A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication

Guy Even and Peter-Michael Seidel

Abstract—A new IEEE compliant floating-point rounding algorithm for computing the rounded product from a carry-save representation of the product is presented. The new rounding algorithm is compared with the rounding algorithms of Yu and Zyner [26] and of Quach et al. [17]. For each rounding algorithm, a logical description and a block diagram is given, the correctness is proven, and the latency is analyzed. We conclude that the new rounding algorithm is the fastest rounding algorithm, provided that an injection (which depends only on the rounding mode and the sign) can be added in during the reduction of the partial products into a carry-save encoded digit string. In double precision format, the latency of the new rounding algorithm is 12 logic levels compared to 14 logic levels in the algorithm of Quach et al. and 16 logic levels in the algorithm of Yu and Zyner.

Index Terms—Floating-point arithmetic, IEEE 754 Standard, floating-point multiplication, IEEE rounding.

1 Introduction

Every modern microprocessor includes a floating-point (FP) multiplier that complies with the IEEE 754 Standard [9]. The latency of the FP multiplier is critical to floating-point performance since a large portion of the FP instructions consists of FP multiplications. For example, Oberman reports that FP multiplications account for 37 percent of the FP instructions in benchmark applications [13].

Floating point multipliers perform the computation in two phases. In the first phase, an addition tree reduces the partial products to a carry-save encoded digit string that represents the exact product. In the second phase, a binary string representing the rounded product is computed from the carry-save encoded string. A lot of research has been devoted to optimizing the latency of adding the partial products to produce the carry-save encoded product, e.g., [1], [2], [5], [11], [12], [14], [15], [16], [21], [23], [24], [25]. More recently, work on rounding the product according to the IEEE 754 Standard has been published [7], [17], [18], [19], [26], [28]. Assuming that the multiplier outputs a carry-save encoded digit string representing the exact product, the following natural question arises: What is the fastest method to compute the rounded product given the exact product represented by carry-save encoded digit string?

We consider and compare three rounding algorithms: 1) the algorithm of Quach et al. [17], which we denote as the QTF algorithm; 2) the algorithm of Yu and Zyner [26], which we denote as the YZ algorithm; and 3) a new algorithm that is based on injection-based rounding [7], which we denote as the ES algorithm. We provide block diagrams of these rounding algorithms, optimized for speed. We measure the latency of the algorithms in a unified fashion, using logic levels to enable technology independent comparisons. The main building blocks of these algorithms are similar and consist of a compound adder and the computation of a sticky and carry bits. Thus, the costs of the three algorithms are similar and the interesting question is finding the fastest algorithm.

We focus on double precision multiplication in which each significand is represented by 53 bits. The algorithms assume that the significands are normalized, namely, in the range $[1, 2)$, and, therefore, their product is in the range $[1, 4)$. We do not consider the cases that deal with denormal or special values since supporting denormal values can be obtained by using an extended exponent range [10], [19], [27] and the computation on special values can be done in parallel [8]. A full design of a floating-point unit that includes a discussion of the internal formats, supporting denormal values, special values, and exceptions appears in [20].

The three algorithms share the following techniques:

1. The product represented by a carry-save encoded digit string of 106 digits in the case of double precision is partitioned into a lower part and an upper part. The upper part is added by a compound adder that computes the binary representations of sum and $sum + ulp$, where ulp denotes a unit in the last position and sum denotes the sum of the upper part. A carry-bit, a round-bit, and a sticky-bit are computed from the lower part.
2. The rounding decision is computed in two paths: The nonoverflow path works under the assumption that the exact product is in the range $[1, 2)$ and the overflow path works under the assumption that the product is in the range $[2, 4)$. Although the sum of the upper part, denoted by sum, does not equal the exact product, the most significant bit of sum controls the selection between these two paths.

• G. Even is with the Department of Electrical Engineering Systems, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel.
E-mail: guy@eng.tau.ac.il.
• P.-M. Seidel is with the Computer Science and Engineering Department, Southern Methodist University, Science Information Center, Dallas, TX 75275. E-mail: seidel@seas.smu.edu.

*Manuscript received 1 Sept. 1999; revised 1 Feb. 2000; accepted 10 Mar. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111785.*

We outline the main differences between the three rounding algorithms:

1. The rounding decision. The QTF and ES algorithms simplify the rounding decision by an early addition of a value (this value is called the *prediction* in the QTF algorithm and the *injection* in the ES algorithm). In the QTF algorithm, the prediction depends on the rounding mode and on the carry-save digit positioned 53 digits to the right of the radix point. In the ES algorithm, the injection depends only on the rounding mode and we assume that it is added in with the partial products and, thus, the product already includes the injection. The rounding decision in the YZ algorithm is based on customary rounding tables.
2. The position in which the carry-save encoded product is partitioned into a lower part and an upper part differs in the three algorithms. In the YZ algorithm, the lower and upper parts are separated by a "buffer" of three carry-save digits in positions $[51:53]$, where the position of a digit denotes how many digits it is to the right of the radix point. In the other two algorithms, the upper part consists of positions $[-1:52]$, and the lower part consists of positions $[53:104]$.

The latencies of the proposed designs that implement these algorithms in terms of logic levels are the following: The latency of the ES algorithm is 12 logic levels, the latency of the QTF algorithm is 14 logic levels, and the latency of the YZ algorithm is 16. Note that we modified and adapted the QTF and YZ algorithms for minimum latency.

Supporting all four rounding modes of the IEEE 754 Standard is an error prone task. We therefore provide correctness proofs of all three algorithms which formalize and clarify the difficult aspects. From this point of view, the YZ algorithm is easiest to prove and the QTF algorithm is the most intricate (especially the rounding decision logic).

The paper is organized as follows: In Section 2, preliminary issues are described, such as: notation, conventions we use regarding IEEE rounding, and the general setting. In Section 3, a straightforward rounding algorithm is reviewed. This algorithm is described to provide an outline of the task of rounding after the exact product is computed. It does not attempt to parallelize the task of rounding and, therefore, has a long latency. In Sections 4-6, each rounding algorithm is described, proven, and analyzed. In Section 7, we discuss how the latencies of the algorithms increase as the precision is increased. In Section 8, a summary and conclusion is given.

2 Preliminaries

2.1 Notation

Let $x_i x_{i+1} \cdots x_j \in \{0,1\}^*$ denote a binary string. By $x[z_1:z_2]$, we denote the binary string $x_{z_1} x_{z_1+1} \cdots x_{z_2}$. We also sometimes refer to x_i as $x[i]$. Since we deal with fractions, we index binary encoded bit strings by $x_0.x_1x_2\ldots$ so that x_i is associated with the weight 2^{-i}. The value encoded by $x[z_1:z_2]$ is denoted by $|x[z_1:z_2]|$ and equals

original rounding mode	sign	reduced rounding mode
round to $+\infty$	$+$	RI
round to $+\infty$	$-$	RZ
round to $-\infty$	$+$	RZ
round to $-\infty$	$-$	RI

Fig. 1. Rounding mode reduction based on the sign.

$$|x[z_1:z_2]| = \sum_{i=z_1}^{z_2} x_i \cdot 2^{-i}.$$

Boolean operators are often denoted as follows: $\vee$ denotes an OR, $\wedge$ denotes an AND, and $\oplus$ denotes an XOR.

2.2 IEEE Rounding

The IEEE-754-1985 Standard defines four rounding modes: round toward 0, round toward $+\infty$, round toward $-\infty$, and round to nearest (even). In Fig. 1, a reduction of the round toward $+\infty$ and round toward $-\infty$ rounding modes to the rounding modes RZ (round to zero) and RI (round to infinity) is depicted. This reduction is based on the sign of the number and follows Quatch et al. [17]. This reduction leaves only three rounding modes: RI, RZ, and RNE (round to nearest-even). Furthermore, Quach et al. [17] suggested implementing RNE by round to nearest (up), denoted by RNU. The rounding mode RNU is defined as follows: If x is between two successive representable numbers $y_1 \le x < y_2$, then

$$r_{RNU}(x) = \begin{cases} y_1 & \text{if } x < (y_1+y_2)/2 \\ y_2 & \text{otherwise.} \end{cases}$$

The reason that RNE can be implemented by RNU is that $r_{RNU}(x) \neq r_{RNE}(x)$ iff $x = (y_1+y_2)/2$ and the least significant bit (LSB) of the binary encoding of y_2 is 1. Therefore, obtaining $r_{RNE}(x)$ from $r_{RNU}(x)$ can be accomplished by "pulling down" the LSB when $x = (y_1+y_2)/2$.

For the sake of clarity, we define round to zero (RZ) and round to infinity (RI) of significands in the range $[1,4)$ in double precision. Note that this definition excludes the postnormalization shift that takes place when the number is in the binade $[2,4)$. We also ignore the overflow exception that can occur in RI. The definition uses the notation $\lfloor x \rfloor$ for the integer floor function and $\lceil x \rceil$ for the integer ceiling function.

