A comparison of three rounding algorithms for IEEE floating-point multiplication

Guy Even* Peter-M. Seidel†

August 29, 1998

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 [31] and of Quach *et al.* [23]. 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.

*Dept. of Electrical Engineering Systems, Tel-Aviv University, Tel-Aviv 69978, Israel. E-mail:guy@eng.tau.ac.il.
Supported in part by Intel Israel LTD. under grants awarded in 1997 and 1998.

†University of the Saarland, Computer Science Department, D-66041 Saarbruecken, Germany. E-mail:pmseidel@cs.uni-sb.de.
Supported by the DFG Graduiertenkolleg “Effizienz und Komplexitaet von Algorithmen und Rechenanlagen”, Universitaet des Saarlandes.

1 Introduction

Every modern microprocessor includes a floating-point (FP) multiplier that complies with the IEEE 754 Standard [13]. The latency of the FP multiplier is critical to the 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 [17].

A lot of research has been devoted to optimizing the latency of adding the partial products to produce the product, e.g. [1, 2, 6, 9, 15, 16, 18, 19, 20, 21, 26, 28, 29, 30]. More recently, work on rounding the product according to the IEEE 754 Standard has been published [4, 7, 10, 22, 23, 24, 25, 31, 33, 34]. 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: (a) the algorithm of Quach *et al.* [23], which we denote by the QTF algorithm; (b) the algorithm of Yu and Zyner [31], which we denote by the YZ algorithm; and (c) a new algorithm that is based on injection based rounding [10], which we denote by the ES algorithm. We provide block diagrams of these rounding algorithms, optimized for speed. We measure the latency of the algorithms in 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 [14, 25, 32] and the computation on special values can be done in parallel [12]. 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 is computed from the lower part.

2. The rounding decision is computed in two paths: the non-overflow path works under the assumption that the exact product is in the range $[1, 2)$, and 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.

The main differences between the three rounding algorithms are outlined, as follows.

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 position $[53:104]$.

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 tricky 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 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. Due to space limitations some of the sections are omitted and can be found in the full version [11].

2 Preliminaries

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

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

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). Based on the sign of the number the rounding modes round toward $+\infty$ and round toward $-\infty$ can be reduced to the rounding modes RZ (round to zero) and RI (round to infinity) [23]. Thus leaving only three rounding modes: RI, RZ, and RNE (round to nearest - even).

Furthermore, Quach et al. [23] suggested to implement 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 $y_1 \leq x < y_2$, 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 rounding to zero (RZ) of significands in the range $[1, 4)$ in double precision. Note, that this definition excludes the post-normalization shift that takes place when the number is in the binade $[2, 4)$.

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

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

where x div $2^{-\ell}$ is the integer q that satisfies:

$$q \cdot 2^{-\ell} \leq x < (q + 1) \cdot 2^{-\ell}$$

General setting In this paper we consider a double precision multiplier. We assume that the significands are prenormalized, 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.

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. This paper discusses implementations of the second phase.

3 Naive IEEE rounding

In this section we review a simple but slow IEEE compliant algorithm for rounding after multiplication [5].

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 EXACT $\in [1, 4)$.

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 one 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 $|X| = |\text{SUM}| + |\text{CARRY}|$. Note that since the exact product is in the range $[1, 4)$, the 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

$$\text{OR}(X'[54], X'[55], \ldots, X'[105]).$$

5. Compute rounding decision. The rounding decision $\text{RD} \in \{0,1\}$ 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 $|X'[0:52]| + \text{RD} \cdot 2^{-52}$. Let $Y[-1:52]$ be the binary string that represents the sum.

7. Post-normalize. If $|Y| = 2$, then $|Y'| = 1$, otherwise $Y' = Y[0:52]$.

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

3.2 Delay analysis

The latency of steps 1, 3, and 5 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 3 pipeline stages.

4 The ES rounding algorithm

In this section we review injection based rounding [10], and present an implementation for double precision that requires (under assumptions specified in Sec. 4.5) only 12 logic levels.

4.1 Injection based rounding

Rounding by injection reduces the rounding modes RI and RNU to RZ [10]. The reduction is based on adding an injection that depends only on the rounding mode, as follows:

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

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

$$X \in [1,2) \quad \Rightarrow \quad round_{mode}(X) = round_{RZ}(X + \text{INJECTION}) \tag{1}$$

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

Figure 1 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 EXACT, is in the range $[2,4)$, then the injection must be fixed in order to make the reduction to RZ correct. The correction amount, denoted by INJ_CORRECT, is defined by

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

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:

$$X \in [2,4) \quad \Rightarrow \quad round_{mode}(X) = round_{RZ}(X + \text{INJECTION} + \text{INJ\_CORRECT}) \qquad (2)$$

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

Our assumption is that INJECTION is added in the multiplier adder array, and therefore

$$|\text{SUM}| + |\text{CARRY}| = \text{EXACT} + \text{INJECTION}.$$

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.

Figure 2 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:

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

   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 with 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 3 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 4 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 figure 2, that have not been fully described yet.

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 EXACT denote the value of the exact product, the "Fix L (novf)" generates a signal *not(pd)* that satisfies:

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

When a tie occurs there are two possibilities: (a) 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. (b) If RNU and RNE disagree, then the LSB of the RNU result must be pulled down.

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:

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

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:

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

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:

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

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 INJ_CORRECT. It produces an increment decision, if $L_X + C[52] + \text{INJ\_CORRECT} \cdot 2^{52} + R/2 \geq 2$. 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 tricky 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, Equations (1) and (2) imply that the INC signal is correct. But one should also consider the cases that $Y0[-1]$ fails to indicate correctly the binade of the exact product. Namely: (a) $Y0[-1] = 0$ and the exact product is greater than or equal to 2; and (b) $Y0[-1] = 1$ 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 the 53 most-significant bits of the exact product. Recall that

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

The lower part of the product (corresponding to positions $[53:104]$ in the registers SUM and CARRY) as well as L_X do not effect the value of $Y0[-1]$. However, the injection might have an effect on $Y0[-1]$ since it is added-in in the multiplier array, depending on how the multiplier array is implemented (Wallace tree, etc.).

