Rounding in Redundant Digit Floating Point Systems

Hossam A. H. Fahmy and Michael J. Flynn

Computer Systems Lab, Stanford University, California, USA.

ABSTRACT

Redundant representations are used to increase the performance of arithmetic units. If redundancy is eliminated, the bits needed to represent a number may increase or decrease depending on the type of redundancy used. In such a redundant representation, finding the exact location and correct decision for rounding without eliminating the redundancy or loosing its performance gains is difficult. This paper discusses the different issues related to rounding in redundant systems. It also presents a solution that was used to maintain the gains of redundancy in a floating point unit while correctly implementing the IEEE rounding modes.

Keywords: Floating point units, rounding, signed digits, redundancy

1. INTRODUCTION

In the current state-of-the-art high performance floating point adders, two-path algorithms[1] are used with integrated rounding.[2] If the two operands have a small difference between their exponents the close path is used. If the difference is large the far path is used and the rounding decision is calculated in parallel with the significand addition. Some recent attempts for improving the adder and simplifying its design include eliminating the need for rounding in the close path.[3] To explain this idea, let us assume that the close path is only chosen in the case of effective subtraction with exponent difference of zero or one while all the additions regardless of the exponent difference going to the far path. In this case, being in the close path, one of the operands is shifted at most by one bit location. Hence, the result might extend at most by vone bit on the least significant side. Having only subtractions means that there can never be an addition overflow and there is never a need to right shift the result for normalization. The MSB of the result is thus at most aligned with the MSB of the inputs. There is still a need for a left shifter to normalize the number in case of a cancellation of the most significant bits. If such a cancellation occurs and a left shift is needed, even by one bit location, then the result is guaranteed to be accurately represented within the precision of the format used and no rounding is necessary. Hence, if the conditions of using the close path become: 1) exponent difference is equal to zero or one, 2) only effective subtraction and 3) cancellation does occur then this set of conditions guarantees that there is no need for rounding logic in the cancellation path and the adder used there could be simpler to design and possibly slightly faster. A recent study[4] looked at various floating point adder designs from the literature and categorized the different optimization techniques used in them.

A simple parametric time delay model[5] allows us to have a sense of the complexity of some parts of the floating point addition algorithms. With n being the number of bits of the operand, shifting and adding are operations whose time delay is an $\mathcal{O}(\log n)$. In conventional designs, rounding adds a small value to the result and could cause a carry propagation making it an $\mathcal{O}(\log n)$ operation. The use of a compound adder[2] overlaps the time delay of the rounding with the addition and effectively hides it. However, in a system where a redundant representation is employed, the significand addition takes a constant time independent of n and prevents us from completely overlapping the rounding time delay. If the rounding is not placed at another location in the adder path it will form the new critical path of the design negating the speedup achieved by redundancy. In other terms, if the inputs to the adder and its output are kept in a redundant form to enhance the speed then the rounding time delay must be masked by another piece of logic that is already on the critical path.

Similar ideas apply in the case of multipliers. In conventional designs, the rounding is integrated and masked by the reduction tree and final carry propagate adder.[6] A recent study for using a redundant representation in a multiplier without relocating the rounding logic results in a design with a comparable delay to the non-redundant case.[7]

Another issue regarding rounding that arises due to the redundancy is the determination of the rounding location. Exact rounding according to the IEEE standard is defined based on the non-redundant representation

Table 1. Rounding value for the four IEEE modes and different fractional ranges

range	*RNE*	*RZ*	*RP*		*RM*	
			$+ve$	$-ve$	$+ve$	$-ve$
$-1 < f < -0.5$	-1	-1	0	-1	-1	0
-0.5	$-L$	-1	0	-1	-1	0
$-0.5 < f < 0$	0	-1	0	-1	-1	0
0	0	0	0	0	0	0
$0 < f < 0.5$	0	0	1	0	0	1
0.5	L	0	1	0	0	1
$0.5 < f < 1$	1	0	1	0	0	1

and rounding must occur at a specific bit location in each of the precisions defined in the standard. We can think of a redundant representation as composed of a main part and some extra bits to achieve the redundancy. Those extra bits can be added or subtracted from the main part to give the non-redundant representation. If redundancy is eliminated, the bits needed to represent a number may increase or decrease depending on the type of redundancy used. Hence in a redundant representation, finding the exact location and correct decision for rounding without eliminating the redundancy or loosing its performance gains is a challenge.