Definition 1. *Let $x \in [1,4)$, then $r_{RZ}(x)$ and $r_{RI}(x)$ are defined by*

$$r_{RZ}(x) = \begin{cases} \lfloor \frac{x}{2^{-52}} \rfloor \cdot 2^{-52} & \text{if } x \in [1,2) \\ \lfloor \frac{x}{2^{-51}} \rfloor \cdot 2^{-51} & \text{if } x \in [2,4). \end{cases}$$

$$r_{RI}(x) = \begin{cases} \lceil \frac{x}{2^{-52}} \rceil \cdot 2^{-52} & \text{if } x \in [1,2) \\ \lceil \frac{x}{2^{-51}} \rceil \cdot 2^{-51} & \text{if } x \in [2,4). \end{cases}$$

2.3 General Setting

In this paper, we consider a double precision multiplier. We assume that the significands are normalized, namely, that the values of the two significands are in the range $[1,2)$ and that each significand is represented by a binary string with bits in positions $[0:52]$. The exact product of the two significands is in the range $[1,4)$ and is encoded by a binary string with bits in positions $[-1:104]$. (Note that the weight of the bit in position $[-1]$ is 2.) For the sake of simplicity, we ignore the exponent and sign-bit paths; a full description of a floating-point multiplier, including the exponent and sign-bit paths, appears in [20].

3 Naive IEEE Rounding

In this section, we review a simple but slow IEEE compliant algorithm for rounding after multiplication [4].

3.1 Description

The input consists of two binary strings SUM and CARRY, each having 106 bits which are indexed from -1 to 104. The sum of the binary numbers represented by SUM and CARRY equals the exact product $\mathrm{EXA} \in [1,4)$.

Rounding is computed as follows (the computation of the exponent string is omitted):

1. Reduce the rounding mode to one of three rounding modes based on the sign of the product.
2. 2:1-compression. The SUM and CARRY strings are added to obtain a single binary string $X[-1:104]$, namely $|X| = |\mathrm{SUM}| + |\mathrm{CARRY}|$. Note that, since the exact product is in the range $[1,4)$, the most significant bit of X is in position $[-1]$.
3. Normalization. If $|X| \geq 2$, then $|X'| = |X|/2$, otherwise $|X'| = |X|$. This is implemented by a conditional shift by at most one position to the right. Note that X' is indexed from 0 to 105.
4. Compute sticky. The sticky-bit equals
$$\mathrm{OR}(X'[54], X'[55], \ldots, X'[105]).$$
5. Compute rounding decision. The rounding decision $\mathrm{RD} \in \{0,1\}$ is based on the rounding mode, the bits, $L = X'[52]$, $R = X'[53]$, and the sticky-bit. Note that the rounding mode at this stage already incorporates the sign.
6. Increment. Compute $|X'[0:52]| + \mathrm{RD} \cdot 2^{-52}$. Let $Y[-1:52]$ be the binary string that represents the sum. Note that the increment may result with an overflow, namely, $|Y| = 2$.
7. Postnormalize. If $|Y| = 2$, then $|Y'| = 1$, otherwise $Y' = Y[0:52]$.

The significand of the rounded product is given by Y'.

3.2 Delay Analysis

The latency of Steps 2, 4, and 6 of the naive rounding procedure is at least logarithmic in the length of the binary strings SUM and CARRY. The other steps require only constant delay. If every pipeline stage can accommodate at most one logarithmic depth circuit, then an implementation of the naive rounding procedure requires at least three pipeline stages.

4 The ES Rounding Algorithm

In this section, we review injection-based rounding [7] and present an implementation for double precision that requires (under assumptions specified in Section 4.5) only 12 logic levels.

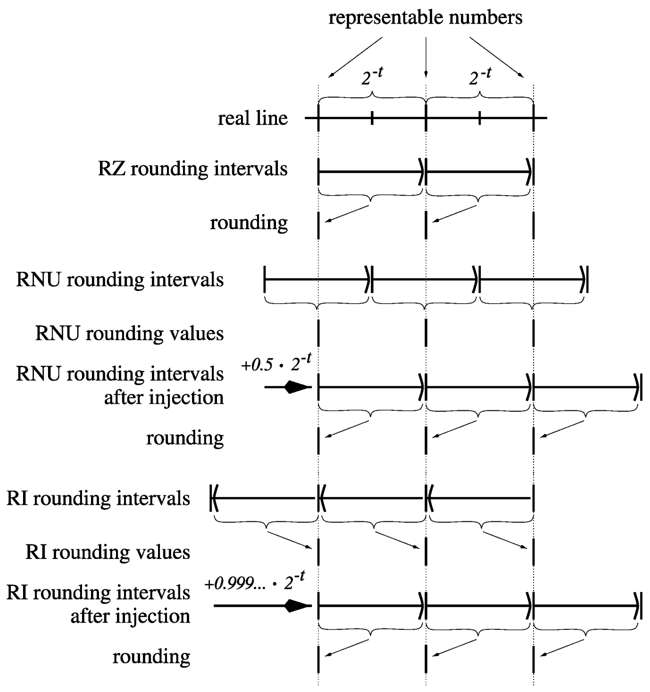

Fig. 2. (a) Reducing RNU to RZ with an injection; (b) reducing RI to RZ with an injection.

4.1 Injection-Based Rounding

Rounding by injection reduces the rounding modes RI and RNU to RZ [7]. The reduction is based on adding an injection that depends only on the rounding mode, as follows:

$$\mathrm{INJ} = \begin{cases} 0 & \text{if RZ} \\ 2^{-53} & \text{if RNU} \\ 2^{-52} - 2^{-104} & \text{if RI.} \end{cases}$$

The effect of adding INJ is summarized in the following equation:

$$X \in [1,2) \quad \Rightarrow \quad round_{mode}(X) = round_{RZ}(X + \mathrm{INJ}), \quad (1)$$

where $mode \in \{RZ, RNU, RI\}$.

Fig. 2 depicts the reduction of RNU and RI to RZ by injection assuming that the number to be rounded is in the range $[1,2)$.

If the exact product, denoted by EXA, is in the range $[2,4)$, then the injection fails to reduce all the rounding modes to RZ. To fix the reduction, an injection correction amount is added. The injection correction amount, denoted by INJCOR, is defined by

$$\mathrm{INJCOR} = \begin{cases} 0 = 0 - 0 & \text{if RZ} \\ 2^{-53} = 2^{-52} - 2^{-53} & \text{if RNU} \\ 2^{-52} = 2^{-51} - 2^{-104} - (2^{-52} - 2^{-104}) & \text{if RI.} \end{cases}$$

Therefore, if X is in the range $[2,4)$, the effect of adding the injection and the correction amount is summarized in the following equation:

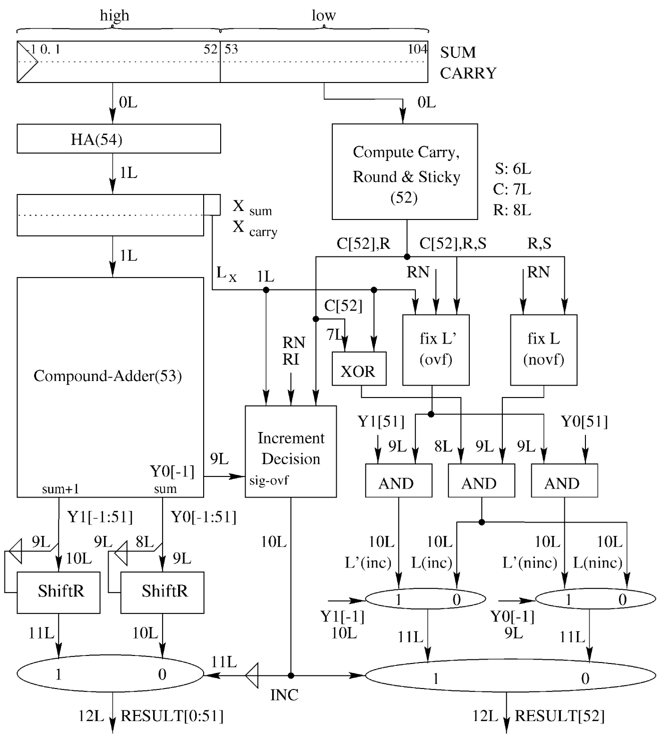

Fig. 3. Block diagram of the ES rounding algorithm annotated with timing estimates ("5L" next to a signal means that the signal is valid after five logic levels).

$$X \in [2,4) \Rightarrow round_{mode}(X) = round_{RZ}(X + \text{INJ} + \text{INJCOR}) \tag{2}$$

where $mode \in \{RZ, RNU, RI\}$.

