Floating Point Square Root under HUB Format

Julio Villalba-Moreno
Dept. of Computer Architecture
University of Malaga
Malaga, SPAIN
jvillalba@uma.es

Javier Hormigo
Dept. of Computer Architecture
University of Malaga
Malaga, SPAIN
fjhormigo@uma.es

***Abstract*—Unit-Biased (HUB) is an emerging format based on shifting the representation line of the binary numbers by half unit in the last place. The HUB format is specially relevant for computers where rounding to nearest is required because it is performed simply by truncation. From a hardware point of view, the circuits implementing this representation save both area and time since rounding does not involve any carry propagation. Designs to perform the four basic operations have been proposed under HUB format recently. Nevertheless, the square root operation has not been confronted yet. In this paper we present an architecture to carry out the square root operation under HUB format for floating point numbers. The results of this work keep supporting the fact that the HUB representation involves simpler hardware than its conventional counterpart for computers requiring round-to-nearest mode.**

***Index Terms*—Digit recurrence square root, HUB format, on-the-fly conversion**

I. Introduction

Half-Unit-Biased (HUB) is a new format based on shifting the representation line of the binary numbers by half unit in the last place [1]. This emerging representation meets important features namely i) the round to nearest is achieved by truncation, ii) bit-wise two's complement of numbers and iii) no double rounding error. It is specially relevant in computer where round to nearest is required. The consequence of these features is that the complexity of the underlined hardware of HUB systems is less than that of its conventional counterpart.

In modern computer processors, compilers and standards, rounding-to nearest is the most extended rounding mode (in fact, it is the default mode of the IEEE754-2008 [2]). Round to nearest involves a final addition of one unit-in-the-last-place (ULP), which slows down the system (it often belongs the critical path). The implementation of this kind of rounding is relatively complex and it involves an increase in both area and delay. Therefore, it is normally found in floating-point circuits. HUB representation does not have this problem since rounding to nearest is carried out by truncation.

The efficiency of using HUB for fixed-point representation has been showed in [3] and [4] and for floating-point in [5], [6] and [7] (HUB format is not valid for pure integer numbers). In [6] a half-precision floating-point HUB unit is used for high dynamic range image and video systems (based on additions and multiplications). By reducing bit-width while maintaining the same accuracy, the area cost and delay of FIR filter implementations has been drastically reduced in [3], and similarly for the QR decomposition in [4].

In [5] the authors carry out an analysis of using HUB format for floating point adders, multipliers and converters from a quantitative point of view. Experimental studies demonstrate that HUB format keeps 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). For division, the delay is exactly the same as its conventional counterpart with a moderate reduction in the area [7]. Nevertheless, to the best of our knowledge, the square root operation has not been confronted yet.

In this paper, we deal with the square root operation for floating point representation under HUB format. We adapt the algorithm of the square root by digit recurrence of [8] to HUB floating point numbers. A similar solution is found for HUB floating point division in [7], and some ideas of that work are used in this paper. In comparison with the counterpart conventional square root by digit recurrence with on-the-fly conversion we prove that the number of iterations and delay are kept whereas the hardware requirements are moderately reduced. Our results for the square root confirm the previously observed fact that the underlined hardware of the HUB floating point alternative is simpler than its conventional counterpart with no time penalty.

Thus, we consider that this emerging format may play an important role the design of the new generations of computers requiring round to nearest.

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 square root by digit recurrence to the HUB format. Next we deal with the number of iterations and calculation of bits required for the data path. A comparison with the standard representation is presented in section IV. Finally, in the last section we give the summary and conclusion.

II. HUB format for floating-point

In [1], the authors present the mathematical fundamentals and a deep analysis of the HUB format. In this section we

This work has been supported by the Ministry of Science and Innovation of Spain under project CICYT TIN2016-80920R

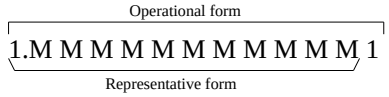


Fig. 1. Representative versus operational forms

summarize the HUB format defined in [1] and particularize it for the floating-point normalized HUB numbers, which are used for the square root in the next sections.