The following claim shows that when such mismatches occur, the rounded product equals 2. Moreover, in these cases both paths: the one working under the assumption that no overflow occurs, and the one working under the assumption that overflow occurs, yield the result 2. Therefore, correct rounding is obtained even when $Y0[-1]$ fails to indicate correctly the binade of the exact product.

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

$$r_{mode}(\text{EXACT}) = \begin{cases} r_{RZ}(\text{EXACT} + \text{INJECTION}) & \text{if } Y0[-1] = 0 \\ r_{RZ}(\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT}) & \text{if } Y0[-1] = 1 \end{cases}$$

Proof: We consider two main cases: (a) $Y0[-1] = 0$; and (b) $Y0[-1] = 1$.

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

$$\text{EXACT} + \text{INJECTION} \in [2, |Y0[0:51]| + \varepsilon), \tag{3}$$

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 $|\text{SUM}[53:104]| + |\text{CARRY}[53:104]| \in [0, 2^{-51})$.

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 Eq. (3) yields

$$\text{EXACT} + \text{INJECTION} \in [2, 2 + 2^{-52}). \tag{4}$$

The correction of the injection satisfies $0 \leq \text{INJ\_CORRECT} \leq 2^{-52}$, therefore:

$$\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT} \in [2, 2 + 2^{-51}). \tag{5}$$

According to Eq. (2), in this case $r_{mode}(\text{EXACT}) = r_{RZ}(\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT})$. However, in this case

$$r_{RZ}(\text{EXACT} + \text{INJECTION}) = r_{RZ}(\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT}) = 2$$

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

(b) Suppose $Y0[-1] = 1$. If $\text{EXACT} \geq 2$ then the claim follows from Eq. (2). If $\text{EXACT} < 2$, then since $\text{INJECTION} \in [0, 2^{-52})$, it follows that

$$\text{EXACT} + \text{INJECTION} \in [2, 2 + 2^{-52}). \tag{6}$$

The proof now follows the proof in case (a). □

Claim 1 proves that $Y0[-1]$ can be used for controlling the selection of the right rounded result. The following claim proves that our implementation of the computation of $r_{mode}(\text{EXACT})$ 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*

$$r_{RZ}(\text{EXACT} + \text{INJECTION}) = \begin{cases} |(Y0[0:51], (L_X \oplus C[52]))| & \textit{if } \text{INC} = 0 \\ |(Y1[0:51], (L_X \oplus C[52]))| & \textit{if } \text{INC} = 1 \ \&\ Y1[-1] = 0 \\ |(Y1[-1:51]| & \textit{if } \text{INC} = 1 \ \&\ Y1[-1] = 1 \end{cases}$$

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

$$r_{RZ}(\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT}) = \begin{cases} |(Y0[-1:51]| & \textit{if } \text{INC} = 0 \\ |(Y1[-1:51]| & \textit{if } \text{INC} = 1 \end{cases}$$

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

$$\text{EXACT} + \text{INJECTION} = |Y0[0:51]| + L_X \cdot 2^{-52} + C[52] \cdot 2^{-52} + tail, \tag{7}$$

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

$$r_{RZ}(\text{EXACT} + \text{INJECTION}) = r_{RZ}(|Y0[0:51]| + L_X \cdot 2^{-52} + C[52] \cdot 2^{-52}). \tag{8}$$

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

$$r_{RZ}(\text{EXACT} + \text{INJECTION}) = |Y0[0:51]| + L_X \cdot 2^{-52} + C[52] \cdot 2^{-52}$$

If $\text{INC} = 1$, 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 lemma.

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

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

Therefore,

$$\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT} = |Y0[-1:51]| + \text{INC} \cdot 2^{-51} + tail, \tag{9}$$

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

$$r_{RZ}(\text{EXACT} + \text{INJECTION} + \text{INJ\_CORRECT}) = r_{RZ}(|Y0[-1:51]| + \text{INC} \cdot 2^{-51}) \tag{10}$$

and the lemma follows. □

4.5 Delay analysis

In the section we present a delay analysis of the 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 incremented sum is $d_{CLA}+1$. This can be obtained by ORing the carry-generate and carry-propagate signals [27, 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 [31].

4. We assume that the delay associated with buffering a fan-out of 53 is one logic level.

Figure 3 depicts the block diagram of the injection based rounding algorithm annotated with timing estimates. We assigned d_{CLA} the value of 8 logic levels. This implies that the sticky bit is valid after 6 logic levels, the carry-bit $C[52]$ is valid after 7 logic levels, and the round-bit is valid after 8 logic levels. Similarly, the sum $Y0$ is valid after 9 logic levels, the MSB $Y0[-1]$ is valid after 8 logic levels, the incremented sum $Y1$ is valid after 10 logic levels, and the MSB $Y1[-1]$ is valid after 9 logic levels.

Figure 4 depicts implementations of the Fix L (novf), Fix L' (ovf), and Increment Decision boxes annotated with timing estimates. These timing estimates 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 [31]. We refer to this algorithm as the YZ rounding algorithm.

5.1 Description

Figure 5 depicts a block diagram of the YZ rounding algorithm. This description differs from the description in [31] in two ways:

1. In [31] 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 [31] is fed to a 4 : 1-mux, which selects one of them either shifted to 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:

$$\begin{aligned} C[53] &= \sigma_{53} \\ S &= \text{OR}(\sigma_{54}, \sigma_{55}, \ldots, \sigma_{104}) \end{aligned}$$

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

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

   .

