Digit Recurrence Floating-point Division under HUB Format

Abstract—**Half-Unit-Biased format is based on shifting the representation line of the binary numbers by half Unit in the Last Place. The main feature of this format is that the round-to-nearest is carried out by a simple truncation, preventing any carry propagation and saving time and area. Algorithms and architectures have been defined for addition/substraction and multiplication operations under this format. Nevertheless, the division operation has not been confronted yet. In this paper we deal with the floating-point division under HUB format, studying the architecture for the digit recurrence method, including the on-the-fly conversion of the signed digit quotient.**

Keywords—division by digit recurrence, HUB format, on-the-fly conversion

I. Introduction

Rounding-to-nearest is the most extended rounding mode in computer arithmetic for processors, compilers and standards (in fact, it is the default mode of the IEEE754-2008 [1]). The round to nearest mode involves a final addition, which slows down the system (it usually leads the critical path). The implementation of this rounding mode is relatively complex and it involves an increase in both delay and area. Thus, it is usually implemented in floating-point circuits.

Many efforts have been devoted by the researchers to reduce the rounding overhead. Recently some works have been proposed which reduce the hardware requirement and the delay by performing the round-to-nearest by simple truncation. In [2][3][4] RN-Representation is presented and analyzed (the *RN* standing for *round-to-nearest*), which allows performing unbiased round-to-nearest by truncation for the four basic operations. On the other hand, in [5] a new family of formats for computing with real-numbers under round to nearest is presented, which also carry out the round-to-nearest by truncation. This new representation is called Half-Unit Biased (HUB) format. Although some architectures under HUB formats have been presented for addition/substraction and multiplication, to the best of our knowledge, the division operation has not been confronted yet. In this paper, we deal with the division operation under HUB format, in such a way that the four basic operations are covered under this new format.

The efficiency of using HUB formats for fixed-point representation has been demonstrated in [6] and [7] and for floating-point in [8] and [9]. By reducing bit-width while maintaining the same accuracy, the area cost and delay of FIR filter implementations has been dramatically reduced in [6], and similarly for the QR decomposition in [7]. In [9] a half-precision floating-point HUB unit is used for high dynamic range image and video systems (based on additions and multiplications). In [8] the authors analyze the benefits of using HUB format for floating point adders, multipliers and converters from a quantitative point of view. Experimental analysis demonstrate that HUB format maintains the same accuracy as conventional format for the aforementioned units, simultaneously improving area, speed and power consumption (14% speed-up, 38% less area and 26% less power for single precision floating point adder, 17% speed-up, 22% less area and slightly less power for the floating point multiplier) .

On the other hand, in the division by digit recurrence the quotient is represented in a radix–r from and one digit of it is obtained per iteration. Fast implementation are obtained if the residual recurrence is carried out in redundant representation and the final conversion of the quotient from signed-digit representation to conventional representation is carried out on-the-fly.

In this paper we adapt the algorithm of the division by digit recurrence to deal with HUB numbers. We carry out the conversion from redundant representation to conventional representation on-the-fly. In comparison with the counterpart conventional division by digit recurrence with on-the-fly conversion we prove that the number of iterations and delay are kept whereas the hardware requirements are reduced.

The rest of the paper is organized as follows: in section II we present the fundamental of the HUB representation. In section III we show the adaptation of algorithm of the division by digit recurrence to the HUB format. Next we deal with the number of iterations and calculation of bits required for the data path. The on-the-fly conversion and rounding is presented in section IV, including the analysis of the tie case. Finally, in the last section we give the summary and conclusion.

II. HUB format for floating-point

The mathematical fundamentals and a deep analysis of the HUB format as well as the addition and multiplication operations under this format can be found in [5]. In this section we summarize the HUB format defined in [5] and particularize it for the floating-point normalized HUB numbers, which are used for the division algorithm in the next sections.

The HUB format is based on shifting the numbers that can be exactly represented under conventional format by adding a bias. The bias is just half unit-in-the-last-place (ULP). Let X denote a HUB number represented by a digit-vector $X = (X_{n-1}, X_{n-2}, \cdots X_1, X_0, X_{-1}, \cdots, X_{-f})$ in a radix β. The value of this HUB number is

$$X = \left[\sum_{i=-f}^{n-1} X_i \cdot \beta^i\right] + \frac{\beta}{2} \cdot \beta^{-f-1} \qquad (1)$$

where the term $\frac{\beta}{2} \cdot \beta^{-f-1}$ represents the bias.