The HUB format is based on shifting to the right the numbers that can be exactly represented under conventional format by adding a bias. The bias is just half 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

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

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

A floating-point HUB number is similar to a regular one but its significand follows the HUB format. Therefore, the only difference is the format of the significand. In this paper we deal with floating-point HUBs operands with a 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) in such a way 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

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

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:

$$M_x = 1.M_{x_{-1}} M_{x_{-2}} \cdots M_{x_{-f}} 1 \quad (3)$$

The binary version is required to operate with HUB numbers. Thus, let *operational form* denote the binary version. Figure 1 shows both the representative and the operational forms. We can see that the least significant bit (LSB) of the operational form of a nonzero 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 ERN (Exactly Represented Number) denote a real number which is exactly represented for a specific floating-point system, and let denote STEP the distance between two

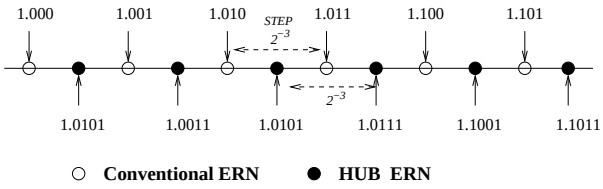


Fig. 2. ERNs for the conventional and the HUB systems, STEP=2^{-3}

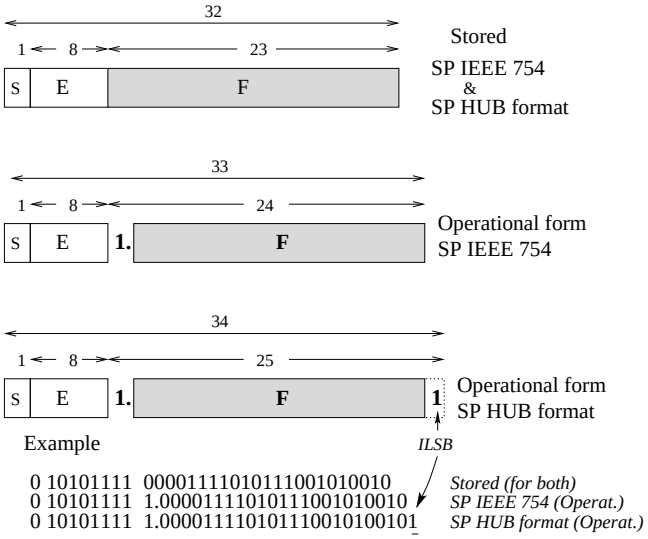


Fig. 3. Single precision (SP) IEEE-754 and its counterpart HUB

consecutive ERNs. For a standard floating-point system with a normalized significand, the counterpart HUB floating point system with the same STEP has their ERNs placed between two consecutive ERN of the standard system (and vice versa). This is an important feature that can be seen in Figure 2. In this figure we can see the position of consecutive ERNs for both representations. The position of each ERN of the standard is represented by a white circle whereas the counterpart HUB is represented in black circle, with a common STEP of 2^{-3} (which involves 4-bit significand for the standard and 5 bits for the counterpart HUB for this example). Notice that both formats have the same number of ERNs, there are not any common ERN among them and the distance between to consecutive ERN is the same (that is, the STEP), keeping the same accuracy [1].

In spite of the operative form of the HUB requires one bit more than its conventional counterpart, the ILSB does not have to be stored or transmitted because it is a constant. Thus, both formats require the same storage space. The ILSB is required explicitly (operational form) only to operate under HUB format. 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 only 23 bits are stored, as in the conventional representation. Figure 3 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 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 ($1 < M_x < 2$) which is composed by p bits ($M[0:p-1]$) with $p > m-1$. We want to round this number to a HUB number of $m-1$ bits (representative form). The rounded normalized HUB number M' is given by the $m-1$ MSB of M (representative form):

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

This equation means that the rounded HUB number is achieved by truncation of the $m-1$ MSB of M. This truncation produces a round to nearest number due to the definition of a HUB number (see equation (2) with $f = m-2$), as proved in [1].

When a number is just between two ERNs (tie condition), the proposed rounding is always performed in the same direction (toward up), which produces a bias. In some applications, this causes annoying statistical anomalies. To avoid this problem in those applications, an unbiased round to nearest is achieved in [5] for HUB without carry propagation. The tie condition takes place when the bits beyond bit $M[m-2]$ are all 0, that is $M[m-1:p-1] = 0...0$. In this case, the unbiased rounded normalized HUB number M' is (representative form):

$$M'[0:m-2] = M[0:m-3,0] \tag{5}$$

Thus, the unbiased rounding is achieved by truncation and forcing the LSB of the representative form to zero, that is the bit $M'[m-2] = 0$, as proved in [5]. Thus, this unbiased round to nearest is not so different from the IEEE 745 tie to even, since it is like a tie to even of the representative form. The main difference is that the HUB format does not involve any carry propagation.

III. Square root by digit recurrence for HUB numbers

In this work we follow the digit recurrence algorithm of the square root proposed in [8] and [9]. We develop the parts of the algorithm required for the computations related to HUB numbers, and what is similar to the conventional one is not detailed but referenced (for example, the computation of the selection function does not change for HUB and then it is not shown in this paper). At this point, it is important to note that the operational form of a HUB significand is treated as a conventional number, that is, the role of the ILSB is similar to the rest of the bits. The only difference from the counterpart IEEE representation is that the HUB significand has one extra bit. Thus, we operate in conventional arithmetic and return to HUB format for rounding.

