ON FAST IEEE ROUNDING

Nhon Quach, Naofumi Takagi, and Michael Flynn

Technical Report: CSL-TR-91-459

January 1991

This work has been supported by the NSF contract No. MIP88-22961.

ON FAST IEEE ROUNDING

by

Nhon Quach, Naofumi Takagi, and Michael Flynn

Technical Report: CSL-TR-91-459

January 1991

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305

Abstract

A systematic general rounding procedure is proposed for floating-point arithmetic operations. This procedure consists of 2 steps: constructing a rounding table and selecting a prediction scheme. Optimization guidelines are given in each step to allow hardware to be minimized. This procedure-based rounding method has the additional advantage that verification and generalization are straightforward. Constructing a rounding table involves examining the range of the result and the shifting possibilities during the normalization step in an operation while selecting a prediction scheme depends on detail of the hardware model used. Two rounding hardware models are described. The first is shown to be identical to that reported by Santoro *et al.* [1]. The second is more powerful, providing solutions where the first fails. Applying this approach to the IEEE rounding modes for high-speed conventional binary multipliers reveals that round to infinity is more difficult to implement than the round to nearest mode; more adders are potentially needed. Round to zero requires the least amount of hardware.

A generalization of this procedure to redundant binary multipliers reveals two major advantages over conventional binary multipliers. First, the computation of the sticky bit consumes considerably less hardware. Second, implementing round to positive and minus infinity modes does not require the examination of the sticky bit, removing a possible worst-case path.

A generalization of this approach to addition produces a similar solution to that reported by Quach and Flynn [2]. Although generalizable to other kinds of rounding as well as other arithmetic operations, we only treat the case of IEEE rounding for addition and multiplication; IEEE rounding because it is the current standard on rounding, addition and multiplication because they are the most frequently used arithmetic operations in a typical scientific computation.

Key Words and Phrases: IEEE rounding, high-speed parallel multipliers, high-speed floating-point adders, redundant binary multipliers, redundant binary representation

Contents

List of Figures

List of Tables

1 Introduction

Not all real numbers are representable in a computer because of the limited precision in hardware. Rounding is a many-to-one mapping that maps an unrepresentable number into a representable one. This mapping process can happen during input/output of the data [3, 4] and during computation. This paper deals with the latter. In particular, we consider the mapping that was specified by the IEEE committee [5] and for addition and multiplication only. Rounding in the other operations is not as critical an issue because of their typically much larger latencies. But the approach to be proposed can be used in these operations as well.

In the IEEE floating-point (FP) format, a (real) number is represented as $(-1)^k \times M \times 2^{e+B}$ where k is the sign, M the significand, e the exponent, and B the bias. Bias is a constant offset needed for representing negative exponents. For a normalized number, $M \in [1, 2)$ (i.e., $1 \leq M < 2$). The number of bits N in the significand depends on the precision of the number. For double-precision $N = 53$, and for single-precision $N = 24$.[1]

In high-speed multipliers, multiplication is typically carried out by first generating many partial products (PP) in parallel, followed by a reduction step reducing these PP to two terms, sum (S) and carry (C) [6, 7]. The final summation is then carried out by a carry propagate adder (CPA). The significand of the product has $2N$ bits, requiring a rounding operation to map it into an N-bit number and possibly a right shift for normalization. In some high-speed multipliers, the operands are first recoded into a redundant binary form to facilitate the PP reduction step. This type of multipliers, hereafter called redundant binary multipliers (RBM), has been shown to be an interesting, viable alternative to conventional binary multipliers (CBM) [8, 9].

The significand in a sum of two numbers may have more than N bits, again requiring a rounding and possibly a normalization operations.[2] Normalization in this case is more complicated. One has the possibilities of a right shift, no shift, and a left shift. The sum has more than N bits when the smaller operand has to be right shifted to align with the larger one before addition or subtraction. For subtraction, the operand to be subtracted also needs to be 2's complemented. The final significand addition step is then carried out by a CPA. Note that in addition the significands of the two operands play the roles of S and C in multiplication. From rounding's point of view, however, there is little difference between addition and multiplication other than the ranges of S and C.

For performance reasons rounding is preferably combined with the CPA step and done in hardware because it is required in each operation. Past rounding works, on multiplication by Santoro *et al.* [1] and on addition by Quach and Flynn [2], are rather *ad hoc*, incomplete, and at too low a level, making their verification and generalization difficult and systematic exploration of the solution space impossible. In this paper, we present a systematic rounding procedure and give optimization guidelines to minimize the hardware used. Our procedure-based rounding approach has the additional advantage that verification and generalization to other types of multipliers (e.g., RBM) are straightforward. Though applicable to other

[1] Including the hidden one bit.

[2] In the case of subtraction, the significand may have less than N bits, but we consider trailing zeroes as part of the significand.

more general forms of rounding, we shall restrict our treatment to the IEEE ones.

In Section 2 we review and discuss the issues surrounding the IEEE rounding modes. In Section 3 we first illustrate our approach on CBM and then, in Section 4, generalize it to RBM. RBM uses a special form of signed-digit representation [10, 11], called redundant binary representation (RBR). We also review RBR in this section. In Section 5 we treat rounding in addition. Section 6 is a summary.

1.1 Notation

For notational simplicity, we define our binary point to be at the $(N-1)^{th}$ bit of S and C. The $(N+1)$-bit integer portions are denoted S_I and C_I and the $(N-1)$-bit fraction portions S_F and C_F (Figure 1). Our numbering convention always starts with 1. Also,

$$R = S + C$$

$$R_I = S_I + C_I$$

and

$$R_F = S_F + C_F$$

where S_I, C_I, and $R_I \in [2^{N-1}, 2^{N+1})$; S_F and $C_F \in [0,1)$; and $R_F \in [0,2)$ for multiplication and $\in [0,1]$ for addition. This is because in addition, either S_F or C_F is equal to zero. W denotes the logical OR of all the bits in R_F below the most significant bit (MSB). Q the carry-out bit of an adder, and $X.Y$ the Y bit of X. The least significant bit (LSB) of R_I, for example, will be denoted $R_I.LSB$.

Figure 1: Explanation of Notation. K represents S, C, or R.

2 IEEE Rounding Modes

Four rounding modes are dictated by the IEEE 754 standard [5]: Round to nearest (RN), round to positive infinity (RP), round to minus infinity (RM), and round to zero (RZ). Implementation*wise*, these 4 rounding modes can be reduced to 3: round up (RU)

Table 1: Implementation of IEEE Rounding Modes

IEEE Rounding Modes	Positive Number	Negative Number
	Treated As	
RN	RU with fix up	
RP	RI	RZ
RM	RZ	RI
RZ	RZ	

with a fix up, round to infinity (RI), and RZ, as shown in Table 1. Mathematically, for RU,

$$x = \begin{cases} \lceil x \rceil & \text{if } x - \lfloor x \rfloor \geq 0.5 \\ \lfloor x \rfloor & \text{otherwise,} \end{cases}$$

$x = \lceil x \rceil$ for RI, and $x = \lfloor x \rfloor$ for RZ, where $\lceil x \rceil$ and $\lfloor x \rfloor$ are the ceiling and floor functions [12], respectively. That RN can be implemented as RU with a fix up can be seen in Table 2. In the table, L is the LSB of the result, G the guard bit, and s the sticky bit. G is the bit of one-bit less significance than the L bit while s is the logical OR of all bits beyond the G bit. The x's in the table are the *don't care* terms. The entries in the table are the values of the rounding bit to be added to the result *at the L bit position.* From the table, we see that RN and RU differ in only one case, which can be fixed up by setting $L = 0$ without generating a carry propagation.