We assume that INJ is already added in the multiplier adder array. As discussed later, in Section 8, this assumption is motivated by the potential of advancing the addition of the injection to the multiplier array with a very small slowdown of the multiplier array. The early addition of the injection causes the carry-save output of the multiplier array to satisfy:

$$|\text{SUM}| + |\text{CARRY}| = \text{EXA} + \text{INJ}.$$

This completes the description of injection-based rounding for numbers in the range $[1,4)$.

4.2 The Rounding Algorithm

In this section, we present the new ES algorithm for rounding in floating-point multiplication that is based on injection-based rounding.

Fig. 3 depicts a block diagram of the ES rounding algorithm. The rounding algorithm works under the assumption that the SUM and CARRY-strings already include the injection (but not the injection correction) and proceeds as follows:

1. The SUM and CARRY-strings are divided into a high part and a low part. The high part consists of positions $[-1:52]$ and the low part consists of positions $[53:104]$.
2. The low part is input to the box that computes the carry, round, and sticky-bits, defined as follows:

$$C[52] = \sigma_{52}$$
$$R = \sigma_{53}$$
$$S = \text{OR}(\sigma_{54}, \sigma_{55}, \ldots, \sigma_{104}),$$

where $\sigma_{52} \cdots \sigma_{104}$ is the binary string that satisfies:

$$|\sigma_{52} \cdots \sigma_{104}| = |\text{SUM}[53:104]| + |\text{CARRY}[53:104]|.$$

3. The higher part is input to a line of Half Adders and produces the output $(X_{sum}[-1:51], L_X)$ and $X_{carry}[-1:51]$. Note that the bit L_X is in position 52 and that no carry is generated to position -2 because the exact product is less than 4 (even after adding the injection).
4. $X_{sum}[-1:51]$ and $X_{carry}[-1:51]$ are input to the Compound Adder, that outputs the sum $Y0[-1:51]$ and the incremented sum $|Y1[-1:51]| = |Y0[-1:51]| + 2^{-51}$.
5. The Increment Decision box receives the round-bit (R), the carry-bit ($C[52]$), the LSB (L_X), the MSB ($Y0[-1]$), and the rounding modes (RN, RI). The output signal INC indicates whether $Y0$ or $Y1$ is to be selected.
6. The most significant bits $Y0[-1]$ and $Y1[-1]$ indicate whether $Y0$ and $Y1$ are in the range $[2,4)$. Depending on these bits, $Y0$ and $Y1$ are normalized as follows:

$$Z0[0:51] = \begin{cases} Y0[0:51] & \text{if } Y0[-1]=0 \\ shift\_right(Y0[-1:50]) & \text{if } Y0[-1]=1 \end{cases}$$
$$Z1[0:51] = \begin{cases} Y1[0:51] & \text{if } Y1[-1]=0 \\ shift\_right(Y1[-1:50]) & \text{if } Y1[-1]=1. \end{cases}$$

7. The rounded result (except for the least significant bit) is selected between $Z0$ and $Z1$ according to the increment decision INC, as follows:

$$\text{RESULT}[0:51] = \begin{cases} Z0[0:51] & \text{if INC}=0 \\ Z1[0:51] & \text{if INC}=1. \end{cases}$$

8. In case the rounding mode is RNE, the least significant bit needs to be corrected since RNE and RNU do not always result in the same least significant bit. The correction of the least significant bit is computed by two parallel paths; one path working under the assumption that the rounded result overflows (i.e., greater than or equal to 2) and the other path working under the assumption that the rounded result does not overflow.

The path that computes the correction of the LSB under the "no-overflow" assumption is implemented by the box called "fix L (novf)". The inputs of the "fix L (novf)" box are the round bit R, the sticky bit S, and a signal RNE indicating whether the rounding mode is round to nearest even. When the output, denoted by $not(pd)$, equals zero, the LSB should be pulled down.

The path that computes the correction of the LSB under the "overflow" assumption is implemented by the box called "fix L′ (ovf)". The inputs of the "fix L′ (ovf)" box are the L_X bit, the carry-bit $C[52]$, the round bit R, the sticky bit S, and the signal RNE. When the output, denoted by $not(pd')$, equals zero, the LSB should be pulled down.

Note that the pull down signals are inactive if the rounding mode is not RNE.

9. The least significant bit of the rounded result before fixing the LSB (in case of a discrepancy between RNE and RNU) equals one of three values:

   a. If the rounded result does not overflow, then the LSB equals $L_X \oplus C[52]$;
   b. If the rounded result overflows and the increment decision is not to increment, then the LSB equals $Y0[51]$; and
   c. If the rounded result overflows and the increment decision is to increment, then the LSB equals $Y1[51]$.

   The fixing of the LSB is implemented by combining (using AND-gates) the pull-down signals with the corresponding candidates for the LSB signals.

   The outputs of the three AND-gates are denoted by: $L'(inc)$, $L'(ninc)$, and $L(inc)$. For the sake of clarity, we introduce the signal $L(ninc)$, which equals $L(inc)$.

10. The LSB of the rounded result equals $L(ninc)$ if no overflow occurred and no increment took place. The LSB of the rounded result equals $L(inc)$ if no overflow occurred and an increment took place. The LSB of the rounded result equals $L'(ninc)$ if an overflow occurred and no increment took place. The LSB of the rounded result equals $L'(inc)$ if an overflow occurred and an increment took place.

    According to the four cases, the LSB of the rounded result is selected depending on the overflow signals and the increment decision.

4.3 Details

In this section, we describe the functionality of three boxes in Fig. 3 that have not yet been fully described.

Fix L (novf). This box belongs to the path that assumes that the product is in the range $[1, 2)$. Recall that there might be a discrepancy between RNE and RNU when a tie occurs, namely, when the exact product equals the midpoint between two successive representable numbers. Let EXA denote the value of the exact product, the "Fix L (novf)" generates a signal $not(pd)$ that satisfies:

$$\text{EXA} \in [1,2) \Rightarrow (not(pd) = 0 \Leftrightarrow \text{a tie occurs and } RNE).$$

When a tie occurs, there are two possibilities: 1) If RNU and RNE agree, then both yield a rounded result with a LSB equal to zero. Pulling down the LSB in this case is not required, but causes no damage. 2) If RNU and RNE disagree, then the LSB of the RNU result must equal 1 and the LSB of the RNE result must equal 0. Hence, the RNE result can be obtained from the RNU result by pulling down the LSB.

Without the addition of the injection, a tie occurs when $R = 1$ and $S = 0$. Since an injection of 2^{-53} is already included, a tie occurs when $R = 0$ and $S = 0$. Therefore, the $not(pd)$ signal is defined by:

$$not(pd) = \text{OR}(R, S, not(RNE)).$$

Fix L′ (ovf). This box belongs to the path that assumes that the product is in the range $[2, 4)$. The "Fix L′ (ovf)" generates a signal $not(pd')$ that satisfies:

$$\text{EXA} \in [2,4) \Rightarrow (not(pd') = 0 \Leftrightarrow \text{a tie occurs and } RNE).$$

The difference between $not(pd')$ and $not(pd)$ is that $not(pd')$ is used under the assumption that the product is greater than or equal to 2. Without the addition of the injection, a tie occurs (in case of overflow) when $L_X \oplus C[52] = 1$, $R = 0$, and $S = 0$. Since an injection of 2^{-53} is already included, a tie occurs when $L_X \oplus C[52] = 1$, $R = 1$, and $S = 0$. Therefore, the $not(pd')$ signal is defined by:

$$not(pd') = \text{OR}(not(L_X \oplus C[52]), not(R), S, not(RNE)).$$

Increment Decision. The increment decision box has two paths, depending on whether an overflow occurs. The path working under the assumption that no overflow occurs (i.e., $Y0[-1] = 0$) produces an increment decision if $L_X + C[52] = 2$. The path working under the assumption that an overflow occurs (i.e., $Y0[-1] = 1$) needs to take into account the correction of the injection, denoted by INJCOR. It produces an increment decision if $L_X + C[52] + \text{INJCOR} \cdot 2^{52} + R/2 \geq 2$. Therefore, the INC signal is defined by:

$$\text{INC} = \begin{cases} L_X \wedge C[52] & \text{if } Y0[-1] = 0 \text{ or } RZ \\ L_X \vee C[52] & \text{if } Y0[-1] = 1 \text{ and } RI \\ R + L_X + C[52] \geq 2 & \text{if } Y0[-1] = 1 \text{ and } RNE. \end{cases}$$

4.4 Correctness Proof

The difficult part in our algorithm is the correctness of the INC signal. As long as the bit $Y0[-1]$ indicates correctly whether the exact product is greater than or equal to 2, (1) and (2) imply that the INC signal is correct. However, the bit $Y0[-1]$ might not indicate correctly the binade of the exact product. The following two cases are possible: 1) $Y0[-1] = 0$ and the exact product is greater than or equal to 2; and 2) $Y0[-1] = 1$ and the exact product (without the injection) is less than 2.