Let x denote a floating-point HUB number such that x is represented by (S_x, M_x, E_x) with M_x magnitude and normalized HUB significand. The result of the square root operation

$$s = \sqrt{x} \tag{6}$$

is a floating-point HUB number represented by (S_s, M_s, E_s), with M_s magnitude and normalized. The exponent of the result E_s is obtained by division by two ($E_s = E_x/2$, using a simple arithmetic right shift of one position) and the significant of the result M_s is the square root of the original significant $M_s = \sqrt{M_x}$. Nevertheless, the actual algorithm involves some scaling in the initial input data x to make easy the computation of the final exponent and to take the initial value into the convergence range of the algorithm.

The algorithm is made of three main steps: initialization, recurrence and termination. In the initialization step the input HUB number is suitably scaled to the range that support the recurrence (second step). In the second step N iterations of a recurrence are carried out, in which each iteration produces one digit of the result [9] (the value of N is discussed later in section III-E), and finally in the termination step a new scaling, normalization and rounding are performed.

Let us start by the main step: the recurrence

A. The recurrence

Let us deal with the square root of the significand $\sqrt{M_x}$. In what follows we show the digit recurrence step for square root for HUB numbers (see [8] for a detailed description for regular numbers). We assume that the initial significant M_x is suitably scaled such that new value M'_x belongs to the convergence range of the algorithm: $\frac{1}{4} \leq M'_x < 1$ (an initial scaling is performed in the initialization step as shown later in subsection III-B).

Let $S(i)$ denote the value of the result after i iterations:

$$S(i) = \sum_{j=0}^{i} s_j r^{-j} \tag{7}$$

where s_0 is calculated in the initiation step, r is the radix of the result and s_j is the j-th digit of the result. We use a symmetric signed-digit set of consecutive integers for the result s such as $s_i \in [-a, a]$, where $a \geq \lceil r/2 \rceil$ to keep a redundant representation. The redundancy factor is

$$\rho = \frac{a}{r-1}, \qquad \frac{1}{2} < \rho \leq 1 \tag{8}$$

The digit s_0 should be 1 for $\rho < 1$ to represent a value greater than ρ, and it can be 1 or 0 for $\rho = 1$.

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

$$w(i) = r^i(x - S(i)^2) \tag{9}$$

The bound to be kept is

$$-2\rho S(i) + \rho^2 r^{-i} < w(i) < 2\rho S(i) + \rho^2 r^{-i} \tag{10}$$

and the initial condition is

$$w(0) = M'_x - S(0)^2 \tag{11}$$

The final recurrence is

$$w(i+1) = rw(i) - 2S(i)s_{i+1} - s_{i+1}^2 r^{-(i+1)} \tag{12}$$

For clarity in the exposition let define the function

$$F(i) = -2S(i)s_{i+1} - s_{i+1}^2 r^{-(i+1)} \tag{13}$$

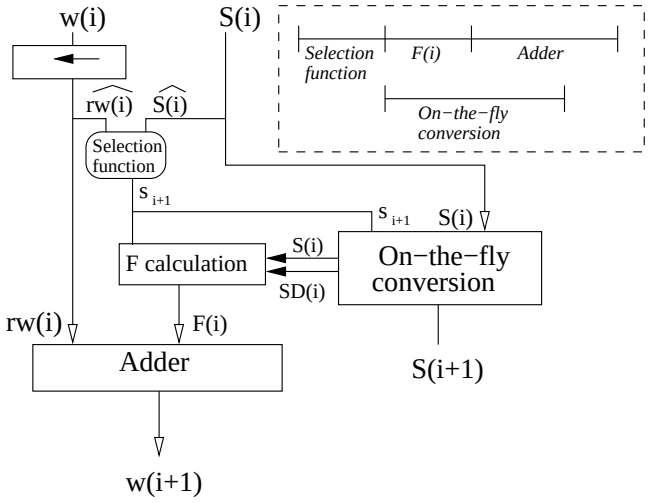


Fig. 4. Basic modules and timing of square root by digit recurrence

such that the recurrence (12) becomes

$$w(i+1) = rw(i) + F(i) \tag{14}$$

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

Figure 4 shows the basic modules and the timing of square root 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 addition of $rw(i)$ and $F(i)$ belongs to the critical path as well as the computation of $F(i)$, see timing in figure 4).

In order to carry out a possible final correction a sign detection of the last residual is needed. For a residual in redundant representation (i.e a carry-save implementation $w = wc + ws$) the sign detection of the last residual is difficult since it involves the conversion from redundant to conventional representation. To solve this problem a sign detection lookahead network for the carry-save representation of the residual is proposed in [9] which avoids the slow conversion.