Table 2: Comparison of Round to Nearest and Round Up

L	G	s	RN	RU
x	0	0	0	0
x	0	1	0	0
0	1	0	0	1
0	1	1	1	1
1	1	0	1	1
1	1	1	1	1

For IEEE rounding, we have to perform the following:

1. Compute R (and therefore C and S) to a precision of $2N$ bits. For addition, this step is simpler because the non-shifted significand has a precision of only N bits. Consequently, the precision of R depends on the number of bits in the right shift during the alignment process and on the operation called for (i.e., addition or subtraction).

2. Normalize R if necessary by shifting it appropriately.

3. Compute the L, G, and s bit. The L bit corresponds to the N^{th} bit of R, and the G bit to the $(N-1)^{th}$ bit of R. The s bit requires ORing of the $N-2$ lower-order bits of R.

4. Denote the higher-order N bits of R as R_H. Based on the rounding mode and the L, G, and s bits, add a rounding "1" to R_H when necessary.

5. If $R_H.Q = 1$ as a result of rounding, right shift R_H by one bit, and adjust the exponent accordingly. R_H is the significand of the final result.

Note that R_H in this case is different from R_I defined earlier. R_H denotes the higher-order N bits of R *after* the normalization step. Several implementational difficulties associated with IEEE rounding can now be identified. First, S, C, and R need to be computed to a precision of $2N$ bits, requiring more hardware. Second, correct rounding in certain modes depends on the s bit, requiring more time because R must first be computed and then rounded using an extra addition step.

Rounding was not an issue before the establishment of the IEEE standard; few processors support RP and RM, and RN was nonexistent. Rather, most processors only implement RU. Even in this case, RU was not truly RU in the sense discussed above. In some high-speed multipliers, for example, the product was only computed to a precision of N plus a couple extra bits. RU is then *approximated* by adding a rounding "1" at the $N-1$ bit position — the G bit position — during the CPA step. Rounding in this case requires no extra addition step.

A natural question is whether we can do away with the extra addition step in IEEE rounding. In the following section, we show that such a rounding scheme is indeed possible and shall establish a procedure for doing so. But for the moment, let's examine the above rounding procedure again and consider the implications. To eliminate the rounding addition step in step 5, it must be combined with step 1, where R is computed. Because the rounding "1" ($r(1)$) is added to R at the N bit position, we must therefore compute R_I and R_F separately, breaking R into two parts. Further, the computation of R_F may propagate an overflow "1" ($o(1)$) into R_I. This means that we have to compute R_I, $R_I + r(1)/o(1)$, and $R_I + r(1) + o(1)$ in parallel and select the correct one. Finally, because rounding is now performed before normalization, an $r(1)$ becomes an $r(2)$ when a right shift is needed during normalization; i.e., we now have to compute $R_I + r(2)$, instead of $R_I + 1$, for rounding. The observation here is that the possible outcomes are small, making possible the rounding scheme to be proposed below.

3 Rounding for Binary Parallel Multipliers

The rounding procedure has 2 steps: constructing a rounding table and selecting a valid prediction scheme. Using RU as an example, we first explain how to construct this rounding table and then describe a simple rounding hardware model before outlining methods to select a valid prediction scheme. Finally, we present an improved model, which, unlike the simple one, has the advantage that it provides a solution for all the IEEE rounding modes.

3.1 Round Up

3.1.1 Constructing a Rounding Table

The number of columns in the table corresponds to the shifting possibilities in the normalization process. For multiplication the result may require a right shift to normalize, giving 2 columns: no right shift (NRS) and right shift (RS). The number of rows in the table, on the other hand, depends on both the rounding modes and the magnitude of R_F. For RU, $R_F \in [0, 2)$ and four rows, called regions, are required. Each entry in the table corresponds to the correction that must be performed to R_I to obtain the final result, rounded properly. A rounding table for the RU mode is given in Table 3.

Table 3: Rounding Table for Round Up in Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$0 \leq R_F < 0.5$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
2	$0.5 \leq R_F < 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
3	$1 \leq R_F < 1.5$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{2, 3\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
4	$1.5 \leq R_F < 2$	$R_I + 2$	$R_I.LSB = 0$: $R_I + \{2, 3\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$

We first consider the entries in Region 1. When R_I is normalized, no adjustment of the result is needed because $R_F < 0.5$. When R_I needs an RS to be normalized, we consider two cases.

- When $R_I.LSB = 0$: after shifting, $R_F \in [0, 0.25)$, and the result needs no correction. However, there is an optimization here. Because $R_I.LSB = 0$ and it is to be discarded after the shift, adding an *equivalent* "1" ($e(1)$) to R_I would not change the result. Thus, the possible correction digits are {0,1}, producing the $R_I + \{0, 1\}$ entry in the table.

- When $R_I.LSB = 1$: after right shifting, the effective $R_F \in [0.5, 0.75)$. Either an $r(2)$ or an $e(1)$ can be added to R_I for correction; therefore, we have the $R_I + \{1, 2\}$ entry.

To obtain Region 2, because $R_F \in [0.5, 1)$, an $r(1)$ needs to be added to R_I when the result does not need an RS. The RS entries are obtained using a similar argument as that for Region 1. To obtain Region 3, consider the NRS case. Because $R_F \in [1, 1.5)$, an $o(1)$ needs to be added to R_I, obtaining the entry $R_I + 1$. In the case of R_I needing an RS, we again consider two cases. When $R_I.LSB = 0$, the effective $R_F \in [0.5, 0.75)$; hence, we can add either an $r(2)$ or an $e(3)$ to R_I. When $R_I.LSB = 1$, we can add either an $o(1)$ or an $e(2)$ as indicated in the table since $R_F \in [1, 1.25)$.

To obtain Region 4, we need to add both an $r(1)$ and an $o(1)$ to R_I in the case of NRS. The RS entries are obtained as follows. After the RS, $R_F \in [0.75, 1)$. When $R_I.LSB = 0$, either an $r(2)$ or an $e(3)$ can be added to R_I, obtaining the $R_I + \{2, 3\}$ entry. When $R_I.LSB = 1$, R_F effective $\in [1.25, 1.5)$; hence, either an $o(1)$ or an $e(2)$ can be added, hence yielding the $R_I + \{1, 2\}$ entry in the table. An intuitive argument might have yielded an $R_I + \{3, 4\}$ entry because both an $r(2)$ and an $o(1)$ need to be added to R_I. But such an argument is incorrect.

Table 3 indicates at least three possible outcomes: R_I, $R_I + 1$, and $R_I + 2$. They must be computed in parallel and the correct one selected if speed is to be minimized as pointed out earlier. A straightforward way to accomplish this is to use three adders. In any events, because we are computing multiple outcomes in parallel, which MSB to observe for the need of a right shift is an interesting question. Preserving event sequentiality is the issue here. The overflow "1" is added to R_I before RS, and rounding comes later; hence, $R_F.Q$ determines the correct MSB. In Regions 1 and 2, for example, $R_F.Q = 0$ because $R_F < 1$; the MSB of the adder computing R_I should be examined. In Regions 3 and 4, because $R_F.Q = 1$, it is the MSB of the adder computing $R_I + 1$.

