The Case For a Redundant Format in Floating Point Arithmetic

Hossam A. H. Fahmy
hfahmy@arith.stanford.edu

Michael J. Flynn
flynn@arith.stanford.edu

Computer Systems Laboratory, Stanford University, USA

Abstract

This work uses a partially redundant number system as an internal format for floating point arithmetic operations. The redundant number system enables carry free arithmetic operations to improve performance. Conversion from the proposed internal format back to the standard IEEE format is done only when an operand is written to memory. A detailed discussion of an adder using the proposed format is presented and the specific challenges of the design are explained. A brief description of a multiplier and divider using the proposed format is also presented. The proposed internal format and arithmetic units comply with all the rounding modes of the IEEE 754 floating point standard. Transistor simulation of the adder and multiplier confirm the performance advantage predicted by the analytical model.

1. Introduction

Addition is the most frequent arithmetic operation in numerically intensive applications. Multiplication follows closely and then division and other elementary functions. This work presents several techniques to improve the effectiveness of floating point arithmetic units in general but with a focus on addition.

A partially redundant number system was previously proposed [5] for use as an internal format within the floating point unit and the associated registers. The format is based on the single and double precision formats of the ANSI/IEEE standard [1]. However, in the proposed format each group of 4 bits of the significand are represented redundantly as a 5 bit signed digit number in $\{-15, \cdots, +15\}$ using two's complement form. The fifth bit (extra bit) represents a negative value with the same weight as the least significant bit of the next higher group. This is shown for the string of bits $a4, a3, a2, a1, a0$ in Fig. 1. This extra bit, $a4$, is saved in the register to the right of the least significant bit in the next higher group and to the left of $a3$. As with IEEE formats, the significand is always positive so there is no need for the extra bit in the most significant digit. The number is also always normalized in the proposed format. Denormalized IEEE numbers are normalized in the conversion process upon loading into the register file. Each group of 5 bits represents one base 16 digit and therefore, the exponent is applied to base 16 rather than base 2 as is used in the normal IEEE format. The proposed format is in the form, $(-1)^{sign}\, first\, digit.remaining\, digits \times 16^{exp-bias}$. The guard, round and sticky (GRS) bits are saved in the register file with the unrounded result. The result is then correctly rounded according to the IEEE standard's rules in the following operation when it is used or saved to the memory. This deferred rounding technique moves the rounding computation off the critical path and allows it to be overlapped with the exponent difference calculation in the adder.

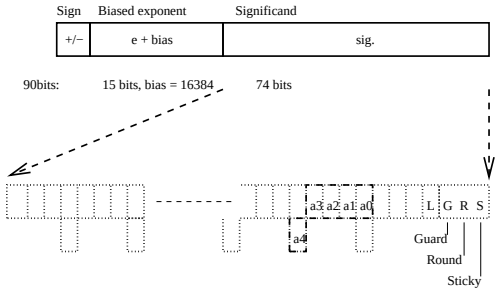

Figure 1. The proposed signed digit format for floating point numbers.

In general, SD numbers allow carry free addition by using redundant number representations. Eliminating the carry propagation significantly reduces the latency of arithmetic operations. The conversion from binary to SD form is trivial since the binary format is usually a valid SD representation. However, converting a SD number back into a non-redundant form involves a carry propagation. SD numbers are not commonly used in arithmetic circuits since the SD to binary conversion requires a carry propagation. The proposed system efficiently hides this time delay by overlapping the SD number to binary conversion with the memory store operations, thus removing it from the critical path.

More details of the conversion to and from the proposed internal format to the IEEE format were presented in our earlier work [5].

The format presented above with its specific use of a base 16 signed digits is obviously a special case of more general signed digits [13] where another base or even a mix of different bases [15] can be used. The case of using another base is discussed briefly in section 4 below. The methodology of thinking about the algorithms and trade-offs discussed in this work apply to the general case as well. The specific format presented above is what we implemented in transistors to prove our claims regarding the speed improvement. We deemed this specific format as a practical compromise to give enough speed improvement with a reasonable increase in the area of the register file. The issue of the optimal redundant format to use will obviously depend on the requirements on speed, area and power consumption. That issue is beyond the scope of the current work.

In the following sections, the presented format is used to build efficient arithmetic circuits. Section 2 explains the details of the floating point addition unit. Two design challenges due to the redundant format, namely the leading digit detection and the rounding, are discussed in section 3. Section 4 describes the analytical time delay modeling of the addition unit and discusses the rationale for postponing the rounding and for using a hexadecimal based exponent. The multiplication, division and other elementary functions are computed using the units presented in section 5. Then section 6 provides the simulation results for both the adder and the multiplier. Finally, in section 7 conclusions of this work are presented.