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 which path should be chosen is made by the *select decision* box. First, an overflow signal ovf is computed as follows:

$$ovf = Y0[-1] \vee (Z_{novf}[50] \wedge Y1[-1]). \tag{11}$$

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:

$$inc = \begin{cases} Z_{ovf}[50] & \text{if } ovf = 1 \\ Z_{novf}[50] & \text{if } ovf = 0 \end{cases} \tag{12}$$

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 provide a proof 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,*

$$Z[50] \cdot 2^{-1} + Z[51] \cdot 2^{-2} + rd'[51] \cdot 2^{-2} < 1 \tag{13}$$

$$Z[50] \cdot 2^{-1} + Z[51] \cdot 2^{-2} + Z[52] \cdot 2^{-3} + rd[52] \cdot 2^{-3} < 1 \tag{14}$$

Proof: The partial compression [8] 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 by the fact that $X_{sum}[i]$ and $X_{carry}[i+1]$ cannot be both 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 therefore, Eq. 13 follows. The contribution of $rd[52] \cdot 2^{-3}$ is in the range $[0, 2/16]$, and therefore, Eq. 14 follows. □

5.3 Delay analysis

Figure 6 depicts the YZ rounding algorithm annotated with timing estimates. We use the same assumptions on the delays of signals that are used in Sec. 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 7 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 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 3 logic levels. In the no-overflow path, only 2 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 3 more logic levels, due to the need to compute the signal ovf in two logic levels (see Eq. 11), and one selection according to Eq. 12.

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.* [23] presented methods for IEEE compliant rounding. Their technique is a generalization of the rounding algorithm of Santoro *et al.* [24]. 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 $\text{SUM}[53]$ and $\text{CARRY}[53]$. The injection of the prediction bit reduces the number of possibilities of the rounded result.

In this section we deviate from Quach *et al.* [23] 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:

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

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

Figure 7 depicts a block diagram of a rounding algorithm that we suggest based on Quach *et al.* [23]. 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.

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 for the case that an overflow occurs which computes the signal inc_{ovf}, and the other path for the case that no overflow occurs which computes the signal $incnovf$. The following equations define the signals inc_{ovf}, inc_{novf}, and inc.

$$inc_{novf} = \begin{cases} (L_{X'} + C[52] \geq 2) & \text{if RZ} \\ (R + L_{X'} + C[52] \geq 2) & \text{if RNU} \\ ((S \vee R) + L_{X'} \geq 2) & \text{if RI and } C[52] = pred \\ 0 & \text{if RI and } C[52] \neq pred \end{cases} \tag{15}$$

$$inc_{ovf} = \begin{cases} (L_{X'} + C[52] \geq 2) & \text{if RZ} \\ (L_{X'} + C[52] \geq 1) & \text{if RNU} \\ (S \vee R \vee L_{X'}) & \text{if RI and } C[52] = pred \\ L_{X'} & \text{if RI and } C[52] \neq pred \end{cases} \tag{16}$$

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 the effect of the wrong prediction is reversed by $(S \vee R) \oplus L_{X'} \oplus pred \oplus C[52]$.

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 sub-cases:
   (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, since only truncation takes place, 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 sub-cases:

(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 signal correctly 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* $\text{EXACT} < 2$*, or* $Y0[-1] = 0$ *and* $\text{EXACT} \geq 2$*. Then,* $inc_{ovf} = inc_{novf}$*.*

Proof: The proof is divided into two cases:

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

$$2 + L_{X'} \cdot 2^{-52} \leq Y0[-1:51] + L_{X'} \cdot 2^{-52} = \text{EXACT}[0:52] + pred \cdot 2^{-52} < 2 + 2^{-52}$$

it follows that $L_{X'} = 0$. This implies that in this case $inc_{ovf} = inc_{novf}$, as required.

2. $Y0[-1] = 0$ and $\text{EXACT} \geq 2$. This discrepancy can only occur if $C[52] = 1$ and $pred = 0$. Therefore,

$$|Y0[-1:51]| + L_{X'} \cdot 2^{-52} = \text{EXACT}[-1:52] - 2^{-52} \geq 2 - 2^{-52}$$

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

Figure 8 depicts the rounding algorithm based on Quach *et al.* [23] 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. 8 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. 8 depicts a non-optimized processing order in which the post-normalization 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 the signals as they arrive. Thus, a total delay of 15 logic levels is obtained. By performing post-normalization 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 4 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.*[23] with the optimization (in which the post-normalization 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 4 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, 2 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: (a) The partial products are usually obtained by Booth recoding and by selecting (e.g. 5:1 multiplexer), and hence, are valid much later than the injection; and (b) 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 6 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 8 additional logic levels.

References

[1] H. Al-Twaijry. *Area and Performance Optimized CMOS Multipliers.* PhD thesis, Stanford University, August 1997. accessible via ftp://umunhum.stanford.edu /tr/hesham.aug97.thesis.ps.

[2] G.W. Bewick. *Fast Multiplication: Algorithms and Implementation.* PhD thesis, Stanford University, March 1994. accessible via 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 Transactions on Computers*, C-31(3):260–264, March 1982.

[4] P.M. Bromham and R.S. Carlton. Method for rounding using redundant coded multiply result. U.S. patent 5680339, 1997.

[5] J.T. Coonen. Specification for a proposed standard for floating point arithmetic. Memorandum ERL M78/72, University of California, Berkeley, 1978.

[6] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[7] H.M. Darley. Method and apparatus for rounding in high-speed multipliers. U.S. patent 5170371, 1992.

[8] M. Daumas and D.W. Matula. Recoders for partial compression and rounding. Technical Report 97-01, Laboratoire de l'Informatique du Paralllisme, Lyon, France, 1997. accessible via ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR97/RR97-01.ps.Z.

[9] B.C. Drerup and E.E. Swartzlander. Fast multiplier bit-product matrix reduction using bit-ordering and parity generation. *Journal of VLSI Signal Processing*, 7:249–257, 1994.

[10] G. Even, S.M. Mueller, and P.M. Seidel. A Dual Mode IEEE multiplier. In *Proceedings of the 2nd IEEE International Conference on Innovative Systems in Silicon*, pages 282–289. IEEE, 1997.

[11] Guy Even and Peter-M. Seidel. A comparison of three rounding algorithms for IEEE floating-point multiplication. Technical Report EES1998-8, Dept. of Elec. Eng. Systems, Tel-Aviv Univ., 1998. http://www.eng.tau.ac.il/Utils/reportlist/reports/repfram.html.

[12] 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. U.S. patent 5339266, 1994.

[13] IEEE standard for binary floating-point arithmetic. ANSI/IEEE754-1985, New York, 1985.

