better to delay the conversion until the multiplication is performed instead of before.

On the other hand, to study the conversion from a HUB number to a conventional one, let us develop eq(17) further:

$$\begin{aligned} Y' &= \left[\sum_{i=-g}^{n} Y_i\cdot\beta^i\right] + \frac{\beta}{2}\cdot\beta^{-g-1} \\ &= \sum_{i=-g-1}^{n} Y_i\cdot\beta^i \end{aligned} \quad (22)$$

being $Y_{-g-1} = \beta/2$. Therefore, from a different point of view, the HUB format appends a new constant LSD (Y_{-g-1}) which is not represented explicitly in the digit vector. For hardware implementation, said digit is not stored or transmitted explicitly, since this hidden digit is constant and equals $\beta/2$. But, if said constant hidden LSD is represented explicitly, an implicit conversion from g-digit HUB format to a $(g+1)$-digit conventional format has actually been performed. This $(g+1)$-digit number may be treated as a conventional one. For example, the HUB four-fractional-digit number $(7,3,5,6)$ represents the value 0.73565 and it is converted to conventional representation by appending the hidden constant LSD, producing the conventional number $(7,3,5,6,5)$. Thus, conversion from HUB format to conventional one is performed trivially by appending the hidden digit explicitly and does not produce any error. Therefore, as we will see later, this trivial conversion may be used to perform operations (arithmetic, conversions or other type) with those numbers using conventional circuits. However, hardware implementation is usually simplified, when it is taken into account that the LSD of those converted numbers are always $\beta/2$.

4.5 Signed fixed-point and floating-point representations

In previous examples, only unsigned real fixed-point values have been considered. However, the application of the main idea to other conventional representation is straightforward. Basically, said main idea is the half-ULP shifting of the ERNs, or viewing it in a different way, the appending of an implicit LSD set to $\beta/2$ to the initial ERNs.

Let us begin by including the sign information. Using a sign-and-magnitude representation, one bit

TABLE 2
Examples of HUB and conventional binary representation for negative real numbers

Real value	HUB	error$_{(10^{-3})}$	conventional	error$_{(10^{-3})}$
sign-and-magnitude				
-0.1	1.0001	-6.25	0.0010	25
-0.2	1.0011	18.75	0.0011	-12.5
two's complement				
-0.1	1.1110	-6.25	1.1110	25
-0.2	1.1100	18.75	1.1101	-12.5

is utilized to encode the sign whereas the magnitude is represented using an unsigned HUB number as defined before. Then, the digit-vector $X' = (S, X_{n-1}, X_{n-2}, \ldots, X_1, X_0, X_{-1}, \ldots X_{-g})$ where $S \in \{0, 1\}$ is

$$X' = (-1)^S \cdot \left(\left[\sum_{i=-g}^{n-1} X_i \cdot \beta^i \right] + \frac{\beta}{2} \cdot \beta^{-g-1} \right) \quad (23)$$

It also could be seen as a conventional sign-and-magnitude number extended with an implicit LSD set to $\beta/2$. Thus, the magnitude preserves the properties of a HUB representation, and the sign has to be handled as under conventional sign-and-magnitude representation.

Similarly to conventional radix complement representation, to use HUB radix-complement representation, the definition of the unsigned HUB number (Eq.(19)) is modified such as the MSD is negatively weighted, i.e.,

$$X' = -X_{n-1} \cdot \beta^{n-1} + \left[\sum_{i=-g}^{n-2} X_i \cdot \beta^i \right] + \frac{\beta}{2} \cdot \beta^{-g-1} \quad (24)$$

Again, it could be seen as a conventional radix complement number extended with an implicit LSD set to $\beta/2$. HUB radix-complement representation has the same properties as unsigned HUB representation, including the simplicity of radix-complement operation which allows constant time negation under this format. Table 2 shows a few examples of signed numbers representation under HUB and conventional formats.