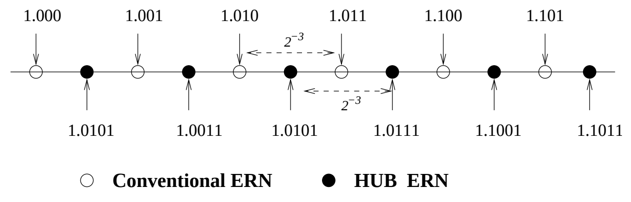

Fig. 1. Example of conventional 4-bit ERNs and their counterpart HUB

A floating-point HUB number is similar to a regular one but its significand follows the HUB format. Thus, the only difference is the format of the significand. In this paper we use floating-point HUBs operands with normalized significand in radix-2. Let us define x as a floating-point HUB number, which is represented by the triple (S_x, M_x, E_x) such that $x = (-1)^{S_x} M_x 2^{E_x}$, where the significand M_x is a HUB magnitude. A normalized HUB significand fulfills that $1 < M_x < 2$. Thus, for radix-2 the digit-vector has the form $M_x = (M_{x_0}, M_{x_{-1}}, M_{x_{-2}}, \cdots, M_{x_{-f}})$, where each digit is now a bit, and so it is composed by $f + 1$ bits. Let *representative form* denote the digit-vector. For a radix–2 HUB significand, expression (1) becomes

$$M_x = \left[\sum_{i=0}^{f} M_{x_i} \cdot 2^{-i}\right] + 2^{-f-1} \tag{2}$$

where 2^{-f-1} is the bias. Thus, although the normalized HUB significand is represented by $f+1$ bits, according to expression (2) the binary version of a normalized HUB significand is composed by $f + 2$ bits:

$$M_x = 1.M_{x_{-1}} M_{x_{-2}} \cdots M_{x_{-f}} 1 \tag{3}$$

The binary version is required to operate with HUB numbers. Thus, let *operational form* denote the binary version. We can see that the least significant bit (LSB) of the operational form of a HUB number is always equal to 1, and it is implicit in the format (similar situation takes place for the most significant bit (MSB) of the significand in the IEEE normalized floating-point numbers). Let ILSB denote the implicit LSB of a HUB number.

Let us call ERN (Exactly Represented Number) to a real number which is exactly represented for a specific floating-point system. Given a standard floating-point system with a normalized significand, its HUB version is obtained by shifting the ERN by half ULP. Figure 1 shows a simple example of ERNs for a conventional floating-point system of 4-bit significand and its HUB operational form. Notice that both formats have the same number of ERNs and the distance between to consecutive ERN is also the same, keeping the same accuracy [5].

Although the operative version of the HUB format requires one bit more than its equivalent conventional one, the ILSB does not have to be stored or transmitted since it is a constant. Thus, both the HUB and the conventional formats are stored with the same number of bits. Only to operate with numbers in HUB format, the ILSB is required explicitly (operational form). For example, the HUB operational form with the same precision as the IEEE-754 simple precision (SP), has 25 bits for the significand, where the first and last bits are implicit and

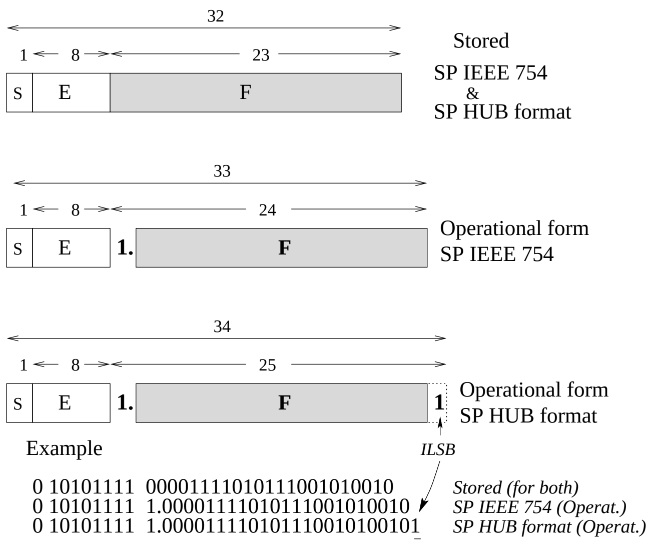

Fig. 2. Single precision (SP) IEEE-754 and its shifted version

only 23 bits are stored, as in the conventional representation. Figure 2 shows the different sizes for storage, the operational form of the conventional SP IEEE-754 and the operational form of HUB, as well as the position of the ILSB.