[14] C. Lee. Multistep gradual rounding. *IEEE Transactions on Computers*, 32(4):595–600, April 1989.

[15] C. Martel, V. Oklobdzija, R. Ravi, and P.F. Stelling. Design strategies for optimal multiplier circuits. *Proceedings 12th Symposium on Computer Arithmetic*, 12:42–49, 1995.

[16] Z.-J. Mou and F. Jutand. Overturned-stairs adder trees and multiplier design. *IEEE Transactions on Computers*, 41(8):940–948, August 1992.

[17] S.F. Oberman. *Design Issues in High Performance Floating Point Arithmetic Units.* PhD thesis, Stanford University, January 1997. accessible via ftp://umunhum.stanford.edu/tr/oberman.nov96.thesis.ps.Z.

[18] S.F. Oberman, H. Al-Twaijry, and M.J. Flynn. The SNAP project: Design of floating point arithmetic units. In *Proceedings of the 13th Symposium on Computer Arithmetic*, volume 13, pages 156–165. IEEE, 1997.

[19] 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 Transactions on Computers*, 45(3):294–306, March 1996.

[20] R.M. Owens, R.S. Bajwa, and M.J. Irwin. Reducing the number of counters needed for integer multiplication. *Proceedings 12th Symposium on Computer Arithmetic*, 12:38–41, 1995.

[21] K.F. Pang, H.-W. Soong, R. Sexton, and P.H. Ang. Generation of high speed CMOS multiplier-accumulators. *Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 217–220, 1988.

[22] W.-C. Park, T.-D. Han, S.-D. Kim, and S.-B. Yang. Floating Point Multiplier Performing IEEE Rounding and Addition in Parallel. *accepted in Journal of System Architecture*, 1998. accessible via http://comsci.yonsei.ac.kr/∼chan/jcs.ps.

[23] N. Quach, N. Takagi, and M. Flynn. On fast IEEE rounding. Technical Report CSL-TR-91-459, Stanford University, January 1991.

[24] M.R. Santoro, G. Bewick, and M.A. Horowitz. Rounding algorithms for IEEE multipliers. In *Proceedings 9th Symposium on Computer Arithmetic*, pages 176–183, 1989.

[25] P.-M. Seidel. How to half the latency of IEEE compliant floating-point multiplication. In *Proceedings of the 24th Euromicro Conference*, volume 24. IEEE, 1998.

[26] N. Takagi, H. Yasura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Transactions on Computers*, C-34(9):217–220, September 1985.

[27] A. Tyagi. A reduced-Area Scheme for Carry-Select Adders. *IEEE Transactions on Computers*, 42(10):1163–1170, 1993.

[28] J. Vuillemin. A very fast multiplication algorithm for VLSI implementation. *INTEGRATION, the VLSI journal*, 1:39–52, 1983.

[29] C.S. Wallace. A suggestion for parallel multipliers. *IEEE Trans. Electron. Comput.*, EC-13:14–17, 1964.

[30] Z. Wang, A. Jullien, and C. Miller. A new design technique for column compression multipliers. *IEEE Transaction on Computers*, 44(8):962–970, August 1995.

[31] R.K. Yu and G.B. Zyner. 167 MHz Radix-4 floating point multiplier. *Proceedings 12th Symposium on Computer Arithmetic*, 12:149–154, 1995.

[32] R.K. Yu and G.B. Zyner. Method and apparatus for partially suporting subnormal operands in floating point multiplication. U.S. patent 5602769, 1997.

[33] R.K. Yu and G.B. Zyner. Shared rounding hardware for multiplier and division/square root unit using conditional sum adder. U.S. patent 5671171, 1997.

[34] G. Zyner. Circuitry for rounding in a floating point multiplier. U.S. patent 5150319, 1992.

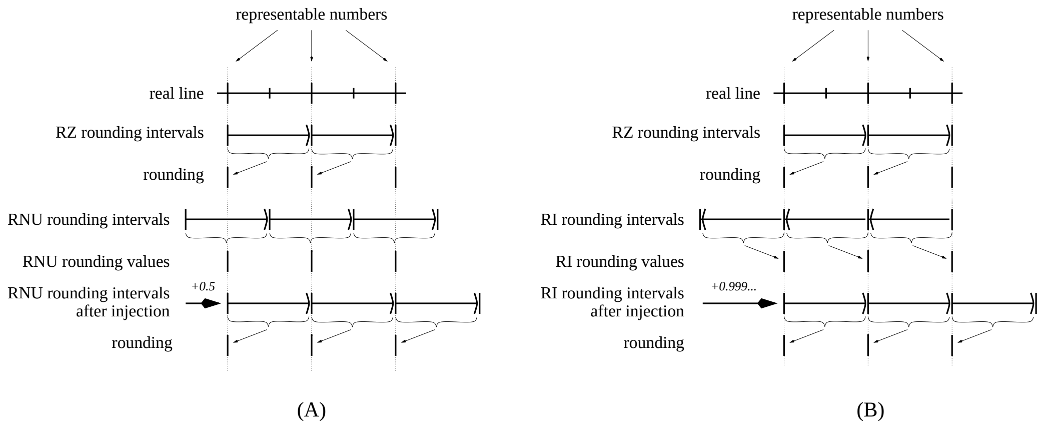

Figure 1: (A) Reducing RNU to RZ with an injection; (B) Reducing RI to RZ with an injection.