In a floating point system previously proposed by the current authors,[8] the numbers are saved before rounding them to the register file. When such a number is used by another functional unit, it is rounded first thus overlapping the rounding with the exponent difference in the adder and with the partial product generation and reduction in the multiplier. The position of the least significant bit of the number where the rounding takes place (the rounding location) is determined by the leading non-zero bit of the Most Significant Digit (MSD) and a signed sticky digit of the part below that leading bit. In the following section the calculation of the rounding decision is presented. The hardware used in the adder to determine the rounding location and deciding on the rounding value is described next. Then, the rounding for the multiplier where three special correction bit vectors are added to the partial products is also explained.

2. ROUNDING DECISION

The system previously proposed is based on signed digits with a hexadecimal radix. Each digit is composed of five bits and uses 2's complement encoding to represent a number between -15 and $+15$. The most significant bit of each digit is the extra bit providing the redudancy and, due to the encoding, is negatively valued. Since the significand of the proposed format is always positive, the MSD has four bits only, let us denote these bits by a (most significant), b, c and d (least significant). On the other hand, the Least Significant Digit (LSD) has five bits that we can denote by e (the extra bit with negative value), f (the most significant positive bit), g, h, i (the least significant). The digit containing the guard, round and sticky digits is encoded so that $Guard = -2g_1 + g_0$ and $S = -2s_1 + s_0$. Hence it has g_1 (a bit from the guard digit with a negative value), g_0 (a bit from the guard digit with a positive value), r, s_1 (a bit from the sticky digit with a negative value) and s_0 (a positive value bit). The relative weight of each bit is as follows:

a	b	c	d	$\cdots$	$\cdots$	$\cdots$	f	g	h	i	g_0	r	s_0
						e				g_1		s_1	

A fractional value f_i at bit location i of a signed digit binary number $\cdots x_{i+1}x_ix_{i-1}\cdots x_0$ can be defined as $f_i = (\Sigma_{j=0}^{j=i-1} 2^j \times x_j)/2^i$. The decision of the digit added for rounding is then determined by the fractional value at the rounding position. However, the value to add in order to achieve the correct rounding does not depend only on the fractional range but also on the IEEE rounding mode. In RP and RM modes, the sign of the floating point number affects the decision as well. The decision is according to Table 1 where L is the bit at the rounding location. It is important to note that in this format the rounding to zero mode is not a simple truncation. If the fractional value is negative a -1 must be added to perform the correct rounding to zero.

In this design the fractional range is estimated and the rounding value decided speculatively for each bit location in the Least Significant Digit (LSD). The resulting potential new LSDs after adding each rounding value are also calculated. Then based on the circuits indicating the leading bit of the MSD and the fine adjustment, the final rounded LSD is chosen.

3. ROUNDING IN THE ADDER

Fig. 1 shows the rounding logic used in the adder given our definition of the bits of the MSD and LSD. At the top left, simple logic gates use the bits of the MSD to determine the signals ld, lc, lb and la indicating the approximate leading bit. Once the signed sticky (ss) signal is known, the exact location is determined. The signed sticky is generated by another block that is not shown in this figure and it indicates whether the bits below the leading one of the MSD constitute a positive or negative number. Hence there are five possibilities for the exact location of the leading bit: either one of the four bits of the MSD or one bit lower than d. This last cases occurs if $a = b = c = 0$, $d = 1$ and $ss = 1$. Let us call this case dm as a short for d minus one. If the leading bit is a then the location of rounding is at bit f of the LSD and so on for the other locations.

The *primitive rounding logic* block to the left of the figure is responsible for generating the speculative bits indicating the rounding value. The rounding value can be positive, negative or zero and hence is represented by two bits r_p and r_n and its value is given by $r_p - r_n$. Since the digits in the proposed format are always between -15 and $+15$, then the digit that was shifted out and from which the guard and round bits were drawn also falls in that range. Hence, if $g_1 = 1$ we are guaranteed that at least one other bit in that digit that was shifted out is one. So, either $g_0 = 1$, $r = 1$ or the sticky digit is positive (i.e. $s_1 = 0$ and $s_0 = 1$). This property insures that the fractional range at bit i the least significant bit of the LSD is bounded between -1 and 1. At g_0, this property insures that the fractional range is bounded between 0 and 1. For the other bit locations, the fractional range is bounded between -0.5 and 1. Given the location of the leading bit, the values of r_p and r_n are derived based on Table 1 resulting in the equations shown in Table 2. Those equations insure that either $r_p = rn = 0$ or only one of them is set to one but never both at the same time.

The *Digit rnd* part in the middle of the figure generates three possible outcomes for the cases of positive, zero or negative rounding value at each possible rounding location. These outcomes are the values of the bits in the locations of e to g_0 which we can call e' to g'_0 to signify the rounded values. In addition to that there might be a carry out (co) to the next higher digit. These values are given by the logic equations shown in Table 3.

