Reproducible Summation under HUB Format

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

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

Francisco Jaime
Dept. Program. Lang. & Computer Science
University of Málaga
Málaga, SPAIN
franj@ac.uma.es

***Abstract*—Floating point reproducibility is a property claimed by programmers and end users. Half-Unit-Biased (HUB) is a new representation format in which the round to nearest is carried out by truncation, preventing any carry propagation and saving time and area. In this paper we study the reproducible summation of HUB numbers by using a error-free vector transformation technique, providing both a specific architecture and the usage of combined HUB/Standard floating point adders to achieve a reproducible result.**

***Keywords*-Reproducible summation, HUB format**

I. Introduction

HUB is the acronym of Half-Unit-Biased format and it is based on shifting the standard numbers by half unit in the last place (ULP). Some of its important features are that the two's complement is carried out by bit-wise inversion, the round-to-nearest is performed by simple truncation, and requires the same number of bits for storage as its conventional counterpart for the same precision [1]. Thanks to those characteristics, it is possible to eliminate the rounding logic which significantly reduces both area and delay [1], [2].

The efficiency of using HUB formats for floating-point approach has been demonstrated in several works. In [1] 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 the 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). Regarding division and square root, the hardware required for HUB is exactly the same as that of conventional (for the same precision), needing the same number of iterations. Moreover, for implementations with on-the-fly conversion, the use of HUB format saves area [3]

The non-reproducibility of the floating point addition is due to the fact that it does not have the associative law. This leads to having different results depending on the order of the operands when a sum is carried out. The dynamic scheduling on processor with parallel computing resources involves different execution order of the operands in a multioperand addition. In this paper we propose a technique to deal with this problem for HUB numbers, in such a way that the final result of a summation is independent of the order of summation. Our algorithm is based on the error-free vector transformation proposed by Rump [4]. Basically we perform a pre-rounding of the data to a common base according to a specific boundary. This splits the operands in a high and low parts, in such a way that the summation of the high parts are free of rounding error and can be added in any order. This technique is proposed in [5] and [6] for standard IEEE floating point operands. For HUB format, the technique can not be directly applied but it requires some not straightforward modifications. In this paper, we propose an algorithm to deal with the summation of floating point HUB numbers, overcoming the problems involved by the particular way of rounding of the HUB approach.

The rest of the paper is organized as follows: in section II we present the fundamentals of the HUB format for floating point representation, in section III we describe the algorithm proposed in [5] and [6] based on the error-free vector transformation, in section IV we propose our algorithm to perform the reproducible summation of floating point HUB numbers, and 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 [7]. In this section we summarize the HUB format defined in [7] and particularize it for the floating-point normalized HUB numbers.

From a format point of view, a normalized floating point HUB number is similar to its IEEE counterpart but the significand is HUB. Thus, the only difference is the format of the significand. Without any loss of generality, we consider radix-2. Let x^{hub} denote a normalized radix–2 floating-point HUB number with a significand HUB of p bits (denoted m_{hub} with $1 < m_{hub} < 2$), a sign bit s and an exponent e

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

such that [7]:

$$x^{hub} = (-1)^s m_{hub} 2^e \tag{1}$$

where m_{hub} is

$$m_{hub} = \left[\mu + \frac{1}{2}\right] 2^{-(p-1)} \tag{2}$$

with $\mu \in \mathbb{N}$ belonging to the bound:

$$2^{p-1} \leq \mu < 2^p \tag{3}$$

Let *integer significand* denote μ ($\mu = \mu_{p-1}\mu_{p-2}...\mu_0$), that is

$$\mu = \sum_{i=0}^{p-1} \mu_i 2^i \tag{4}$$

From now on the triple (s, μ, e) denotes a normalized HUB number such that:

$$x^{hub} = (-1)^s \left[\mu + \frac{1}{2}\right] 2^{e-(p-1)} \tag{5}$$

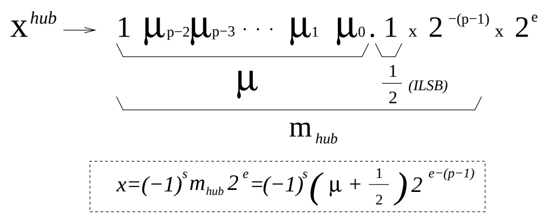
Due to the bound (eq. (3)) we have that $\mu_{p-1} = 1$, and from (2) and (4) the normalized HUB significant m_{hub} has the form:

$$m_{hub} = 1\mu_{p-2}...\mu_0.1 \ \times 2^{-(p-1)} \tag{6}$$

This expression is called the *operational form* of a HUB significand and is used in actual circuits [7]. From expression (6) and as shown in [7], a normalized HUB significand can be represented by the bits $\mu_{p-2}...\mu_0$ since the most significant bit (MSB) and the less significant bit (LSB) are both 1 and they are treated as implicit (the value $1\mu_{p-2}...\mu_0 \ \times 2^{-(p-1)}$ is called *representative form* in [7]). The implicit LSB of the operational form (fractional bit in expression (6)) is called ILSB in [7] and corresponds to the term $\frac{1}{2}$ in expressions (2) and (5). Consequently, according to equations (5), (6) and (4) x^{hub} is a normalized floating point HUB number given by:

$$x^{hub} = (-1)^s m_{hub} 2^e = (-1)^s \left[\sum_{i=0}^{p-1} \mu_i 2^i + \frac{1}{2}\right] 2^{e-(p-1)} \tag{7}$$

with $\mu_{p-1} = 1$ and $1 < m_{hub} < 2$. Figure 1 summarize the expression of a normalized HUB number x.



$x^{hub} \longrightarrow \underbrace{\underbrace{1\ \mu_{p-2}\mu_{p-3}\cdots\ \mu_1\ \mu_0}_{\mu}.\underbrace{1}_{\frac{1}{2}\ (ILSB)}}_{m_{hub}} \times 2^{-(p-1)} \times 2^e$
$x=(-1)^s m_{hub} 2^e = (-1)^s \left(\mu + \frac{1}{2}\right) 2^{e-(p-1)}$

Figure 1. Normalized floating point HUB format

A. Round to nearest for HUB numbers

Let us deal with round to nearest for HUB format for floating-point numbers. Consider a normalized non-HUB number $y = (s_y, \mu_{y\_non\_hub}, e_y)$ (that is $2^{p-1} \leq \mu_{y\_non\_hub} < 2^p$ but $\mu_{y\_non\_hub} \notin \mathbb{N}$) whose significand is composed by p integer bits and f fractional bits ($f > 0$):

$$\mu_y = 1\mu_{y_{p-2}} \cdots \mu_{y_0} \ . \ \mu_{y_{-1}} \cdots \mu_{y_{-f}} \tag{8}$$

We want to round this number to the nearest HUB number denoted by $y^{hub} = R^{hub}(y) = (s'_y, \mu'_y, e'_y)$, where $R^{hub}(*)$ denotes the operator that rounds a value to the nearest HUB number. $R^{hub}(y)$ is a HUB number with the same sign and exponent as y and with its integer significand μ'_y given by the p MSB of $\mu_{y\_non\_hub}$, that is the integer part of $\mu_{y\_non\_hub}$. Thus, $R^{hub}(y) = (s_y, \mu'_y, e_y)$ with μ'_y given by the next expression:

$$\mu'_y = \lfloor \mu_{y\_non\_hub} \rfloor = 1\mu_{y_{p-2}}...\mu_{y_0} \tag{9}$$

and the corresponding operational HUB significand $m'_{y\ hub}$ is

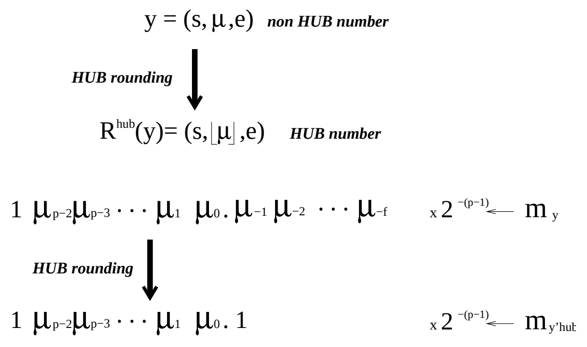
$$m'_{y\ hub} = \left[\lfloor \mu_{y\_non\_hub} \rfloor + \frac{1}{2}\right] 2^{-(p-1)} \tag{10}$$

Therefore, the rounded HUB number $R^{hub}(y)$ is achieved by keeping both the sign and the exponent and obtaining the integer part of the original number (that is, by truncating the p MSBs of the significand of y). Due to the definition of a HUB number this truncation produces a round to nearest number, as proved in [7]. For example, consider a 4 bit (plus ILSB) HUB format and two consecutive HUB numbers $a^{hub} = 1.010\ 1$ and $a{+}^{hub} = 1.011\ 1$ (the separate 1 is the ILSB). The middle point is $mdl = 1.011\ 0$ (mdl is not a HUB number; the separate 0 has same weigh as the ILSB). Consider the point $x = 1.010\ 1\ 01$ which fulfills $a^{hub} < x < mdl$. The nearest HUB number is 1.010 1 (that is a^{hub}) which can be obtained by truncation of the 4 MSB of 1.010 1 01. Now consider the number $x = 1.011\ 0\ 01$ which fulfills $mdl < x < a{+}^{hub}$. The nearest HUB number is 1.011 1 (that is $a{+}^{hub}$) which can be also obtained by truncation of the 4 MSB of 1.011 0 01 (note that the bit at the position of the ILSB is always 1 in a HUB number). Thus, truncation produces a round to nearest in both cases.