2. Floating point addition

In the current state-of-the-art high performance floating point adders, two-path algorithms are used with integrated rounding similar to the designs proposed by Farmwald [7] and Quach [17]. These adders perform both addition and subtraction. An effective subtraction occurs when both operands are of the same sign and the required operation is a subtraction or when the operands differ in their signs and the operation is an addition. In the case of effective subtraction and an exponent difference of zero or one, a few of the leading digits of the result might become zero. For this case of leading zero digits, there is a need for a left shift of the result for normalization. The two-path algorithm separates this specific case into a path with a specialized left shifter while the general case of operands passes through the regular path. The special path is called the cancellation path (where the leading digits are possibly canceled) or the close path (where the exponents are close to each other) while the regular path is called the far path. The adder design presented here is following a similar approach and is using a

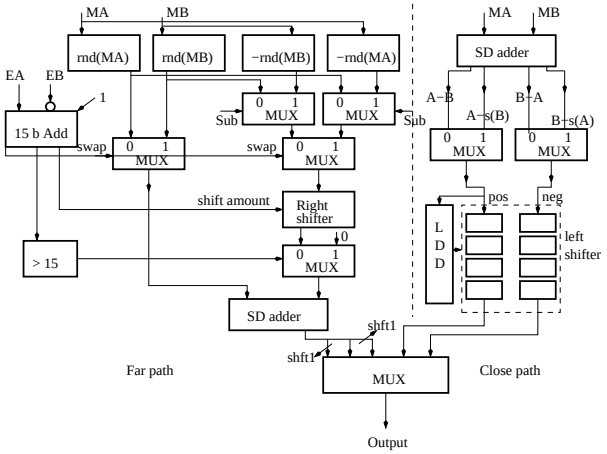

Figure 2. Block diagram of the two-path adder.

two-path algorithm as shown in the block diagram of Fig. 2. In the presented design, the cancellation path is used only in the case of an effective subtraction with an exponent difference of zero or an effective subtraction with an exponent difference of one and a cancellation of some of the leading digits occurring in the result. In all other cases, the far path is used.

The far path of the proposed adder is similar to the far path of other algorithms presented in the literature. The unique aspects of the proposed adder are: first, the use of a hexadecimal base for the exponents; second, the location of the rounding logic in parallel with the exponent difference and third the use of signed digit numbers in the significand. The hexadecimal base of the exponents makes the right shifting for alignment of the two operands a shift to a 4-bit boundary only. So, instead of using an n-way shifter in the conventional adders an $\lceil n/4 \rceil$-way shifter is used here. Such a reduction in the complexity of the shifter reduces its time delay as discussed below. The parallel execution of the rounding logic with the exponent difference logic takes the rounding away from the critical path of the adder. It is possible to simultaneously round and negate the number and this is what is done in the presented design to prepare the operand for the SD (signed digit) adder.

A signal indicating an effective subtraction selects the operand or its negative and a signal indicating which of the operands has a larger exponent allows for swapping them. Then, the operand with the smaller exponent is shifted to the right for alignment and the bits that are flowing out of the shifter are used to calculate the guard, round and sticky bits. Only the least significant bits of the exponent difference are used to indicate the shift amount. If the exponent difference is large enough to completely shift out the smaller operand a zero is forced as the second operand into the adder.

The result of the SD adder may need a normalization

shift by one digit to the right for the case of effective addition and an overflow. The result may otherwise need a normalization shift by one digit to the left for the case of an effective subtraction and cancellation of the Most Significant Digit (MSD). This cancellation of only the MSD can occur even when the exponent difference is larger than one. The SD adder block produces the result and three signals indicating the need for no shift, a shift to the left or a shift to the right. The guard, round and sticky bits are calculated speculatively dependent on the shifting possibilities. The multiplexer unit responsible for choosing between the far and cancellation paths receives those different signals and speculative results and chooses the final result among them in case it chooses the far path.

In the cancellation path the exact exponent difference is not calculated but the least significant bit of each of the two exponents is examined. If the two exponent bits are found to be identical the difference of the exponents is speculated to be zero and the direct subtraction of the operands is needed. If, on the other hand, the two exponent bits are not identical the difference is assumed to be one and the subtracter should produce a result equal to one operand minus the other operand shifted by one bit to the right. The direct subtraction in the case of zero exponent difference may lead to a negative result if the significand of the second operand is larger than the significand of the first operand. To remedy for this negative result in conventional adders, the sign of the floating point result is flipped and the bits representing the result are negated. In the presented design, the subtracter produces the result and its negative and then chooses one of them at the end depending on the sign of the result.