A. *Round-to-nearest for HUB numbers*

Let us deal with round-to-nearest for HUB format for floating-point numbers. Let m denote the number of bits of the operational form of a normalized HUB number (that is, including the ILSB). Consider a normalized non-HUB number M which is composed by p bits ($M[0 : p-1]$) with $m-1 < p$. We want to round this number to a HUB number. The biased rounded normalized HUB number M' is given by the $m - 1$ MSB of M (representative form):

$$M'[0 : m - 2] = M[0 : m - 2] \tag{4}$$

Thus, the rounded HUB number M' is achieved by simple truncation of the $m-1$ MSB of M. Due to the definition of a HUB number (see equation (2) with $f = m-2$) this truncation produces a round-to-nearest number, as proved in [5].

III. Division by digit recurrence for HUB numbers

In this paper we follow the digit recurrence algorithm that can be found in [10]. We describe the parts of the algorithms needed for the calculations related to HUB numbers, and what is similar to the standard case is not described but referenced (for example, the selection function does not change for HUB numbers and it is not described in this paper).

Let x and d denote the floating-point HUB operands of a division, such as x and d are represented by (S_x, M_x, E_x) and (S_d, M_d, E_d) respectively, with M_x and M_d magnitude and normalized HUB significands. The result

$$q = \frac{x}{q} \tag{5}$$

is a floating-point HUB number represented by (S_q, M_q, E_q), with M_q magnitude and normalized. The operational form of a normalized floating-point HUB number is $1.xxx...xxx1$, with a total of m bits. To carry out the division, the exponents are subtracted, the significands are divided and the resulting quotient is normalized and rounded.

Let us deal with the division of the significands. Since the significant of the operands are HUB normalized numbers ($1 < M_x < 2$ & $1 < M_d < 2$) the resulting significand is in the range $(\frac{1}{2}, 2)$. Consequently, normalization is required when the quotient is less than 1. In this case, a left shift of one position is required as well as a decrement of the exponent. The digit recurrence algorithm for division consists of N iterations of a recurrence, in which each iteration produces one digit of the quotient [10] (the value of N is discussed later). This is preceded by an initialization step and followed by a termination step.

Following we show the digit recurrence step for division for HUB numbers (see [10] for a detailed description for regular numbers). Let $q(i)$ denote the value of the quotient after i iterations:

$$q(i) = q(0) + \sum_{j=1}^{i} q_j r^{-j} \tag{6}$$

where q(0) is calculated in the initiation step, r is the radix of the quotient and q_j is the j-th digit of the quotient. We use a symmetric signed-digit set of consecutive integers for the quotient q such as $q_i \in [-a, a]$, where $a \geq \lceil r/2 \rceil$ to keep a redundant representation. The redundancy factor is defined as

$$\rho = \frac{a}{r-1}, \qquad \frac{1}{2} < \rho \leq 1 \tag{7}$$

The digit recurrence algorithm is based on keeping a partial remainder (residual) inside a convergence bound in each iteration. The residual w is defined as

$$w(i) = r^i(x - dq(i)) \tag{8}$$

The bound to be kept is

$$|w(i)| \leq \rho \cdot d \tag{9}$$

and the final recurrence is

$$w(i+1) = rw(i) - dq_{i+1} \tag{10}$$

The initial value of $w(0)$ is

$$w(0) = x - dq(0) \tag{11}$$

The recurrence is carried out in such a way that $w(i)$ is bounded by equation (9). The value of q_{i+1} is selected according to the quotient-digit selection function, which is obtained as a function of a truncated version of $rw(i)$ and d (see [10] for details). If the final residual is negative, a correction step is required (by substracting one ulp to the quotient).