The adder proposed in figure 4 works in redundant arithmetic to speedup the computation. We propose to use the carry-save representation such that the residual $w(i)$ is composed by a sum and carry words ($w(i) = ws(i) + wc(i)$), and the adder of figure 4 is a 3-2 CSA. The right input of the adder of figure 4 is $F(i)$, which is a function of $S(i)$ (see Eq. (13)). The problem is that $S(i)$ is a signed digit number which is not compatible with any input of a 3-2 CSA. In order to have a compatible value at the right input of the adder, $S(i)$ has to be converted from signed-digit representation to two's complement form by using an on-the-fly conversion technique. On the other hand, the final result $S(N)$ has to be converted from redundant to conventional representation too. Both processes (generation of a two's complement form of $F(i)$ and on-the-fly conversion of the result) are related and work in parallel in boxes "F calculation" and "On-the-fly conversion" of figure 4. Now we describe both processes.

1) On-the-fly conversion of the result: Consider a new form $SD(i)$ (Decremented form) which is defined as

$$SD(i) = S(i) - r^{-i} \tag{15}$$

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

$$S(i+1) = \begin{cases} (S(i) \parallel s_{i+1}) & \text{if } s_{i+1} \geq 0 \\ (SD(i) \parallel (r - |s_{i+1}|)) & \text{if } s_{i+1} < 0 \end{cases} \tag{16}$$

$$SD(i+1) = \begin{cases} (S(i) \parallel s_{i+1} - 1) & \text{if } s_{i+1} > 0 \\ (SD(i) \parallel (r - 1 - |s_{i+1}|)) & \text{if } s_{i+1} \leq 0 \end{cases} \tag{17}$$

where the symbol $\parallel$ means concatenation and $S(0) = SD(0) = 0$. Basically this algorithm keeps a copy of $S(i)$ and a its decremented value $SD(i)$, which are suitably updated in every iteration.

At this point is important to note that the square root algorithm proposed in [8] includes a third equation similar to equation (15) (with an addition of r^{-i} instead of a substraction and with the associated updating equations similar to equations (16)) which is needed to perform the final rounding on-the-fly (see section IV). Rounding involves a final addition of one ULP in a standard representation system. Nevertheless, in HUB this addition is not required any more since rounding is carried out simply by truncation. Thus, the HUB implementation of the square root save area since the third equation proposed in [8] is not required here.

Figure 5 shows a simple architecture to carry out the on-the-fly conversion for HUB result as well as the termination step and $F(i)$ generation. The network of shifers and multiplexors and the corresponding control unit of figure 5 make possible the updating of $S(i)$ and $SD(i)$.

2) Computation of F(i): Consider $s_{i+1} > 0$ in expression (13). Operating with expression (13), we have

$$F(i) = -(2S(i) + s_{i+1} r^{-(i+1)}) s_{i+1} \tag{18}$$

If we analyze this expression we can see that the addition of $2S(i)$ and $s_{i+1} r^{-(i+1)}$ of expression (18) can be performed by simple concatenation [8]. Thus, in terms of concatenations, expression (13) for $s_{i+1} > 0$ becomes

$$F(i) = -(2S(i) \parallel s_{i+1}) s_{i+1} \tag{19}$$

Now consider $s_{i+1} < 0$ in expression (13). Equation 13 becomes

$$F(i) = 2S(i)|s_{i+1}| - s_{i+1}^2 r^{-(i+1)} \tag{20}$$

If we take into account expression (15) in equation (20) we have

$$F(i) = (2SD(i) + (2r - |s_{i+1}|) r^{-(i+1)}) |s_{i+1}| \tag{21}$$

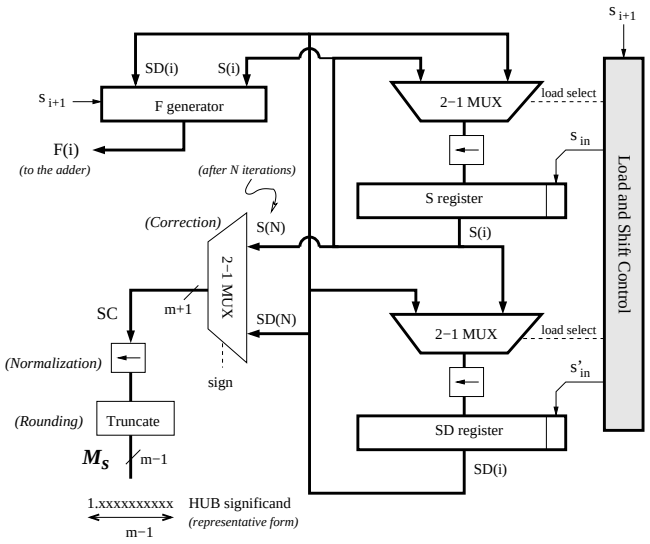


Fig. 5. Architecture for the on-the-fly conversion of the result and for F(i)

