This is the author's version of an article that has been published in IEEE Transactions on Computers. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TC.2018.2807429

Unbiased Rounding for HUB Floating-point Addition

Julio Villalba, Javier Hormigo and Sonia González-Navarro

***Abstract*—Half-Unit-Biased (HUB) is an emerging format based on shifting the represented numbers by half Unit in the Last Place. This format simplifies two's complement and round-to-nearest operations by preventing any carry propagation. This saves power consumption, time and area. Taking into account that the IEEE floating-point standard uses an unbiased rounding as the default mode, this feature is also desirable for HUB approaches. In this paper, we study the unbiased rounding for HUB floating-point addition in both as standalone operation and within FMA. We show two different alternatives to eliminate the bias when rounding the sum results, either partially or totally. We also present an error analysis and the implementation results of the proposed architectures to help the designers to decide what their best option are.**

***Index Terms*—HUB format, unbiased rounding**

I. Introduction

Nowadays, the development of special purpose systems is exploding due to the emergence of new application areas, such as machine learning [1], [2], Internet-of-Things [3], green computing [4], [5], [6] and others. Many of these applications requires floating-point (FP) computation which is traditionally very costly. As a consequence, in the last years, there is a growing interest in finding new FP approaches which achieve better figures when implementing these specific systems.

Probably the most radical FP approach proposed until now is the Flexible Floating-Point (FFP) format [7] launched by Synopsys, which is totally different to the IEEE FP standard [8]. FFP changes the representation of both significand and exponent, uses truncation as the only rounding mode and does not normalize, among other changes. A more subtle modification is proposed by using HUB approach [9].

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). This representation format has important features [9] like the two's complement is carried out by bit-wise inversion, the round-to-nearest is performed by simple truncation and the double rounding error [10] never happens. Furthermore, it requires the same number of bits for storage as its conventional counterpart for the same precision.

In [11], the authors analyze the benefits of using HUB format for FP adders, multipliers, and converters. 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 FP adder, 17% speed-up, 22% less area and slightly less power for the FP multiplier).

The basic HUB adder presented in [11] performs round-to-nearest, but it may introduce some bias. The importance of having an unbiased rounding mode (i.e., having equal probability of rounding up or down for the tie case) has been highlighted in the literature since long time ago [12], [13]. It eliminates the drift effect [14] and reduces the power of quantification error in signal processing [15]. In fact, an unbiased rounding mode is the default mode in the IEEE FP standard [8].

In this paper, we analyze the different sources of bias which are present in previous HUB adder architecture and propose two new architectures to avoid these sources of bias. We also provided an error analysis and an implementation comparison of the different adders.

The rest of the paper is organized as follows: Section II summarizes the fundamental of the HUB representation and reviews the basic HUB adder [11]. Section III is devoted to analyze the different sources of bias whereas Section IV presents the new architectures proposed to perform partial or total unbiased rounding. Section V analyzes the implementation of unbiased rounding for a FMA unit. The error analysis and the implementation results are presented in Sections VI and VII respectively. Finally, in the last section, we give the summary and conclusions.

II. HUB format and adder for FP numbers

A. HUB format for FP numbers

The mathematical fundamentals and a deep analysis of the HUB format can be found in [9]. In this section, we summarize the HUB format defined in [9] and particularize it for normalized HUB FP numbers.

A HUB FP number is similar to a regular one but its significand follows the HUB format. Thus, the only difference compared with the binary FP standard [8] is the format for the significand. Without any loss of generality, in this paper, we use HUB FP operands with normalized significand in radix–2. Let us define x as a FP radix–2 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 normalized HUB magnitude. We define a normalized HUB significand as a value M_x such that $1 < M_x < 2$. The HUB significand is

J. Villalba, J. Hormigo and S. González-Navarro are based at the Department of Computer Architecture, Universidad de Málaga, Málaga, Spain, E-29071.
E-mail: {jvillalba,fjhormigo,sgn}@uma.es

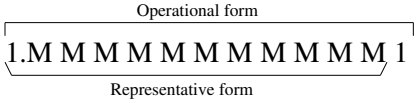


Fig. 1. Representative versus operational forms

a $f+1$ bit vector with the form $M_x = (M_{x_0}, M_{x_{-1}}, M_{x_{-2}}, \cdots, M_{x_{-f}})$ such that:

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

where 2^{-f-1} is the shifting (i.e., half ULP) and $M_{x_0} = 1$. Let *representative form* denote the vector. Thus, although the normalized HUB significand is represented by $f+1$ bits, according to expression (1) the binary version of a normalized HUB significand is composed by $f+2$ bits:

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

The binary version is required only to operate with HUB numbers. Thus, let *operational form* denote the binary version (see Fig. 1). We can see that the least significant bit (LSB) of the operational form of any nonzero HUB number is always equal to 1. Thus, we make this bit implicit for the HUB format (as the implicit most significant bit (MSB) of the significand in the IEEE normalized FP numbers). Let ILSB denote the implicit LSB of a HUB number. The ILSB is not needed to represent, to store, or to transmit a HUB number. It could be explicitly required only to perform operations.