The source of such errors is due to the fact that $|Y0[-1:51]|$ does not always equal the 53 most-significant bits of the exact product. Recall that:

$$|Y0[-1:51]| = |X_{\text{sum}}[-1:51]| + |X_{\text{carry}}[-1:51]|.$$

Hence, there may be two reasons for a discrepancy between $Y0[-1:51]$ and the 53 most-significant bits of the exact product: 1) The carry-in bit to position $[51]$ is ignored in $Y0[-1:51]$; and 2) the injection is incorporated in the SUM and CARRY strings.

The following claim proves that, even in the presence of mismatches described above, $Y0[-1]$ can be used for controlling which path should be selected: the "no-overflow" path or the "overflow" path. The reason this is true is that, when $Y0[-1]$ fails to indicate the binade of the exact product, the following holds: 1) The rounded result equals 2; and 2) both paths compute the result 2.

Claim 1. *Let EXA denote the exact product and let SUM and CARRY satisfy* $|\mathrm{SUM}| + |\mathrm{CARRY}| = \mathrm{EXA} + \mathrm{INJ}$. *Then, correct rounding of EXA can be computed as follows:*

$$r_{mode}(\mathrm{EXA}) = \begin{cases} r_{RZ}(\mathrm{EXA}+\mathrm{INJ}) & \text{if } \overline{Y0[-1]} \\ r_{RZ}(\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR}) & \text{if } Y0[-1]. \end{cases}$$

Proof. There are two cases to consider: 1) $Y0[-1]=0$ and 2) $Y0[-1]=1$.

1. Suppose $Y0[-1]=0$. If $\mathrm{EXA}<2$, then the claim follows from (1). If $\mathrm{EXA}\geq 2$, then

$$\mathrm{EXA}+\mathrm{INJ} \in [2, |Y0[0:51]| + \varepsilon), \tag{3}$$

with $\varepsilon = 3\cdot 2^{-52}$. The reason for this is the possible contribution of $L_X \cdot 2^{-52} \in \{0, 2^{-52}\}$ and $|\mathrm{SUM}[53:104]| + |\mathrm{CARRY}[53:104]| \in [0, 2^{-51})$.

Since $Y0[-1]=0$, it follows that

$$|Y0[0:51]| \leq 2 - 2^{-51}.$$

Since $|Y0[0:51]| + \varepsilon \geq 2 + 2^{-52}$, we have

$$|Y0[0:51]| \geq 2 - 2^{-51}.$$

Therefore, $|Y0[0:51]| = 2 - 2^{-51}$ and (3) yields

$$\mathrm{EXA}+\mathrm{INJ} \in [2, 2+2^{-52}). \tag{4}$$

The correction of the injection satisfies $0 \leq \mathrm{INJCOR} \leq 2^{-52}$, therefore:

$$\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR} \in [2, 2+2^{-51}). \tag{5}$$

According to (2), in this case, $r_{mode}(\mathrm{EXA}) = r_{RZ}(\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR})$. However, in this case,

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}) = r_{RZ}(\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR}) = 2$$

because rounding to zero maps both intervals $[2, 2+2^{-52})$ and $[2, 2+2^{-51})$ to 2.

2. Suppose $Y0[-1]=1$. If $\mathrm{EXA}\geq 2$, then the claim follows from (2). If $\mathrm{EXA}<2$, then since $\mathrm{INJ} \in [0, 2^{-52})$, it follows that

$$\mathrm{EXA}+\mathrm{INJ} \in [2, 2+2^{-52}). \tag{6}$$

The proof now follows the proof in case 1. □

The following claim proves that our implementation of the computation of $r_{mode}(\mathrm{EXA})$ is correct. Note that the claim does not deal with fixing the LSB to obtain RNE from RNU.

Claim 2.

1. *If* $Y0[-1]=0$, *then*

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}) = \begin{cases} |(Y0[0:51], (L_X \oplus C[52]))| & \text{if } \overline{\mathrm{INC}} \\ |(Y1[0:51], (L_X \oplus C[52]))| & \text{if } \mathrm{INC} \wedge \overline{Y1[-1]} \\ |(Y1[-1:51]| & \text{if } \mathrm{INC} \wedge Y1[-1]. \end{cases}$$

2. *If* $Y0[-1]=1$, *then*

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR}) = \begin{cases} |(Y0[-1:51]| & \text{if } \mathrm{INC}=0 \\ |(Y1[-1:51]| & \text{if } \mathrm{INC}=1. \end{cases}$$

Proof. Suppose $Y0[-1]=0$, then

$$\mathrm{EXA}+\mathrm{INJ} = |Y0[0:51]| + L_X\cdot 2^{-52} + C[52]\cdot 2^{-52} + tail,$$

where $tail \in [0, 2^{-52})$. This implies that

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}) = r_{RZ}(|Y0[0:51]| + L_X\cdot 2^{-52} + C[52]\cdot 2^{-52}).$$

The INC signal in this case equals 1 iff the addition of L_X and $C[52]$ generates a carry to position 51. If $\mathrm{INC}=0$, then simple addition takes place:

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}) = |Y0[0:51]| + L_X\cdot 2^{-52} + C[52]\cdot 2^{-52}.$$

If $\mathrm{INC}=1$, there are two cases: In the first case, the increment does not cause an overflow and, again, simple addition takes place. If an overflow is caused, then, since only 53 bits are output, the bit $L_x \oplus C[52]$ is discarded. This completes the proof of the first part of the claim.

Suppose $Y0[-1]=1$, then $\mathrm{INC}=1$ iff

$$L_X\cdot 2^{-52} + |\mathrm{SUM}[53:104]| + |\mathrm{CARRY}[53:104]| + \mathrm{INJCOR} \geq 2^{-51}.$$

Therefore,

$$\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR} = |Y0[-1:51]| + \mathrm{INC}\cdot 2^{-51} + tail,$$

where $tail \in [0, 2^{-51})$. This implies that

$$r_{RZ}(\mathrm{EXA}+\mathrm{INJ}+\mathrm{INJCOR}) = r_{RZ}(|Y0[-1:51]| + \mathrm{INC}\cdot 2^{-51})$$

and the claim follows. □

4.5 Delay Analysis

In this section, we present a delay analysis of the ES rounding algorithm depicted in Fig. 3. Our analysis is based on the following assumptions:

1. Consider a carry look-ahead adder and let d_{CLA} denote the delay of the 53-bit adder measured in logic levels. We assume that the MSB of the sum has a delay of at most $d_{CLA}-1$ logic levels. This assumption is easy to satisfy if the carry look-ahead adder of Brent and Kung is used [3]. Otherwise, satisfying this assumption may require arranging the parallel-prefix network so that the MSB is ready one logic level earlier.
2. The compound adder is implemented so that the delay of the sum is d_{CLA} and the delay of the

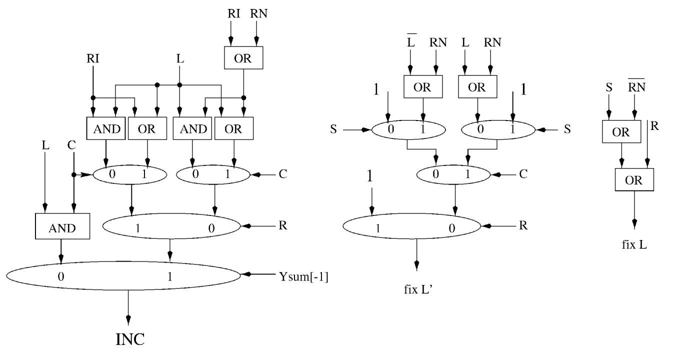

Fig. 4. Details of boxes in the ES rounding algorithm: (a) the Increment Decision box; (b) the fix L' (ovf) box; and (c) the fix L (novf) box.

incremented sum is $d_{CLA}+1$. This can be obtained by ORing the carry-generate and carry-propagate signals [22, Lemma 1].

3. Consider the box in which the carry, round, and sticky bits are computed. According to the first assumption, since the widths of this box and the compound adder are similar, the delay of the carry bit is $d_{CLA}-1$ logic levels and the delay of the round bit is d_{CLA} logic levels. The delay of the sticky bit is estimated to be $d_{CLA}-2$ logic levels, based on the fast sticky bit computation presented in [26].
4. We assume that the delay associated with buffering a fan-out of 53 is one logic level.

Fig. 3 depicts the block diagram of the injection-based rounding algorithm annotated with timing estimates. We assigned d_{CLA} the value of eight logic levels. This implies that the sticky bit is valid after six logic levels, the carry-bit $C[52]$ is valid after seven logic levels, and the round-bit is valid after eight logic levels. Similarly, the sum $Y0$ is valid after nine logic levels, the MSB $Y0[-1]$ is valid after eight logic levels, the incremented sum $Y1$ is valid after 10 logic levels, and the MSB $Y1[-1]$ is valid after nine logic levels.