So, assuming the two operands to be labeled A and B, all the possible combinations are produced in the presented design: $A - B$, $A - shift(B)$, $B - A$ and $B - shift(A)$. Then, depending on the speculation of the exponent difference, either the direct subtraction or the one involving a shift is chosen. Since a complete calculation of the exponent difference does not occur in the cancellation path, the rounding is done in conjunction with the significand subtraction. A round digit is computed for each operand and is used within a signed digit subtracter to perform the subtraction. A Leading Digit Detector (LDD) is used to calculate the shift amount needed to normalize the result. This shift amount is applied to two shifters one shifting the result and the other shifting the negative of the result. The sign of the leading digit is detected and the correct sign for the floating point result is decided. The result and its negation as well as a signal indicating the sign of the leading digit are forwarded to the multiplexer unit selecting between the cancellation path and the far path. This unit then makes the decision on the path to choose and appropriate result from each path.

3. Challenges: The leading digit detection and rounding

In an effective subtraction in the close path one or more of the leading digits in the result may become zero. Then, in order to normalize the result, the leading non-zero digit must be detected and the result must be normalized by left shifting the significand by the number of leading zeros. All floating point adders include circuits to either detect or predict the position of the leading non-zero digit after the subtraction is performed. The prediction circuits like the work of Bruguera and Lang [2] or the work of Quach and Flynn [16] operate on the adder's operands in parallel with the significand addition. It is to note that in both schemes the original operands are not redundant while the prediction circuits are working on a redundant representation because the prediction is done before the result of the adder is available. In a redundant number there are several patterns that evaluate to a string of leading zeros. A prediction circuit must then involve a technique for matching all such patterns and taking appropriate actions. If, on the other hand, a detection scheme was used on the non-redundant result of an adder, there will be no need for complicated pattern matching. Since in the proposed design for the floating point adder signed numbers are used for the inputs and output, even a detection scheme must perform some pattern matching.

A few possible patterns become the hard cases in detecting the first non-zero digit. In fact, the leading zeros may be expressed directly as zeros or indirectly as leading insignificant digits: a leading 1 followed by -15s or a leading -1 followed by 15s. The leading 1 (-1) can be converted to a zero and borrowed into the neighbor -15 (15) digit position as a 16 (-16). Since $16 - 15 = 1$ $(-16 + 15 = -1)$, the zero propagation may continue into lower significance digits. The following example illustrates how leading non-zero digits may be leading insignificant digits. Assuming $| l | < 15$,

$$\begin{array}{rrrcrcccc} 1 & -15 & -15 & \cdots & -15 & l & m & \cdots & = \\ 0 & 0 & 0 & \cdots & 1 & l & m & \cdots & \\ -1 & 15 & 15 & \cdots & 15 & l & m & \cdots & = \\ 0 & 0 & 0 & \cdots & -1 & l & m & \cdots & \end{array}$$

Another pattern is $100\cdots00 - ve = 011\cdots10 + ve$. This pattern and its dual $(-1)00\cdots00 + ve$ are what causes a fine adjustment in the case of the previous work [16, 2]. The fine adjustment is basically to indicate that the location of the leading digit should be shifted by one position. We can mentally think of detecting the leading zeros and the leading insignificant digits as the first step followed by a fine adjustment step. In the fine adjustment step if the leading digit is 1 and is not followed by another positive digit but by 0 then we must detect the sign of the remainder of the number. If that sign is negative, a fine adjustment is needed. The dual case holds for a leading -1.

The need to detect special cases for $+1$ followed by -15 or -1 followed by $+15$ can be eliminated by the use of some recoding techniques similar to what was presented in the work on recoders for partial compression [3, 4]. In the section discussing rounding and leading digit detection, Daumas and Matula state [4]: "Partial compression also realizes virtually all the benefits of leading digit deletion." The word virtually is important; partial compression does not provide a solution for the fine adjustment cases. In fact, from a complexity and time delay point of view, getting the exact bit location of the leading one is essentially the same as doing a carry propagation. The fine adjustment is hence equivalent to transforming the redundant representation into a non-redundant one. As described by Quach and Flynn [16], parallel addition and leading one prediction are both problems of bit pattern detection. They also identified sticky bit computation as the third problem in the category of bit pattern detection.