A few important points played a role in deriving those equations:

1. Since r_p and r_n are mutually exclusive, it is possible to use only one signal, co, to denote a positively valued carry in the case of $r_p = 1$ and a negatively valued carry otherwise.

2. For the rest of the units, it is important to guarantee that the generated digits are between -15 and $+15$. Hence, even in the case of truncation of the LSD there might be a carry out of it. This occurs if $e = 1$ and the other bits that were to remain are all zeros. In this case, a $co = 1$ signal is generated and e' is reset to zero.

3. The case of rounding at g_0 is slightly different.

   $r_p = 1$**:** Then $g'_0 = \overline{g}_0$, $g'_1 = g_1\overline{g}_0$ and a carry of $\overline{g}_1 g_0$ must be added to the LSD. A multiplexer with $\overline{g}_1 g_0$ as a select line is used to choose between the outcome of rounding at i when the rounding value is 1 or 0.

   $r_p = r_n = 0$**:** This is a case of truncation and, as above, we must guarantee that the new digit is within the permissible range. Hence, $g'_0 = g_0$ but $g'_1 = g_1 g_0$ and a negatively valued carry of $g_1\overline{g}_0$ is added to the LSD. Again, a multiplexer with select line equal to $g_1\overline{g}_0$ is used to choose between the rounded LSD at location i when the rounding value is -1 or 0.

   $r_n = 1$**:** Then $g'_0 = \overline{g}_0$, $g'_1 = \overline{g}_0$ and a negative carry of g_1 is added to the LSD. The select line of the multiplexer is then g_1 and the inputs are the rounded LSD at i when the rounding value is -1 or 0.