For systems where round-to-nearest is required, a rounding bit is normally needed. Since the round-to-nearest for HUB format is carried out by truncation, the rounding bit is not required anymore. Thus, in spite of having an extra bit in the operational form, this extra bit of HUB numbers is compensated by the lack of a specific rounding bit. As a consequence, the data–path is not always incremented for HUB approach, as shown for the division in [16] or addition in [11].

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

$$M'[0:m-2] = M[0:m-2] \quad (3)$$

Thus, the rounded HUB number M' is achieved by truncation of the $p-(m-1)$ LSB of M. Due to the definition of a HUB number (see (1) with $f = m-2$) this truncation produces a round-to-nearest number, as proved in [9].

On the other hand, when a number is just between two Exactly Represented Numbers (ERNs) (tie condition), the proposed rounding is always carried out in the same direction (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 proposed in [9] for HUB. 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):

$$M'[0:m-2] = M[0:m-3], 0 \quad (4)$$

Thus, the unbiased rounding is achieved by truncation and forcing the LSB of the representative form to zero, i.e. bit $M'[m-2] = 0$, as proved in [9]. Thus, this unbiased round-to-nearest is not so different from the IEEE-754 tie to even, since it is like a tie to even of the representative form (with no carry propagation).

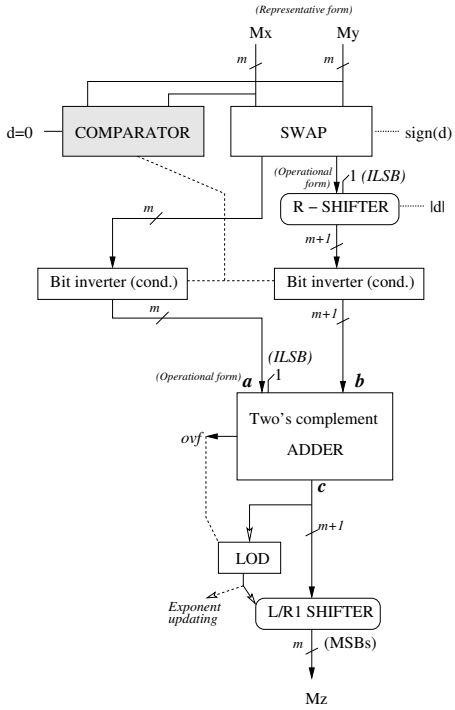


Fig. 2. HUB Floating-point adder proposed in [11] (architecture A)

B. Basic adder for HUB FP numbers

A simple design of a HUB FP adder is presented in [11] and shown in Fig. 2. In comparison with the conventional numbers FP architecture counterpart, the HUB architecture does not use the circuits required for calculating the sticky bit as well as the round-to-nearest circuit since the HUB round-to-nearest is carried out simply by truncation. As a consequence, an important reduction both in area and power consumption is achieved as proved in [11]. Let A denote the architecture proposed in [11].

Architecture A carries out the addition of two HUB FP numbers with rounding-to-nearest for the tie-away-from-zero case. Therefore, the rounding of the significand is biased for the tie case, i.e., the rounding is always performed in the same direction: up. In the next sections, we analyze deeply when the tie condition occurs (or not) and propose two designs to prevent bias when rounding.

III. Sources of bias when rounding

Some bias in the rounding may be produced when the tie condition occurs. For HUB numbers, the tie condition occurs when the result of a operation after normalization has a bit 0 in the position of the ILSB and the rest of bits on its right are all 0, i.e. when the discarded bits after truncation are all zero. The parenthesis ( ) denotes the position of the ILSB. Next, we show some examples of possible actual results after normalization:

```
          ILSB
           |
  01.01101(1)     Not tie (discarded: 1000)
  01.01101(0)     Tie     (discarded: 0000)
  01.01100(1)001  Not tie (discarded: 1001)
  01.01101(0)000  tie     (discarded: 0000)
  01.01100(0)010  Not tie (discarded: 0010)
         |
        LSB*
```

From now on, let LSB* denote the LSB of the representative form (i.e the underlined bit in 01.01100(0)010), x means either 0 or 1 and d the difference between the exponents of the operands (alignment). The effective operation (Eop) is a function of the operation ($op = 0$ for addition, $op = 1$ for subtraction) and the sign of the operands. In the next subsections, we analyze the tie condition (which depends on the effective operation and the alignment).