Figure 3 shows the basic modules and the timing of division by digit recurrence. The residual $w(i)$ can be represented in non redundant (i.e. conventional two's complement) or redundant form (carry-save or signed-digit). Normally a redundant representation is preferred since it results in a faster iteration and that is what we assume in this paper (the substraction $rw(i) - dq_{i+1}$ belongs to the critical path, see timing in figure 3). A sign detection of the last residual is needed to carry out a possible correction, and the zero-remainder condition

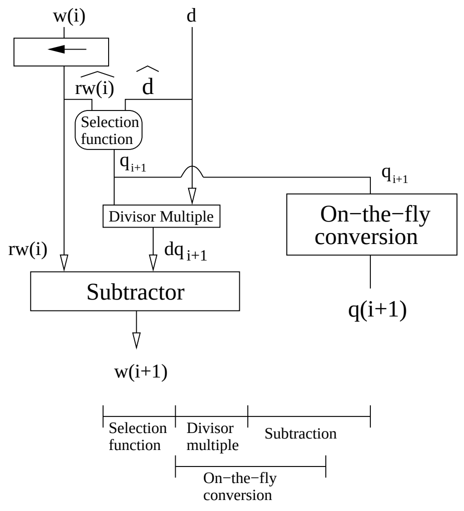

Fig. 3. Basic modules and timing of division by digit recurrence

may be also required. In a redundant representation of the residual (i.e a carry save implementation $w = wc + ws$) both the zero and sign detection of the last residual are difficult since it involves the conversion from redundant to conventional representation. To solve this problem a sign and zero detection lookahead network for the carry-save representation of the residual is proposed in [10] which avoid the slow conversion.

To update the quotient $q(i)$ to $q(i+1)$ we use the on-the-fly conversion proposed in [10], which produces at each recurrence step the corresponding quotient value in a non redundant form, as shown below. The on-the-fly conversion is performed in parallel with the recurrence and it does not belong to the critical path (see figure 3). Before dealing with on-the-fly conversion for HUB numbers, let us see the initialization and termination steps.

A. Initialization steps

The initial value $w(0)$ of the recurrence equation (10) have to fulfill the bound given by equation (9), that is $|w(0)| \leq \rho \cdot d$. Let us study the case of $\rho = 1$ and $(1/2) < \rho < 1$ separately.

- $\rho = 1$. Since d is a HUB normalized number $1 < d < 2$ and then

$$1 < \rho d \tag{12}$$

On the other hand, we select $w(0) = x/2$. Since x is a HUB normalized number $1 < x < 2$, and dividing by 2 we have:

$$\frac{1}{2} < w(0) < 1 \tag{13}$$

From expressions (12) and (13) we conclude that

$$|w(0)| < \rho d \tag{14}$$

- $\frac{1}{2} < \rho < 1$. Since d is a HUB normalized number $1 < d < 2$ and then

$$\frac{1}{2} < \rho d \tag{15}$$

On the other hand, we select $w(0) = x/4$. Since x is a HUB normalized number $1 < x < 2$, and dividing by 4 we have:

$$\frac{1}{4} < w(0) < \frac{1}{2} \tag{16}$$

From expressions (15) and (16) we we conclude that

$$|w(0)| < \rho d \tag{17}$$

As consequence, to initialize w the value of x has to be left shifted either one position ($w(0) = x/2$ if the selected digit set has maximum redundancy) or two positions ($w(0) = x/4$ digit set without maximum redundancy).

B. Termination step

For the termination step, we have to take into account several issues.

- Since there is a initial shift of one or two positions, the quotient has to be shifted correspondingly.
- The recurrence can produce a negative final residue. In this case, the quotient has to be corrected by subtracting one ulp (correction step).
- After the shifting and correction the final quotient is in the range $(\frac{1}{2}, 2)$ so that to obtain a normalized HUB number another left shift is required if the quotient is less that one, and the exponent has to be updated (normalization step).
- Rounding-to-nearest: for HUB it is carried out by simple truncation after normalization. Thus, rounding after normalization never involves an overflow (see section IV).
- If we want to know if the quotient is exact, we have to check the zero condition of the last residual.

C. Number of iterations and data path width

The number of bits to be computed by the iterations (h) is the number of bits of the normalized HUB significand (m) (operational form) plus some extra bits, namely:

- One guard bit due to normalization. Since $q \in (\frac{1}{2}, 2)$, if $q < 1$ a left shift of one position is required for normalization
- One bit if $\rho = 1$ or two bits is $\rho \in (\frac{1}{2}, 1)$. This can be denoted by the function $(1 + \lfloor \rho \rfloor)$.