A floating-point HUB format should include a significand represented using any of the HUB signed fixed-point representations described before, along with an exponent represented in any conventional way. For example, using a sign-and-magnitude representation for the significand and, E and B being the exponent and base, respectively, $(X, E_x)'$ represents

$$(X, E_x)' = (-1)^S \cdot \left(\left[\sum_{i=-g}^{n-1} X_i \cdot \beta^i \right] + \frac{\beta}{2} \cdot \beta^{-g-1} \right) \cdot B^E \quad (25)$$

Again, a floating-point HUB number could be seen as a conventional floating-point number, where its significand is extended with an implicit LSD set to $\beta/2$. We should note that, as it is shown in Eq.(25), the weight of this implicit digit is half ULP respect to the significand, for all possible values of the exponent. Since the exponent does not affect the rounding process, this floating-point HUB representation also allows rounding to the nearest by truncation along with maintaining the remaining characteristics described before. Moreover, the definition of special cases or denormalized numbers is not modified by the implicit LSD.

5 Arithmetic operations with HUB numbers

Arithmetic operations with numbers under a HUB format could be performed by converting them first to a conventional format, then operating the conventional numbers using conventional circuits and, finally, converting the result back to a HUB format. Supposing a HUB number with n digits, the first conversion is performed by appending the hidden LSD to the number, such as a conventional number with $n+1$ digits is obtained (see Section 4.4). This new LSB is constant and equal to $\beta/2$. It is clear that this conversion is performed with no error and negligible cost.

The desired operation is performed using a conventional circuit for $n+1$ digits. Therefore, support for an extra digit is required, compared with the case of having conventional numbers initially. The increase of the cost will depend on the concrete operation. It is important to note that the described process also allows operating with HUB and conventional numbers together, since the actual operation is performed in the conventional way. In this case, HUB numbers are converted before, whereas conventional numbers go directly to the conventional circuit.

The conversion of the result back to the HUB format is carried out simply by truncating it to obtain the number of digits desired. This produces a round-to-nearest result (the biased version) under the HUB format. Thus, the conversion and rounding is performed with almost no cost. However, for the general case, if unbiased rounding is desired, the sticky bit of the discarded digits has to be computed. Then, one ULP is subtracted from the LSD of the converted result, if it is odd and the sticky bit is zero. For some operations, this additional computation is not required, since the sticky bit is known in advance for all cases, as we will see later.

The process described above is a general way to operate with HUB numbers. Then, theoretically, it could be possible to use actual processors or systems working with conventional number systems to operate with HUB number by software. However, that requires to make explicit the hidden LSD (for operation, storage and transmission), which eliminates any

advantage of using HUB formats. Although, it would be more reasonable to modify the actual designs at hardware level following the process described above to deal with HUB number, simpler process could be performed by studying each particular case. Following, some main operations are studied.

5.1 Addition and Subtraction

Here we focus on addition since subtraction is performed by negating one of the operands, which only involves the diminished radix complement of each digit thanks to the use of HUB format. For simplicity, and without any loss of generality, let us consider two fixed-point HUB numbers X' and Y', with g and f fractional digits, respectively, being $g \leq f$ and only one integer digit for both numbers (the left-aligning of both numbers by following conventional techniques and the generalization to a different number of integer digits are straightforward). Considering eq(17), the result of the addition of these two numbers is

$$X' + Y' = X + \frac{\beta}{2} \cdot \beta^{-g-1} + Y + \frac{\beta}{2} \cdot \beta^{-f-1}. \quad (26)$$

Now, two cases are considered. First, let us suppose $g < f$, which represents the case when both operands are not right-aligned. This is a typical circumstance in floating-point addition for numbers with different exponents. Let us consider that the number Y' represents the significand of the number with the lower exponent. Then, Y' is right-shifted using conventional left extension, and thus, it has more fractional digits than X'. On the other hand, this situation may also happen in fixed-point digital processing systems, since operands may be unaligned or have different word-length [6]. In this case, it is fulfilled

$$X' + Y' = \left[\sum_{i=-g}^{0} (X_i + Y_i) \cdot \beta^i\right] + \left(\frac{\beta}{2} + Y_{-g-1}\right) \cdot \beta^{-g-1} + \left[\sum_{i=-f}^{-g-2} Y_i \cdot \beta^i\right] + \frac{\beta}{2} \cdot \beta^{-f-1} \quad (27)$$