A. Effective addition with $d \geq 1$ (Eop=0)

In this case, the tie condition is never produced since there is at least a bit 1 beyond the ILSB position. The next examples show the situation:

```
  01.xxxxx(1) [operational form]
+ 00.1xxxx(x)1 [d=1]
-------------
  01.xxxxx(x)1 not tie case

  01.1xxxx(1) [operational form]
+ 00.1xxxx(x)1 [d=1]
-------------
 010.xxxxx(x)1 [overflow]
 -->
  01.0xxxx(x)x1 not tie
```

B. Effective subtraction with $d \geq 2$ (Eop=1)

If $d \geq 2$ the result of the subtraction is either normalized or it requires a left shift of one position [17] (that is, the pattern of the result is 01.xxx... or 00.1xxx...). The tie condition is never fulfilled in the result since there is at least a bit 1 beyond the position of the ILSB (due to the shift of the second operand). The next example shows the situation:

```
  01.1xxxx(1)   [operational form]
- 00.01xxx(x)x1 [d=2]
-------------
  01.xxxxx(x)x1 not tie
```

Another example (left shift required):

```
  01.00000(1)   [operational form]
- 00.01xxx(x)x1 [d=2]
-------------
  00.1xxxx(x)x1 not tie
<--
  01.xxxxx(x)1  not tie
```

C. Effective aligned addition (d=0, Eop=0)

In this case, an overflow is always produced, and a right shift of one position is required. The tie condition takes place if the LSB* of the result of the addition is 0 (before shifting). Otherwise, there is not a tie:

```
  01.xxxxx(1) [operational form]
+ 01.xxxxx(1)
-------------
 010.xxxyz(0) [overflow]
 -->
  01.0xxxy(z)0 [z=0 tie, z=1 not tie]
```

Thus, the tie condition depends on bit z. If z=0, the tie case is produced and it is a source of bias.

D. Effective aligned subtraction (d=0, Eop=1)

Let M_u, M_v denote the two normalized significands of two HUB numbers (U, V) with the same exponent. Assume that $M_u \geq M_v$. Since $1.000\cdots00(1) \leq M_v \leq M_u \leq 1.111\cdots11(1)$, the subtraction fulfills $0 \leq M_u - M_v \leq 0.111\cdots111(0)$. Consequently, the result always fulfills the tie condition because a left shift of at least one position is always required for normalization, which involves the injection of a 0 at the position of the ILSB of the normalized result, as shown in the next example:

```
Mu>Mv
  01.xxxxx(1) [operational form]
- 01.xxxxx(1)
-------------
  00.1xxxx(0)
 <--
  01.xxxx0(0) [tie case]
```

Similarly, if $M_u \leq M_v$ we have that $-0.111\cdots1101 \leq M_u - M_v \leq 0$ and thus, the tie condition always takes place, as shown in the next example:

```
Mu<Mv
  01.xxxxx(1) [operational form]
- 01.xxxxx(1)
-------------
  11.110xx(0)
 <--
  10.xx000(0) [tie case]
```

Thus, the tie condition is always produced and this is another possible source of bias.

E. Effective non-aligned subtraction with d=1 (Eop=1)

If the result of the subtraction is already normalized, the tie condition never fulfills since the ILSB(=1) of the second operand is out of the word-size, which involves a bit 1 at least in those positions, as shown in the next example:

```
  01.10010(1) [operational form]
- 00.10000(0)1 [d=1]
```

```
-------------
  01.00010(0)1 [Normalized, no tie case]
```

If the result of the subtraction is not normalized and the most significant fractional bit is 1, the tie condition does not fulfill, as shown in the next example:

```
  01.10010(1) [operational form]
- 00.11000(0)1 [d=1]
-------------
  00.11010(0)1
 <--
  01.10100(1)  [Normalized, no tie case]
```

On the other hand, the tie case is possible if the result of the subtraction has the integer bit 0 and the most significant fractional bit 0 (that is, the result has the pattern 00.0xxx...x(x)1), as shown in the next example:

```
  01.00010(1) [operational form]
- 00.11100(1)1 [d=1]
-------------
  00.00101(1)1
         |
        LSB*
 <--
  01.01110(0)0 [Normalized, tie case]
```

In this case, the resulting HUB number would be always rounded up (001.01110(1)), which produces a bias. This is the last source of bias.

To sum up, the three possible sources of bias are: i) aligned addition (case d=0, Eop=0), ii) aligned subtraction (case d=0, Eop=1) and iii) subtraction and d=1 (case d=1, Eop=1). Next, we describe the proposed architectures to deal with these sources of bias.

IV. Adders for unbiased rounding

The two new designs proposed in this paper are shown in Fig. 3 and Fig. 4. Architecture A+ (Fig. 3) is designed to perform partial unbiased rounding (only aligned addition case), whereas architecture A++ (Fig. 4) resolves all the previous source of bias, producing unbiased rounding.

A. Architecture for partial unbiased rounding: A+

The architecture A+ shown in Fig. 3 only prevents the bias due to one of the sources: aligned addition. The way to solve the situation is described in [16]: rounding by forcing the LSB* of the shifted result (bit y in the example of subsection III-C) to zero. The next example shows the tie case (z=0), and how to achieve an unbiased rounding:

```
  01.xxxx0(1) [operational form]
+ 01.xxxx1(1)
-------------
 010.xxxy0(0) [overflow]
 -->
  01.0xxxy(0)0  tie case, exact value
  Results as a function of y in 01.0xxxy(0)0
  case y=0
    01.0xxx0(0)0 [tie case, exact value]
    01.0xxx0(1) [final rounded up]
```