For similar reasons this expression really involves a concatenation. Thus, in terms of concatenation, expression (13) for $s_{i+1} < 0$ becomes

$$F(i) = (2SD(i) \parallel (2r - |s_{i+1}|))|s_{i+1}| \tag{22}$$

Thus, these expressions can be implemented by concatenation and multiplication by one radix-r digit. Figure 5 shows the architecture to obtain $S(i)$ and $SD(i)$ which are used to generate the value of F(i) in two's complement representation ($F(i)$ is the right input of the adder of the general architecture of figure 4).

B. Initialization step

On the one hand, the algorithm converges if the input data belongs to the interval $[\frac{1}{4}, 1)$, whereas our significand $M_x \in (1, 2)$. On the other hand, to compute the square root in an easy way the exponent has to be an even number. To meet these two conditions it is needed to scale the significand and peform the suitable adjust of the initial exponent E_x. If $(0, M_x, E_x)$ is the initial HUB floating point number, we transform this number to a new one $(0, M'_x, E'_x)$ such that $(0, M_x, E_x) = (0, M'_x, E'_x)$ with $\frac{1}{4} < M'_x < 1$ and E'_x even as follow:

$$M'_x = \begin{cases} \frac{1}{4}M_x & \text{if } LSB(E_x) = 0 \quad (E_x \quad even) \\ \frac{1}{2}M_x & \text{if } LSB(E_x) = 1 \quad (E_x \quad odd) \end{cases} \tag{23}$$

and the exponent E'_x is

$$E'_x = \begin{cases} E_x + 2 & \text{if } LSB(E_x) = 0 \quad (E_x \quad even) \\ E_x + 1 & \text{if } LSB(E_x) = 1 \quad (E_x \quad odd) \end{cases} \tag{24}$$

We can see that E'_x is even and $\frac{1}{4} < M'_x < 1$. For example, if the HUB number is given by the triple $(0, 1.110..., 0010) \Rightarrow x = 1.11... \times 2^2$ we transform this triple to the new triple $(0, 0.01110..., 0100) \Rightarrow x = 0.0111... \times 2^4$.

The initial value $w(0)$ of the recurrence equation (12) is given by equation (11). Taken into account expression (7) we have that $S(0) = s_0$ and then

$$w(0) = M'_x - s_0^2 \tag{25}$$

The value of s_0 is selected as function of ρ such that $s_0 = 1$ for $\rho < 1$ to represent a result value greater than ρ; it can be either 0 or 1 for $\rho = 1$ (maximum redundancy). Thus, if we select $s_0 = 0$ or $s_0 = 1$ expression (25) becomes

$$w(0) = M'_x - s_0 \tag{26}$$

As consequence, to initialize w the value of s_0 is subtracted from the scaled version of the HUB significad M'_x.

C. Termination step

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

- Correction: the recurrence can produce a negative final residue. In this case, the result has to be corrected by subtracting one unit in the LSB of the just calculated result. To do that, we select the value of result $S(N)$ if the residual w is non negative, or $S(N) - 1$ if the residual is negative. Let $sign$ denote the sign of the final remainder $w(N)$, such as $sign = 0$ if the remainder is non negative, and $sign = 1$ if the remainder is negative. In this last case, the decremented value $S(N) - 1$ can be taken from $SD(N)$ directly. Thus, the value of the result after Correction (SC) is:

$$SC = \begin{cases} S(N) & \text{if } sign = 0 \\ SD(N) & \text{if } sign = 1 \end{cases} \tag{27}$$

- Normalization: after the correction, the final result is in the range $(\frac{1}{2}, 1)$ since the initial significand is in the range $(\frac{1}{4}, 1)$. Thus, to obtain a normalized HUB number (range $(1, 2)$) a left shift of one position is required, and the exponent has to be conveniently updated . The normalized HUB result $s = (S_s, M_s, E_s)$ is (representative form):

$$\begin{aligned} M_s[0 : m-2] &= SC[1 : m-1] \\ E_s &= \tfrac{E'_x}{2} - 1 \\ S_s &= 0 \end{aligned} \tag{28}$$

  Note that the final exponent E_s is carried out in parallel with the recurrence and thus, its calculation does not belong to the critical path. Figure 6 shows a possible circuit for for obtaining the final exponent E_s as well as the scaled initial significand M'_x. Since the exponent is normally an integer of few number of bits (8 for simple precision, 11 for double), the calculation of the new exponent can be also realized by a simple lookup table.
- Rounding-to nearest: for HUB it is carried out by truncation after normalization. Thus, rounding never involves an overflow. Consequently, no any modification of the result of the equation (28) is needed for rounding.