The last two addends of Eq.(27) are simply the $f - (g+1)$ explicit LSDs of Y' and its implicit LSD, respectively. Thus, if those digits of the result are required, they could be simply copied at the output. The MSDs of the result, corresponding to the first and second addend could be computed by using a conventional $(g+1)$-digit adder and a special logic circuit(which adds the constant $\beta/2$ to one digit), respectively. Thanks to the last addend of Eq.(27), the exact result could be represented by using a HUB number of f fractional explicit digits. Moreover, it is guaranteed that the LSD of the exact result is always non-zero. Thus, the result could be unbiasedly rounded to the desired length (typically using g fractional digits) simply by truncation, without using hardware for computing the sticky bit. Figure 3 shows an example of an adder with the exact result for the binary representation. In this case, the special logic to compute the second addend is simply an inverter. Figure 4 shows another example with the result rounded to the size of the shortest operand by using unbiased round-to-nearest.

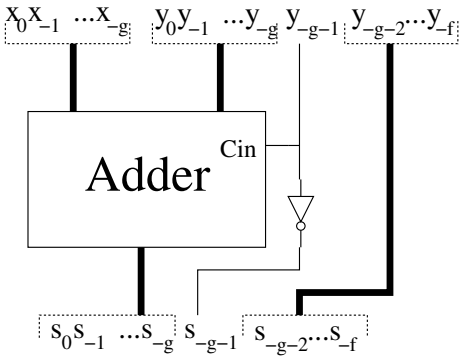


Fig. 3. Binary adder for unaligned HUB numbers

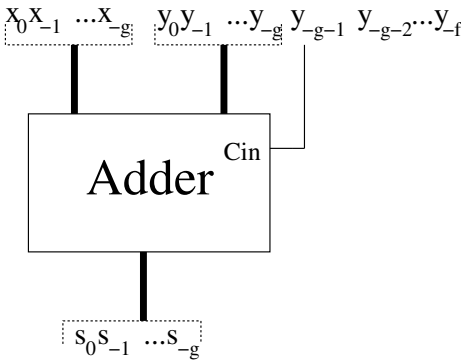


Fig. 4. Binary adder for unaligned HUB numbers with rounding

Now, let us suppose $f = g$, i.e., both operands are aligned. This is a typical circumstance in fixed-point processors or in floating-point computation when the operands have the same exponent, then

$$\begin{aligned} X' + Y' &= \left[\sum_{i=-g}^{0} (X_i + Y_i) \cdot \beta^i\right] + \left(\frac{\beta}{2} + \frac{\beta}{2}\right) \cdot \beta^{-g-1} \\ &= X + Y + \beta^{-g} \end{aligned} \quad (28)$$

In this case, the addition could be computed using a conventional $(g+1)$-digit adder with the carry input set to one. Unfortunately, the result could not be exactly represented using a HUB number and several options could be considered. Taking the resulting output digit string directly from the adder as a HUB number actually performs a round-to-nearest which produces an error of 0.5 ULP (the maximum for this rounding mode). Since this error is always in the same direction, a bias is produced. However, setting the

LSD of the digit string result to zero (in fact, saving hardware in many cases) turns this rounding into an unbiased one, although the error of 0.5 ULP is kept. For a floating-point implementation, the latter should be the preferred solution when aligned addition occurs, i.e. when the operands have the same exponent. For a specific application fixed-point datapath, a better option may be delivering the result in a conventional format and returning to the HUB format whenever rounding is required.

In the case of multi-operand addition, the implicit LSD corresponding to all input HUB numbers could be gathered to be added together as a constant value. For example, let us suppose a multi-operand addition of K HUB input operands, all of them having the same number of digits and being aligned. The result of the addition of the explicit digits of all input numbers could be computed using any conventional circuit. Then, the final result could be computed by adding the constant $\lfloor K \cdot \beta/2 \rfloor$ (which is the addition of the implicit LSD of all inputs), aligned to the right, to the previous result. If K is odd the result is exact considering a HUB output, whereas it is exact under conventional representation if K is even. Similarly, if the sizes of the input numbers are not the same, the weight of the LSD corresponding to each number needs to be taken into account to generate the constant value.