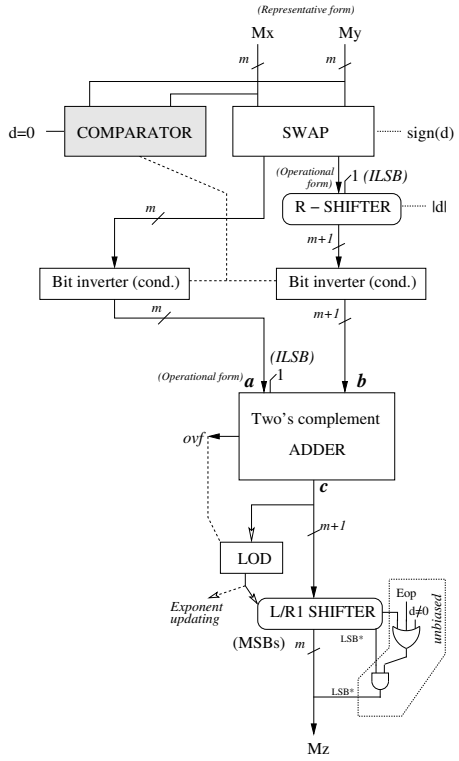


Fig. 3. Partial unbiased HUB FP adder: architecture A+

```
case y=1
  01.0xxx1(0)0 [tie case, exact value]
  01.0xxx0(1) [final rounded down]
```

If the probability of having y=0 or 1 is the same, it produces an unbiased rounding in this case.

The hardware required to perform this unbiased rounding is very simple: one AND and one OR gates placed along the final L/R1-SHIFTER of architecture A+ (see Fig. 3). As can be noted, A+ is a slight modification of the adder A.

B. Architecture for unbiased rounding: A++

The architecture A++ shown in Fig. 4 eliminates the three sources of bias described in previous section. It prevents the significand comparator required in architecture A (Fig. 2). In this way, the result of the operation can be positive or negative (this feature allows preventing the tie cases not supported by A+). A leading zero-one detector (LZOD) is required in the new architecture to detect if the result is negative, as well as the final conditional inverter, in order to obtain the magnitude of the result if this were negative.

The conditional inverter after the swap module of Fig. 4 and the ILSB at the input of the R-Shifter are in charge of obtaining the two's complement of the right operand when an effective subtraction is required (box "logic" in Fig. 4 controls the effective operation).

Note that the two's complement of a HUB number does not involve a carry propagation [9]. When an effective subtraction is performed, the conditional upper inverter carries out a bit-wise inversion of the m MSB of the significand (that is, the representative form). On the other hand, when an effective addition is performed, the conditional inverter does not modify

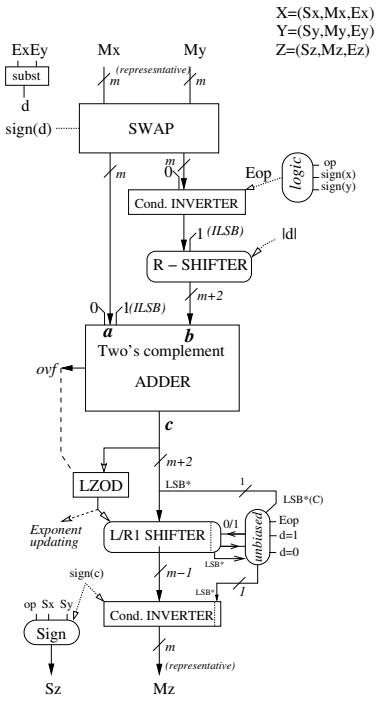


Fig. 4. Unbiased HUB FP adder: architecture A++

the data. At the input of the R-shifter, the value ILSB=1 is concatenated to form the right operand to the adder ($m+2$ bits) which represents the two's complement of the number in case of a subtraction.

For the case of aligned addition (d=0, Eop=0), the hardware required to obtain unbiased results is the same as the described in adder A+. This hardware is enclosed in the logic box entitled "unbiased" in Fig. 4. This "unbiased" box also contains the logic required to manage the tie condition produced in the case of performing a subtraction when d=1 (case d=1, Eop=1). To avoid this source of bias, we propose the following strategy: if the LSB* of the result is 0 (before shifting), we will carry out a rounding up, whereas if it is 1, we will perform a rounding down (at first, the value of the LSB* of the result may be 0 or 1 with the same probability).

The rounding up is achieved easily by truncation after shifting (regular case). For the rounding down, we follow a simple algorithm in the left shifting process. It consists of inserting a first bit 0 and the rest of bit 1 (that is, in the shifting we follow the pattern 0111...). This is shown in the next example:

```
  01.00000 0100(1) [operational form]
- 00.11111 1000(1)1 [d=1]
-----------------
  00.00000 1011(1)1 [exact result, not normal.]

 ALGORITHM for normalization and rounding down
 Shifts step by step:
 <--
  00.00001 0111(0) [1st shift, 0 inserted]
 <--
  00.00010 1110(1) [2nd shift, 1 inserted]
 <--
  00.00101 1101(1) [3rd shift, 1 inserted]
 <--
  00.01011 1011(1) [4th shift, 1 inserted]
 <--
  00.10111 0111(1) [5th shift, 1 inserted]
 <--
  01.01110 1111(1) [6th shift, 1 inserted]

 FINAL NORMALIZED HUB NUMBER (AFTER ROUNDING):
  01.01110 1111(1)
 EXACT RESULT (NOT ROUNDED, normalized)
  01.01111 0000(0)
```

Hence, if "d=1", and the effective operation is 1, the box "unbiased" carries out the injection of the corresponding sequence depending on the value of LSB*(c) (c is output of the two's complement adder of Fig. 4). Otherwise, it always inserts 0.