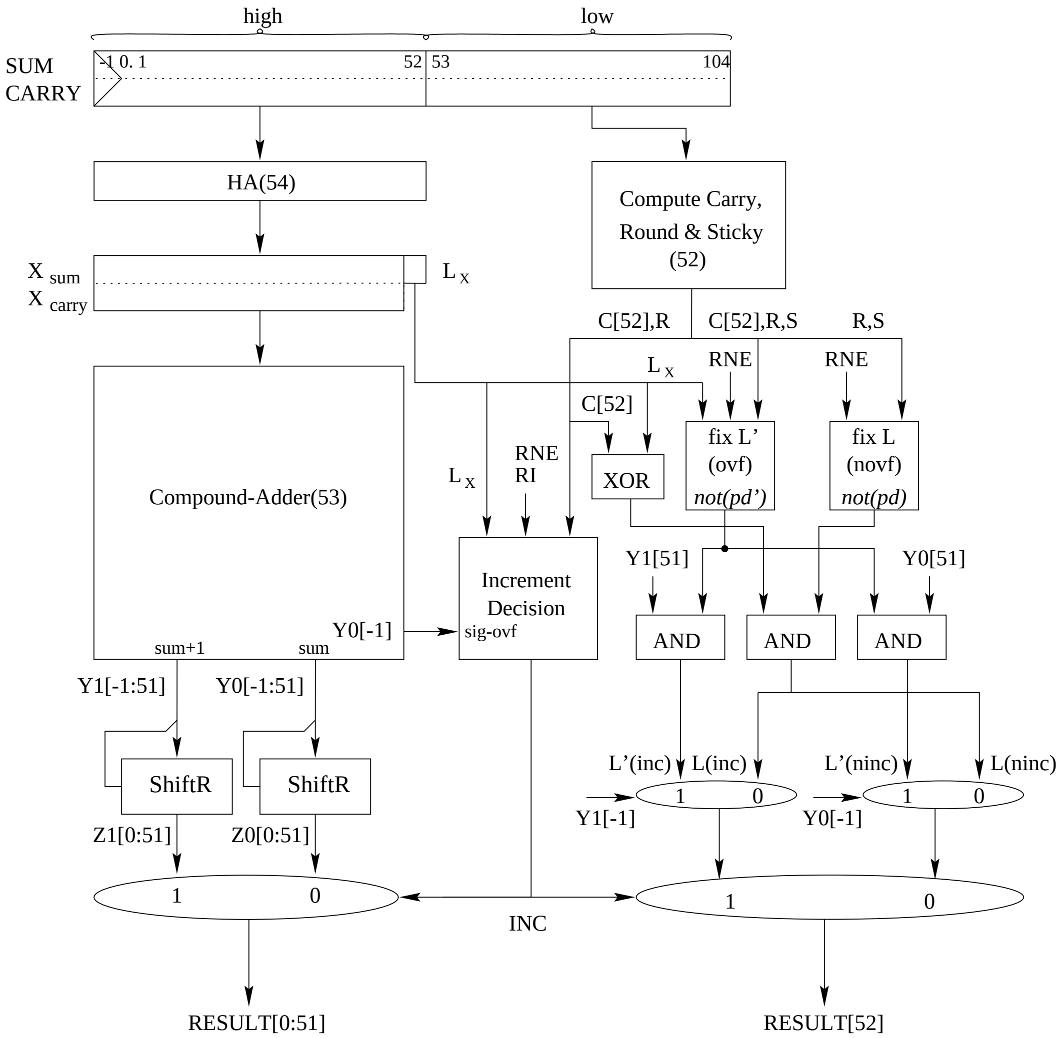

Figure 2: Block diagram of the ES rounding algorithm

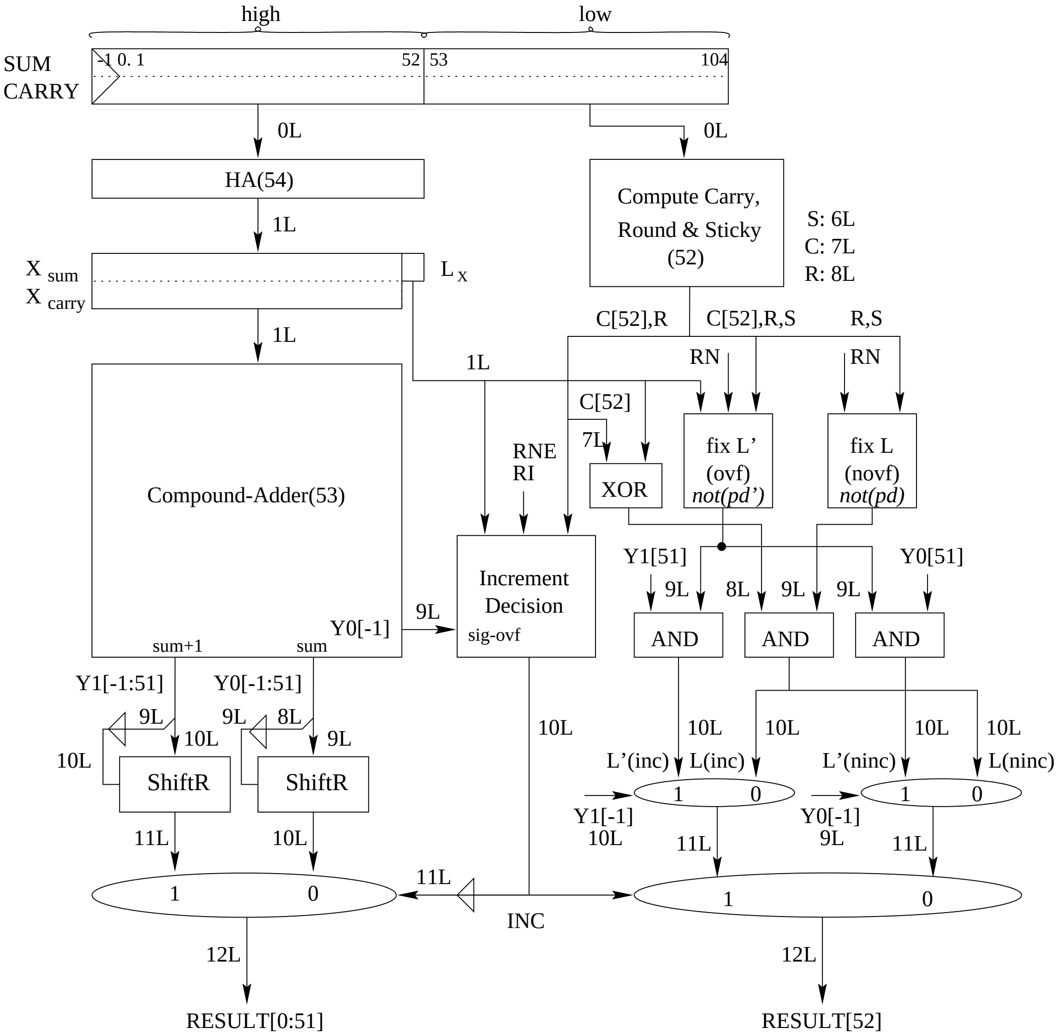

Figure 3: Block diagram of the ES rounding algorithm annotated with timing estimates