5.2 Multiplication

Let us consider again the same two numbers of the previous section, X' and Y'. Using eq(17) the result of the multiplication of these two numbers is

$$\begin{aligned} X' \cdot Y' &= \left(X + \frac{1}{2}\cdot\beta^{-g}\right)\cdot\left(Y + \frac{1}{2}\cdot\beta^{-f}\right) \\ &= X\cdot Y + \frac{X}{2}\cdot\beta^{-f} + \frac{Y}{2}\cdot\beta^{-g} + \frac{1}{4}\cdot\beta^{-g-f} \\ &= X\cdot Y + \left\lfloor \sum_{i=-g}^{0} \frac{X_i}{2}\cdot\beta^{i-f} \right\rfloor + \\ &+ \left\lfloor \sum_{i=-f}^{0} \frac{Y_i}{2}\cdot\beta^{i-g} \right\rfloor + \frac{1}{4}\cdot\beta^{-g-f} \end{aligned} \quad (29)$$

Taking into account that the product $X \cdot Y$ has $f + g$ fractional bits, the first three addends of the final addition in eq(29) are aligned and the last one is a constant value. However, the feasibility of halving all digits to compute the second and third addends depends on the radix and the digit representation. If we particularize for the binary case, eq(29) results

$$X' \cdot Y' = X \cdot Y + X \cdot 2^{-f-1} + Y \cdot 2^{-g-1} + 2^{-g-f-2} \quad (30)$$

Therefore, a conventional multiplier may be used, but two new addends may be included into the partial product compressor tree. These addends correspond to the input operands left-shifted one bit more that he LSB of the conventional multiplication result, whereas the last term (β^{-g-f-2}) corresponds to the implicit LSB of the HUB format and it is omitted. Thus, at first, the cost increase of a HUB multiplier is the inclusion of these two values in the partial product array. However, in many applications, including floating-point units, the exact result of the multiplier has to be rounded to fit into the size of one of the input operands. In this case, since the LSB of the exact result is always one, no sticky bit computation is required. Then, for those LSDs which are going to be discarded, only a logic to generate the carry signal corresponding to those lest significant part is sufficient. Hence, the use of HUB format not only avoids the cost of the rounding circuit, but also the computation of the LSDs of the results, and the overall cost may be significantly reduced in many cases.

6 HUB FORMAT VERSUS CANONICAL RN-REPRESENTATION

Although canonical RN-representation for other radices is also defined in [5], they are not studied deeply enough to be compared here. Thus, we focus only on radix-2 representation.

To perform a fair comparison, let us consider following a HUB two's complement fixed-point format and a canonical radix-2 RN-representation both with the same number of bits (i.e., they require the same amount of resources to be stored or transmitted). Then, considering n fractional bits for the RN-representation, since RN-representation also includes the round-bit, the corresponding HUB format has $n + 1$ fractional bits (the hidden bit is not taken into account). An example of this situation is represented in Fig. 5. It is clearly seen that the same binary string in both formats represent the same set of non-ERNs (the ones which produce the same result after truncation). The difference between them is which value is selected to represent this set of values. For RN-representation, it is the closest n-fractional-bit number and the round-bit allows to distinguish between the right and the left interval. In contrast, for HUB format, it is the value in the half-point of the interval, which is a $(n + 2)$-fractional-bit number with the LSB set to one.

As we have seen in Section 4, HUB formats could be defined for unsigned or signed numbers, including two's complement or sign-and-magnitude. Similarly, although canonical RN-representation was only defined for two's complement in [5], we have extended it to unsigned and sign-and-magnitude representation. Following, only two's complement is considered, but the extension of the presented ideas to the others cases is straightforward.