For the case of aligned subtraction (d=0, Eop=1), we can prove that depending on the relative position of the operands in architecture A++ (U and V), an effective rounding up or rounding down is performed, which means no bias is produced (assuming that the input data has the same probability of being placed anywhere left or right). This means that $U+(-V)$ involves a rounding up whereas $(-V)+U$ involves a rounding down (considering the same probability for the events $U+(-V)$ and $(-V)+U$).

Let us show a simple example of 5-bit significands over the architecture A++. Assume two aligned HUB numbers U and V with significands $M_u = 01.1010(1)$ and $M_v = 01.0100(1)$ (bit sign included) and exponent 0 for both numbers. We want to perform the operation $U+(-V)$. First of all, the exact result of this operation is $01.10000 \;\; exp-2$ (tie condition), which is not a HUB number and it is just between the next two HUB ERNs: $01.01111 \;\; exp-2$ and $01.10001 \;\; exp-2$. Let us follow the evolution of the subtraction of $U+(-V)$ for the architecture A++ in Fig. 5 in two different ways: i) $M_u+(-M_v)$ (Fig. 5.a) and ii) $(-M_v)+M_u$ (Fig. 5.b).

i) $M_u+(-M_v)$. In this case, $M_x = M_u$ (left input at Fig. 5.a) and $M_y = M_v$ (right input), $S_x = S_y = 0$ and op=1 (Eop=1). Thus, the left and right inputs of the 2's complement adder are $01.1010(1)$ and $10.1011(1)$ respectively. The output $c = 00.0110(0)$ and the result after the L Shifter module is $01.1000(0) \;\; exp-2$ (exact value, sign bit included). Since the result is positive, the conditional inverter module does not modify the value, resulting in the final HUB ERN $01.1000(1) \;\; exp-2, \;\; sign = 0$ which involves a positive bias of $0.00001 \;\; exp-2$ (rounding up).

```
Mu+(-Mv)
  01.1010(1) [operational form]
- 01.0100(1)
-------------
  00.0110(0)
 <--
  01.1000(0) [tie case]
  01.1000(1) [Rounded up]
```

ii) $(-M_v)+M_u$. In this case, $M_x = M_v$ (left input at Fig. 5.b) and $M_y = M_u$ (right input), $S_x = 1, S_y = 0$ and op=0 (Eop=1). Thus, the left and right inputs of the 2's complement adder are $01.0100(1)$ and $10.0101(1)$

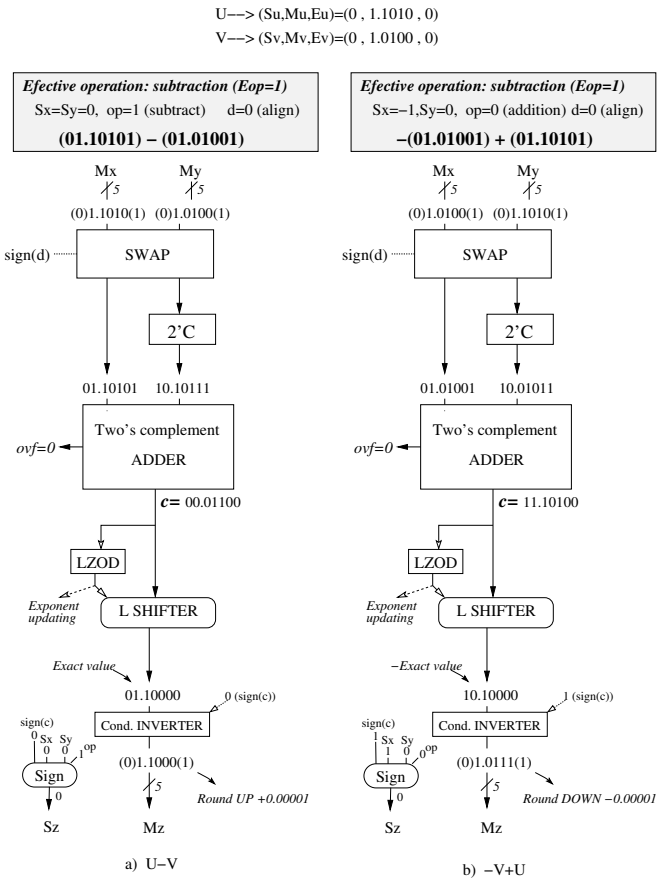


Fig. 5. Example of U+V and (-V)+U for two aligned numbers for effective aligned subtraction

respectively. The output $c = 11.1010(0)$ and the result after the L Shifter module is $10.1000(0)$ $exp - 2$ (exact value, sign bit included). Since the result is negative, the conditional inverter module inverts the bits, resulting in the HUB number $01.0111(1)$ $exp = -2$, $sign = 0$ which involves a negative bias of 0.00001 $exp - 2$ (rounding down).

```
(-Mv)+Mu
 -01.0100(1) [operational form]
+ 01.1010(1)
-------------
  11.1010(0)
 <--
  10.1000(0) [tie case]
  01.0111(1) [Rounded down]
```

Thus, $U + (-V)$ produces a rounding up and $(-V) + U$ produces a rounding down of the same magnitude.