Fig. 4 depicts implementations of the Fix L (novf), Fix L' (ovf), and Increment Decision boxes. These implementations are used in Fig. 3 to obtain the estimated delay of 12 logic levels for the rounded product.

5 The YZ Rounding Algorithm

In this section, we review and analyze the rounding algorithm of Yu and Zyner, which was reported to have been implemented in the ULTRASparc RISC microprocessor [26]. We refer to this algorithm as the YZ rounding algorithm.

5.1 Description

Fig. 5 depicts a block diagram of the YZ rounding algorithm. This description differs from the description in [26] in two ways:

1. In [26], the sum output by the 3-bit adder has only three bits. We believe that this is a mistake and that the sum should have four bits (we denote this sum by $Z[50:53]$).
2. The sum and the incremented sum in [26] is fed to a $4:1$-mux, which selects one of them either shifted to

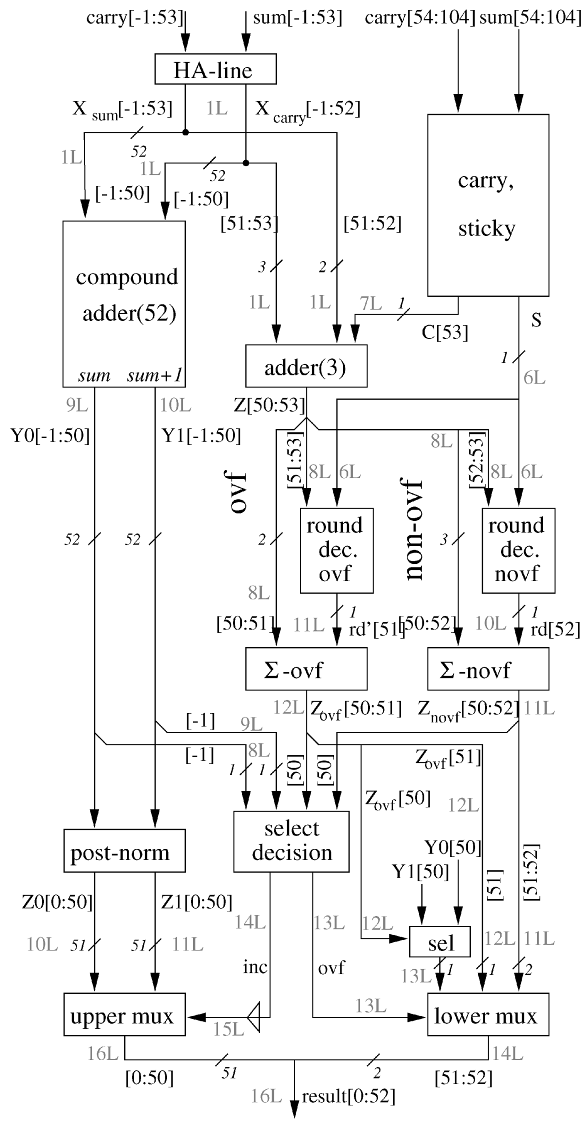

Fig. 5. Block diagram of the YZ rounding algorithm annotated with timing estimates.

the right or not. We propose to normalize the sum and the incremented sum before the selection takes place. This early normalization helps reduce the delay of the rounding circuit at the cost of two shifters rather than one.

The algorithm is described below:

1. The SUM and CARRY-strings are divided into a high part and a low part. The high parts consist of positions $[-1:53]$ and the low parts consist of positions $[54:104]$.
2. The low part is input to the box that computes the carry and sticky bits, defined as follows:

$$C[53]=\sigma_{53}$$
$$S=\mathrm{OR}(\sigma_{54},\sigma_{55},\ldots,\sigma_{104}),$$

where $\sigma_{53}\cdots\sigma_{104}$ is the binary string that satisfies:

$$|\sigma_{53}\cdots\sigma_{104}|=|\mathrm{SUM}[54:104]|+|\mathrm{CARRY}[54:104]|.$$

3. The higher part is input to a line of Half Adders and produces the output $X_{sum}[-1:53]$ and $X_{carry}[-1:52]$. Note that no carry is generated to position $[-2]$ because the exact product is less than 4.
4. The high part $X_{sum}[-1:53]$ and $X_{carry}[-1:52]$ is divided into two parts. Positions $[-1:50]$ are fed into the Compound Adder that outputs the sum $Y0[-1:50]$ and the incremented sum$|Y1[-1:50]| = |Y0[-1:50]| + 2^{-50}$. Positions $[51:53]$ are added with the carry bit $C[53]$ to produce the sum $Z[50:53]$.
5. The processing of $Z[50:53]$ is split into two paths; one working under the assumption that the rounded product will not overflow (i.e., less than 2) and the other path working under the assumption that the rounded product will overflow.

   The no-overflow path computes a rounding decision, *rd*[52], in the *round dec. (novf)* box. The rounding decision *rd*[52] is added with $Z[50:52]$ in the Σ-*novf* box to produce the sum $Z_{novf}[50:52]$. In Claim 3, we prove that this 3-bit addition does not produce a carry bit in position 49. The sum $Z_{novf}[50:52]$ has two roles: Positions $[51:52]$ are the result bits in positions $[51:52]$ if no overflow occurs and position $[50]$ is used to detect if a carry is generated in position $[50]$ if no overflow occurs. The bit $Z_{novf}[50]$ decides whether the upper sum $Y0[0:50]$ or the incremented sum $Y1[0:50]$ should be selected in the no-overflow case.

   The overflow path computes a rounding decision, *rd*′[51], in the *round dec. (ovf)* box. The rounding decision *rd*′[51] is added with $Z[50:51]$ in the Σ-*ovf* box to produce the sum $Z_{ovf}[50:51]$. In Claim 3, we prove that this 2-bit addition does not produce a carry bit in position 49. The sum $Z_{ovf}[50:51]$ has two roles: Position [51] serves as the result bit in position [52] if overflow occurs and position [50] is used to decide whether an increment should take place in the upper part.
6. The decision of which path should be chosen is made by the *select decision* box. First, an overflow signal ovf is computed as follows:

$$ovf = Y0[-1] \vee (Z_{novf}[50] \wedge Y1[-1]). \tag{7}$$

   The overflow signal ovf determines whether $Z_{ovf}[50]$ or $Z_{novf}[50]$ is chosen as the carry-bit that effects position [50] and, therefore, determines the increment decision inc:

$$inc = \begin{cases} Z_{ovf}[50] & \text{if } ovf = 1 \\ Z_{novf}[50] & \text{if } ovf = 0. \end{cases} \tag{8}$$

7. The two least significand bits of the rounded product are computed as follows: If no overflow occurs ($ovf = 0$), then

$$result[51:52] = Z_{novf}[51:52].$$

   Therefore, the *lower mux* selects these bits for $result[51:52]$ when $ovf = 0$ signal.

   If an overflow occurs ($ovf = 1$), then $result[52] = Z_{ovf}[51]$. The bit $result[51]$ depends on whether an increment takes place or not:

$$result[51] = \begin{cases} Y1[50] & \text{if } inc = 1 \\ Y0[50] & \text{if } inc = 0. \end{cases}$$

   Note that $inc = Z_{ovf}[50]$ if $ovf = 1$. Since the signal $Z_{ovf}[50]$ is ready earlier than inc, we use $Z_{ovf}[50]$ to control the selection:

$$result[51] = \begin{cases} Y1[50] & \text{if } Z_{ovf}[50] = 1 \\ Y0[50] & \text{if } Z_{ovf}[50] = 0. \end{cases}$$

   The selection between $Y1[50]$ and $Y0[50]$ is done by the sel multiplexer in Fig. 5.
8. The most significant bits $Y0[-1]$ and $Y0[-1]$ indicate whether $Y0$ and $Y1$ are in the range $[2,4)$. Depending on these bits, $Y0$ and $Y1$ are normalized as follows:

$$Z0[0:50] = \begin{cases} Y0[0:50] & \text{if } Y0[-1] = 0 \\ shift\_right(Y0[-1:49]) & \text{if } Y0[-1] = 1 \end{cases}$$
$$Z1[0:50] = \begin{cases} Y1[0:50] & \text{if } Y1[-1] = 0 \\ shift\_right(Y1[-1:49]) & \text{if } Y1[-1] = 1. \end{cases}$$

9. The rounded result (except for the least significant bit) is selected between $Z0$ and $Z1$ according to the increment decision inc signal, as follows:

$$\text{RESULT}[0:51] = \begin{cases} Z0[0:51] & \text{if } inc = 0 \\ Z1[0:51] & \text{if } inc = 1. \end{cases}$$

5.2 Correctness

In this section, we prove that adding the rounding decision does not generate a carry-bit in position 49. This claim applies both to the no-overflow path and to the overflow path.