Given a real value (A), when encoding it using the canonical RN-representation, and taking into account

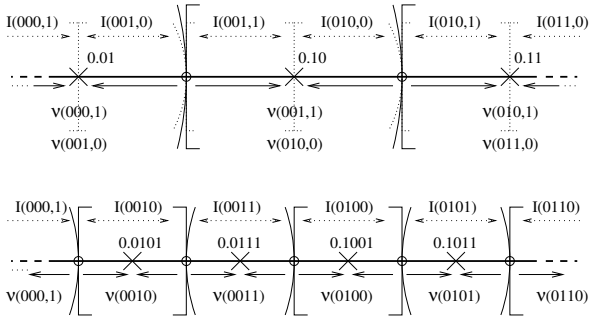


Fig. 5. Binary HUB format vs. binary canonical RN-representation

the interval interpretation, the rounding error ($E_{RN} = A - \nu(a, r_a)$) fulfills

$$\begin{cases} 0 \leq E_{RN} < 2^{n-1} & if \quad r_a = 0 \\ 0 > E_{RN} \geq -2^{n-1} & if \quad r_a = 1 \end{cases} \tag{31}$$

Taking into account only the value represented, then, it fulfills

$$|E_{RN}| \leq 2^{n-1} \tag{32}$$

In contrast, when the same value is encoded under the HUB format, the rounding error (E_{HUB}) fulfills

$$|E_{HUB}| \leq 2^{n-2} \tag{33}$$

Therefore, the maximum rounding error is halved using the HUB format instead of the RN-representation, when the same numbers of bits are considered.

Both representations allow rounding to nearest and negation without carry propagation. The former operation simply requires a truncation and the latter, inverting all bits. Similarly, both formats prevent the double rounding error. Moreover, we have defined biased and unbiased rounding for both formats (although the rounding operation defined in [5] is only biased, since the tie case is always rounding up (see equation(5)). Nevertheless, although directed rounding modes are easily implemented under RN-representation, they require much more effort under HUB formats. Due to the bias added, round toward minus infinity may necessitate to perform a subtraction of one ULP, whereas round toward plus infinity, an addition. Thus, directed rounding modes are at least logarithmic time for HUB numbers. Therefore, HUB formats are only recommended when round-to-nearest is the only rounding mode. However, this is the preferred rounding mode for most applications.

Regarding conversion to and from a conventional two's complement representation, there are important differences. Although conversion from two's complement to either the RN-representation, or the HUB format, is a trivial operation, said conversion is not exact for the latter. This fact should not be an important problem because this conversion is generally associated with a rounding which absorbs this error. It would be desirable that the input values were converted to HUB format directly from the input sources such as ADCs or the numbers in different radix (e.g., converting from conventional decimal to HUB binary). Otherwise, the conversion could be delayed until a rounding is actually required.

On the other hand, conversion from HUB format to two's complement is also a trivial operation, which simply requires appending a LSB set to one. However, conversion from RN-representation requires, at least, logarithmic time, since the number has to be incremented if the round-bit is equal to one. In both cases, no error is produced due to the conversion.

In general, arithmetic operations with RN-represented numbers require special circuits designed by carefully studying both the value and interval interpretation of those numbers. However, HUB numbers could be operated using regular circuits by trivially converting them to conventional numbers before. Nonetheless, it is recommendable to optimize said circuits instead, taking into account that the LSB of the converted numbers is always equal to one.

Focusing on the operations studied in detail, addition of two aligned RN-Represented numbers requires a regular n-bit adder plus an AND-gate and an OR-gate to provide the carry-in of the adder and the round-bit of the result, respectively. Similarly, the corresponding addition for the HUB case, could be implemented by also using an n-bit adder plus an OR-gate and an inverter to provide the carry-in of the adder and the round-bit of the result, respectively. Thus, the cost in both cases is practically the same.