In Table I we show the direction of the rounding as a function of the position of the operands for effective aligned subtraction. From the analysis of this table we deduce that a rounding up(down) is obtained if the left operand is positive(negative).

As a conclusion, depending on the position of the operands, a rounding up or down is performed. Since we do not know the position of the operands a priori, we do not know the direction of the final rounding (up or down) which ensures that the output is not biased (assuming the same probability of having operations $U + (-V)$ and $(-V) + U$).

TABLE I
DIRECTION OF THE BIAS AS A FUNCTION OF THE POSITION OF THE OPERANDS FOR EFFECTIVE ALIGNED SUBTRACTION

sign(A) Left operand	op	sign(B) Right operand	Eop	Rounding ↑↓
+	+	-	-	↑
+	-	+	-	↑
-	+	+	-	↓
-	-	-	-	↓

On the other hand, in the rare case in which we a priori know the input pattern, another alternative solution can be given by the swap module of Fig. 4. When d=0 and a subtraction is carried out, this module can swap the inputs depending on the value of the LSB* of the input (as performed in the classic case of unbiased systems), achieving the desired unbiased rounding. Note that this case of bias can not be prevented in the architecture A since the operation U+(-V) is always performed with the greatest absolute number of the left input of the adder due to the comparator.

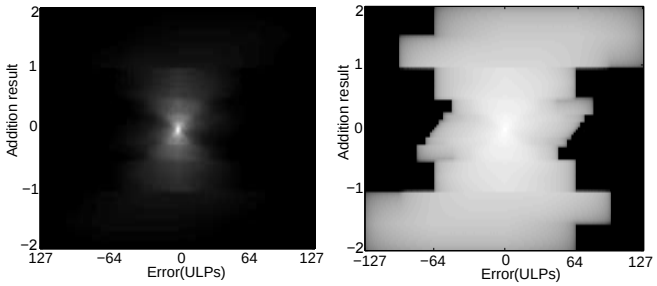
V. UNBIASED ROUNDING FOR A FMA DATA-PATH

Nowadays, addition is performed through a Fused Multiply-Add (FMA) data-path in many processors. This case is analyzed in this section.

The addition operation in a FMA unit may occur in two different cases: from a real FMA operation involving three operands (case 1) or from an actual addition operation involving only two operands (case 2). In case 1, the result of the multiply operation (double size) is added to the other significand. In case 2, one of the multiplier input is set to one and the addition is between two operands of the same size just as in an isolated addition operation.

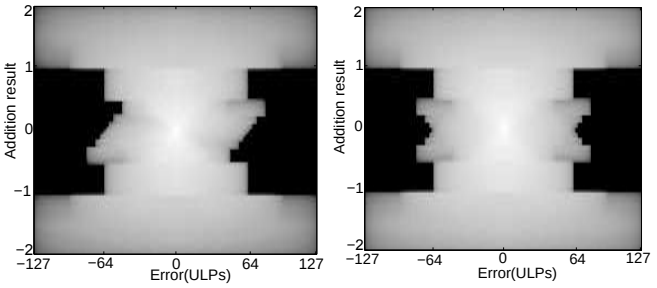
Regarding HUB numbers, in case 1, there is only one possible source of bias when rounding. Since operands have different sizes, and because of the ILSB, there is almost always a bit set to 1 among the discarded bits when truncating. The only different situation is when a catastrophic cancellation occurs, and the result of addition has to be left shifted more bits than the size of the significand and, as a consequence, zeros are introduced to the right. However, since the result of addition may be either positive or negative, this is the same situation as in effective aligned subtraction (case d=0, Eop=1). Therefore, the negative sign and the possible bias is corrected simply by bit-inverting the shifted value when it is negative. No other possible source of bias exists for actual FMA operation.

Regarding case 2 due to the ILSB, this operation has to be treated as a special "simply addition" operation. This may be accomplished by either not including the ILSB at one of the input of the multiplier, or bypassing the multiplier. In both cases, this is a regular addition and bias sources are exactly the same as described in previous section. Hence, these bias can be prevented by using the same pieces of logic as in architecture A++ (see subsection IV-B). We have to note that the FMA left-shifter has triple size and the result of addition is situated in the middle third of it in catastrophic cancellation case. Therefore, the sequences (0111...) or (1000...) described in the previous



(a) Lineal (b) Log

Fig. 6. Histogram of the error when using A architecture: linear scale (a) and logarithmic scale (b)



(a) A+ architecture (b) A++ architecture

Fig. 7. Histogram of the error (logarithmic scale) when using: A+ architecture (a) and A++ architecture (b)

section have to be inserted in the lower third of the shifter input when a "simply addition" operation is performed.