Thus, the width (number of bits) of the datapath (h) is

$$h = m + 2 + \lfloor \rho \rfloor \tag{18}$$

Notice that no any bit is required for rounding since in HUB number the rounding is carried out by simple truncation. In a conventional floating-point representation with the same precision as its counterpart HUB, the number of bits of the normalized significand is $m - 1$, one bit is needed for normalization (guard bit G), one bit is required for rounding (rounding bit R) and $(1 + \lfloor \rho \rfloor)$ bits for scaling [10]. This results in $h = m + 2 + \lfloor \rho \rfloor$, that is, the same number as its counterpart HUB (see equation (18)). As a conclusion, for the same precision both the HUB and the conventional representation require the same number of bits to be computed by the iterations and the data path for the recurrence has the same width. This is an interesting feature of the HUB division since for HUB multiplication the data path are one bit larger than their conventional representation counterpart [5].

The number of iterations N depends on the number of bits to be computed by the iterations (h) and on the radix as follows:

$$N = \left\lceil \frac{h}{\log_2 r} \right\rceil \tag{19}$$

Since the value of h is the same for both conventional and HUB representations, the number of iterations is the same (which is consistent since both representation have the same precision). As a conclusion, the HUB representation and its conventional counterpart have the same precision, the same number of iterations and the same data path width for the residual recurrence.

IV. On-the-fly conversion and rounding

The on-the-fly conversion prevents the final addition to convert from signed-digit representation to conventional representation. Furthermore, in [10] the rounding is integrated in the on-the-fly conversion of the last digit as well as the correction and normalization, such that no any extra cycles are required to carry out these steps. First we present the on-the-fly algorithm for the conventional format and then we deal with the algorithm for the HUB case.

Consider a normalized conventional significand of m-1 bits. Let Q(i) denote the digit vector of the converted quotient consisting of the i most significant digits

$$Q(i) = \sum_{j=1}^{i} q_i r^{-i} \tag{20}$$

To avoid carry propagation a new form $QD(i)$ (Decremented form) is defined as

$$QD(i) = Q(i) - r^{-i} \tag{21}$$

The on-the-fly conversion algorithm is based on performing concatenations of digits instead of addition of digits, in such a way that carry propagation is prevented. In terms of concatenations the algorithm is

$$Q(i+1) = \begin{cases} (Q(i) \parallel q_{i+1}) & \text{if } q_{i+1} \geq 0 \\ (QD(i) \parallel (r - |q_{i+1}|)) & \text{if } q_{i+1} < 0 \end{cases} \tag{22}$$

$$QD(i+1) = \begin{cases} (Q(i) \parallel q_{i+1} - 1) & \text{if } q_{i+1} > 0 \\ (QD(i) \parallel (r - 1 - |q_{i+1}|)) & \text{if } q_{i+1} \leq 0 \end{cases} \tag{23}$$

where the symbol $\parallel$ means concatenation and $Q(0) = QD(0) = 0$. Basically what this algorithm does is, from certain iteration, to keep the binary value of $QD(i)$ with the decremented binary value of $Q(i)$ ($QD(i) = Q(i) - 1$).

Round-to-nearest involves the addition of one after the regular iterations which can take the last digit out of range of the digit set [-a,a]. A new form $QR(i)$ (Rounding form)[1] is

[1]This form is only required when $a \geq r/2$

defined to combine the correction, normalization and rounding in one cycle, together with the on-the-fly conversion of the last digit:

$$QR(i) = Q(i) + r^{-i} \tag{24}$$

The updating of this expression is done according to the following expression (by concatenation):

$$QR(i+1) = \\ = \begin{cases} (QR(i) \parallel 0) & \text{if } q_{i+1} = r-1 \\ (Q(i) \parallel q_{i+1}+1) & \text{if } -1 \le q_{i+1} \le r-2 \\ (QD(i) \parallel (r+1-|q_{i+1}|)) & \text{if } q_{i+1} < -1 \end{cases} \tag{25}$$

The rounded significand before normalization and truncation is

$$MMq = \begin{cases} (QR(N-1) \parallel u) & \text{if } q_N^* \ge r \\ (Q(N-1) \parallel u) & \text{if } 0 \le q_N^* \le r-1 \\ (QD(N-1) \parallel u) & \text{if } q_N^* < r \end{cases} \tag{26}$$

with $q_N^* \in \{-a, a+1\}$ and $u = q_N^* mod\ r$. The final normalized significand is

$$Mq[0:m-2] = \begin{cases} MMq[0:m-2] & \text{if } MMq[0] = 1 \\ MMq[1:m-1] & \text{if } MMq[0] = 0 \end{cases} \tag{27}$$

In the case of the HUB format, the round to nearest is carried out by simple truncation of the normalized number. This simplify the correction, rounding and normalization steps.