Claim 3. *Let* $Z[50:53]$ *denote the sum that is output by the 3-bit adder, as depicted in Fig. 5. Let* $rd[52]$ *denote the rounding decision for the no-overflow path and let* $rd'[51]$ *denote the rounding decision for the overflow path. Then,*

$$Z[50] \cdot 2^{-1} + Z[51] \cdot 2^{-2} + rd'[51] \cdot 2^{-2} < 1$$
$$Z[50] \cdot 2^{-1} + Z[51] \cdot 2^{-2} + Z[52] \cdot 2^{-3} + rd[52] \cdot 2^{-3} < 1.$$

Proof. The partial compression [6] caused by the half-adder line implies that

$$(|X_{sum}[51:53]| + |X_{carry}[51:52]|) \cdot 2^{49} \in [0, 5/8].$$

This follows from the fact that $X_{sum}[i]$ and $X_{carry}[i+1]$ cannot both be equal to one. Adding $C[53]$ increases the above range by 2^{-4}, yielding that

$$|Z[50:53]| \cdot 2^{49} \in [0, 11/16].$$

The contribution of $rd'[51] \cdot 2^{-2}$ is in the range $[0, 4/16]$ and the contribution of $rd[52] \cdot 2^{-3}$ is in the range $[0, 2/16]$ and, therefore, the claim follows. □

5.3 Delay analysis

Fig. 5 depicts the YZ rounding algorithm annotated with our timing estimates. We use the same assumptions on the delays of signals that are used in Section 4.5. We argue that at least 16 logic levels are required. The path in which the sum and incremented sum are computed does not lie on the critical path. The critical path consists of the carry-bit computation, the 3-bit adder, the *round dec. (novf)* box, the Σ-*novf* box, the *select decision* box, a driver, and the *upper mux*.

We considered the following optimizations to minimize delay for a lower bound on the required number of logic levels:

1. The 3-bit adder is implemented by conditional sum adder; the late carry-in bit $C[53]$ selects between the sum and the incremented sum. This is a fast implementation because the bits of X_{carry} and X_{sum} are valid after one logic level and the carry-bit $C[53]$ is valid after seven logic levels.
2. The rounding decision boxes are implemented by cascading two levels of multiplexers that are controlled by $Z[52:53]$ in the no-overflow path and by $Z[51:52]$ in the overflow path. In the overflow path, $Z[53]$ is combined with sticky-bit and, hence, the rounding decision required three logic levels. In the no-overflow path, only two logic levels are required.
3. The addition of the rounding decision bit required only one logic level using a conditional sum adder.
4. The inc signal is valid after three more logic levels, due to the need to compute the signal ovf in two logic levels (see (7)), and one selection according to (8).
5. The inc signal passes through a driver due to the large fanout. This driver incurs a delay of one logic level and controls the *upper mux* to output the result after 16 logic levels.

6 The QTF Rounding Algorithm

Quach et al. [17] presented methods for IEEE compliant rounding. Their technique is a generalization of the rounding algorithm of Santoro et al. [18]. In this section, we present a rounding algorithm that is based on the method of Quach et al. while aiming for minimum delay.

Apart from reducing the rounding modes to RZ, RNU, and RI, the key idea used in the methods of Quach et al. and Santoro et al. is to inject a prediction bit that is based on the rounding mode and the values of $\mathrm{SUM}[53]$ and $\mathrm{CARRY}[53]$. The injection of the prediction bit reduces the number of possibilities of the rounded result.

In this section, we deviate from Quach et al. [17] in the following points:

1. The presentation in the paper of Quach et al. is separated according to the rounding mode. Since we are investigating rounding algorithms that support all the rounding modes, we integrated the rounding modes into one algorithm.
2. Quach et al. suggest several options for the choice of the prediction logic in RNU. Only one possibility was suggested in modes RZ and RI. Since the prediction logic lies on the critical path, we chose to simplify the prediction logic as much as possible by defining:

$$pred = \begin{cases} 1 & \text{if } RI \wedge (\mathrm{SUM}[53] \vee \mathrm{CARRY}[53]) \\ 0 & \text{otherwise.} \end{cases}$$

3. Quach et al. separate the rounding decision and the compound adder. They use a 3-way compound adder that computes sum, $sum+1$, and $sum+2$. The correct sum is selected by the control logic. We are interested in a faster design and, therefore, we break the 3-way adder into a Half-Adder line, a 2-way compound adder, and a mux. The control logic uses an output of the 2-way compound adder and the LSB (in case of no overflow) is generated by the control logic, as well as the increment decision.

6.1 Description

Fig. 6 depicts a block diagram of a rounding algorithm that we suggest based on Quach et al. [17]. There are many similarities between the rounding algorithm based on injection rounding and the rounding algorithm based on Quach et al., so we point out the differences and the new notations.

Before being input to the compound adder, the high part of the SUM and CARRY pass through two lines of Half-Adders. The first line makes room for the prediction bit. The second pass enables separating the bit $L_{X'}$ in position $[52]$ (this is, in fact, part of a 3-way compound adder). The increment decision has two paths: one for overflow and the other for no-overflow. The MSB $Y0[-1]$ selects which path outputs the increment decision inc. In addition, the increment decision computes the LSB (before fixing for RNE) in case an overflow does not occur.

6.2 Details

In this section, we describe the details of the *increment decision* box and the *LSB-fix for RNE* box.

6.2.1 Increment Decision Box

The outputs of the *increment decision* box are the increment decision inc and the bit L that equals the LSB of the rounded product before fixing in case no overflow occurs. The increment decision is partitioned into two paths. One is for the case that an overflow occurs which computes the signal inc_{ovf} and the other path is for the case that no overflow occurs which computes the signal inc_{novf}. The following equations define the signals inc_{ovf}, inc_{novf}, and inc:

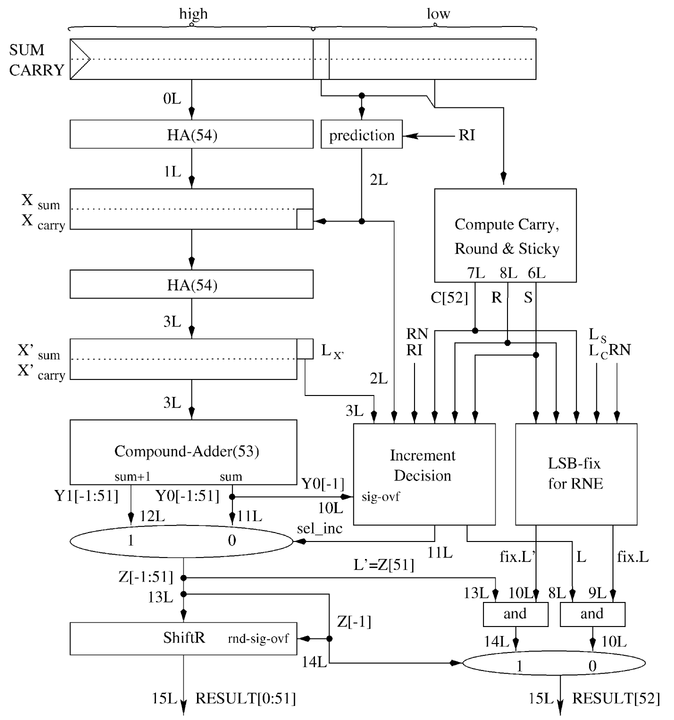

Fig. 6. Block diagram of the QTF rounding algorithm annotated with timing estimates.

$$inc_{novf} = \begin{cases} (L_{X'} + C[52] \geq 2) & \text{if RZ} \\ (R + L_{X'} + C[52] \geq 2) & \text{if RNU} \\ ((S \vee R) + L_{X'} \geq 2) & \text{if RI } \wedge C[52] = pred \\ 0 & \text{if RI } \wedge C[52] \neq pred \end{cases}$$

$$inc_{ovf} = \begin{cases} (L_{X'} + C[52] \geq 2) & \text{if RZ} \\ (L_{X'} + C[52] \geq 1) & \textit{if RNU} \\ (S \vee R \vee L_{X'}) & \text{if RI} \wedge C[52] = pred \\ L_{X'} & \text{if RI} \wedge C[52] \neq pred. \end{cases}$$

The output inc equals inc_{novf} or inc_{ovf} bit according to bit $Y0[-1]$.

$$inc = \begin{cases} inc_{novf} & \text{if } Y0[-1] = 0 \\ inc_{ovf} & \text{if } Y0[-1] = 1. \end{cases}$$

The bit L, which equals the LSB of the rounded product (before fixing) in case no overflow occurs, is defined by:

$$L = \begin{cases} L_{X'} \oplus C[52] & \text{if RZ} \\ R \oplus L_{X'} \oplus C[52] & \text{if RNU} \\ (S \vee R) \oplus L_{X'} \oplus C[52] \oplus pred & \text{if RI.} \end{cases}$$

Note that the case of RI is complicated due to the possibility that $pred \neq C[52]$. If $pred = C[52]$, then $L = (S \vee R) \oplus L_{X'}$, but if $pred \neq C[52]$, the effect of the wrong prediction is reversed by $(S \vee R) \oplus L_{X'} \oplus pred \oplus C[52]$.