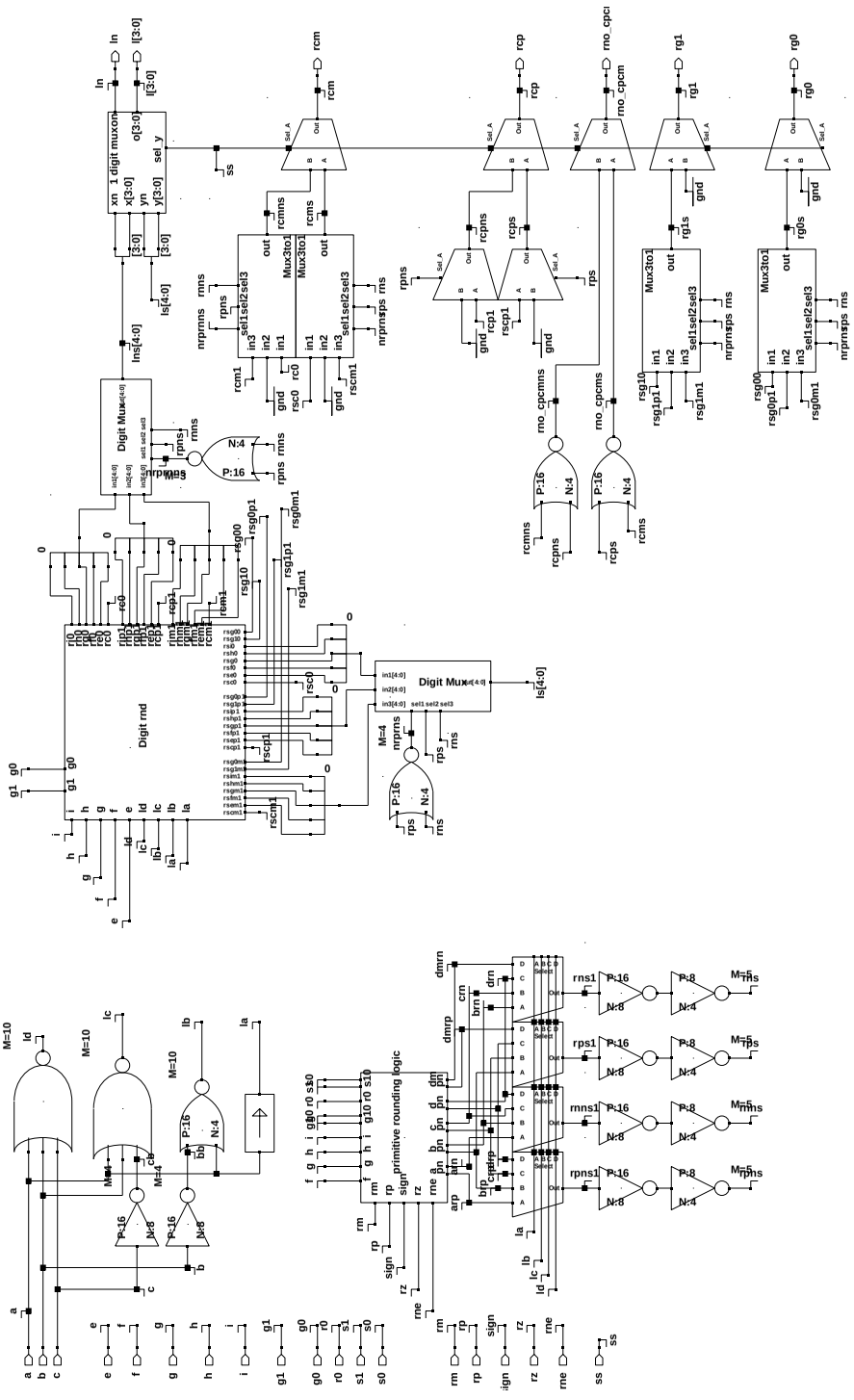

Figure 1. Rounding of the least significant digit.

Table 2. Logic equations for r_p and r_n

Leading bit	Round at	Rounding value
dm	g_0	$r_p = RNE\, r\overline{s}_1(s_0+g_0)$ $+(RP\,\overline{sign}+RM\,sign)(r+\overline{s}_1 s_0)$ $r_n = (RP\,sign+RM\,\overline{sign}+RZ)\overline{r}s_1$
d	i	$r_p = RNE\,\overline{g}_1 g_0\overline{\overline{r}s_1}(r+s_0+i)$ $+(RP\,\overline{sign}+RM\,sign)\overline{g}_1(g_0+r+\overline{s}_1 s_0)$ $r_n = RNE\, g_1(\overline{g}_0+\overline{r}s_1+\overline{r}\,\overline{s}_0 i)$ $+(RP\,sign+RM\,\overline{sign}+RZ)(g_1+\overline{g}_0\overline{r}s_1)$
c	h	$r_p = RNE\, i\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)}(g_0+r+s_0+h)$ $+(RP\,\overline{sign}+RM\,sign)(i+\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)}(g_0+r+s_0))$ $r_n = (RP\,sign+RM\,\overline{sign}+RZ)\overline{i}(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)$
b	g	$r_p = RNE\, h(i+\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)})(i+g_0+r+s_0+g)$ $+(RP\,\overline{sign}+RM\,sign)(h+i+\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)}(g_0+r+s_0))$ $r_n = (RP\,sign+RM\,\overline{sign}+RZ)\overline{hi}(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)$
a	f	$r_p = RNE\, g(h+i+\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)})(h+i+g_0+r+s_0+f)$ $+(RP\,\overline{sign}+RM\,sign)(g+h+i+\overline{(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)}(g_0+r+s_0))$ $r_n = (RP\,sign+RM\,\overline{sign}+RZ)\overline{ghi}(g_1+\overline{g}_1\overline{g}_0\overline{r}s_1)$

Table 3. Logic equations for the rounded LSD

Round at	$r_p - r_n$	co	e'	f'	g'	h'	i'	g'_1	g'_0
	$+1$	$\overline{e}fghi$	$e\overline{(fghi)}$	$f\oplus ghi$	$g\oplus hi$	$h\oplus i$	$\overline{i}$	0	0
i	0	0	e	f	g	h	i	0	0
	-1	$e\overline{f}\,\overline{g}\overline{h}$	$\overline{e}\overline{f}\,\overline{g}\overline{h}i+e\overline{(\overline{f}\,\overline{g}\overline{h}i)}$	$\overline{f}\oplus(g+h+i)$	$\overline{g}\oplus(h+i)$	$\overline{h}\oplus i$	$\overline{i}$	0	0
	$+1$	$\overline{e}fgh$	$e\overline{(fgh)}$	$f\oplus gh$	$g\oplus h$	$\overline{h}$	0	0	0
h	0	$e\overline{f}\,\overline{g}\overline{h}$	$e(f+g+h)$	f	g	h	0	0	0
	-1	$e\overline{f}\,\overline{g}$	$\overline{e}\overline{f}\,\overline{g}\overline{h}+e\overline{(\overline{f}\,\overline{g}h)}$	$\overline{f}\oplus(g+h)$	$\overline{g}\oplus h$	$\overline{h}$	0	0	0
	$+1$	$\overline{e}fg$	$e\overline{(fg)}$	$f\oplus g$	$\overline{g}$	0	0	0	0
g	0	$e\overline{f}\,\overline{g}$	$e(f+g)$	f	g	0	0	0	0
	-1	$e\overline{f}$	$\overline{e}\overline{f}\,\overline{g}+e\overline{(\overline{f}g)}$	$\overline{f}\oplus g$	$\overline{g}$	0	0	0	0
	$+1$	$\overline{e}f$	$e\overline{f}$	$\overline{f}$	0	0	0	0	0
f	0	$e\overline{f}$	ef	f	0	0	0	0	0
	-1	e	$\overline{f}$	$\overline{f}$	0	0	0	0	0