For the on-the-fly conversion for the HUB format, we follow a similar strategy, such that equations (22) and (23) are performed. Nevertheless, equation (25) is not required due to the fact that the round-to-nearest for HUB format (truncation) does not involve a final addition of one ULP, and then, unlike the conventional format, the last digit is never out of digit set range.

After the iterations, we have to select the value of quotient $Q(N)$ if the residual w es positive, or $Q(N)-1$ if the residual is negative (correction step). Let $sign$ denote the sign of the final remainder $w(N)$, such as $sign = 0$ if the remainder is positive, and $sign = 1$ if the remainder is negative. In this last case, the decremented value $Q(N)-1$ can be taken from $QD(N)$ directly since $QD(N) = Q(N)-1$. Thus, the value of the quotient after Correction (QC) is:

$$QC = \begin{cases} Q(N) & \text{if } sign = 0 \\ QD(N) & \text{if } sign = 1 \end{cases} \tag{28}$$

Finally, the normalized and rounded-to-nearest final HUB quotient (q) is given by (representative form)

$$q[0:m-2] = \begin{cases} QC[0:m-2] & \text{if } QC[0] = 1 \\ QC[1:m-1] & \text{if } QC[0] = 0 \end{cases} \tag{29}$$

For the HUB format the round to nearest is carried out by simple truncation, as shown in equation (29). Figure 4 shows a simple architecture to carry out the on-the-fly conversion for HUB quotient. In this figure we can see that the correction step is carried out by an output 2-1 multiplexor, the normalization by a left shift of one position if the MSB is zero and the rounding by simple truncation of the $m-1$ MSBs.

The architecture proposed in figure 4 carries out the correction, normalization and rounding after the last digit has been converted (Q(N)), whereas for the conventional representation

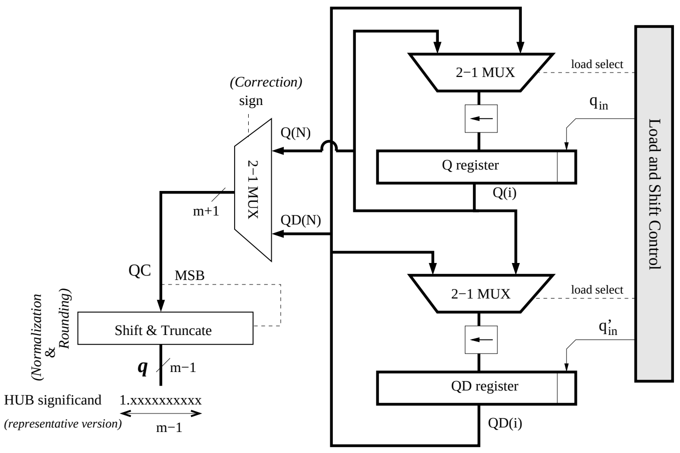

Fig. 4. A possible architecture for the on-the-fly conversion for HUB

the rounded significand before normalization and truncation is obtained from $Q(N-1), QD(N-1), QR(N-1)$ (see equations (26)). Following we prove that this strategy is also possible for the HUB case, and show that an important reduction in the hardware requirements is achieved.