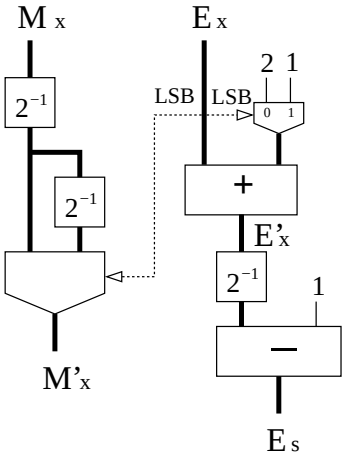


Fig. 6. A possible circuit for obtaining the final exponent and M'_x

An important advantage of the HUB alternative in comparison with the square root of the standard representation [8] is that the computation of the exponent can be performed at the beginning of the full process and no any exponent update is required after rounding because an overflow of the significand is never produced. In a regular representation we can not know the final exponent of the square root until final rounding whereas in a HUB representation we know the final exponent from the beginning. Thus, in comparison with the HUB circuit of figure 6, the circuit required for the exponent in the standard representation involves an extra circuit for updating the exponent in case of overflow.

- If we want to know if the result is exact, we have to check the zero condition of the last residual.

Therefore, equations (28) represent the final normalized and rounded to nearest result. Figure 5 includes the hardware required for the termination step: the correction step is carried out by the vertical output 2-1 multiplexer, the normalization by the shifter placed after the multiplexor, and the rounding by direct truncation of the $m-1$ MSBs of the output of the shifter.

D. Number of bits of the data path

The number of bits to be computed by the iterations (h) is the number of bits of the normalized HUB significand (m) (operational form, that is including the ILSB) plus the guard bits required for initialization and normalization. When the exponent is even, the initial significand M_x is scaled by a factor $1/4$ (see expression (23)) and thus, two guard bits are required. On the other hand, since the final result after recurrence $SC \in (\frac{1}{2}, 1)$ (that is, SC has the pattern 0.1xxx..., see expression (28)), then no extra bits are required and a left shift of one position is enough for normalization. Thus, the width (number of bits) of the datapath (h) is

$$h = m + 2 \tag{29}$$

Note that the rounding bit (R) (required in the conventional architecture) is no longer necessary under HUB format since in HUB representation the rounding is carried out by truncation. In the standard representation the number of bit required is given by the size of the input data (m-1) (that is one less bit than HUB) plus two bits for the initial scaling (similar to HUB), plus one bit for rounding (R bit), which produces $m+2$ bits. Thus, the same number of bits of the data path is requited for obtaining the square root of a number when a rounded to nearest result is required. This is an important conclusion since the role of the R bit in the standard representation is now played by the ILSB of the HUB representation.

Figure 7 shows an example for HUB and for its IEEE counterpart, where the value of $w(0)$ is obtained from $M'x$ ($w(0) = M'_x/4$, assuming $s_0 = 0$). The size of the data path is $m + 2$ for both representations.

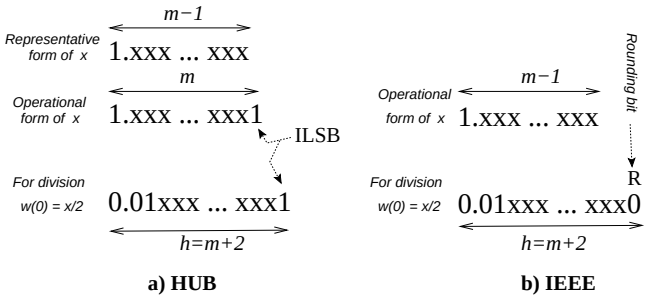


Fig. 7. Size of the data path (h) for HUB and IEEE for the same precision

On the other hand, $w(0)$ is not a HUB number but a regular one, as deduced from figure 7.a, and the rest of the iterations work in a regular way. The intermediate operations (obtaining $w(i)$) work as regular computation, and once the final result is obtained (after N iterations), the corresponding rounded-to nearest HUB number is achieved by truncation (similar situation takes place for partial products in HUB multiplication, see [1] or for HUB division in [7]). Figure 8 shows the sequence of $w(i)$ and s_i (which are not HUB numbers) and how the rounded-to nearest HUB numbers are obtained just from the final result by truncation.