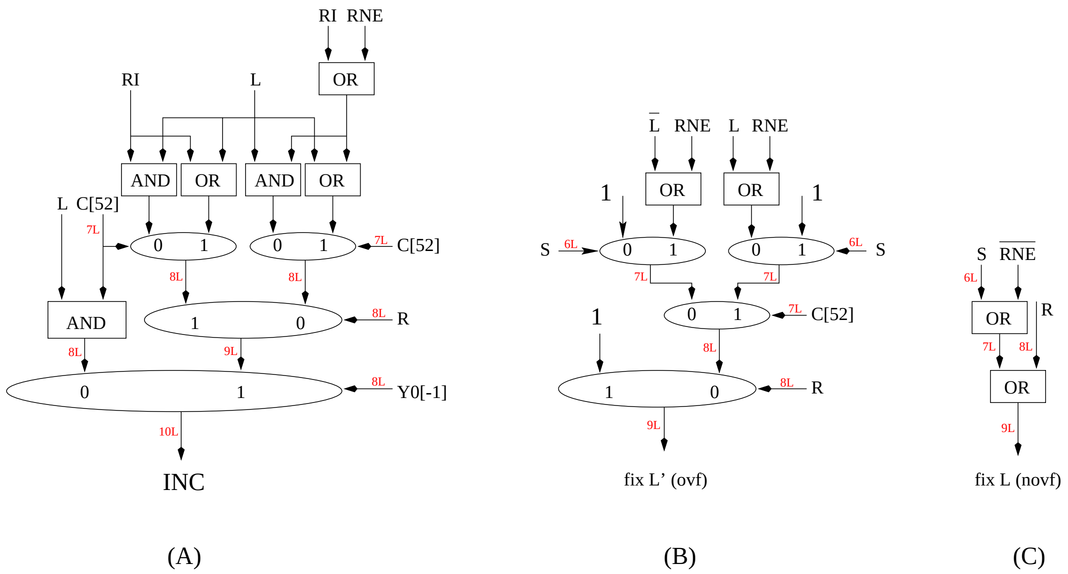

Figure 4: Details of boxes in the ES rounding algorithm: (A) The Increment Decision box; (B) The fix L' (ovf) box; and (C) The fix L (novf) box.