Figure 2 summarize the HUB rounding process: given a normalized non HUB number (s, μ, e), the round-to-nearest HUB number $(s, \lfloor \mu \rfloor, e)$ is performed by simple truncation of the significand (integer part).

III. Reproducible summation based on error-free vector transformation

In this section we have a look to the algorithms proposed in [5] and [6] for standard IEEE floating summation which are based on the error-free vector transformation presented by Rump [4]. The main idea is to pre-round the numbers to a common base in such a way that the addition of the



y = (s, μ, e) non HUB number
HUB rounding
R^hub(y)= (s,⌊μ⌋,e) HUB number
1 μp-2 μp-3 · · · μ1 μ0. μ-1 μ-2 · · · μ-f x 2^-(p-1) ← my
HUB rounding
1 μp-2 μp-3 · · · μ1 μ0. 1 x 2^-(p-1) ← my'hub

Figure 2. HUB rounding operator $R^{hub}(*)$

numbers can be computed accurately without any rounding error.

Given a vector of floating point numbers, all elements are split in high order parts and low order parts following the well-know algorithm FastToSum [8]. The algorithm is called an error-free transformation because it computes the addition as well as its rounding error. The upper parts can now be added and the result is reproducible. From this basis, the authors propose several algorithms for reproducible summation based on searching a common base to perform the split of the elements and how to increase the accuracy using the K-fold reproducible summation.

The next algorithm is used in [5] for reproducible sequential sum for a vector $v = \{v_1, v_1, ..., v_n\}$ of n floating point numbers of p bit precision

```
1: $m = max(|v_i|)$
2: $\delta = fl(n * m/(1 - 2n\epsilon))$
3: $M = 2^{\log_2(\delta)}$
4: $T = 0$
5: for $i = 1$ to $n$ in any order do
6:     $q_i = fl(fl(M + v_i) - M)$
7:     $T = fl(T + q_i)$
8: end for
```

where fl() denotes the evaluated result of an expression in floating point arithmetic (round to nearest mode), m is the maximum of $|v_i|$, the machine $\epsilon = 2^{-p}$, q_i is the high order part of v_i, and T is the accumulator. M is used to operate with each element to obtain the high part q_i in step 6. Thus, after running the algorithm it is ensured that T is the reproducible sum of high order parts.

The fundamentals of these algorithms are two lemmas (providing the conditions for exact computation), some theorems for error bound of conventional sum and the error-free vector transformation.

IV. Pre-rounding technique for HUB systems

In this paper we use both the standard IEEE 754-2008 floating point representation and the HUB representation. We do not consider the subnormal numbers. Also, overflow and underflow is not treated here. Only rounding to nearest is considered for HUB number and rounding toward zero for the standard representation. In both cases, the rounding is achieved by simple truncation. This is a very important feature throughout this paper since rounding never involves a change of exponent or a carry propagation when using HUB format.

The precision of a standard floating point number is given by the number of bits of its significand. For a HUB number, it is represented for the number of bits of its integer significant (μ). The counterpart of the floating point HUB numbers are the standard one with the same precision (and viceversa). Let p denote the precision of a HUB/standard floating point number (HUB numbers of p bit precision and standard numbers of p bit precision involve the same amount of numbers in both systems, the same distance between consecutive numbers for the same exponent and the same number of bits in the representation format [7]).

The unit in the last place of a floating point number x (HUB and standard) is the spacing between two consecutive floating point numbers of the same exponent and is denoted by $ulp(x)$.

We define machine epsilon ϵ as $\epsilon = 2^{-p}$. [1]

The upper bound of the relative error due to HUB rounding is $u = \epsilon$ since HUB rounding is to nearest.