Besides the hardware cost, we have empirically studied the accuracy of the aligned addition for both formats. In our experiment, we have utilized 16-bit two's complement fixed-point numbers within the range $(-1\ 1)$ (i.e., one sign bit and 15 fractional bits) as exact real numbers. They have been converted to RN-representation and HUB format, for both cases with 8 explicit bits. Thus, a rounding has been required for this conversion, and round-to-nearest through truncation has been used. To test the addition operation, 250000 exact results corresponding to the addition of two real numbers generated randomly (excluding the results which produces overflow) have been calculated by using 16-bit fixed-point arithmetic. Moreover, the additions of the same pairs of numbers, but previously converted to both studied formats, have been computed using the corresponding algorithm on each case, i.e. the algorithms described in [5] for RN-representation and the ones in this paper for HUB format. These results have been converted back to their exact values (16-bit conventional fixed-point representation) and compared to the exact results obtained using 16-bit fixed-point arithmetic. Besides the ERNs, since the interval interpretation of RN-representation numbers provides more information than only their value interpretation, the error of said

interval interpretation by considering both bounds of the intervals have been also calculated .

In Fig. 6, the probability distributions of the errors are shown by the histogram of the rounding error for both HUB format and RN-representation. Solid lines correspond to the computed value, whereas dashed and dashed-dotted lines correspond to the lower and higher bounds of the associated interval, respectively. Since the addition of aligned HUB operands has three possible implementations, as we described in Section 5.1, all of them have been considered here for comparison: HUBconv corresponds to an adder with its output delivered under conventional representation, HUBbiased corresponds to an adder with its output biased-rounded to HUB format and HUBunbiased corresponds to an adder with its output unbiased-rounded to HUB format. To facilitate comparison, table 3 shows the main statistical parameters corresponding to those errors.

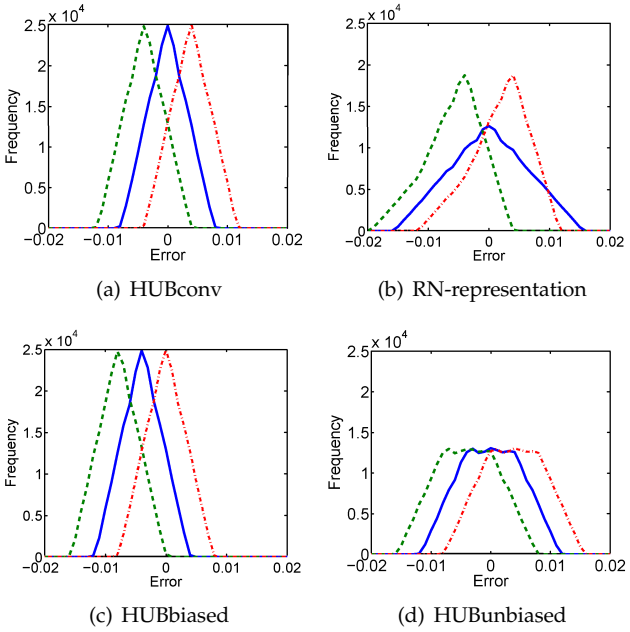
It is clearly observed that the three cases of addition using HUB numbers produce less error than using RN-representation. That is explained by the fact that RN-representation algorithms use the ERNs (which are one of the end-points of the interval and, consequently, double the rounding error amount) to compute the ERN of the result and the round bits only to select the two possible intervals for this ERN. Therefore, since the initial operands accumulated more rounding error, the results are less accurate than those for HUB representation.

As expected, the best result is produced when adding HUB numbers and the output is provided in a conventional format. In this case, the error that is produced is half that of the one produced by using RN-representation. If the output is preferred under HUB format, the compulsory final rounding, increases the total error, but it is still lower than the error when using RN-representation. Regarding floating-point computation, these results are directly applicable to the case of operands with the same exponent, since this study only considers the addition of aligned operands. The unaligned addition will produce an equal or lesser amount of error, since the aligned case produces the maximum one (see Section 5.1). The unbiased rounding would be usually the utilized rounding, since it is the preferred one for floating-point computation.