6.2.2 LSB-Fix for RNE

The *LSB-fix for RNE* box outputs two signals: $not(pd)$ is used to pull-down the LSB if a "tie" occurs but no overflow occurs, and $not(pd')$ is used to pull-down the LSB if a "tie" and an overflow occur. These signals are defined as follows:

In contrast to injection-based rounding, no injection or prediction is contained in the $L_{X'}$, R, and S-bit computation in RNE. If no overflow occurs, a "tie" occurs iff $R = 1$ and $S = 0$, in which case the LSB should be pulled down for RNE. Therefore,

$$not(pd) = \text{OR } \ (not(R), S, not(RNE)).$$

If overflow occurs, a "tie" occurs iff $L_{X'} + C[52] = 1$, $R = 0$, and $S = 0$, in which case the LSB should be pulled down for RNE. Therefore,

$$not(pd') = \text{OR } \ (not(L_{X'} \oplus C[52]), R, S, not(RNE)).$$

6.3 Correctness

In this section, we prove the correctness of the selection signal INC. The proof is divided into two parts. In the first part, we assume that $Y0[-1] = 1$ iff the exact product is in the range $[2, 4)$. In the second part, we prove that, even if the $Y0[-1]$ signals overflow incorrectly, then the selection signal inc is still correct.

Claim 4. *Suppose that $Y0[-1]$ signals correctly whether the exact product is in the range $[2, 4)$. Then, the inc signal signals correctly whether an increment is required for rounding.*

Proof. We consider separately the cases of overflow and no overflow. For each case, we consider the three possible rounding modes. The question which we address is whether the rounding decision in conjunction with the compression of the lower part of the carry-save representation produces a carry into position 51. The inc signal should be 1 iff a carry is generated into position 51.

Suppose no overflow occurs, namely $Y0[-1] = 0$.

1. In rounding mode RZ, only truncation takes place and, therefore, a carry into position 51 is generated iff $L_{X'} + C[52] \geq 2$.
2. In rounding mode RNU, the rounding decision is to increment (in position 52) if the round-bit equals 1. This increment generates a carry to position 52 and, hence, a carry is generated into position 51 iff $R + L_{X'} + C[52] \geq 2$.
3. In rounding mode RI, the rounding decision is to increment (in position 52) if $R = 1$ or $S = 1$. One needs to take into account the prediction that was already added to the product. We consider two subcases:

   a. If $C[52] = pred$, then the contributions of $pred$ and $C[52]$ cancel out and, therefore, $C[52]$ should be ignored. The rounding decision generates a carry into position 51 iff $(R \vee S) \wedge L_{X'}$.
   b. If $C[52] \neq pred$, then this implies that $C[52] = 0$, $pred = 1$, and $R = 1$. Therefore, the rounding decision without the prediction would have been to increment in position 52. Since $pred = 1$, this increment already took place, and an additional carry should not be generated into position 51.

Suppose that overflow occurs, namely, $Y0[-1] = 1$.

1. In rounding mode RZ, only truncation occurs, so this case is identical to the case of no overflow.

2. In rounding mode RNU, the rounding decision is determined by the bit in position 52 which equals $L_{X'} \oplus C[52]$. Therefore, there are two cases: either a carry is generated into position 51 since $L_{X'} + C[52] \geq 2$ or a carry is generated into position 51 by the rounding decision. Combining these cases implies that a carry is generated into position 51 iff $L_{X'} + C[52] \geq 1$.
3. In rounding mode RI, we consider two cases: i) If $C[52] = pred$, then we may ignore $C[52]$ and the prediction since their contributions cancel out. In this case, the rounding decision is to increment iff $\text{OR}\ (L_{X'}, R, S)$. ii) If $C[52] \neq pred$, then $C[52] = 0$, $pred = 1$, and $R = 1$. We consider two subcases:

   a. If $L_{X'} = 1$, then the effect of the prediction was restricted to changing $L_{X'}$ from 0 to 1. Therefore, the rounding decision is based on $R \vee S$. Since $R = 1$, the rounding decision is to increment.
   b. If $L_{X'} = 0$, then the effect of the prediction was to generate a carry into position 51 in the second half-adder line and to change $L_{X'}$ from 1 to 0. This means that, without the prediction, $L_{X'}$ would have been equal to 1, which implies that the rounding decision would have been to increment. Since an increment already took place, an additional increment is not required. □

The selection between inc_{ovf} and inc_{novf} is controlled by $Y0[-1]$, although $Y0[-1]$ might not correctly signal the case of overflow. The following claim shows that, when $Y0[-1]$ does not signal overflow correctly, both choices are equal and, hence, the inc signal is correct.

Claim 5. *Suppose that $Y0[-1]$ bit does not signal an overflow correctly, namely, $Y0[-1] = 1$ and $\text{EXA} < 2$ or $Y0[-1] = 0$ and $\text{EXA} \geq 2$. Then, $inc_{ovf} = inc_{novf}$.*

Proof. The proof is divided into two cases:

1. $Y0[-1] = 1$ and $\text{EXA} < 2$. This case can only occur when $pred = 1$ and $C[52] = 0$. Therefore, it is restricted to rounding mode RI. Since

$$\begin{aligned} 2 + L_{X'} \cdot 2^{-52} &\leq Y0[-1:51] + L_{X'} \cdot 2^{-52} \\ &= \text{EXA}[0:52] + pred \cdot 2^{-52} < 2 + 2^{-52}, \end{aligned}$$

   it follows that $L_{X'} = 0$. This implies that, in this case, $inc_{ovf} = inc_{novf}$, as required.
2. $Y0[-1] = 0$ and $\text{EXA} \geq 2$. This discrepancy can only occur if $C[52] = 1$ and $pred = 0$. Therefore,

$$\begin{aligned} |Y0[-1:51]| + L_{X'} \cdot 2^{-52} &= \text{EXA}[-1:52] - 2^{-52} \\ &\geq 2 - 2^{-52}. \end{aligned}$$

   Since $|Y0[-1:51]| + L_{X'} \cdot 2^{-52}$ is smaller than 2 and a multiple of 2^{-52}, it follows that

$$|Y0[0:51]| + L_{X'} \cdot 2^{-52} = 2 - 2^{-52}.$$

   This implies that $L_{X'} = 1$. Consider the three rounding modes: In RZ, $inc_{ovf} = inc_{novf}$. In RI, if $C[52] = 1$, then $pred = 1$, excluding the possibility of this case. In RNU, since $C[52] = 1$ and $L_{X'} = 1$, it follows that $inc_{ovf} = inc_{novf}$, and the claim follows. □

6.4 Delay Analysis

Fig. 6 depicts the rounding algorithm based on Quach et al. [17] with delay annotation. The delay assumptions that are used here are similar to those used in the two previous rounding algorithms. The rounding algorithm depicted in Fig. 6 uses a prediction logic which lies on the critical path. The delay of the prediction logic is two logic levels. Following Quach et al., Fig. 6 depicts a nonoptimized processing order in which the postnormalization shift takes place after the round selection. The increment decision box is assumed to be organized as follows: The bits S, $C[52]$, R, and $Y0[-1]$ are valid after $6, 7, 8, 10$ logic levels, respectively. To minimize delay, we implement the rounding equations by 4 levels of multiplexers so that the results can be selected conditionally as the signals arrive. Thus, a total delay of 15 logic levels is obtained. By performing postnormalization before the round selection takes place, one logic level can be saved to obtain a total delay of 14 logic levels.

7 Higher Precisions

How do these rounding algorithms scale when higher precisions are used? One can see that the parts in the presented rounding algorithms that depend on the length of the significands are: the half-adders, the compound adder, the sticky, round, and carry-bit computation, the selection multiplexers, and the drivers for amplifying the signals that control the wide multiplexers.

When precision is increased, the widths of upper and lower parts of the carry and save strings grow, but they still stay almost equal to each other. This implies that our assumptions on the relative delay of the carry-bit computation and the compound adder do not need to be changed. Moreover, it is expected that, as precision grows, the gap between the delay of computing the carry-bit and the sticky-bit grows so that the sticky-bit computation will not lie on the critical path. This implies that a first order estimate (ignoring additional delay due to increased fanout and interconnection length) of the delays of the rounding algorithms for precision p can be stated as follows:

1. The delay of the injection-based rounding algorithm is four logic levels plus the delay of the sum computation of the p-bit compound adder $d_{CLA}(p)$.
2. The delay of the YZ rounding algorithm is $8 + d_{CLA}(p)$.
3. The delay of the rounding algorithm based on Quach et al. [17] with the optimization (in which the postnormalization takes place before the selection) is $6 + d_{CLA}(p)$ logic levels.