The logic deciding the ss signal detects the sign of each digit and whether it is zero or not. Then, a priority encoder ($\mathcal{O}(\log n)$ operation) is used to determine the value of ss. Hence ss is a late signal and the *Digit rnd* block speculatively uses the approximate location of the leading bit to multiplex all the possible outcomes and to generate two sets of digits. A set for the case of a zero signed sticky and another set for the case of a signed sticky equal to one. As shown in Fig. 1, the set assuming that the signed sticky is one enters into the multiplexer in the center of the figure that selects the correct digit depending on the values of r_p and r_n given that $ss = 1$. The other set is channeled to the multiplexer to the right of the *Digit rnd* block which also selects the correct digit depending on the values of r_p and r_n given that $ss = 0$. Finally, the multiplexer to the top right of the figure gives the final output digit depending on the value of ss. The signed sticky ss is also used as a select line for the other multiplexers to the right of the figure giving the values of the rounded g_1 and g_0 as well as the possible three values for the carry out to the next higher digit: $rcp = 1$ if it is positive, $rcm = 1$ if it is negative otherwise if it is zero the signal *rno-cpcm* is set to one.

4. ROUNDING IN THE MULTIPLIER

A subtle difference between the multiplier in the proposed system and the conventional multipliers is the fact that each operand has a number of bits that are considered to have a negative value. These are the extra bits in each digit of the number. The significands of the two operands X and Y are thus assumed to have three components: P the positively valued bits, E the negatively valued extra bits and r the value of the rounding digit which is either positive or negative (i.e. $r \in \{-1, 0, 1\}$.) The end result is $X \times Y$ but, $X = P_x - E_x + r_x$ and $Y = P_y - E_y + r_y$ and hence a number of possibilities for the design were studied.[9]

Similar to the case of the floating point adder, rounding in the presented floating point multiplier proved to be a challenge. The operands can be rounded first and then the rounded significands used to perform the multiplication. This puts the rounding logic on the critical path of the multiplier and delays the start of the partial products reduction. Rounding first without any other parallel work done on the significands is unacceptable if a high speed multiplier is required. It adds a time delay comparable to the time needed for the partial product reduction tree using $[4 : 2]$ compressors.[9]

Another possibility is to chop both operands X and Y at the rounding location. Then, simultaneously, some additional partial products with the values $r_y(P_x - E_x)$, $r_x(P_y - E_y)$ and $r_x r_y$ are generated. In this second case, the reduction of the partial products starts directly but it takes more hardware because of the added partial products.

As mentioned in the discussion on rounding in section 2, this rounding location is determined by the leading bit of the MSD and the signed sticky digit. The operand chopping takes away all the bits below the approximate rounding location determined *only* by the leading bit of the MSD. The multiplication is carried as $(X_{chopped} + (b_x + r_x))(Y_{ch} + (b_y + r_y))$ where b_x and b_y are the bits directly after the rounding location in case the signed sticky indicates a negative fractional value while r_x and r_y are the rounding digits. Hence the multiplication can be divided into three parts:

$$
\begin{array}{lll}
X \times Y = & X_{chopped} Y_{chopped} & \text{By Booth recoding, generating partial products and summing} \\
& +(b_x + r_x) Y_{chopped} + (b_y + r_y) X_{chopped} & \text{2 special partial products added in the tree} \\
& +(b_x + r_x)(b_y + r_y) & \text{special correction added to the tree}
\end{array}
$$

Fig. 2 shows how $(b_y + r_y)X_{chopped}$ is formed.* The rounding portion $(b_y + r_y)$ is done by feeding the bits of Y to the correction block in the bottom right of the figure to generate five shifting signals sh_4 to sh_0 and the correction rounding value. The output of the correction rounding value or is coded by two bits or_p and or_n and its value is $or_p - or_n$. In the figure, the complement of those two signals are shown as *orpb* and *ornb*. This correction block at the bottom right basically recodes the multiplication by $(b_y + r_y)$ as a multiplication by 2^{shift} followed by a selection of either the positive or negative of the shifted operand based on the value of the or_p and or_n signals.