The rounded significand before normalization and truncation for conventional representation (MMq, see equation (26)) is determined from the already converted parts $Q(N-1), QD(N-1), QR(N-1)$ and the incoming digit q_N^*, and the digit q_N^* depends on the sign of the remainder ($sign$) [10][2]. Similarly, for the HUB format we have to determine the quotient after correction QC from the already converted parts $Q(N-1), QD(N-1)$, the sign of the remainder ($sign$) and the digit q_N. Let u denote the LSD. For HUB, in the last cycle, according to equations (22), $Q(N)$ is

$$Q(N) = \begin{cases} (Q(N-1) \parallel q_N) & \text{if } q_N \ge 0 \\ (QD(N-1) \parallel (r-|q_{i+1}|)) & \text{if } q_N < 0 \end{cases} \tag{30}$$

Now, consider the correction step. If the remainder is positive ($sign = 0$), the quotient after correction is given by (30) directly $QC = Q(N)$. If the remainder is negative ($sign = 1$), then we have to subtract one to $Q(N)$, that is we select the decremented form $QC = QD(N)$, as shown in expression (28). Let analyze the value of $QD(N)$ by taking $i = N-1$ in expression (23)

$$QD(N) = \begin{cases} (Q(N-1) \parallel q_N - 1) & \text{if } q_N > 0 \\ (QD(N-1) \parallel (r-1-|q_N|)) & \text{if } q_N \le 0 \end{cases} \tag{31}$$

If we analyze equations (28), (30) and (31) and we can rewrite expression (28) as:

$$QC = \begin{cases} (Q(N-1) \parallel q_N - sign) & \text{if } q_N \ge 0 \\ (QD(N-1) \parallel r - |q_N| - sign) & \text{if } q_N < 0 \end{cases} \tag{32}$$

For a better comparison with the conventional format (expression (26)) let us rewrite expression (32) as

$$QC = \begin{cases} (Q(N-1) \parallel u & \text{if } q_N \ge 0 \\ (QD(N-1) \parallel u & \text{if } q_N < 0 \end{cases} \tag{33}$$

[2]In [10] it is assumed that the sign of $w(N)$ is anticipated in the last cycle in such a way that it can be used for the calculation of the LSD in the same cycle. Since the residue recurrence is the same for HUB, we keep this assumption in this paper

where $u = q_N - sign$ if $q_N \geq 0$ and $u = r - |q_N| - sign$ if $q_N < 0$.

Expression (33) shows that it is possible to obtain the quotient after correction QC in the cycle of the conversion of the last digit, as happens with the conventional format by expression (26). Moreover, the calculation of the incoming digit u is simpler for HUB format.

Figure 5.a shows the architecture required for the on-the-fly conversion for HUB format and figure 5.b shows the architecture for the conventional format, where the differences between both architectures have been highlighted in grey color. For both the HUB and the conventional formats equations (22) and (23) are carried out by two 2-1 multiplexors, two shifters and registers Q and QD and the corresponding load and shift control. In addition to these equations, the conventional format requires the implementation of equation (25), which involves one extra 3-1 multiplexor, a shifter, the register QR and a more complex control of the incoming concatenated digit (comparison of q_{i+1} with $r-1, r-2$ and -1), as shown in figure 5.b. On the other hand in the conversion of the last digit, the incoming least significant digit (LSD) u is obtained as a function of $sign$ and q_N for HUB whereas it depends on $sing$, q_N and the MSB of $Q(N-1)$ and $QD(N-1)$ for the conventional representation (see the bottom of Figure 5). Therefore, we conclude that there is an important reduction in the hardware of the on-the-fly conversion for the HUB representation.

A. *The tie case*

In floating-point division for IEEE standard representation and round to nearest mode, the tie case never happens, as shown in [10]. Nevertheless, for HUB numbers the tie case occurs if the remainder is zero and the LSB of the quotient is also 0 (before rounding). If this situation takes place, the final quotient is just in the middle of two ERNs.

In a biased implementation the tie case is always rounded up (the ILSB is always one). To obtain an unbiased implementation the rounding has to be random for the tie case. Let KL and $L'K'$ denote the two LSBs of the normalized quotient (operational form) before rounding and after rounding respectively, and let z denote the zero-remainder condition ($z = 1$ means $w(N) = 0$). We propose the next algorithm:

```
z  KL --> K'L'
0  xy     x 1 (not tie)
1  00     0 1 (tie, rounding up)
1  10     0 1 (tie, rounding down)
```

In the proposed algorithm, for the tie case, if KL=00, the resulting rounded value is K'L'=01 that is, rounded up, whereas if KL=10 the resulting rounding is K'L'=01 that is, rounding down. Since the value of K is random, the up or down rounding is also random (note that L' is the ILSB and its value is always 1 for a HUB number). The function for K' is

$$K' = K \vee (z \wedge \overline{L}) \tag{34}$$

where the symbols $\vee$ and $\wedge$ mean the logic operations OR and AND respectively. As a conclusion, an unbiased implementation is possible by a simple logic operation on the bit K.