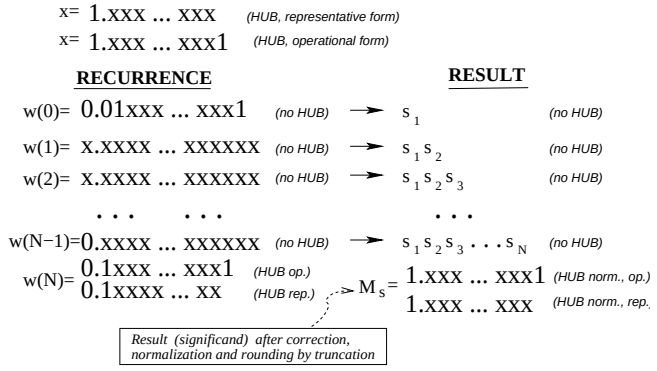


Fig. 8. Recurrence and result ($\rho = 1$)

Therefore, both formats involve the same number of bits for an actual implementation. Basically, the extra bit of the significand of the HUB format (ILSB) is compensated by the lack of the R bit for this format, as shown in figure 7.

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 conclusion is the same as that obtained for HUB floating point division in [7].

E. Number of iterations

The number of iterations N depends on the number of bits to be computed by the iterations (h) and on the radix as follows [9]:

$$N = \left\lceil \frac{h}{\log_2 r} \right\rceil \tag{30}$$

Since the value of h is the same for both the conventional and the counterpart 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 number of iterations and the same data path width for the residual recurrence.

IV. Differences between HUB and conventional architectures

Both the conventional and the HUB architectures have the same number of iterations and the same size of the data path of the recurrence. This means that in the architecture of figure 4 the shifter, selection function module, F calculation module and adder are totally similar, including the timing. The differences are due to two elements: the module "On-the-fly conversion" of figure 4 and the hardware required for the computation of the exponent.

On the one hand, the on-the-fly conversion for HUB involves equations (16) and (17). Apart from equations (16) and (17), the on-the-fly conversion in the standard representation involves a third group of equations like following [9]:

$$QR(i) = S(i) + r^{-i} \tag{31}$$

The updating of this expression is (by concatenation):

$$QR(i+1) = \begin{cases} (QR(i) \parallel 0) & \text{if } s_{i+1} = r-1 \\ (S(i) \parallel s_{i+1}+1) & \text{if } -1 \le s_{i+1} \le r-2 \\ (SD(i) \parallel (r+1-|s_{i+1}|)) & \text{if } s_{i+1} < -1 \end{cases} \tag{32}$$

These equations are required in the standard representation since round-to-nearest requires the addition of one ULP after the regular iterations which can take the last digit out of the range of the selected digit set [-a,a]. In HUB, these equations are not needed since rounding to nearest does not involve the addition of one; it is achieved simply by truncation and the the last digit always belongs to the digit set.

Figure 9.a shows the architecture required for the on-the-fly conversion for HUB format and figure 9.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 (16) and (17) are carried out by two 2-1 multiplexors, two shifters and registers Q and SD and the corresponding load and shift control. In addition to these equations, the conventional format requires the implementation of equation (32), which involves one extra 3-1 multiplexer, a shifter, the register QR and a more complex control.

On the other hand and regarding the exponent, in conventional representation it is possible to have an overflow after rounding, which involves an update of the exponent. For HUB, updating is not required since overflow is never produced. Thus, we save some area, as shown in figure 10. In this figure, the conventional circuit has been optimized since the compulsory substraction of 1 and addition of 1 in case of overflow has been compensated through the left input of the final multiplexor. On the other hand, if a lookup table is used to implement the generation of the final exponent, the conventional representation has double size because it involves an extra input (the overflow signal).

Be that as it may, we conclude that there is an clear (and moderate) reduction in the hardware requirements of the HUB representation, keeping the delay.