For the standard, an important lemma is found in [5] and [6] and a slightly modified version of it is used in this paper (not valid for HUB):

Lemma I. *Let Q, x, y be three standard floating point numbers with a significand of p bits such that x and y are multiple of $\frac{1}{2}ulp(Q)$. If $|x+y| < \epsilon^{-1}\frac{1}{2}ulp(Q)$ then x+y can be represented as an exact p-bit signfificand floating point number and $x + y$ can be exactly computed.*

Proof: A number is exactly representable in a p-bit signifcand floating point format if the exponent is inside the required range and the number of significant bits is equal or less than p. In other words, if the difference of exponents of the MSB and the LSB is less than p. Then, let us focus on the weigh of the LSB and MSB:

- The minimum LSB (LSB_{min}) corresponds with the addition of one operand whose ulp is the minimum multiple possible, that is $\frac{1}{2}ulp(Q)$. Assume Q has an exponent k so that $ulp(Q) = 2^{k-(p-1)}$. Thus
$$LSB_{min} = min\{LSB(x), LSB(y)\} = \frac{1}{2}2^{k-(p-1)} \quad (11)$$
- Tanking into account that $\epsilon = 2^{-p}$ and $ulp(Q) = 2^{k-(p-1)}$, expression $|x + y| < \epsilon^{-1}\frac{1}{2}ulp(Q)$ becomes $|x + y| < 2^k$ and thus the maximum MSB possible

[1]The definition of ϵ in [5] and [6] is different from ours, although the numerical value is the same. In our paper, rounding never involves a change in the exponent (since it is carried out by truncation) and the integer 1 is not exactly representable

(MSB_{max}) of the result is

$$MSB_{max} = 2^{k-1} \tag{12}$$

Now, the difference of exponent between the MSB_{max} and the LSB_{min} is $\log_2 2^{k-1} - \log_2(\frac{1}{2}2^{k-(p-1)}) = p - 1 < p$, and thus, it is representable as an exact standard floating point number.

Q.E.D

The underlying practical interpretation of this lemma is that the addition of two standard floating point numbers is reproducible if both are multiple of $\frac{1}{2}ulp(Q)$ and the number of significant bits of the result is less than or equal to the size of the significand (p).

Now we adapt the pre-rounding technique proposed in [5] and [6] for HUB systems. The key idea is to perform a pre-rounding of the input values to a common base in such a way that their sum can be computed accurately with no rounding error. The error depends only on the input numbers and the selected common base and the result does not depend on the order of summation.

With this technique the floating HUB numbers are split into a high and low independent parts according to a common base. The summation of the high parts is reproducible, whereas the lower part keeps the rounding error. For HUB, this technique can be implemented in two ways: using a direct manipulation of the exponent and significant in a specific architecture or by using HUB adders, as shown below.

A. Splitting the HUB number

The next theorem shows the way to split a normalized HUB number in two independent upper and lower parts such that the upper part is suitable for reproducible summation, as shown below. For clarity in the exposition and from now on, we consider positive numbers (since HUB follows sign-magnitude representation, the treatment of negative number is similar).

Theorem I *Let $Q = 2^k$ and consider its standard normalized floating point representation with a significand of p bits. Let $ulp(Q)$ denote the unit in the last place of the standard floating point representation of Q, that is $ulp(Q) = 2^{k-(p-1)}$. Let x^{hub} be a normalized HUB number ($x^{hub} \equiv (s, \mu, e)$), with $Q > x^{hub}$. If we define q and r as the upper and lower part respectively of x^{hub} such that:*

$$x^{hub} = q + r \tag{13}$$

where q is

$$q = \left(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2}\right) 2^{k-(p-1)} \tag{14}$$

then

$$q = l\frac{1}{2}ulp(Q) \tag{15}$$

$$r < \frac{1}{2}ulp(Q) \tag{16}$$

with $l \in \mathbb{N}$

Proof:

- $q = l\frac{1}{2}ulp(Q)$. Since $\lfloor \mu 2^{e-k} \rfloor \in \mathbb{N}$ then $\exists l' \in \mathbb{N}$ such that $\lfloor \mu 2^{e-k} \rfloor = l'$. Thus, $\left(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2}\right) = l' + \frac{1}{2} = \frac{1}{2}(2l' + 1)$. Since $l' \in \mathbb{N}$ then $l = 2l' + 1$ is also an integer, that is $\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2} = l\frac{1}{2}$. From this expression and taking into account that $ulp(Q) = 2^{k-(p-1)}$, expression (14) becomes

$$q = l\frac{1}{2}ulp(Q) \tag{17}$$

- $r < \frac{1}{2}ulp(Q)$. From expression (13) we have that $r = x^{hub} - q$. Thus, from expressions (5) and (14) we have:

$$r = \left(\mu + \frac{1}{2}\right) 2^{e-(p-1)} - \left(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2}\right) 2^{k-(p-1)} =$$
$$= \left[\mu 2^{e-k} + \frac{1}{2}2^{e-k} - \lfloor \mu 2^{e-k} \rfloor - \frac{1}{2}\right] 2^{k-(p-1)}$$