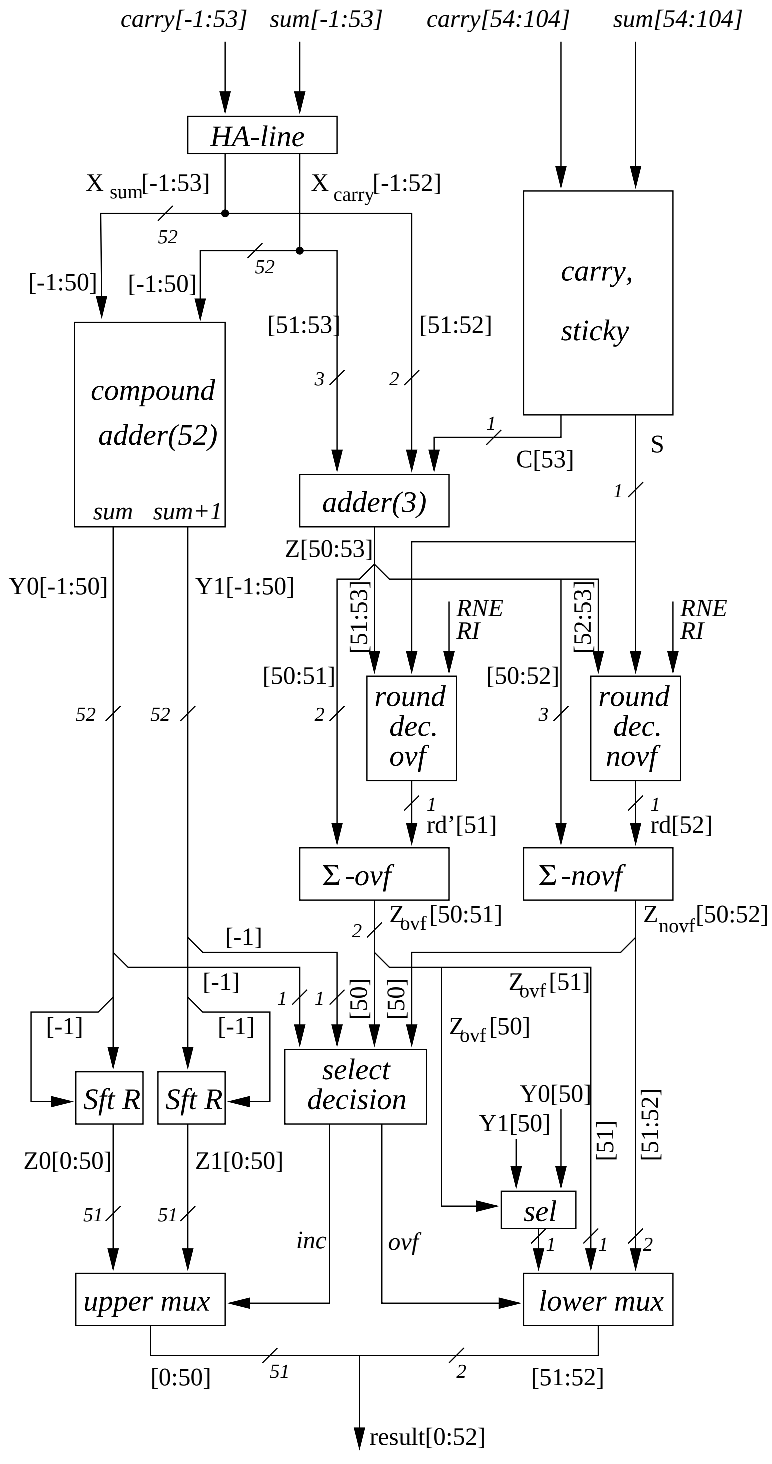

Figure 5: Block diagram of the YZ rounding algorithm

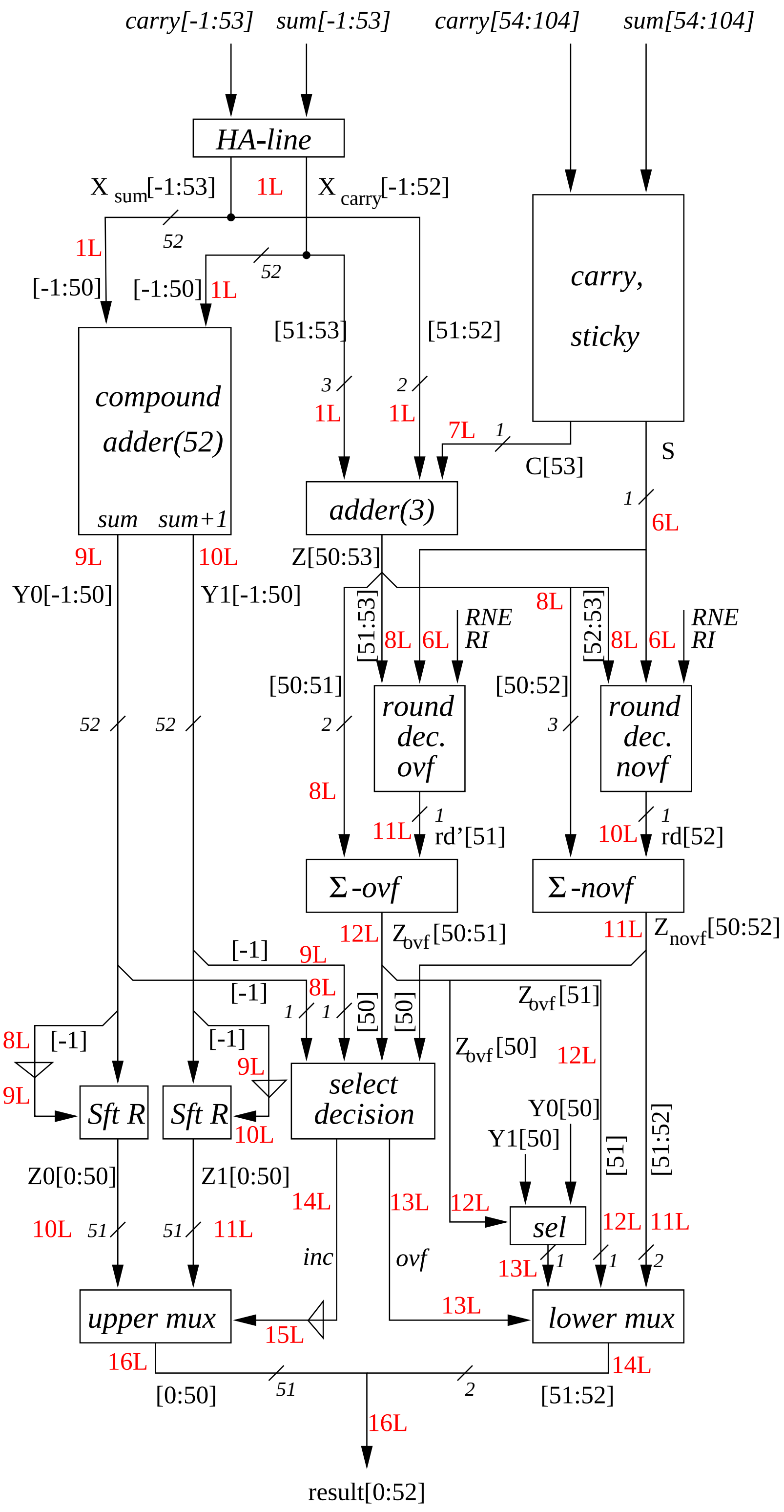

Figure 6: Block diagram of the YZ rounding algorithm annotated with timing estimates