VI. Experimental error analysis

Several numerical experiments have been carried out to test the effectiveness of the architecture presented in this paper. In these experiments, we utilized 32 bits two's complement fixed-point format within the range $(-2,\ 2)$, i.e., the format has one sign bit, one integer bit, and 30 fractional bits. Input operands were pseudo-randomly generated within the range $(-1,\ 1)$ by using C. These operands were associated in pairs and added producing a 32-bit fixed-point result for each pair. These results are always in the range $(-2,\ 2)$ (overflow may not occur due to the selected range for input operands). These fixed-point values are the reference values; the input operands were also converted into 32-bit HUB FP format with only 24 bits for the significand; those HUB numbers were added utilizing each of the HUB architectures (A, A+ and A++); after the addition, the HUB FP results were converted back to 32-bit fixed-point format. The error of the results obtained through HUB FP arithmetic compared to the one obtained directly in fixed-point arithmetic was studied.

Fig. 6 shows the three-dimensional histogram of one billion additions performed using the basic adder A, where white represents the maximum occurrence and black the minimum one. The X-axis represents the error of the HUB results in ULPs, whereas Y-axis represents the value of the exact addition. Although the errors are bounded by ± 127 ULPs, in Fig. 6(a) it is observed that most of the errors are concentrated between ± 64. Looking carefully, certain asymmetry around the center can be perceived. To observe more easily the extreme cases, Fig. 6(b) shows the logarithm of the frequency, instead of the frequency itself. Two different zones of asymmetry between positive and negative results can be seen: one for results bounds by ± 0.5 which mostly corresponds to case of subtraction and d=0 or d=1; another for significand results greater than 1, corresponding to effective additions. In both cases, positive results tend to have positive errors and the opposite for negative ones.

TABLE II
Statistical parameters of rounding error distribution.

Parameters:	mean	σ	mean(+)	mean(-)
A	1.86e-13	2.85e-08	5.43e-09	-5.43e-09
A+	-9.31e-14	2.85e-08	1.23e-09	-1.23e-09
A'++	4.66e-13	2.85e-08	4.25e-09	-4.25e-09
A++	-3.73e-13	2.85e-08	-1.68e-12	2.42e-12

Fig. 7(a) shows the results obtained using A+. As expected, it is easily seen that the asymmetry at the top and bottom of the image has disappeared whereas the one at the center remains. A+ is designed to avoid only the bias for the aligned addition by allowing negative rounding errors for these cases. However, A++ eliminates all the different sources of bias, as can be seen in Fig. 7(b), where positive and negative results have the same probability of having a positive or negative errors.

A more quantitative analysis is provided in Table II, which shows several statistical parameters obtained in these experiments for each architecture. To study the influence of the different cases of bias during rounding, this table includes also a new architecture, A'++, which eliminates only the cases of bias due to subtractions (when d=0 and d=1), instead of only eliminating the bias due to aligned addition (when d=0) as architecture A+ does. As expected, the mean of the whole set of numbers is always practically zero and the difference between different architectures is negligible. Similarly, the standard deviation is identical for all architecture which indicates that the "amount" of errors remains the same through the different architectures. However, the differences appear when considering the error for only positive or only negative results (the two last columns respectively). For the basic architecture A [11], the mean of positive results has a slight positive bias and the opposite for negative ones. This bias is only slightly reduced by A'++ and much more clearly by A+. It is seen that eliminating the bias produced by the aligned addition has much more impact than eliminating the bias produced by subtraction. When both possible cases of bias are removed by utilizing A++ the means drop to typical values.

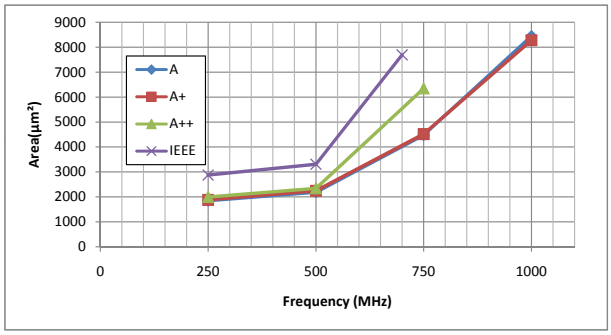
VII. Implementation Results

The architectures A, A+ and A++ have been implemented using VHDL and synthesized with Synopsys Design Compiler H-2013.03-SP2 and the TSMC 65nm library with typical-case operating conditions in which the temperature is 25C and voltage Vdd = 1.0V. The combinational 32-bit version of these architectures has been synthesized for the same target clock frequencies and the area and power consumption results gathered for comparison.