Regarding multiplication, first, this operation is only defined for positive RN-represented numbers. As a consequence, negative RN-represented numbers must be negated before operating them. In contrast, they may be signed or unsigned for HUB representation. Then, let us consider only unsigned multiplication here. For RN-representation case, this operation requires an n-bit multiplier with two additional rows consolidated into the partial product array and the computation of the round-bit, which means an amount of hardware similar to a regular n+1-bit multiplier. However, for HUB format, it requires an $(n+1)$-bit multiplier with two additional input addends in the compressor tree. Hence, multiplication of HUB numbers requires more hardware than RN-represented ones for the proposed format examples with the same number of bits.

TABLE 3
Statistical parameters of rounding error distribution.

Operation:	**Addition**			
Parameters:	$\min_{(10^{-3})}$	$\text{mean}_{(10^{-5})}$	$\max_{(10^{-3})}$	$\sigma_{(10^{-3})}$
HUBconv	-7.81	-2.31	7.75	3.19
RN-rep.	-15.63	-2.67	15.53	6.36
HUBbiased	-11.72	-393	3.85	3.19
HUBunbiased	-11.72	-3.12	11.66	5.05
Operation	**Multiplication**			
HUB	-3.69	-0.214	3.60	0.839
RN-rep.	7.12	0.724	7.37	1.68



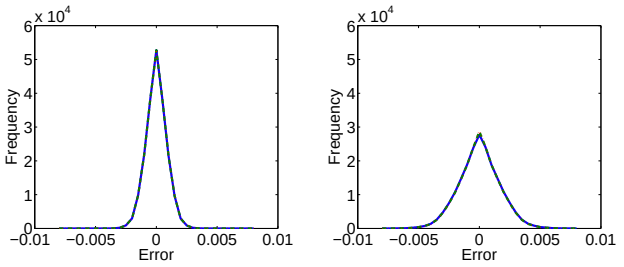
(a) HUBconv (b) RN-representation

(c) HUBbiased (d) HUBunbiased

Fig. 6. Histogram of the rounding errors for addition.

Similarly to addition, we have experimentally measured the correctness of multiplication for both number representations. The only difference compared to the addition experiment process is that the results are 32 bit-width for the exact computation, and 16 bit-width for others. The probability distribution of the error is shown in Fig.7 by the histogram of the rounding error for both HUB format and RN-representation. In this case, the histogram for the error when considering the bounds of the intervals practically coincides with the one corresponding to the values. It is clearly seen that the error for RN-representation doubles the one for HUB format.

Therefore, seen from a different perspective, RN-representation requires one bit more than HUB-format to obtain the same precision. Thus, for the same



(a) HUB format (b) RN-representation

Fig. 7. Histogram of the rounding errors for multiplication.

precision, a HUB number multiplier requires slightly less hardware than an RN-represented multiplier.

Finally, the hardware implementation of unbiased rounding after multiplication or unaligned addition for RN-represented numbers requires the computation of the sticky bit (see Section 5). However, this computation is not required in the case of HUB numbers. This may be an important hardware saving for implementing floating-point adders or multipliers.

Summarizing, HUB formats and canonical RN-representation present the same complexity to perform round-to-nearest and radix complement, i.e., constant time. Moreover, both representation systems are defined for general even radix, for floating- and fixed-point numbers, and, after our extension, for unsigned and signed values, including sign-and-magnitude and two's complement. Despite these similarities, they also have some differences. Perhaps the most important difference is that RN-representation numbers require one bit more than HUB numbers to obtain the same precision. Although conversion from conventional formats is trivial in both cases, it is not exact for HUB formats. However, conversion in the other way is only trivial for HUB formats whereas it is, at least, logarithmic time for RN-representations. Considering the same precision, the modifications required by arithmetic units to process RN-representation operands mean a cost slightly greater than the modifications for HUB operands. On the other hand, HUB numbers could also be operated using regular circuits by extending those with their hidden constant LSB. Taking these characteristic into account, HUB formats are generally more adequate to implement real number computation when it requires frequent round-to-nearest rounding.

7 Conclusion

In this work we have proposed and analyzed a new family of formats for computing with real numbers under round-to-nearest. By shifting the ERNs of the standard real representations, the HUB format allows performing radix complement and round to nearest with negligible cost. HUB formats are defined for unsigned or signed numbers, including two's complement or sign-and-magnitude, and they could be used as significand of floating-point numbers. Arithmetic operations could be performed by trivially converting the HUB operands to a conventional representation, but optimized circuits are obtained by further analysis as it is shown for addition and multiplication. Biased and unbiased rounding has been defined and it has been shown that the computation of the sticky bit is generally not required when operating with HUB numbers.