Taking into account expression (4) and developing the integer part of $\mu 2^{e-k}$:

$$r = \left[\sum_{i=0}^{p-1} \mu_i 2^i 2^{e-k} + \frac{1}{2}2^{e-k} - \right.$$
$$\left. \sum_{i=k-e}^{p-1} \mu_i 2^i 2^{e-k} - \frac{1}{2}\right] 2^{k-(p-1)} =$$
$$= \left[\sum_{i=0}^{k-e-1} \mu_i 2^i 2^{e-k} + \frac{1}{2}2^{e-k} - \frac{1}{2}\right] 2^{k-(p-1)}$$

An upper bound is achieved if we consider $\mu_i = 1, \forall i$ in the previous expression and taking into account that $\sum_{i=0}^{k-e-1} 2^i 2^{e-k} = 1 - 2^{e-k}$:

$$r \le \left[\frac{1}{2} - \frac{1}{2}2^{e-k}\right] 2^{k-(p-1)} \tag{18}$$

Since k is a constant and $e < k$ then $\frac{1}{2} - \frac{1}{2}2^{e-k} < \frac{1}{2}$. From this expression and taking into account the definition of $ulp(Q)$ as $ulp(Q) = 2^{k-(p-1)}$, expression (18) becomes:

$$r < \frac{1}{2}ulp(Q) \tag{19}$$

Q.E.D

Theorem I involves a disjoint splitting of a HUB number x^{hub} into two standard numbers q and r where there is not any overlap between them. Moreover, if the resulting upper parts of a set of HUB numbers fulfill the condition of Lemma 1, then their summation is exact and reproducible. Next we prove this.

Given a vector of n HUB numbers $v^{hub} = [v_1^{hub}, ..., v_n^{hub}]$, we split its elements according to Theorem I (that is $v_i^{hub} = q_i + r_i, \forall i$) using a common base $Q = 2^k$ such that $Q > q_i, \forall i$. To meet the conditions of Lemma 1 the next expression have to be fulfilled:

$$|\sum_{i=1}^{n} q_i| < \epsilon^{-1}\frac{1}{2}ulp(Q) \tag{20}$$

Now we prove that this condition is accomplished. From Theorem I we have $Q = 2^k$ so that $ulp(Q) = 2^{k-(p-1)}$ and taking into account that $\epsilon = 2^{-p}$ we have that $\epsilon^{-1}\frac{1}{2}ulp(Q) = 2^k$. Thus, the condition (20) of Lemma I becomes:

$$|\sum_{i=1}^{n} q_i| < 2^k \tag{21}$$

Now we have to find a suitable value of k that fulfills equation (21). First we take into account that $|\sum_{i=1}^{n} q_i| \leq \sum_{i=1}^{n} |q_i|$ so that we are going to prove a stronger condition:

$$|\sum_{i=1}^{n} q_i| \leq \sum_{i=1}^{n} |q_i| < 2^k \tag{22}$$

Since $v_i^{hub} = q_i + r_i$, then $q_i = v_i^{hub} - r_i$ and $|q_i| = |v_i^{hub} - r_i| \leq |v_i^{hub}| + |r_i|$. Since each v_i^{hub} fulfills the condition of Theorem I, we can use equation (16) to ensure that $|q_i| < |v_i^{hub}| + \frac{1}{2}ulp(Q)$ and then $\sum_{i=1}^{n} |q_i| < \sum_{i=1}^{n} |v_i^{hub}| + n\frac{1}{2}ulp(Q)$. To ensure a good bound for $|v_i^{hub}|$ we choose the maximum of them (denoted $|v_i^{hub}|_{max}$). Futhermore, the maximum of a set of HUB numbers is reproducible since floating point maximum is associative. Thus, we can write

$$\sum_{i=1}^{n} |q_i| < n|v_i^{hub}|_{max} + n\frac{1}{2}ulp(Q) \tag{23}$$

Taking into account that $ulp(Q) = 2^{k-(p-1)}$ and $\epsilon = 2^{-p}$, we can write that $ulp(Q) = 2\epsilon \cdot 2^k$. Substituting this expression in equation (23) we have

$$\sum_{i=1}^{n} |q_i| < n|v_i^{hub}|_{max} + n\epsilon \cdot 2^k \tag{24}$$

If we select 2^k such that $n|v_i^{hub}|_{max} \leq (1 - n\epsilon) \cdot 2^k$ then equation (24) becomes:

$$\sum_{i=1}^{n} |q_i| < 2^k \tag{25}$$

Thus, the value of k can be chosen as

$$k = \left\lceil \log_2 \frac{n|v_i^{hub}|_{max}}{1 - n\epsilon} \right\rceil \tag{26}$$

In summary, given a set of HUB numbers to be added $\{v_i^{hub}\}$ we select the maximum of them and calculate the value of Q as $Q = 2^k$ with k according expression (26). Now, we apply the Theorem I to each HUB number obtaining a set $\{q_i\}$ whose addition is reproducible.