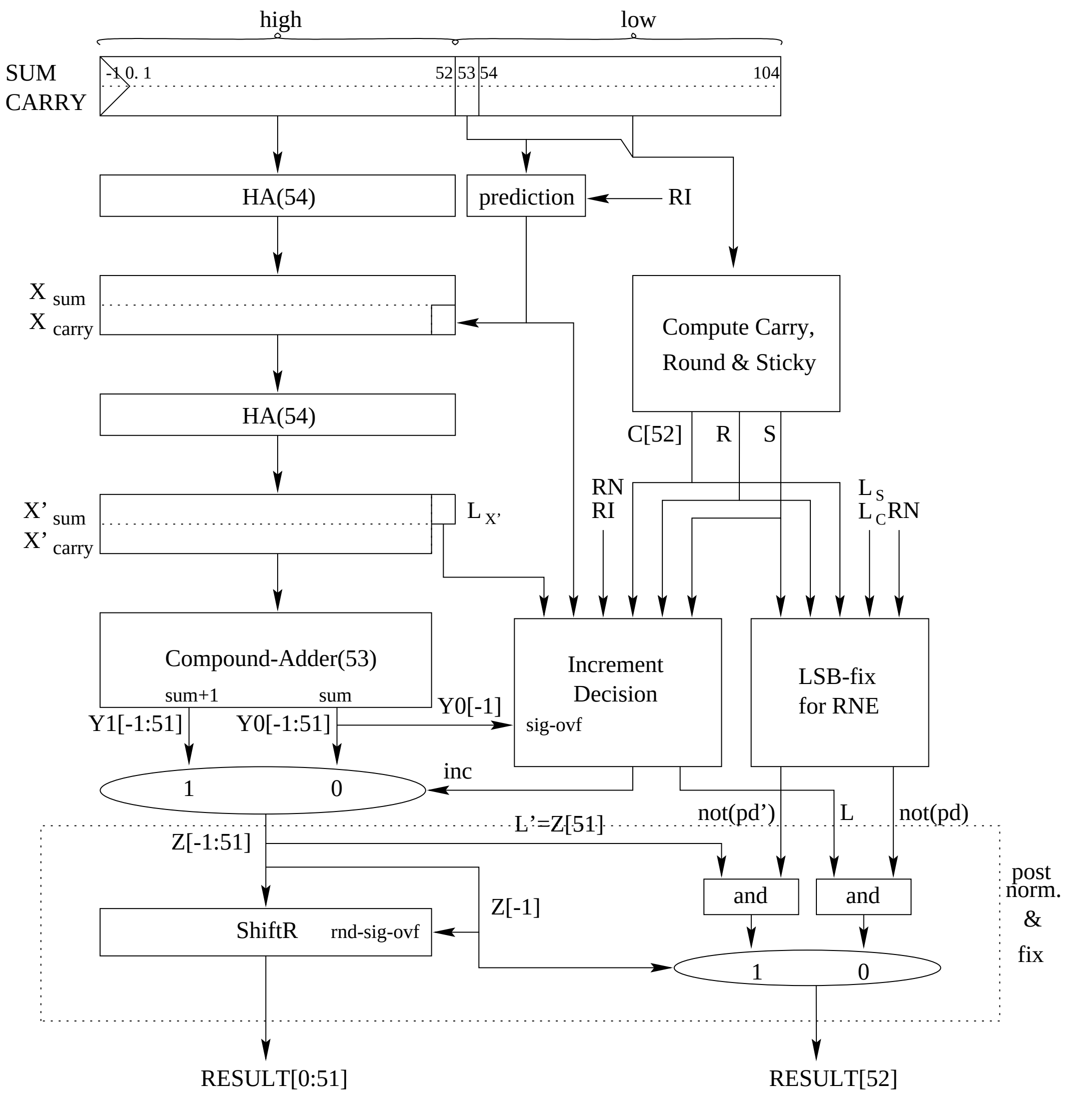

Figure 7: Block diagram of the QTF rounding algorithm

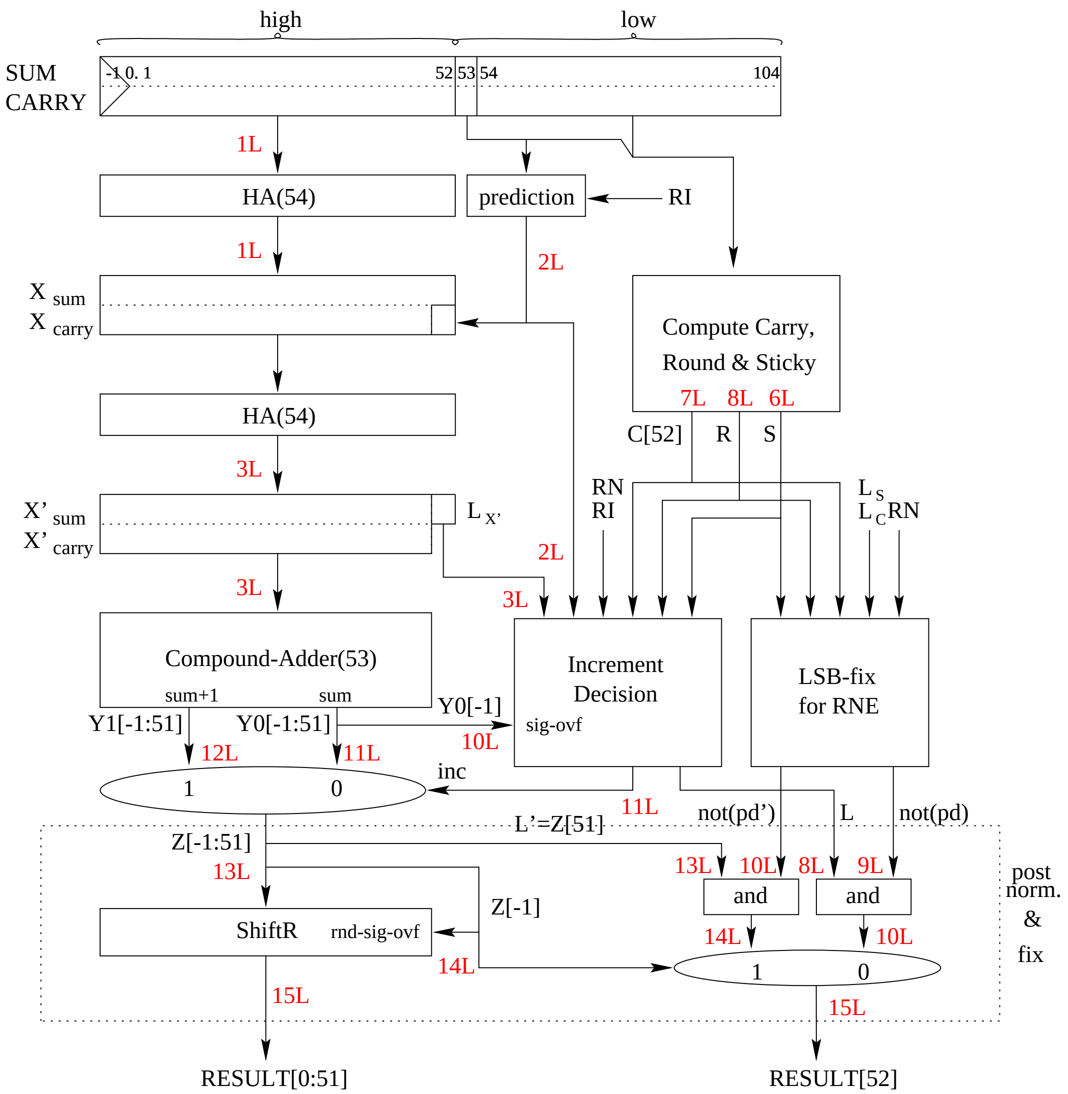

Figure 8: Block diagram of the QTF rounding algorithm annotated with timing estimates