*Obviously, the same circuit with the inputs reversed can do $(b_x + r_x)Y_{chopped}$.

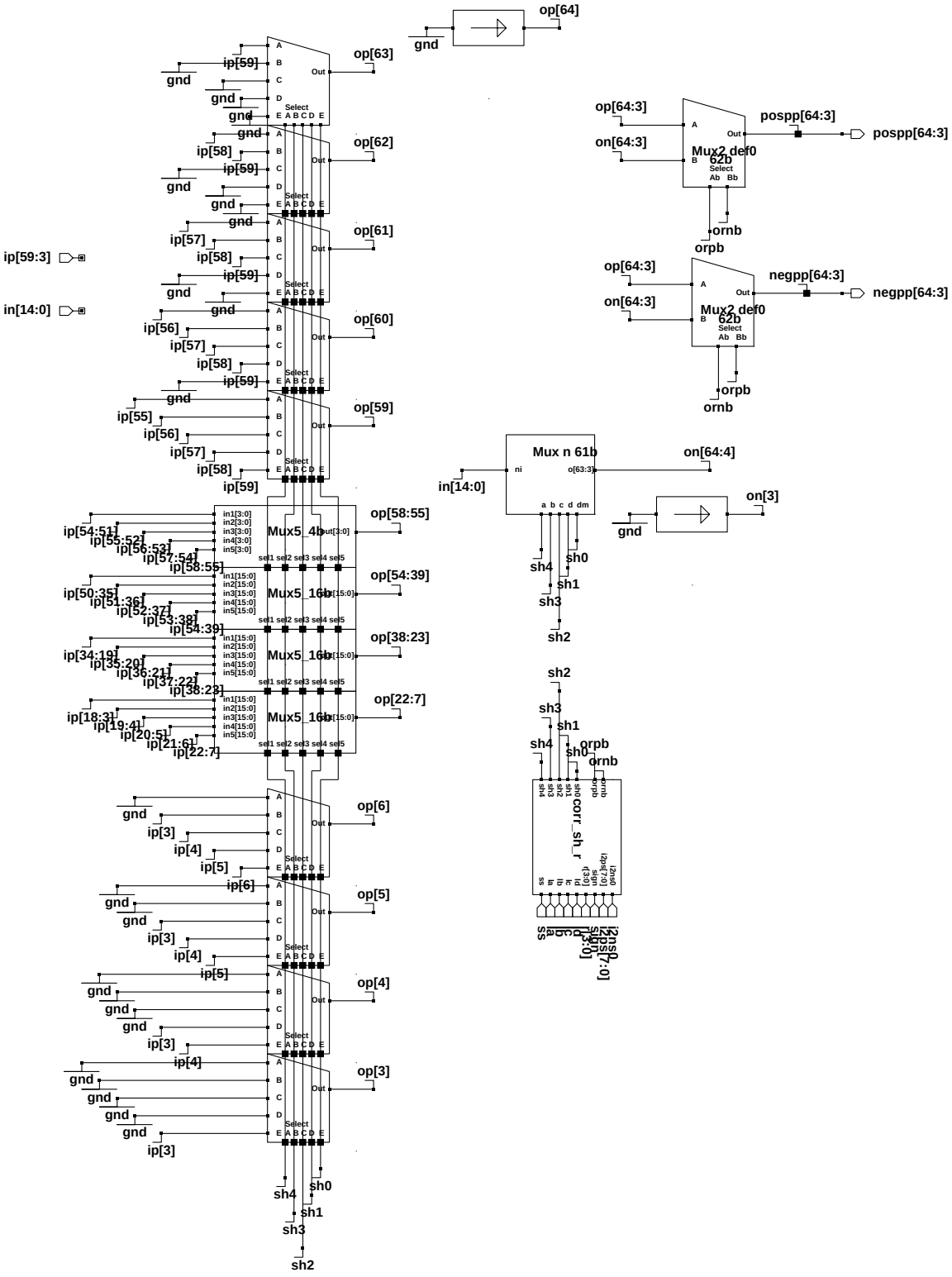

Figure 2. Multiplying one operand by the rounding part of the other.