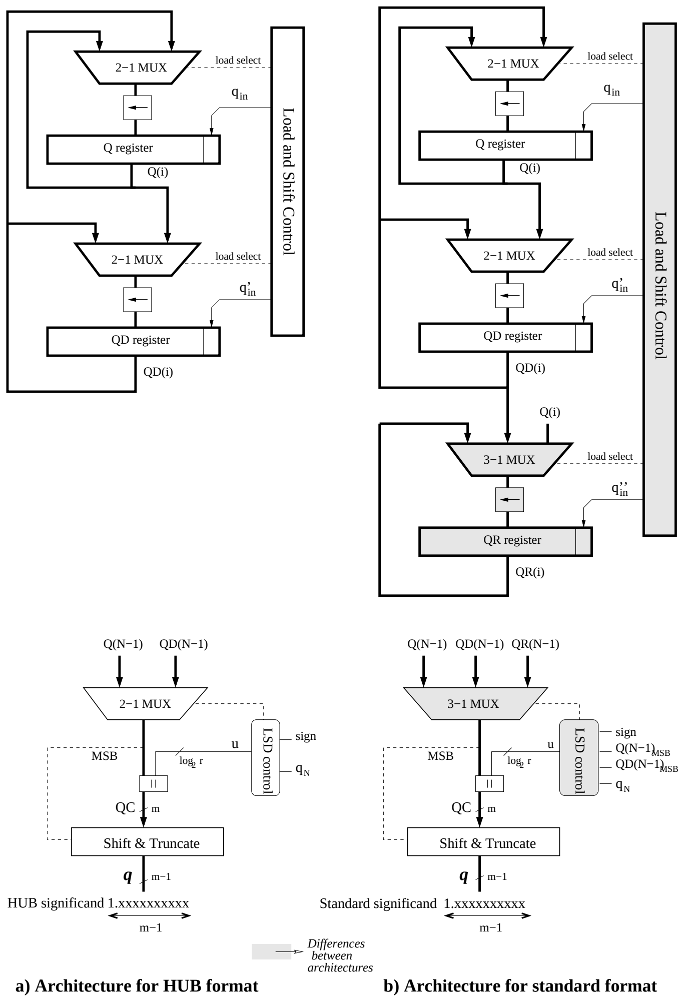

Fig. 5. On-the-fly conversion for HUB (a) and for conventional (b)

V. Summary and Conclusion

In this paper we have presented the division of two HUB numbers using the digit-recurrence algorithm and on-the-fly conversion of the quotient with rounding to nearest. We have proved that, for the same precision, the HUB representation and its conventional counterpart have the same number of iterations and the same data path width for the residue recurrence. Thus, the conventional architecture for the residue recurrence can be used without modification. For the on-the-fly conversion we have designed an architecture that reduces the hardware requirements since one of the three ways used in conventional on-the-fly converters is not required, as well as the logic for the last digit is simplified (see figure 5 to see the differences). This simplification of the hardware is due to the fact that the HUB numbers are rounded to nearest by simple truncation. Finally, the unbiased rounding is also proposed, which only involves a simple logic operation on the LSB (no carry propagation).

References

[1] "IEEE standard for binary floating-point arithmetic," *ANSI/IEEE Std 754-1985*, 1985.

[2] 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.

[3] 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.

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

[5] J. Hormigo and J. Villalba, "New formats for computing with real-numbers under round-to-nearest," *Computers, IEEE Transactions on*, vol. PP, no. 99, 2015.

[6] J. Hormigo and J. Villalba-Moreno, "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] S. D. Muñoz and J. Hormigo, "Improving fixed-point implementation of QR decomposition by rounding-to-nearest," in *Consumer Electronics (ISCE 2015), 19th IEEE International Symposium on*, June 2015, pp. 1–2.

[8] "Measuring the improvement when using HUB formats to implement floating-point systems under round-to-nearest," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pp. –, 2015, in press.

[9] J. Hormigo and J. Villalba, "Simplified floating-point units for high dynamic range image and video systems," in *Consumer Electronics (ISCE 2015), 19th IEEE International Symposium on*, June 2015, pp. 1–2.

[10] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, San Francisco, 2004.