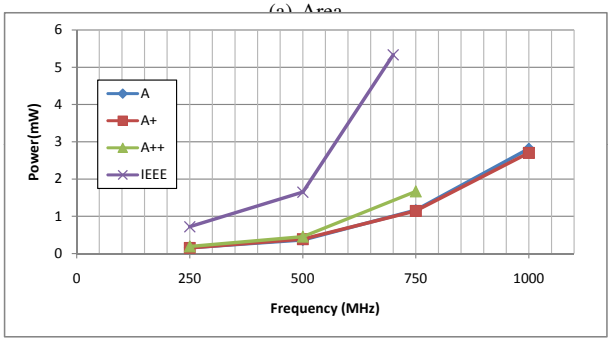
Fig. 8 shows the area and power consumption results for the three adder architectures. A conventional adder fulfilling the IEEE default rounding mode has been added as a reference. Let remember the difference among them: adder A is the basic HUB adder [11] which may produce biased results when rounding; adder A+ can produce unbiased results only for the case of aligned addition (when d=0 and Eop=0) but does not resolve the cases of biased results when subtracting and d=0 or d=1; and adder A++ which performs rounding without introducing bias.

As it can be seen, the cost of having an architecture able to produce unbiased or partially unbiased results (A++ and A+, respectively) is small for low frequencies. In more detail, the area increase to eliminate the bias due to aligned addition (A+) is below 3% and even negative (i.e. there is an area reduction) for 1 GHz. Similarly, there is an improvement in power consumption, except for 500MHz where it increases about 5%, as shown in Fig. 8(b). Regarding the area increase to produce unbiased results (A++), it ranges between 7% and 41%. However, as it is shown in Fig. 8(b), regarding the power consumption, these bounds significantly increase to 23% and 43%, respectively. These ratios should be lower for a FMA, since the added logic is very similar but the original resources are much higher.

We have to note that the source of bias due to the tie condition in aligned addition (d=0, Eop=0) is much more usual than the other sources of bias studied in this paper. Taking this into account, the partial unbiased adder A+ prevents much bias involving a very moderate increase in the hardware cost. This is a good candidate for applications where bias is not an important issue. However, for applications where the prevention of bias is mandatory, the unbiased adder A++ comes at a moderate increase of cost compared to the biased adder A.



(a) Area



(b) Power consumption

Fig. 8. Implementation results: Area (a) and power consumption (b)

VIII. Conclusion

In this paper, we study the different sources of bias when rounding after FP HUB addition and FMA. We present and compare two architectures to deal with these sources of bias. From the error analysis and the implementation results we conclude that the adder A++ is a solution (with a moderate increased cost compared to biased adder A) for applications where the prevention of the bias is mandatory. On the other hand, the partial unbiased architecture A+ should be generally used since it prevents much bias and involves a negligible increase in the hardware cost. Similar solutions could be applied to FMA architectures.

Acknowledgments

This work has been supported in part by the following Spanish projects: TIN2013-42253-P, TIN2016-80920-R, JA2012 P12-TIC-1692.

References

[1] E. Alpaydin, *Machine learning : the new AI*, 2016.
[2] M. Kubat, *An Introduction to Machine Learning*, 2015, no. August.
[3] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, feb 2014.
[4] F. K. Shaikh, S. Zeadally, and E. Exposito, "Enabling technologies for green Internet of Things," *IEEE Systems Journal*, vol. PP, no. 99, pp. 983–994, jun 2015.
[5] R. Murphy, T. Sterling, and C. Dekate, "Advanced architectures and execution models to support green computing," *Computing in Science and Engineering*, vol. 12, no. 6, pp. 38–47, nov 2010.
[6] P. P. Pande, A. Ganguly, and K. Chakrabarty, *Design technologies for green and sustainable computing systems*, P. P. Pande, A. Ganguly, and K. Chakrabarty, Eds. New York, NY: Springer New York, 2013.
[7] Synopsys, "DWFC Flexible Floating Point Overview," *https://www.synopsys.com/dw/doc.php/doc/dwfc/datasheets/dwfc__overview_ffp.pdf*, no −6, 2016.
[8] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1 –58, 29 2008.
[9] J. Hormigo and J. Villalba, "New formats for computing with real-numbers under round-to-nearest," *IEEE Transactions on Computers*, vol. 65, no. 7, pp. 2158–2168, 2016.
[10] D. R. Lutz and N. Burgess, "Overcoming double-rounding errors under IEEE 754-2008 using software," in *2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*. IEEE, nov 2010, pp. 1399–1401.
[11] 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, 2016.
[12] H. L. Garner, "A survey of some recent contributions to computer arithmetic," *IEEE Transactions on Computers*, vol. C-25, no. 12, pp. 1277–1282, Dec 1976.
[13] W. J. Cody, "Static and dynamic numerical characteristics of floating-point arithmetic," *IEEE Transactions on Computers*, vol. C-22, no. 6, pp. 598–601, June 1973.
[14] J. F. Reiser and D. E. Knuth, "Evading the drift in floating-point addition," *Information Processing Letters*, vol. 3, no. 3, pp. 84–87, jan 1975.
[15] H. Hilton, "Bias-free rounding in digital signal processing," Apr. 22 2004, uS Patent App. 10/277,419. [Online]. Available: https://www.google.com/patents/US20040078401
[16] J. Villalba-Moreno, "Digit recurrence floating-point division under HUB format," *23rd IEEE Symposium on Computer Arithmetic, Silicom Valley (California, USA)*, July 2016.
[17] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, San Francisco, 2004.