The correction block must decide on the exact amount of shifting to be done. For this to be achieved, the leading one of the MSD of Y is detected. Similar to the rounding in the adder, let us assume that the bits of the MSD are $a\ b\ c\ d$ and let us assume that the signals indicating which bit is the leading one are labeled la, lb, lc and ld. Those signals are among the inputs to the correction block. The other inputs are the value of the signed sticky ss, the rounding mode selectors, the sign of the operand and the bits of the LSD and guard, round and sticky digits of Y. As in the discussion of the adder, let us define the bits of Y as:

$$\begin{array}{cccc|ccc|cccc|ccc} a & b & c & d & \cdots & \cdots & \cdots & f & g & h & i & g_0 & r & s_0 \\ & & & & & & e & & & & g_1 & & s_1 & \end{array}$$

If the signed sticky is not set ($ss = 0$), the value of b_y the bit directly after the rounding location is irrelevant, the output corrected value *or* is given by r_y alone and the shifting amount is determined only by the leading one of the MSD. On the other hand, if $ss = 1$ then b_y must be considered. The evaluation of $(b_y + r_y)$ is tricky since r_y can be negative, zero or positive while b_y is positive or zero except at the g_0 location where it might be negative, zero or positive. Hence, when $ss = 1$ and the rounding is

not at the g_0 location: If both b_y and r_y are positive then $(b_y + r_y) = 2$ which means that, effectively, it is carried to the next higher bit location and the shift amount is increased by one canceling the effect of ss. Otherwise, the effect of ss is taken into account and the shift amount derived based on it.

at the g_0 location: The value of g_1g_0 is given by $-2g_1 + g_0$ and can be in $\{-2, -1, 0, 1\}$

- $g_1g_0 = 01 = (1)$ then, as above, if $r_y = 1$ it is a carry to the next higher bit. Otherwise, the effect of ss is taken into account.
- $g_1g_0 = 00 = (0)$ then $b_y = 0$ and the effect of ss is taken into account.
- $g_1g_0 = 11 = (-1)$ then if $r_y = -1$ (note the sign) it is a carry to the next higher bit. Otherwise, the effect of ss is taken into account.
- $g_1g_0 = 10 = (-2)$ in this case $r_y \in \{0, 1\}^\dagger$ and if it is zero a carry to the next higher bit occurs. Otherwise, the effect of ss is taken into account.

Since the rounding location could be anywhere from the g_0 location to the f location, the shift amount produced by the correction block is assumed to be zero if the value of $(b_y + r_y)$ evaluates to be at the g_0 location. Let the rounding values at any location be represented by r_p and r_n with a subscript referring to the leading bit of the MSD corresponding to this location. These rounding values are evaluated by the *primitive rounding logic* block described in the discussion of Fig. 1. The signals for the shift are then given by the following equations:

$$\begin{aligned}
sh_4 &= \overline{ss}\, la + ss\, la(gr_{p_b}) \\
sh_3 &= \overline{ss}\, lb + ss\, la\overline{(gr_{p_b})} + ss\, lb(hr_{p_c}) \\
sh_2 &= \overline{ss}\, lc + ss\, lb\overline{(hr_{p_c})} + ss\, lc(ir_{p_d}) \\
sh_1 &= \overline{ss}\, ld + ss\, lc\overline{(ir_{p_d})} + ss\, ld(\overline{g}_1 g_0 r_{p_{dm}} + g_1 g_0 r_{n_{dm}} + g_1 \overline{g}_0 \overline{r}_{p_{dm}}) \\
sh_0 &= ss\, ld\, \overline{(\overline{g}_1 g_0 r_{p_{dm}} + g_1 g_0 r_{n_{dm}} + g_1 \overline{g}_0 \overline{r}_{p_{dm}})}
\end{aligned}$$

The output corrected rounding value *or* corresponds to the value of $b_y + r_y$ regardless of the shift. So, $or = 0$ if $b_y + r_y = 0$, $or = 1$ if $b_y + r_y = 1$ or 2 and $or = -1$ if $b_y + r_y = -1$ or -2. To calculate or_p and or_n, we can use two intermediary signals r_p and r_n indicating the value of r_y,

$$\begin{aligned}
r_p &= \overline{ss}(la\, r_{p_a} + lb\, r_{p_b} + lc\, r_{p_c} + ld\, r_{p_d}) + ss(la\, r_{p_b} + lb\, r_{p_c} + lc\, r_{p_d} + ld\, r_{p_{dm}}) \\
r_n &= \overline{ss}(la\, r_{n_a} + lb\, r_{n_b} + lc\, r_{n_c} + ld\, r_{n_d}) + ss(la\, r_{n_b} + lb\, r_{n_c} + lc\, r_{n_d} + ld\, r_{n_{dm}})
\end{aligned}$$

†Due to the condition on the number system that a digit cannot be -16, if $g_1g_0 = 10$ then in the bits representing the guard round and sticky digits either the round bit $r = 1$ or the sticky digit is positive. This leads to only a positive fractional value at g_0 which means that the rounding value at this location cannot be negative as noted in section 3.