Methods to determine the minimum number of regions are also of interest because it may effect the number of bits that we have to examine for prediction (as explained below). Minimum number of regions is ensured by starting off with as many regions as needed and then coalescing them whenever possible. The necessity of a region is determined by examining events that can happen in the range of R_F and may effect R_I. In this case, rounding, overflow, and RS are the events of significance. Overflow here means that the value of R_F exceeds 1.

Informally speaking, the goal of rounding is to implement the rounding table with as little hardware as possible. Optimization is performed through judicious selection of the correction digits, the optimization targets being the number of adders and the complexity of the selection logic selecting the final result. Reducing the former saves hardware, and reducing the latter improves speed. Reducing the former may have an adverse effects on the latter; however, such an event is unlikely for current mainstream ECL and CMOS technologies. High-speed *parallel* adders are area-intensive, and low-speed adders are time-consuming. Besides this area and/or time savings, a smaller number of adders also means less power consumption in ECL and less capacitive loading in CMOS. Both have a positive effect on speed. Our first goal is therefore to reduce the number of adders.

The straightforward approach mentioned above is unattractive because of the number of adders required. To reduce the number of adder, one obvious way is to reduce the number of possible outcomes in a rounding table. But this is not possible because the number of possible outcomes depends mainly on the rounding mode. Seemingly a hopeless situation, but there is a way. High-speed adders generally employ some form of carry-lookahead network for carry propagation. Prime examples of this type of adders are carry-select [13, 14], carry-lookahead, and conditional sum [15]. For these adders, only the carry-lookahead network needs to be duplicated for simultaneous computation of R_I and $R_I + 1$

(denoted here as R_I+(0, 1)).[3]) Hence, adders computing adjacent results can be combined to produce a so-called compound adder, reducing the number of adders needed to implement a rounding table. This method is especially effective when coupled with the following observations on Table 3.

1. To know exactly which columns (i.e., the RS or the NRS columns) R_I is in requires a full $2N$-bit addition because we have to compute R.

2. To know exactly which rows R_F is in requires an $(N-1)$-bit addition because we have to compute R_F. However,

3. To predict *approximately* which rows R_F is in requires only an examination of $S_F.MSB$ and $C_F.MSB$. This is illustrated in Table 4. When both MSBs are 0, we know that R_F must be less than 1, corresponding to Regions 1 and 2 (Group 1). Similarly when one of the MSBs is 1, R_F must be in the range of $[0.5, 1.5)$, corresponding to Regions 2 or 3 (Group 2). Finally, when both MSBs are 1, R_F must be in the range of $[1, 2)$, or in Regions 3 or 4 (Group 3).

Table 4: Predicting the Regions with the MSBs of C_F and S_F

$S_F.MSB$ $C_F.MSB$	Regions	Group
0 0	1 or 2	1
0 1	2 or 3	2
1 0	2 or 3	2
1 1	3 or 4	3

Hence, rather than having to implement the whole rounding table, prediction allows us to only implement a group. Within a group, the outcomes are computed by a compound adder, and we select the correction digits such that all outcomes in the group are computable by this adder. Different groups may compute outcomes that are numerically different. This is resolved by a predictor, which *predictively* adds a fixed constant (usually the difference) to S_I and C_I before sending them to the compound adder. A hardware model for rounding emerges naturally at this point. We digress to present it because selection of a prediction scheme depends on its detail.

3.1.2 A Simple Hardware Model

Figure 2 shows a simple hardware model for rounding. In this model, we have an $(N+1)$-bit compound adder computing $C_I + S_I + p + (0, 1)$ where $p \in \{0, 1\}$ and is determined by the predictor based on a certain prediction scheme (to be established later), a row of half adder that produces an empty slot for later insertion of p at the compound adder, a selector that

[3] In fact, simultaneous computation of $R_I + (0, 1, 2)$ is possible as we shall see shortly.

controls the final selection multiplexors selecting among the possible outcomes, an R_F-adder that computes R_F, and some logic for determining the L, G, and s bits. The selector also has to perform the implicit task of shifting the selected result as needed. The final selection multiplexors may get unwieldy if the number of results is large, necessitating a two-step multiplexing scheme at this time. The following discussion assumes a single-stage selection scheme.

Figure 2: A Hardware Model for IEEE Rounding

In general, the selector needs to examine the *appropriate* $R_I.MSB$ bit, the $R_I.LSB$ bit, the $R_F.Q$ bit, the $R_F.MSB$ bit, and the W bit to select the correct result. To know exactly the region R_I is in, we need the $R_F.Q$ and $R_F.MSB$ bits; and to know exactly the column R_I is in, we need the $R_F.Q$ and $R_I.MSB$ bits. This step is not optimizable because of our inability to predict the future. The $R_I.LSB$ bit allows selection of the correct result in the RS column. This step is optimizable in that it may be simplified, or completely avoided, through judicious selection of a prediction scheme and the correction digits. The need to examine the W bit depends on the sizes of the regions in the rounding table and is not always needed. But when it is needed, it is not optimizable. The size of the region in Rounding Table 3, for example, is 0.5. The smaller the region, the more bits one potentially has to examine. A single-point region requires an examination of all bits.

For RU, only the former 4 bits need to be examined. Normally, these bits are known at approximately the same time with the exception of the $R_I.LSB$ bit, which is known much sooner. A circuit-level optimization is possible here. The paths computing the $R_I.MSB$ bit in the R_I-adder and the $R_F.Q$ and $R_F.MSB$ bits in the R_F-adder can be selectively sped up for early determination of the column and region. After the correct result is selected, the L, G, and s bits can then be determined. These bits are needed for the fix-up step in

RN.

3.1.3 Selecting a Valid Prediction Scheme

The number of prediction schemes available to the predictor is an exponential function of the number of regions in a rounding table. Not all of these prediction schemes are valid, however, because selection of the correction digits under one has to satisfy the requirement that outcomes in all groups be computable by the compound adder. Investigating possible prediction schemes is an important issue because they affect the complexity of the selector logic. The next goal of the design should be to reduce the complexity of this selector logic, the general guideline being the avoidance of the need to examine the $R_I.LSB$ bit. The complexity of the predictor logic, on the other hand, depends on the prediction scheme and is relatively insignificant because of the small number of literals involved. But a simple predictor logic — when happens to result — never hurts.

Using Table 3 as an example, because there are 4 regions, we have a total of 16 possible prediction schemes, but only one is valid using the simple hardware model, as shown in the prediction table (Table 5). In this particular case, the prediction scheme, which corresponds to a simple ORing of $C_F.MSB$ and $S_F.MSB$, can be easily guessed from Table 4 because a prediction "1" needs to be added whenever we know that we may be in Region 3. In general, an exhaustive search may be needed before a suitable predictation scheme can be found. It is interesting to note that this prediction scheme is unique for the simple model and corresponds to Algorithm 3 reported in [1].