8 Summary and Conclusions

A new IEEE compliant floating-point rounding algorithm for computing the rounded product from a carry-save representation of the product is presented. The new rounding algorithm is compared with two previous rounding algorithms. To make the comparison as relevant as possible, we considered optimizations of the previous algorithms which improve the delay. For each rounding algorithm, a logical description and a block diagram is given, the correctness is proven, and the latency is analyzed.

Our conclusion is that the new ES rounding algorithm is the fastest rounding algorithm provided that an injection is added in during the reduction of the partial products into a carry-save encoded digit string. With the ES algorithm, the rounded product can be computed in 12 logic levels in double precision (i.e., when the significands are 53 bits long). In "precision independent" terms, the critical path consists of a compound adder and four additional logic levels.

If the injection is not added in during the reduction of the partial products into a carry-save encoded digit string, then an extra step of adding in the injection is required. This step amounts to a carry-save addition and the latency associated with it is that of a full-adder, namely, two logic levels. Thus, if the injection is added in late, then the latency of the ES rounding algorithm is 14 logic levels.

The addition of the injection during the reduction of the partial products can be accomplished without a slowdown or with a very small slowdown. The justification for this is: 1) The partial products are usually obtained by Booth recoding and by selecting (e.g., $5{:}1$ multiplexer) and, hence, are valid (i.e., available) much later than the injection; and 2) the delay of adding the partial products does not increase strictly monotonically as a function of the number of partial products. The delay incurred by adding in the injection, if any, depends on the length of the significands and on the organization of the adder tree.

The other two rounding algorithms do not require an injection and, in double precision, the latency of the QTF rounding algorithm is 14 logic levels. The critical path consists of a compound adder and six additional logic levels. The YZ rounding algorithm ranks as the slowest rounding algorithm, with a latency of 16 logic levels, and the critical path consists of a compound adder and eight additional logic levels.

Acknowledgments

A preliminary version of this work appeared as "A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication" in the *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 225-232, Adelaide, Australia, 14-16 April 1999. Guy Even's work was supported in part by Intel Israel Ltd. under grants awarded in 1997 and 1998. Peter-Michael Seidel's work was supported by the DFG Graduiertenkolleg "Effizienz und Komplexität von Algorithmen und Rechenanlagen," Universität des Saarlandes.

References

[1] H. Al-Twaijry, "Area and Performance Optimized CMOS," PhD thesis, Stanford Univ., Aug. 1997, ftp://umunhum.stanford.edu/tr/hesham.aug97.thesis.ps.
[2] G.W. Bewick, "Fast Multiplication: Algorithms and Implementation," PhD thesis, Stanford Univ., Mar. 1994, ftp://umunhum.stanford.edu/tr/bewick.apr94.thesis.ps.Z.
[3] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, pp. 260-264, Mar. 1982.
[4] J.T. Coonen, "Specification for a Proposed Standard for Floating-Point Arithmetic," Memorandum ERL M78/72, Univ. of California, Berkeley, 1978.
[5] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza,* vol. 34, pp. 349-356, 1965.
[6] M. Daumas and D.W. Matula, "Recoders for Partial Compression and Rounding," Technical Report 97-01, Laboratoire de l'Informatique du Parallelisme, Lyon, France, 1997, ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR97/RR97-01.ps.Z.
[7] G. Even, S.M. Mueller, and P.M. Seidel, "A Dual Mode IEEE Multiplier," *Proc. Second IEEE Int'l Conf. Innovative Systems in Silicon,* pp. 282-289, 1997.
[8] C.N. Hinds, E.V. Fiene, D.T. Marquette, and E.E. Quintana, "Parallel Method and Apparatus for Detecting and Completing Floating-Point Operations Involving Special Operands," US patent 5339266, 1994.
[9] *IEEE Standard for Binary Floating-Point Arithmetic.* New York: ANSI/IEEE 754-1985, 1985.
[10] C. Lee, "Multistep Gradual Rounding," *IEEE Trans. Computers,* vol. 32, no. 4, pp. 595-600, Apr. 1989.
[11] C. Martel, V. Oklobdzija, R. Ravi, and P.F. Stelling, "Design Strategies for Optimal Multiplier Circuits," *Proc. 12th Symp. Computer Arithmetic,* pp. 42-49, 1995.
[12] Z.-J. Mou and F. Jutand, "Overturned-Stairs Adder Trees and Multiplier Design," *IEEE Trans. Computers,* vol. 41, no. 8, pp. 940-948, Aug. 1992.
[13] S.F. Oberman, "Design Issues in High Performance Floating Point Arithmetic Units," PhD thesis, Stanford Univ., Jan. 1997, ftp://umunhum.stanford.edu/tr/oberman.nov96.thesis.ps.Z.
[14] S.F. Oberman, H. Al-Twaijry, and M.J. Flynn, "The SNAP Project: Design of Floating-Point Arithmetic Units," *Proc. 13th Symp. Computer Arithmetic,* pp. 156-165, 1997.
[15] V.G. Oklobdzija, D. Villeger, and S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Trans. Computers,* vol. 45, no. 3, pp. 294-306, Mar. 1996.
[16] R.M. Owens, R.S. Bajwa, and M.J. Irwin, "Reducing the Number of Counters Needed for Integer Multiplication," *Proc. 12th Symp. Computer Arithmetic,* pp. 38-41, 1995.
[17] N. Quach, N. Takagi, and M. Flynn, "On Fast IEEE Rounding," Technical Report CSL-TR-91-459, Stanford Univ., Jan. 1991.
[18] M.R. Santoro, G. Bewick, and M.A. Horowitz, "Rounding Algorithms for IEEE Multipliers," *Proc. Ninth Symp. Computer Arithmetic,* pp. 176-183, 1989.
[19] P.-M. Seidel, "How to Half the Latency of IEEE Compliant Floating-Point Multiplication," *Proc. 24th Euromicro Conf.,* 1998.
[20] P.-M. Seidel, "The Design of IEEE Compliant Floating-Point Units and Their Quantitative Analysis," PhD thesis, Univ. of Saarland, Dec. 1999.
[21] N. Takagi, H. Yasura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Trans. Computers,* vol. 34, no. 9, pp. 217-220, Sept. 1985.
[22] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers,* vol. 42, no. 10, pp. 1,163-1,170, Oct. 1993.
[23] J. Vuillemin, "A Very Fast Multiplication Algorithm for VLSI Implementation," *INTEGRATION the VLSI J,* vol. 1, pp. 39-52, 1983.
[24] C.S. Wallace, "A Suggestion for Parallel Multipliers," *IEEE Trans. Electronic Computers,* vol. 13, pp. 14-17, 1964.
[25] Z. Wang, A. Jullien, and C. Miller, "A New Design Technique for Column Compression Multipliers," *IEEE Trans. Computers,* vol. 44, no. 8, pp. 962-970, Aug. 1995.
[26] R.K. Yu and G.B. Zyner, "167 MHz Radix-4 Floating-Point Multiplier," *Proc. 12th Symp. Computer Arithmetic,* pp. 149-154, 1995.
[27] R.K. Yu and G.B. Zyner, "Method and Apparatus for Partially Suporting Subnormal Operands in Floating-Point Multiplication," US patent 5602769, 1997.
[28] G. Zyner, "Circuitry for Rounding in a Floating-Point Multiplier," US patent 5150319, 1992.

Guy Even received his BSc in mathematics and computer science from the Hebrew University in Jerusalem in 1988 and his MSc and DSc in computer science from the Technion in Haifa in 1991 and 1994, respectively. During 1995-1997, he was a postdoctoral fellow in the Chair of Prof. Wolfgang Paul at the University of the Saarland at Saarbruecken. Since 1997, he has been a faculty member in the Electrical Engineering-Systems Department at Tel-Aviv University. His current areas of research include: computer arithmetic and the design of IEEE compliant floating-point units, approximation algorithms for NP complete problems related to VLSI design, and the design of systolic arrays.

Peter-Michael Seidel studied computer science and electrical engineering at the University of Hagen, Germany, from which he graduated in 1996. He completed his PhD in computer science in 1999 at the chair of Prof. Wolfgang Paul at the University of the Saarland, Germany, where he was supported by a fellowship of the German National Science Foundation (DFG). Currently, he is an assistant professor in the Computer Science and Engineering Department at Southern Methodist University in Dallas, Texas. His current areas of research include: computer arithmetic, computer architecture, and the design of IEEE compliant floating-point units.