B. Accuracy

The general rule for accuracy is the smaller Q is, the more accurate the final sum is. For the pre-rounding technique proposed, the accuracy depends on the common base used for splitting, that is Q. Since the lower parts r_i all fulfill equation (16), the maximum absolute error (A_error) is bounded by the product of the number of numbers to be added and the bound of equation (16):

$$A\_error < n \cdot \frac{1}{2}ulp(Q) \tag{27}$$

Taking into account that $ulp(Q) = 2^{k-(p-1)}$ we have

$$A\_error < n \cdot 2^{k-p} \tag{28}$$

This error bound can be improved by using the K-fold technique described in [5], which applies K passes of error-free transformation to extract K leading parts of each input floating point instead of just 1 leading part to improve the computed accuracy. This technique increases the running time substantially when n is large. In [6] the 1-Reduction technique is proposed to avoid the computation of the global maximum absolute value (specially useful in multiprocessor environment since no communication is required). It is based on a pre-computation of the boundaries independently of input data so that each processor uses its own local boundary. We think that these techniques can be applied for HUB environment since the techniques proposed in [5] and [6] are based on Lemma I and Theorem I. Nevertheless, in this paper we do not study it and it is leaved for future works. See [5] and [6] to find a deep analysis of improving the accuracy.

V. Architectures for implementation

The splitting algorithm proposed in this paper can be implemented by an architecture that manipulates the signficand and the exponent directly or by using HUB/standard adders. Let see both alternatives.

A. Direct manipulation of the significand by a pre-rounding unit

The pre-rounding of the HUB numbers to be added can be carried out by simple manipulation of the input data. For a better explanation of the technique Figure 3 shows the relative position of some different terms used in Theorem I.

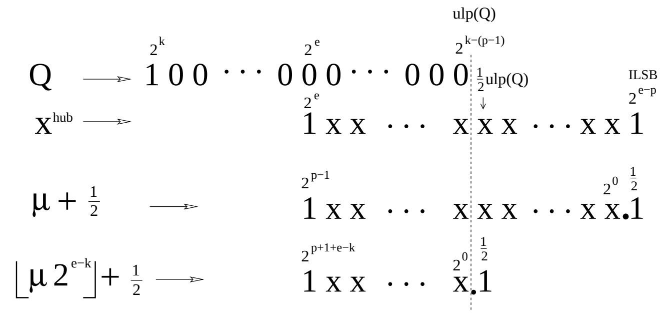
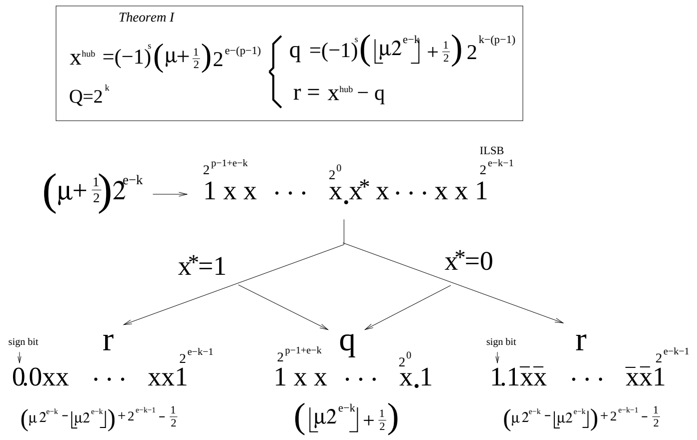


Figure 3. Relative position of some terms used in Theorem I

Next, in figure 4, we can see the mechanism to split a HUB number x^{hub} into q and r, according to Theorem I (see equations (14) and (13)). The main feature of this algorithm is that it does not involve any carry propagation.

As shown in figure 4, from the value $(\mu + \frac{1}{2})2^{e-k}$ of x^{hub} (with the form $1xx \cdots x.x^*x \cdots xx1$) we can obtain q by taking the integer part of $\mu 2^{e-k}$ (that is $\lfloor \mu 2^{e-k} \rfloor$ by truncation) and forcing the fractional bit to one (weigh 2^{-1}).