Table 5: Prediction Table for Simple Model

$S_F.MSB$ $C_F.MSB$	Regions	Predictor Action
0 0	1 or 2	R_I
0 1	2 or 3	$R_I + 1$
1 0	2 or 3	$R_I + 1$
1 1	3 or 4	$R_I + 1$

After a prediction scheme is chosen, we can now select the correction digits for Table 3. Based on the prediction table, the allowable outcomes in Group 1 is $R_I + (0, 1)$; hence, the digit "2" in Regions 1 and 2 must be discarded. Similarly, the digit "0" in Region 2 and the digit "3" in Region 3 must be discarded because the allowed outcomes in this group are now $R_I + (1, 2)$ as a result of prediction. The digit "3" in Region 4 also needs to be discarded for the same reason, leaving only 3 pairs of selectable correction digits in the table. Table 6 lists all 8 possible digit selections for the simple model. In the table, only the RS columns are shown; all NRS columns are the same as those in Table 3. To simplify notation, the actions corresponding to the cases of $R_I.LSB = 0$ and $R_I.LSB = 1$ are denoted as $R_I + c1/c2$ where $c1$ and $c2$ are the selected correction digits. Digit selection scheme 8 does not require examination of $R_I.LSB$. Apparently, there is an advantage in using this scheme over the others.

Table 6: Possible Digit Selections for Round Up in Simple Model

Range of R_F	1	2	3	4	5	6	7	8
$0 \leq R_F < 0.5$	$R_I + 0/1$	$R_I + 0/1$	$R_I + 0/1$	$R_I + 0/1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$
$0.5 \leq R_F < 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$	$R_I + 1$
$1 \leq R_F < 1.5$	$R_I + 2/1$	$R_I + 2/1$	$R_I + 2$	$R_I + 2$	$R_I + 2/1$	$R_I + 2/1$	$R_I + 2$	$R_I + 2$
$1.5 \leq R_F < 2$	$R_I + 2/1$	$R_I + 1$	$R_I + 2/1$	$R_I + 1$	$R_I + 2/1$	$R_I + 1$	$R_I + 2/1$	$R_I + 2$

3.1.4 Improving the Simple Model

Other solutions are possible if we relax the constraint that we can only compute $R_I + (0, 1)$ in the compound adder as in the simple model. An improved rounding hardware model is given in Figure 3. This model differs from the simple one in two ways:

- The compound adder is reduced by 1 bit. The inputs to the adder come from the most significant N bits of S_I and C_I, giving it the ability to compute $R_I + (0, 1, 2)$. The outcome $R_I + 1$ is accounted for as follows. When $R_I.LSB = 0$, it is set to 1, and R_I is selected. When $R_I.LSB = 1$, it is set to 0, and $R_I + 2$ is selected. In either case no carry propagation is generated.

Figure 3: An Improved Hardware Model for IEEE Rounding

Note that if one views the last element in the half adder array in the simple model as an adder of length n ($n = 1$ in this case) and define m as the length of the compound adder, then we can compute $R_I + (0, \cdots, 2^n)$ as long as $n + m = N + 1$. When viewed in this light, another optimization reveals itself. A prediction "1" controlled by a predictor can be added to the n-bit adder (n-adder). This option may seem redundant for some rounding modes like RU. For others (e.g., RI), it is absolutely necessary as we shall see.

- Additional logic must be used to generate $R_I.LSB$. n is a major factor in determining the complexity of this $R_I.LSB$ logic. For this reason, a small n is to be preferred. If desired, the fix-up logic for RN mentioned in Section 2 can be incorporated into this $R_I.LSB$ logic.

The advantage of this improved model is that it allows more possible outcomes in a group, resulting in a greater degree of freedom in selecting a prediction scheme and the correction digits. More important, it often provides solutions to a rounding table where the simple model fails.

With this improved model, 8 prediction schemes are possible for Table 3, as listed in Table 7. Some of these prediction schemes are more interesting than others. Prediction schemes 3 and 5, for example, use $C_F.MSB$ and $S_F.MSB$ as the predictor while Prediction scheme 8 uses no prediction at all. Other prediction schemes may yield a simpler selector logic, however. Based on this prediction table, it can be easily shown that there are a total of 576 ways to select the correction digits as opposed to 8 for the previous simple model.

Table 7: Possible Prediction Schemes for Round Up for the Improved Model

$S_F.MSB$ $C_F.MSB$	1	2	3	4	5	6	7	8
0 0	R_I	R_I	R_I	R_I	R_I	R_I	R_I	R_I
0 1	R_I+1	R_I+1	R_I+1	R_I+1	R_I	R_I	R_I	R_I
1 0	R_I+1	R_I+1	R_I	R_I	R_I+1	R_I+1	R_I	R_I
1 1	R_I+1	R_I	R_I+1	R_I	R_I+1	R_I	R_I+1	R_I

3.1.5 Summary of Procedure

We summarize our procedure as follows.

1. For a particular rounding mode, identify the shifting possibilities in the normalization step. Determine the ranges of R_I and R_F. This step may require a careful analysis of the algorithm.

2. Determine the maximum number of regions needed by examining events that can happen in the range of R_F and can effect R_I. Examples of such events are shifts during the normalization step and rounding. Construct the rounding table, coalescing the regions whenever possible.

3. Determine the number of bits in R_F that need to be examined and construct a prediction table.

4. Based on the rounding table, develop a rounding hardware model.

5. Using this model and the rounding and prediction tables, find all optimal solutions. An optimal solution is one that uses the minimum number of adders and yields the simplest selector logic. In general, more than one optimal solutions are possible. Lesser constraints, such as the complexity of the predictor logic, can be used to further discriminate these solutions.

3.2 Round to Infinity

To derive a rounding scheme for RI, we use the procedure developed earlier in Section 3.1.5. In step 1 the range of R_I is the same as that for RU, requiring 2 columns: NRS and RS. In step 2 the range of R_F is again the same as that for RU, but the rounding threshold is now different. For RU the rounding threshold is 0.5, and for RI it is 0. The range of R_F must be divided into three: $R_F = 0$, $R_F \in (0, 1]$, and $R_F \in (1, 2)$. A rounding table for RI is given in Table 8. The considerations for obtaining the entries in the table are similar to those for RU. Table 9 is the prediction table for this rounding mode.

Table 8: Rounding Table for Round to Infinity in Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$R_F = 0$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
2	$0 < R_F \leq 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{2, 3\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
3	$1 < R_F < 2$	$R_I + 2$	$R_I.LSB = 0$: $R_I + \{2, 3\}$ $R_I.LSB = 1$: $R_I + \{3, 4\}$

Several observations can be made in relation to Table 8. First, $R_F = 0$ has it own region. This has a speed implication because the selector logic now has to await full computation of R_F and determination of the G and s bits before being able to select the final result. Second, in Regions 2 and 3 corresponding to Group 2 (Table 9), there are a total of three possible outcomes: $R_I + 1$, $R_I + 2$, and $R_I + 3$. Thus, there is no solution using the simple model. Third, there are a total of 4 possible outcomes in the table; therefore, even with the improved model, prediction is needed. Since we can only compute up to $R_I + 2$ in the compound adder, we need to add a prediction "1" to the n-adder as soon as we know that we *may be* in Region 3. This corresponds to the condition of at least one of the MSBs of S_F or C_F being true (Table 9). This prediction scheme is unique and corresponds to a simple ORing operation. We have now completed steps 3 and 4 of our procedure.

For step 5, there are a total of 16 different solutions because of the freedom in selecting the correction digits.