What is proposed in this work is to perform only the coarse adjustment of finding the leading digit by eliminating any leading zeros or insignificant digits. The main advantage of using a signed digit number system is to eliminate the carry propagation from the critical path. Thus, introducing another circuit similar in complexity (fine adjustment) instead of the carry propagation in the critical path will defeat the purpose. Hence, the fine adjustment is left to the rounding unit and a signed sticky digit (similar to what Matula and Nielsen [10, 12] proposed) is used there. The rounding occurs in parallel with the exponent difference and is not sequentially after the addition, so it is out of the critical path.

Based on the work of Daumas and Matula [3, 4], two recodings are defined to delete the leading insignificant digits. The N-recoding is where a negative one could be added to the digits and the P-recoding is where a positive one could be added. More specifically, for two consecutive digits of the result,

$$\begin{array}{cccccccc} \cdots & s_{i_3} & s_{i_2} & s_{i_1} & s_{i_0} & & s_{i-1_3} & s_{i-1_2} & \cdots \\ s_{i_4} & & & & s_{i-1_4} & & & & \end{array}$$

The N-recoding is defined as reseting s_{i-1_4} and s_{i_0} to 0 if they were both 1. This is mathematically correct since s_{i-1_4} has a negative value. This recoding can create digits that are equal to -16, however, this is not important since the recoded format is only used within the leading digit detector (LDD) circuits and even with such "out of bound" digits the position of the leading digit can be correctly estimated. The N-recoding when applied to the case of repeated -15 would be as follows assuming that the digit l is positive:

$digits$	k	-15	$\cdots$	-15	l	$\cdots$
$equiv.$	$k_3k_2k_1k_0$	0001	$\cdots$	0001	$l_3l_2l_1l_0$	$\cdots$
$bits$	1	1	$\cdots 1$	l_4	$\cdots$	$\cdots$
$result$	$k-1$	0	$\cdots$	1	l	$\cdots$

If the digit k is equal to 1, then all the insignificant 1 followed by negative digits have been eliminated by this N-recoding. If the digit l is negative then the result of the recoding will not be $1\,l\cdots$ but rather l_4 will be canceled effectively giving a result of $0\,(16+l)\cdots$. This feature is of value since it ensures that the number is truly normalized. A leading digit of 1 with a negative fractional part is not the normalized format. The condition for the N-recoding to change the bits is $s_{i_0} = s_{i-1_4} = 1$ and hence its output is given by $s^n_{i_0} = s_{i_0}\overline{s}_{i-1_4}, s^n_{i-1_4} = \overline{s}_{i_0}s_{i-1_4}$ while the remaining bits of the digit pass unchanged.

The P-recoding on the other hand is defined to eliminate the case of insignificant leading -1 followed by positive digits. Referring to the two consecutive digits above, if $s_{i_0} = 1$ and $s_{i-1_4} = 0$ then we can split s_{i-1_4} to 1 and -1, add the 1 to s_i and keep the -1 with s_{i-1} as its new s_{i-1_4}. This split is to occur only if s_{i-1} is not exactly equal to zero in order to prevent the new s_{i-1} from becoming -16. Applying P-recoding to the case of repeated digits of 15 the result is:

$digits$	0	-1	15	$\cdots$	15	l	$\cdots$
$equiv.$	0	1111	1111	$\cdots$	1111	$l_3l_2l_1l_0$	$\cdots$
$bits$	1	0	0	$\cdots$	l_4	$\cdots$	$\cdots$
$result$	0	0	0	$\cdots$	-1	l	$\cdots$

In this, the digit l is assumed negative ($l_4 = 1$) and the whole result is negative at the end. If l is positive then one more digit would become zero and the result will be $0\,(l-16)\cdots$. Note that even in this case, the sign of the result is still negative since $\mid l \mid \leq 15$. In general due to the choice of base and possible values in this number system, any number is of the sign of its leading non-zero digit [14]. The N and P-recodings do not alter that.

The condition mentioned above for the P-recoding to change the bits is $s_{i_0} = 1, s_{i-1_4} = 0$ and $\overline{z}_{i-1} = 1$, where $\overline{z}_{i-1}$ is an indicator to show if the digit $i-1$ is not zero. Let $u_i = s_{i_0}\overline{s}_{i-1_4}\overline{z}_{i-1}$, then if $u_i = 1$ the output of the P-recoding for digit i should be $s_i + 1$ instead of s_i. Obviously, u_i could be added to s_i or, better, it could be used as a select line in a multiplexer which has s_i and $s_i + 1$ as its inputs. This u_i signal also affects the most significant bit of the lower adjacent digit s_{i-1}. If $u_i = 0$ then the output of the P-recoding for this bit, $s^p_{i-1_4}$, is determined just by the outcome of the multiplexer choosing between s_{i-1} and $s_{i-1} + 1$ depending on u_{i-1}. If, on the other hand, $u_i = 1$ then the two possible cases of u_{i-1} need to be analyzed.