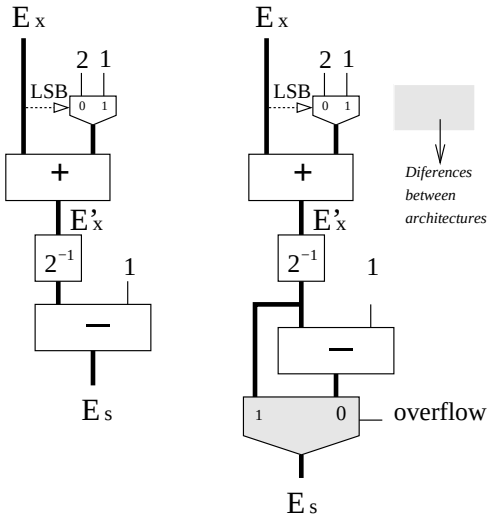
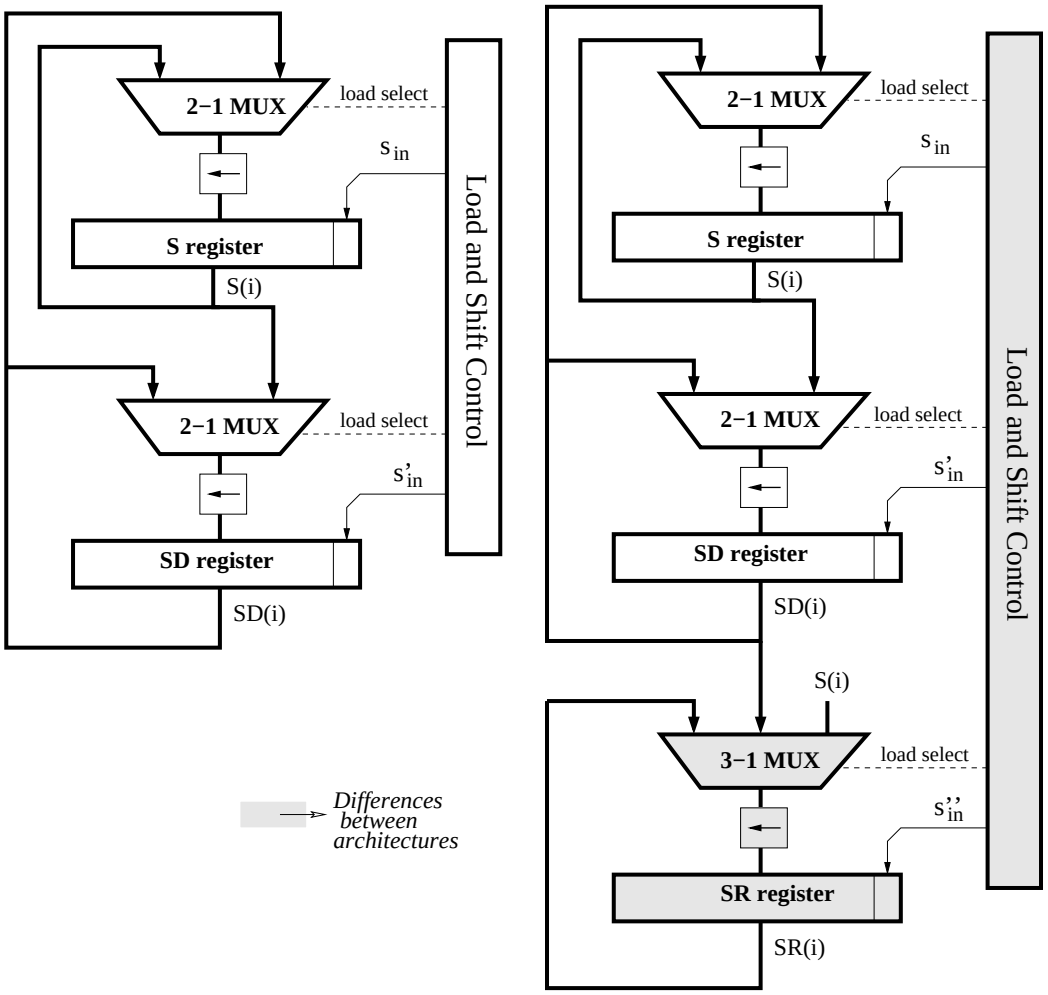


Fig. 10. Computation of the exponent (a) for HUB and (b) for conventional (optimized)

V. Summary and Conclusion

In this paper we have presented the square root of a HUB floating point number using the digit-recurrence algorithm and on-the-fly conversion of the result 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 any modification for HUB computation. For the on-the-fly conversion we have proposed an architecture that needs less hardware resources



a) Architecture for HUB format **b) Architecture for standard format**

Fig. 9. On-the-fly conversion for HUB (a) and for conventional (b)

since one of the three equations used in conventional on-the-fly converters is not required for HUB, and the computation of the exponent is also simplified. This simplification of the hardware is due to the fact that the HUB numbers are rounded to nearest simply by truncation.

As conclusion, the square root computation under HUB format presented in this paper confirms the fact that the underlined hardware of the HUB floating point alternative is simpler than its conventional counterpart, for the same precision and keeping the same delay. Thus, we consider that this emerging format may play an important role the design of the new generations of computers.

REFERENCES

[1] 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.

[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] 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.

[4] 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.

[5] J. Hormigo and J. Villalba, "Measuring improvement when using hub formats to implement floating-point systems under round-to-nearest," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2369–2377, June 2016.

[6] ——, "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.

[7] J. Villalba-Moreno, "Digit recurrence floating-point division under hub format," in *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, July 2016, pp. 79–86.

[8] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, 1994.

[9] ——, *Digital Arithmetic*. Morgan Kaufmann, San Francisco, 2004.