Table 9: Prediction Table for the Round to Infinity Mode

$S_F.MSB$ $C_F.MSB$	Regions	Group
0 0	1 or 2	1
0 1	2 or 3	2
1 0	2 or 3	2
1 1	3	3

The number of possible outcomes, their spread[4], and the sizes of the regions in a rounding table are often an indicator of the degree of difficulty in implementing a certain rounding mode. In this sense, RI is harder to implement than RN.

3.3 Round to Zero

A rounding table for RZ is shown in Table 10. Being a simple truncation, its table is the simplest among the rounding modes. Only two regions are required. Both hardware models have solutions, but the improved one has more because of the larger degree of freedom in selecting the correction digits. Not much more can be said about this rounding table other than the fact that if we select the digit "0" in Region 0 and the digit "1" in Region 1, then the selector does not have to examine the $R_I.MSB$. It only needs to right shift the result when $R_I.MSB = 1$.

Table 10: Rounding Table for Round to Zero in Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$0 \leq R_F < 1$	R_I	$R_I + \{0, 1\}$
2	$1 \leq R_F < 2$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$

[4]Spread is defined here as the absolute difference between the maximum and the minimum outcomes. In Table 8, for example, the minimum is R_I and the maximum is $R_I + 3$. The spread is therefore equal to 3 in this case.

3.4 Timing and Hardware Expenditure for IEEE Rounding in Binary Multipliers

It is informative to examine the sequence of events and the critical paths in the various rounding modes discussed above. We assume use of the improved model and start at the point when S and C arrive from the PP reduction hardware. The first three events are the same for all rounding modes. They are:

1. The predictor determines p. For some prediction schemes, this step may not be needed. In parallel, the R_F-adder computes R_F.
2. The half adder array computes $S_I + C_I + p$ after the determination of the prediction bit p.
3. The compound adder computes $C_I + S_I + p + (0, 1, 2)$.

The subsequent events will depend on the rounding modes. For RU, the events are:

4. Because the $R_F.Q$ and $R_F.MSB$ bits have been selectively sped up in the R_F-adder, these bits arrive sooner, waiting for the $R_I.MSB$ to select the final result.
5. The lower-order bits of R_F become available at this time.
6. The $R_I.MSB$ bit becomes available, and the selector selects the final result.
7. The L, G, and s logic determines the bits, and the fix-up step begins. In parallel, the $R_I.LSB$ logic determines the $R_I.LSB$.

For RI, they are:

4. Because the $R_F.Q$ and $R_F.MSB$ has been selectively sped up, these bits arrive sooner. But selection of the result cannot begin yet because Region 1 requires the full computation of R_F and then the W bit. Depending on the implementation detail, the W bit may or may not be available at the same time as the $R_I.MSB$. Hence, speeding up the paths mentioned above may not be worthy for this rounding mode.
5. the W bit is known, and the final result is selected.

And for RZ, they are:

4. Because the $R_F.Q$, $R_F.MSB$, and $R_I.MSB$ bits have been selectively sped up, these bits arrive sooner, and selection of the final result begins. The s bit and the G bits do not have to be computed for this rounding mode.

Table 11 lists the critical paths for all the rounding modes, with CA denoting the compound adder. In relation to the table, several observations can be made. First, for RN, the $R_F.Q$ and $R_F.MSB$ paths in the R_F-adder are not in the critical path. Since the final selection multiplexors present a large capacitive load to the selector, using a two-stage multiplexing scheme here helps because the load can be distributed between the stages and the $R_F.Q$ and $R_F.MSB$ bits can be used to select the multiplexors in the first stage. Second, the L, G, and s bit logic represents only a small portion of the overall delay. Speeding it up is not likely to matter a great deal. Third, the s bit is only needed for RN; RI needs the W bit. There is a subtle difference between the two; the s bit is determined based on the W bit $after$ the selection of the final result. This means that the W path is more of a critical path in RI than the s path in RN. Fourth, RI has two potential critical paths. Path 2 is likely worse because R_F has to be computed, making RI a possible overall worst-case path in the multiplier. Finally, RZ has the fastest execution time because it is the simplest.

Table 11: Possible Critical Path in the IEEE Rounding Modes

<table>
<tr><th rowspan="2">RN</th><th colspan="2">RI</th><th rowspan="2">RZ</th></tr>
<tr><th>Path 1</th><th>Path 2</th></tr>
<tr><td rowspan="2">Predictor
Half Adder Array
$R_I.MSB$ Path of CA
Selector
Multiplexors
L,G,s Logic</td><td>Predictor
Half Adder Array
$R_I.MSB$ Path of CA</td><td>R_F-Adder
W Logic</td><td rowspan="2">Predictor
Half Adder Array
$R_I.MSB$ Path of CA
Selector
Multiplexors</td></tr>
<tr><td colspan="2">Selector
Multiplexors</td></tr>
</table>

In terms of hardware expenditure, RU requires an $(N-1)$ bit adder for computing R_F, an additional carry-lookahead network for the compound adder, a selector, a predictor, and some logic for the fix-up step. The latter two are relatively inexpensive because they involve a small number of literals. The selector is only slightly more complicated logically, but does have to drive a large number of multiplexors. Most of the hardware penalty for IEEE rounding is in the R_F-adder and in the (duplicate) carry-lookahead chain, with the size of the latter roughly a quarter of that of the former. RI requires roughly the same functionality, and therefore the same amount of hardware. RZ does not need to examine the s and G bits, hence the simplest. For multipliers, there will be an additional item of hardware expense coming from the PP reduction hardware, which may dominate all the items discussed so far, depending on detail of the implementation — whether the design is iterative or full tree.

4 Rounding for Redundant Binary Multipliers

4.1 Redundant Binary Representation

As noted earlier RBR is a special form of signed-digit number representation [10, 11]. In this representation a digit is selected from the digit set $\{-\rho, \cdots, 0, \cdots, \rho\}$, subject to the constraint $r > \rho \geq \frac{r-1}{2}$ where r is the radix. For RBR, $r = 2$ and $\rho = 1$. The most important property of this number system is its allowance for carry-free addition. The one-time conversion cost from CBR to RBR and back can be more than offset by the speed enhancement and/or hardware reduction that the RBR system brings. In RBR, each digit in a number $\in \{-1, 0, 1\}$ and is represented with 2 bits. The fourth redundant state gives rise to many possible encoding schemes. Table 12 gives 2 such examples. In the table, $\overline{1}$ denotes -1. In Scheme 1, T_i is used to represent the sign of a digit and in Scheme 2, the negative digit itself. Scheme 1 corresponds to that reported in [8] and will be used for the remainder of this paper. Using this notation, then, possible RBRs for 46 and -46 are $T = 00000$ and $D = 101110$, and $T = 101110$ and $D = 101110$, respectively. Note that in this particular encoding scheme, the conversion from CBR to RBR is free.