Theorem I
$x^{hub} = (-1)^s(\mu+\frac{1}{2})2^{e-(p-1)}$
$Q=2^k$
$q = (-1)^s(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2})2^{k-(p-1)}$
$r = x^{hub} - q$
$(\mu+\frac{1}{2})2^{e-k}$
1 x x ··· x.x* x ··· x x 1
ILSB
2^{e-k-1}
x*=1
x*=0
r
q
r
sign bit
0.0xx ··· xx1
1 x x ··· x.1
1.1x̄x̄ ··· x̄x̄1
$(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor) + 2^{e-k-1} - \frac{1}{2}$
$(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2})$

Figure 4. Splitting a HUB number x^{hub} as upper q and lower r parts. No any carry propagation is involved

The involved operation to calculate $r = x^{hub} - q$ is (considering a common factor of $2^{k-(p-1)}$):

$$\left(\mu + \frac{1}{2}\right) 2^{e-k} - \left(\lfloor \mu 2^{e-k} \rfloor + \frac{1}{2}\right) =$$
$$= (\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor) + 2^{e-k-1} - \frac{1}{2} \qquad (29)$$

Let us analyze this expression in detail. The term $(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor)$ of this expression is a fractional value of $k-e$ bits (notice that $\mu \in \mathbb{N}$) to which we add the amount 2^{e-k-1}. The result of this addition does not involve a carry propagation since the LSB of $(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor)$ is 2^{e-k}. Thus, the expression $(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor) + 2^{e-k-1}$ has the form $0.x^*xx \cdots xx1$. The substraction of the term $1/2$ in expression (29) produces either a positive number if the bit $x^* = 1$ or a negative one if $x^* = 0$. Lets analyze both cases to prove that carry propagation is always prevented:

- Bit $x^* = 1$ In this case, the term $(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor)$ has the form $0.1xxx \cdots xx1$. Thus, the substraction of the amount 1/2 gives a positive number. No any carry propagation is involved since the only operation is to force to zero the first fractional bit (final result: $0.0xxx \cdots xx1$)
- Bit $x^* = 0$ In this case, the term $(\mu 2^{e-k} - \lfloor \mu 2^{e-k} \rfloor)$ has the form $0.0xxx \cdots xx1$. Thus, the substraction of the amount 1/2 gives a negative number. To obtain the negative number is enough to invert the first fractional bit and consider the sign bit as 1 (final result: $1.1xxx \cdots xx1$)

The architecture that splits a HUB number is shown in Figure 5. We can see that only a shifter and a 1-bit inverter are required. The shifter carries out a shift of $k-e$ bits to the right, obtaining q. The outgoing bits are used to calculate r, where the bit inverter is needed to obtain both the sign bit and the first fractional bit. No carry propagation is involved since all the operations are bit wise.

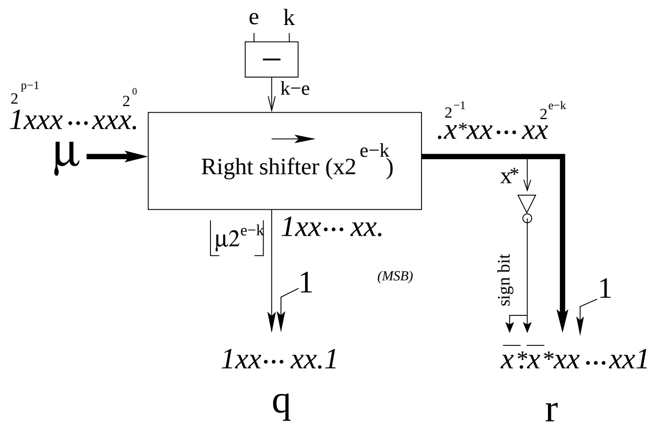


Figure 5. Architecture for splitting a HUB number x^{hub} as upper q and lower r parts with no carry propagation

B. *Using a combined HUB/standard adder*

The second way to obtain the values of q and r is by using a combined HUB/standard floating point (FP) adder as that shown in Figure 6. The only difference between a p-bit standard FP adder (with rounding by truncation) and its counterpart FP HUB adder is that HUB adder has an extra bit at its significand inputs: the ILSB bit. Thus, for having a combined HUB/standard FP adder we use a conventional adder of p+1 bits where the value of ILSB at each significand input is 0 for a standard number and 1 for a HUB number. The output (significand) has p-bits, where the rounding was performed by truncation for both the standard and the HUB result (for the addition of two standards (p-bits) the result has p bits, whereas for HUB the extra ILSB is always 1)