On the other hand, the binary canonical RN-representation presents some similar characteristics to HUB formats, mainly rounding to nearest by truncation and constant time negation. In this paper we extend the features of the canonical RN-representation to unsigned and sign-and-magnitude numbers, and unbiased rounding. Moreover, a deep qualitative and quantitative analysis of both representations is performed. Targeting systems which compute real numbers using round-to-nearest, we conclude that, for the same precision, our proposal reduces the hardware required, since it utilizes one bit less to store and transmit the numbers, basic arithmetic operations are simpler and does not need to compute the sticky bit.

Summarizing, HUB formats may be well suited for general floating-point applications and application-specific fixed-point data-paths with round-to-nearest. Some suitable applications may be DSP, industrial control or physics simulation, but it is not appropriated for cryptography or other applications in which exact computation is required. A few patent applications have been filed for different issues regarding this representation.

Acknowledgments

This work was supported in part by the Ministry of Education and Science of Spain under contracts TIN2013-42253-P.

References

[1] P. Kornerup and D. W. Matula, *Finite Precision Number Systems and Arithmetic*. Cambridge University Press, 2010.

[2] IEEE Task P754, *IEEE 754-2008, Standard for Floating-Point Arithmetic*, Aug. 2008. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4610933

[3] P. Kornerup and J.-M. Muller, "RN-coding of numbers: Definition and some properties," in *Proc. Intl Meeting on Automated Compliance Systems (IMACS 05)*, Jul 2005.

[4] J.-L. Beuchat and J.-M. Muller, "Multiplication algorithms for radix-2 RN-codings and two's complement numbers," in *Int. Conf. on Application-Specific Systems, Architectures and Processors*, 2005, pp. 303–308.

[5] P. Kornerup, J.-M. Muller, and A. Panhaleux, "Performing arithmetic operations on round-to-nearest representations," *Computers, IEEE Transactions on*, vol. 60, no. 2, pp. 282–291, Feb 2011.

[6] J. Hormigo and J. Villalba, "Optimizing DSP circuits by a new family of arithmetic operators," in *Signals, Systems and Computers, 2014 Asilomar Conference on*, Nov 2014, pp. 871–875.

[7] P. Kornerup, J.-M. Muller, and A. Panhaleux, "Floating-point arithmetic on round-to-nearest representations," *arXiv preprint arXiv:1201.3914*, 2012.
[8] A. Panhaleux, "Contributions to floating-point arithmetic: Coding and correct rounding of algebraic functions," Ph.D. dissertation, Ecole normale supérieure de lyon-ENS LYON, 2012.
[9] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.
[10] S. Boldo and G. Melquiond, "Emulation of a FMA and correctly rounded sums: Proved algorithms using rounding to odd," *Computers, IEEE Transactions on*, vol. 57, no. 4, pp. 462–471, April 2008.

Javier Hormigo received an M.Sc and a Ph.D., both in Telecommunication Engineering, from the Universidad de Malaga, Spain, in 1996 and 2000, respectively. He was a member of the Image and Vision Department of the Instituto de Optica, Madrid, Spain, in 1996. He joined the Universidad de Malaga in 1997 and is currently Associate Professor in the Computer Architecture Department. His research interests include computer arithmetic, specific application architectures, and FPGA.

Julio Villalba received a B.Sc degree in Physics in 1986 (Universidad de Granada, Spain) and a Ph.D in Computer Engineering in 1995 (Universidad de Malaga, Spain). During 1986-1991, he worked in the R&D Department of Fujitsu Spain and was an assistant professor. Since 2007, he has been a Full Professor in the Department of Computer Architecture at the Universidad de Malaga. Currently he is an Associate Editor of IEEE Trans. on Computers. His research interests include computer arithmetic and specific application architectures.