Table 12: Encoding Schemes for Redundant Binary Representation

Redundant Binary Representation	Scheme 1 T_i D_i	Scheme 2 T_i D_i
$\overline{1}$	1 1	1 0
0	0 0	0 0
1	0 1	0 1

In [10] it was shown that totally parallel addition is possible for $r \geq 3$, and that carry-free addition in RBR requires an examination of the neighboring lower-order bit. In RBR, addition is performed in two steps. In the first step an intermediate carry and sum, u_i and v_i, are computed, satisfying the equation

$$x_i + y_i = 2u_i + v_i \tag{1}$$

where x_i and y_i are the i^{th} digits of the augend and addend, respectively. In the second step the final sum z_i is obtained by

$$z_i = v_i + u_{i-1} \tag{2}$$

In Eqns (1) and (2), x_i, y_i, z_i, u_i, and $v_i \in \{\overline{1}, 0, 1\}$. The key to carry-free addition in RBR is to select u_{i-1} and v_i such that they are neither all 1 nor all -1 in the first step (Eqn (1)), thereby prohibiting a carry propagation in the second step (Eqn (2)). Table 13 shows the addition rule for the first step. In the first row, both x_i and y_i equal 1, a $v_i = 0$ will prohibit a positive carry propagation into the $(i+1)^{th}$ position in the second step regardless of the values of x_{i-1} and y_{i-1}. Similarly, when x_{i-1} and y_{i-1} are nonnegative (i.e., 0 or 1), a $v_i = -1$ will again prohibit a positive carry propagation into the $(i+1)^{th}$ digit position. Other rows can be obtained using a similar argument. u_i and v_i, therefore, can be determined by examining only x_i, y_i, x_{i-1}, and y_{i-1}.

The final conversion from RBR to CBR can be done using the following equation:

$$A(= \sum_{i=0}^{n-1} a_i 2^i) = A^+(= \sum_{a_i=1} a_i 2^i) - A^-(= \sum_{a_i=-1} |a_i| 2^i) \tag{3}$$

Table 13: First Step Rule for Carry-free Addition in Redundant Binary Representation

x_i	y_i	$x_{i-1}y_{i-1}$	u_i	v_i
1	1	——	1	0
1	0	Nonnegative	1	$\bar{1}$
0	1	Otherwise	0	1
0	0	——	0	0
1	$\bar{1}$	——	0	0
$\bar{1}$	1	——	0	0
0	$\bar{1}$	Nonnegative	0	$\bar{1}$
$\bar{1}$	0	Otherwise	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	——	$\bar{1}$	0

where $a_i \in \{-1, 0, 1\}$ and each digit in A^+ and $A^- \in \{0, 1\}$. This conversion is typically done by using a conventional binary adder. No special hardware is needed.

An example of carry free addition taken from [8] is given below. In the example, the final conversion step is not shown.

Example: Adding 87 and 101

Augend:		$[10\bar{1}0\bar{1}00\bar{1}]$	(87)
Addend:	$+$	$[1\bar{1}10011\bar{1}]$	(101)
Intermediate Sum:		$[01001\bar{1}\bar{1}10]$	
Intermediate Carry:	$+$	$[1\bar{1}000101\bar{1}]$	
Sum:		$[1\bar{1}1000\bar{1}00]$	(188)

4.2 Redundant Binary Multipliers

Because of storage inefficiency, RBR is mainly used to represent the intermediate results in a computation. In RBM, for example, the binary operands are first encoded into RBR during the PP generation and reduction steps and later decoded into CBR at the final CPA step for external storage and compatibility to the binary world. Unlike CBM, the result of the PP reduction step has only one term, but is represented with 2 bit vectors T and D playing the roles of S and C in CBM. The final conversion step from CBR to RBR plays the role of the CPA step in CBM. Using the notation introduced earlier for S and C, T_F and $D_F \in (-1, 1)$, and T_I and $D_I \in (-2^{N+1}, 2^{N+1})$. Like CBM, an RS may be needed during normalization in RBM.

The improved hardware model only needs to be slightly modified for RBM. In the R_I path, the half adder array provides a convenient place for the RBR to CBR conversion. Suppose we have a number V_I represented in RBR with T_I and D_I. How do we convert them into S_I and C_I such that $S_I + C_I$ is equal or close to R_I? Equation (3) provides a basis for this conversion: we have to subtract the negative digits from the positive ones, all weighted properly. If V_{Ii} (the i^{th} bit of V_I) is equal to $\overline{1}$, we know that $A_i^+ = 0$ and $A_i^- = 1$. But to subtract A^- from A^+, we must first bit-invert A^-; hence, A_i^- becomes 0. Using the same argument, we have $A_i^+ = 0$ and $A_i^- = 1$ for $V_{Ii} = 0$; and $A_i^+ = 1$ and $A_i^- = 1$ for $V_{Ii} = 1$. The first three columns in Table 14 summarize these findings. In the table, we have used S_{Ii} and C_{Ii} for A_i^+ and A_i^-, respectively. The fourth column is similar to the third one, but it allows a carry-in to be added, effectively simulating a full adder in the improved model. Note actually that $S_I + C_I = R_I - 1$ because we had not added a complementation "1" during the conversion process. This fact can be used to advantage as we shall see shortly.

Table 14: Conversion from RBR to Binary Sum and Carry Representation

V_I	T_{Ii} D_{Ii}	C_{Ii} S_{Ii}	C_{Ii+1} S_{Ii}
$\overline{1}$	1 1	0 0	0 0
0	0 0	0 1	0 1
1	0 1	1 1	1 0
—	—	—	—

From the table, we obtain

$$S_{Ii} = \overline{D_{Ii}} \tag{4}$$

and

$$C_{Ii+1} = \overline{T_{Ii}} D_{Ii} \tag{5}$$

The full adder logic is only slightly more complicated:

$$S_{Ii} = \overline{C_{in}} \oplus D_{Ii} \tag{6}$$

and

$$C_{Ii+1} = \overline{T_{Ii}}(C_{in} \vee D_{Ii}) \tag{7}$$

where C_{in} is the carry-in to the "full adder". In the equations, $\oplus$ is used for exclusive OR, juxtaposition for logical AND, $\vee$ for logical OR, and overbar for logical inversion. These conversion logics (Eqns (4)-(7)) are simpler than the half and full adder logics used in CBM, showing an advantage of RBM.

In principle, the R_F path can be left unchanged. The R_F-adder converts T_F and D_F into a 2's complement number, allowing the selector to select the final result. The L, G, and s bits are determined in a similar fashion as in CBM. But there is a better way. In

CBM, the R_F-adder sums up S_F and C_F for later determination of the s bit. In RBM, this summation step is not needed because R_F is represented in RBR with T_F and D_F. To determine the equivalent of s for the fix-up step, we only need to know whether the number represented in the lower-order $(N-2)$ bits of T_F and D_F (denoted here as TD^{N-2}) is equal to 0 or is negative. Determining the former is easy; a simple ANDing operation suffices. Determining the latter requires the following recurrence. TD^i is negative if and only if the i^{th} digit is negative, or if it is zero and TD^{i-1} is negative. Translating, we have for $i \geq 1$

$$Neg_i = T_i + \overline{D}_i Neg_{i-1} \tag{8}$$

where Neg_i indicates the condition of TD^i being negative, and $Neg_0 \equiv 0$ (or false). Eqn (8) is similar in form to the well-known carry recurrence

$$c_i = g_i + p_i c_{i-1} \tag{9}$$

where c_i is the carry into the $(i+1)^{th}$ bit, g_i the bit generation, and p_i the bit propagation function. Note that both T_i and $\overline{D}_i$ are simpler than g_i and p_i, respectively. Many parallel techniques exist for solving this recurrence [16, 17, 18], and carry-lookahead serves as a good example. In short, the R_F-adder needs not be implemented in RBM, only the detection of zero and negativity is sufficient for the fix-up step, saving hardware and improving speed at the same time. In light of the previous discussion on RI being a potential worst-case path, this advantage is especially important for RBM.