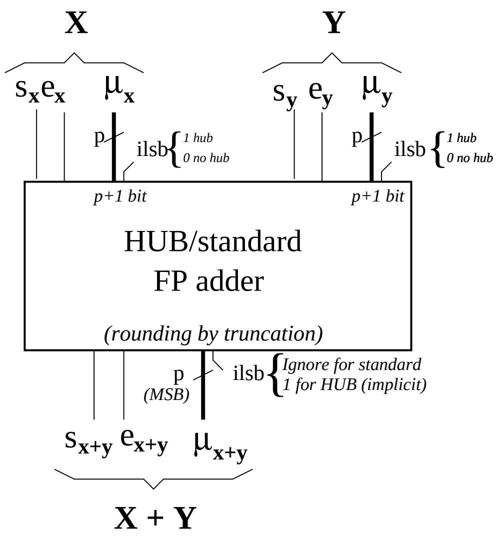


Figure 6. HUB/standar FP adder

To obtain q we have to carry out two operations with the combined adder. Once the value of $Q = 2^k$ is selected, we carry out the next operation for every element v_i:

$$s_i^{hub} = Q + v_i^{hub} \tag{30}$$

where Q is not a HUB number, v_i^{hub} is a HUB number and s_i^{hub} is a HUB number (rounding by truncation). The value of q_i is obtained after the next operation:

$$q_i = s_i^{hub} - Q \tag{31}$$

where Q is not a HUB number, s_i^{hub} is a HUB number and q_i is not a HUB number (rounding by truncation).

To obtain r_i we perform the next operation:

$$r_i = v_i^{hub} - q_i \tag{32}$$

where q_i is not a HUB number, v_i^{hub} is a HUB number and r_i is not a HUB number (rounding by truncation).

Figure 7 shows how to use the combined HUB/standard FP adder to obtain q_i and r_i

VI. Summary, conclusion and future works

In this paper we have presented a reproducible summation algorithm based on error-free vector transformation for floating point HUB representation. This transformation was successfully used in [5] and [6] to obtain a reproducible summation on the standard floating point representation. In this paper we adapt these algorithms to the case of HUB representation, where the round to nearest is carried out by truncation. This feature force us to reformulate the theorems and lemmas on which reproducible summation is based, proving that the split of the HUB floating point number is also possible. As consequence we propose an algorithm to split the numbers in high and low parts in which the high parts are suitable for reproducible summation.

The no carry propagation feature of the HUB rounding makes possible the design of a simple architecture to split the incoming elements. Apart from this, we also propose the usage of standard and HUB adders to run the algorithm (combined standard/HUB adder). As future works, we will work on the improvement of the accuracy of the algorithm by exploring the K-fold technique in deep (we think this is possible since K-folder technique follows the same theoretical base).

As conclusion, HUB format offers a similar performance in terms of summation reproducibility based on error-free vector transformation when using regular adders and makes possible the design of simple specific architectures to split the incoming arguments.

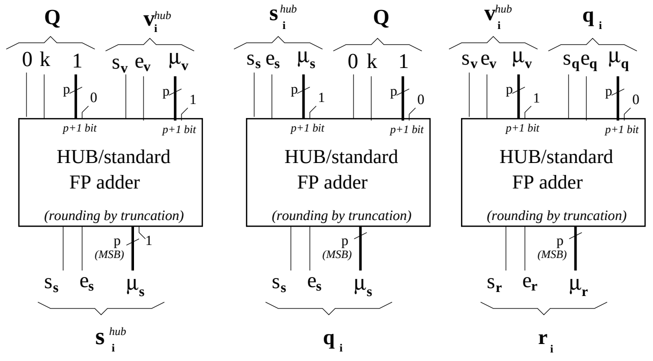


Figure 7. Obtaining q_i & r_i

References

[1] J. Hormigo and J. Villalba-Moreno, “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, 2016.

[2] J. Villalba-Moreno, J. Hormigo, and S. Gonzalez-Navarro, “Fast HUB Floating-point Adder for FPGA,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2018.

[3] J. Villalba-Moreno, “Digit Recurrence Floating-point Division under HUB Format,” *23rd IEEE Symposium on Computer Arithmetic, Silicom Valley (California, USA)*, July 2016.

[4] S. M. Rump, T. Ogita, and S. Oishi, “Fast High Precision Summation,” *Nonlinear Theory and Its Applications, IEICE*, vol. 1, no. 1, pp. 2–24, 2010.

[5] J. Demmel and H. D. Nguyen, “Fast Reproducible Floating-Point Summation,” in *2013 IEEE 21st Symposium on Computer Arithmetic*, April 2013, pp. 163–172.

[6] ——, “Parallel Reproducible Summation,” *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2060–2070, July 2015.

[7] J. Hormigo and J. Villalba-Moreno, “New Formats for Computing with Real-Numbers under Round-to-Nearest,” *Computers, IEEE Transactions on*, vol. 65, no. 7, pp. 2158 – 2168, 2016.

[8] T. J. Dekker, “A floating-point technique for extending the available precision,” *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.