The signals for or_p and or_n are mutually exclusive to fit their role as multiplexer select signals in Fig. 2. The signal or_p is set to 1 if $r_y = 1$ and b_y is neglected in the case of $ss = 0$ or $b_y = 0$. The other case where or_p is set is if r_y is not a negative one and $b_y = 1$. In boolean logic this translates to:

$$or_p = \overline{ss}\, r_p + ss\, r_p(la\, \overline{g} + lb\, \overline{h} + lc\, \overline{i} + ld\, \overline{g}_1 \overline{g}_0) + ss\, \overline{r}_n(la\, g + lb\, h + lc\, i + ld\, \overline{g}_1 g_0)$$

Similarly, or_n is set to 1 if $r_y = -1$ and b_y is neglected in the case of $ss = 0$ or $b_y = 0$. The second case where or_n is set is if r_y is not a positive one and $b_y = -1$ ($g_1 g_0 = 11$) or $b_y = -2$ ($g_1 g_0 = 10$). The third case is if $r_y = 1$ and $b_y = -2$. In boolean logic this translates to:

$$or_n = \overline{ss}\, r_n + ss\, r_n(la\, \overline{g} + lb\, \overline{h} + lc\, \overline{i} + ld\, \overline{g}_1 \overline{g}_0) + ss\, \overline{r}_p\, ld\, g_1 + ss\, r_p\, ld\, g_1 \overline{g}_0$$

With the values of sh_4 to sh_0, or_p and or_n the multiplexers of Fig. 2 shift and choose the correct positive and negative outputs as seen in the top left part of the figure.

The calculation of the special correction $(b_x + r_x)(b_y + r_y)$ is done in a similar fashion. The same correction block described above is used for both operands X and Y. A few logic gates determine that the result is positive if the output corrected rounding values of both X and Y are of the same sign. The result is negative if those values are of opposite sign. The special correction is represented redundantly by a positive and a negative bit vector where the correct bit location is set depending on the output corrected rounding values and the sign of the result.

5. CONCLUSIONS

Due to the redundancy, the rounding location may shift by one bit depending on the signed sticky of the part below the leading bit of the number being rounded. Since the generation of this signed sticky depends on all the bits of the number, it is slow. In the adder, a special attention is given to optimize the speed of the hardware given the slow signed sticky by speculatively calculating different possibilities for the rounded LSD. Once the exact rounding location is determined the corresponding rounded LSD is chosen. For the multiplier, the generation of the rounding correction is explained and an efficient implementation is presented.

For both the adder and the multiplier the rounding hardware has been relocated to have its time delay masked by another essential operation. This masking takes the rounding delay off the critical path of the units.

REFERENCES

1. P. M. Farmwald, *On the Design of High Performance Digital Arithmetic Units*. PhD thesis, Stanford University, Aug. 1981.
2. N. T. Quach, *Reducing the latency of floating-point arithmetic operations*. PhD thesis, Stanford University, Dec. 1993.
3. A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency IEEE floating-point standard adder architectures," in *Proceedings of the 14th IEEE Sympsoium on Computer Arithmetic, Adelaide, Australia*, pp. 35–42, Apr. 1999.
4. P.-M. Seidel and G. Even, "On the design of fast IEEE floating-point adders," in *Proceedings of the 15th IEEE Sympsoium on Computer Arithmetic, Vail, Colorado, USA*, pp. 184–194, July 2001.
5. H. A. H. Fahmy, A. A. Liddicoat, and M. J. Flynn, "Parametric time delay modeling for floating point units," in *The International Symposium on Optical Science and Technology, SPIE's 47th annual meeting (Arithmetic session), Seattle, Washington, USA*, July 2002.
6. M. R. Santoro, G. Bewick, and M. A. Horowitz, "Rounding algorithms for IEEE multipliers," in *Proceedings of the 9th IEEE Sympsoium on Computer Arithmetic, Santa Monica, California, USA*, pp. 176–183, Sept. 1989.
7. M. I. Furgeson and M. D. Ercegovac, "The IEEE rounding for multiplier with redundant operands," in *Thirty-Fourth Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA*, **2**, pp. 1334–1338, Oct. 2000.
8. H. A. H. Fahmy and M. J. Flynn, "The case for a redundant format in floating point arithmetic," in *Proceedings of the 16th IEEE Sympsoium on Computer Arithmetic, Santiago de Compostela, Spain*, June 2003.
9. H. A. H. Fahmy, *A Redundant Digit Floating Point System*. PhD thesis, Stanford University, June 2003.