4.3 Round Up

A rounding table for RU in RBM is given in Table 15. The table is constructed in much the same way as that for CBM. We consider the first row. In the NRS case, $R_I - 1$ is closer to the true result than R_I; hence, it is the desired result. In the RS case, when $R_I.LSB = 0$, the effective $R_F \in (-0.5, -0.25)$. So, R_I is closer to the true result. But because $R_I.LSB = 0$, $R_I + 1$ is also a solution. Hence, we have the $R_I + \{0, 1\}$ entry in the table. When $R_I.LSB = 1$, the R_F effective is (0,0.25). The closer result is again R_I. $R_I - 1$ is also a solution because $R_I.LSB = 1$ and changing a bit to be discarded will not affect the result. The entries in other regions can be obtained in a similar fashion.

Prediction is slightly different for RBM as shown in Table 16. When both MSBs of T_F and D_F are 0, we know that R_F must be $(-0.5, 0.5)$ or in Regions 2 or 3. Other rows are obtained similarly. As in the case of RU in CBM, at least three possible outcomes must be computed in parallel: $R_I - 1$, R_I, and $R_I + 1$.

For RU in RBM, eight prediction schemes are possible, but only 4 are valid as shown in Table 17, where $Z_I = R_I - 1$. Schemes 2 and 3 use the inverse of $T_F.MSB$ and $D_F.MSB$, respectively, as the predictors. As an example, we select the correction digits based on Prediction scheme 1 for the improved (modified) model; others can be obtained similarly. For this prediction scheme, because the allowable results in Group 1 (Table 16) are $R_I + (-1, 0, 1)$, the digit "2" in Region 3 must be discarded and similarly for the digit "2" in Region 4, leaving 6 pairs of selectable correction digits in the table. Hence, there are a total

Table 15: Rounding Table for Round Up in Redundant Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$-1 < R_F < -0.5$	$R_I - 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{0, -1\}$
2	$-0.5 \leq R_F < 0$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{0, -1\}$
3	$0 \leq R_F < 0.5$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
4	$0.5 \leq R_F < 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$

Table 16: Predicting the Regions in Redundant Binary Representation

$T_F.MSB$ $D_F.MSB$	Region	Group
0 0	2 or 3	1
0 1	3 or 4	2
1 1	1 or 2	3
—	—	—

of 2^6 solution for this prediction scheme. It can be easily shown that there are a total of 384 solutions for this rounding table using the (modified) improved model.

Table 17: Prediction Table for Round Up in Redundant Binary Representation

$T_F.MSB$ $D_F.MSB$	1	2	3	4
0 0	Z_I	Z_I+1	Z_I+1	Z_I
0 1	Z_I	Z_I	Z_I+1	Z_I+1
1 1	Z_I	Z_I	Z_I	Z_I
—	—	—	—	—

4.4 Round to Infinity

A rounding table for RI for RBM is given in Table 18. Table 19 is the prediction table. From the tables, it can be seen that Group 1 places the most stringent constraint on digit selection. Three possible outcomes are again needed: R_I, R_I+1, and R_I+2 as in CBM. A prediction "1" is always needed, corresponding to no prediction. This has a slight advantage over CBM. The digit "3" in Region 2 must be discarded because R_I+3 is not computable. There are a total of 8 solutions for this rounding table using the (modified) improved model.

Table 18: Rounding Table for Round to Infinity in Redundant Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$-1 < R_F \leq 0$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$
2	$0 < R_F < 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{2, 3\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$

Another advantage of RBM can now be identified. In CBM, the resolution of the regions requires the computation of R_F and the W bit. In contrast, only the $R_F.Q$ and $R_F.MSB$ bits are needed in RBM; i.e., the W bit path is no longer critical.

4.5 Round to Zero

A rounding table for RZ is shown in Table 20. Again, we can select the digit "-1" in Region 1 and 0 in Region 2 to simplify the selector logic as in CBM. Another interesting observation is that $R_I - 2$ is among the possible outcomes. This outcome is hard to compute and is to be avoided.

Table 19: Prediction table for Round to Infinity in Redundant Binary Representation

$T_F.MSB$ $D_F.MSB$	Region	Group
0 0	1 or 2	1
0 1	2	2
1 1	1	3
—	—	—

Table 20: Rounding Table for Round to Zero in Redundant Binary Multipliers

Region	Range of R_F	Normalization Shifts	
		No Right Shift	Right Shift
1	$-1 < R_F < 0$	$R_I - 1$	$R_I.LSB = 0$: $R_I + \{-2, -1\}$ $R_I.LSB = 1$: $R_I + \{-1, 0\}$
2	$0 \leq R_F < 1$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{-1, 0\}$

5 Rounding for Floating-Point Adders

5.1 Floating-Point Addition

For adding two FP numbers, a simple method has been given in [15]:

1. Exponent subtraction: Subtract the exponents and denote the difference d.

2. Alignment: Right shift the significand of the smaller operand by d bits and adjust the exponent accordingly. Denote the larger exponent E_f.

3. Significand addition: Perform addition or subtraction according to the effective operation, E_o, which is the arithmetic operation actually carried out by the adder in the FP unit.

4. Conversion: Convert to sign-magnitude representation if the result is negative. The conversion is done with an addition step. Denote the result R.

5. Leading one detection: Based on R, compute the amount of left or right shift needed and denote it E_n.

6. Normalization: Normalize the significand by shifting R by E_n bits and add E_n to E_f.

7. Rounding: Based on the rounding mode and on the L, G, and s bits, add a rounding "1" when necessary to the LSB of R_H.

8. Renormalization: If $R_H.Q = 1$ as a result of rounding, right shift R_H by one bit, and adjust the exponent accordingly. R_H is the significand of the final result.

In this algorithm, the computation of the L, G, and s bits for IEEE rounding can be performed at any time after the alignment step. Comparing the steps after the significant addition step in this algorithm with those in the rounding procedure presented in Section 2, one sees an almost one-to-one correspondence. So, the rounding method presented earlier for multiplication applies directly, with only a slight modification in the improved hardware model. The change is mainly in the predictor logic because either S_F or C_F is 0.

Assuming S_F is the shifted significand during alignment, the range of R_F can be determined as follows. For addition, the lower order bits of S_F, hence R_F, $\in [0,1)$. For subtraction, S_F needs to be bit inverted and a complementation "1" needs to be added at the $2N$ bit position; thus, $R_F \in [0,1]$.

5.2 Round Up

A rounding table for RU for addition is given in Table 21. Because normalization may require a left shift (LS), the table has 3 columns and four rows even if the range of R_F is smaller ($\in [0,1]$). Also, we now have the possibility of $R_I + 0.5$. This outcome can be accounted for as follows. If $R_F.MSB = 0$, it is simply set to 1, and if it is 1, it is set to 0 and the result $R_I + 1$ is selected. In either case, no carry propagation is generated. Consequently, an $R_I + 0.5$ entry can be converted into an R_I and an $R_I + 1$ entries when considering the possible outcomes in a group.

In Table 21, one sees that only two outcomes need to be implemented: R_I and $R_I + 1$. This is in agreement with the observation reported in [2], in which it was stated that for RU, the half adder array is not needed.

Some high-speed FP adders adopted a 2-path implementation. In the addition algorithm presented above, observe that a normalization requiring a 1-bit LS can only happen when the alignment step requires an RS of more than 1 bit. Conversely, a normalization requiring a massive left shift can only occur when the alignment step requires no shift or a 1-bit RS. Hence, the critical path is either an RS followed by a significand add or a significand add followed by an LS, but never both. For this implementation, each path can be implemented separately using a different and usually simpler rounding table. The reader is referred to [2] for a more detailed discussion on this 2-path implementation.

5.3 Round to Infinity for Addition

A rounding table for RI is given in Table 22. Three outcomes need to be implemented: R_I, $R_I + 1$, and $R_I + 2$. A row of half adder array is needed for this rounding mode, again in agreement with [2]. It is interesting to note that even for addition, RI requires mode hardware than RU.

Table 21: Rounding Table for Round Up in FP Adders

Region	Range of R_F	Normalization Shifts		
		No Shift	Right Shift	Left Shift
1	$0 < R_F < 0.25$	R_I	$R_I.LSB = 0$: $R_I + \{0,1\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	R_I
2	$0.25 \leq R_F < 0.5$	R_I	$R_I.LSB = 0$: $R_I + \{0,1\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	$R_I + 0.5$
3	$0.5 \leq R_F < 0.75$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0,1\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	$R_I + 0.5$
4	$0.75 \leq R_F \leq 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0,1\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	$R_I + 1$

Table 22: Rounding Table for Round to Infinity in FP Adders

Region	Range of R_F	Normalization Shifts		
		No Right Shift	Right Shift	Left Shift
1	$R_F = 0$	R_I	$R_I.LSB = 0$: $R_I + \{0,1\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	R_I
2	$0 < R_F \leq 0.5$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{2,3\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	$R_I + 0.5$
3	$0.5 < R_F \leq 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{2,3\}$ $R_I.LSB = 1$: $R_I + \{1,2\}$	$R_I + 1$

5.4 Round to Zero for Addition

A rounding table for RZ is given in Table 23. Not much more can be said about this table other than the fact that $R_F = 1$ is harder to predict than other regions.

Table 23: Rounding Table for Round to Zero in FP Adders

Region	Range of R_F	Normalization Shifts		
		No Shift	Right Shift	Left Shift
1	$0 \leq R_F < 0.5$	R_I	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: R_I	R_I
2	$0.5 \leq R_F < 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: R_I	$R_I + 0.5$
3	$R_F = 1$	$R_I + 1$	$R_I.LSB = 0$: $R_I + \{0, 1\}$ $R_I.LSB = 1$: $R_I + \{1, 2\}$	$R_I + 1$

6 Summary

In floating-point addition and multiplication, the right shift needed during the normalization step gives rise to many possible rounding schemes. In this paper, we have presented a systematic procedure for selecting among these schemes. This procedure starts with the construction of a rounding table and then uses prediction to reduce the number of possible outcomes in the rounding table. Constructing a rounding table involves examining the shifting possibilities during the normalization step in an operation and the range of the result while selecting a prediction scheme depends on the detail of a hardware model. Two rounding hardware models are presented. The first has been shown to be identical to that reported by Santoro *et al* [1]. The second is more powerful, providing solutions where the first fails. Both models compute all outcomes needed for a rounding table in parallel and select the correct one. Optimization guidelines have been outlined in each step to reduce the number of adders and the complexity of the selector logic.

Applying this approach to the four IEEE rounding modes for conventional binary multipliers reveals that round to positive and minus infinity modes are more difficult to implement than the round to nearest mode in terms of the number of possible outcomes and their maximum differences. Round to zero requires the least amount of hardware.

Generalizing this procedure to redundant binary multipliers [8, 9], we show that there are two major advantages in using redundant binary number representation in rounding. First, the logic for computing the sticky bit is simpler because only a carry chain is needed. A lesser simplification comes from the logic for the half adder array. Second, in round to positive and minus infinity modes, the W path is not a critical path as in conventional binary multipliers.

We then generalize our approach to addition, obtaining a similar solution as that reported by Quach and Flynn [2]. Though generalizable to other kinds of rounding and other arithmetic operations, we only treat IEEE rounding for addition and multiplication in this paper; IEEE rounding because it is the current standard on rounding, addition and multiplication because they are the most frequently used operations in a typical scientific computation. Division, remainder, and square root are the other operations in the IEEE standard not considered in this paper. We expect rounding in these operations to be simpler because of their larger latency and because of the fact that normalization in these operations does not involve a right shift.

In this work, we only consider methods for rounding at the final summation step. Other possibilities exist. In multipliers, for example, it is possible to inject a constant into the sum and carry terms during the partial product reduction step (e.g., into the Wallace tree). We believe, however, that the methodology presented in this paper can be easily modified to handle these cases as well.

References

[1] M. R. Santoro, G. Bewick, and M. A. Horowitz, "Rounding Algorithms for IEEE Multipliers," *Proc. of the* 9^{th} *Symposium on Computer Arithmetic*, pp. 176–183, 1989.

[2] N. T. Quach and M. J. Flynn, "An Improved Floating-Point Addition Algorithm," Tech. Rep. CSL-TR-90-442, Stanford University, Aug. 1990.

[3] W. D. Clinger, "How to Read Floating-Point Number Accurately," in *ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, pp. 92–101, 1990.

[4] J. Guy L. Steele and J. L. White, "How to Print Floating-Point Number Accurately," in *ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, pp. 112–126, 1990.

[5] *An American National Standard: IEEE Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Standard No. 754, American National Standrads Institute, Washington, DC, 1988.

[6] C. S. Wallace, "A Suggestion for Fast Multipliers," *IEEE Transactions on Electronic Computers*, no. EC-13, pp. 14–17, Feb. 1964.

[7] D. D. Gajski, "Parallel Compressors," *IEEE Transactions on Computers*, vol. C-29, pp. 393–398, May 1980.

[8] N. Takagi, H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Transactions on Computers*, vol. C-34, no. 9, pp. 789–796, Sep. 1985.

[9] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of High Speed MOS Multiplier Using Redundant Binary Representation," in *Proc. of the* 8^{th} *Symposium on Computer Arithmetic*, pp. 80–86, 1987.

[10] A. Avizieni, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Transactions on Computers*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[11] B. Parhami, "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representation," *IEEE Transactions on Computers*, vol. 39, no. 1, pp. 89–98, Jan. 1990.

[12] R. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathametics – A Foundation for Computer Science*. Menlo Park, California: Addison-Wesley, 1989.

[13] M. Uya, K. Kaneko, and J. Yasui, "A CMOS Floating Point Multiplier," *IEEE Journal of Solid-State Circuit*, vol. SC-19, no. 8, pp. 697–702, Oct. 1984.

[14] P. Y. Lu, A. Jain, J. Kung, and P. H. Ang, "A 32-MFLOP 32b CMOS Floating-Point Processor," in *In Proc. of the IEEE International Solid-State Circuit Conference*, pp. 28–29, 1988.

[15] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*. New-York: Holts, Rinehart and Winston, 1982.

[16] P. Kogge and H. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[17] R. E. Ladner and M. J. Fisher, "Parallel Prefix Computation," *Journal of Ass. Comput. Mach*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[18] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, Mar. 1982.