$\mathbf{u_{i-1} = 0}$**:** $s^p_{i-1_4} = 1 = \overline{s}_{i-1_4}$ (remember that $s_{i-1_4} = 0$ for $u_i = 1$).

$\mathbf{u_{i-1} = 1}$**:** then two conditions are possible:

sign bit of $s_{i-1} + 1$ **is** 0**:** Then, as above, $s^p_{i-1_4} = 1$.

sign bit of $s_{i-1} + 1$ **is** 1**:** This means that due to the added 1 an overflow occurred which has a value of $+1$. That positive overflow is canceled out by

the -1 resulting from the P-recoding splitting of the original 0 in s_{i-1_4}, thus $s^p_{i-1_4} = 0$.

Hence in all the cases, if $u_i = 1$ the resulting $s^p_{i-1_4}$ bit is the inverse of the bit coming out of the multiplexer choosing between s_{i-1} and $s_{i-1} + 1$.

As is the case for the N-recoding an "out of bound" digit value can occur. In this case, a value of $+16$ results if the pattern of digits $k\ 15\ l$ with $k_0 = 0$ and $l_4 = 0$ are entered into a P-recoder. Again, this is not problematic since this recoded format is only within the LDD circuits and the position of the leading non-zero digit will be correctly detected as described below. However, this is why it was important to note above that in P-recoding, this split of the zero in s_{i-1_4} is to occur only if s_{i-1} is not exactly equal to zero in order to prevent the new s_{i-1} from becoming -16. Otherwise, there would have been a difficulty to distinguish between the out of bound $+16$ and the -16 resulting from subtracting 16 from a digit that is already zero.

The condition $u_i = s_{i_0}\overline{s}_{i-1_4}\overline{z}_{i-1}$ is not a tight condition. The P-recoding causing insignificant digits deletion occurs when $s_{i-1} = 15\ (= 01111_{bin})$ and $s_i = -1\ (= 11111_{bin})$ or $s_i = 15$ (assuming that it is preceded by some $s = -1$) . So, the strictest condition is $u_i = s_{i_3}s_{i_2}s_{i_1}s_{i_0}\overline{s}_{i-1_4}s_{i-1_3}s_{i-1_2}s_{i-1_1}s_{i-1_0}$. Other conditions between those two extremes can also be used like for example $u_i = s_{i_3}s_{i_2}s_{i_1}s_{i_0}\overline{s}_{i-1_4}s_{i-1_3}$.

For each digit after the recodings, an indicator n_i is used to specify the sign of the digit. Another indicator $\overline{z}_i$ is kept to indicate if the digit is not zero. The final decision of the LDD is based on those n_i's and $\overline{z}_i$'s. The first digit that has $\overline{z}_i = 1$ is the leading digit and its sign is the corresponding n_i.

Either the N-recoding or the P-recoding can be done first, there is no strict order. For example, using $u_i = s_{i_3}s_{i_2}s_{i_1}s_{i_0}\overline{s}_{i-1_4}s_{i-1_3}$. The case of $P(N(s))$ yields:

$$\begin{aligned} u_i^{pn} &= u_i \\ n_i^{pn} &= (\overline{s}_{i+1_0}s_{i_4} + u_{i+1})\overline{u}_i \\ \overline{z}_i^{pn} &= (\overline{s}_{i+1_0}s_{i_4} + s_{i_3} + s_{i_2} + s_{i_1} + s_{i_0}\overline{s}_{i-1_4})\overline{u}_i \\ &\quad + u_i\overline{u}_{i+1}(s_{i+1_0} + \overline{s}_{i_4}) \end{aligned}$$

And the case of $N(P(s))$ yields:

$$\begin{aligned} u_i^{p} &= u_i \\ n_i^{np} &= (\overline{s}_{i+1_0}s_{i_4} + u_{i+1})\overline{u}_i \\ \overline{z}_i^{np} &= (\overline{s}_{i+1_0}s_{i_4} + s_{i_3} + s_{i_2} + s_{i_1} \\ &\quad + s_{i_0}(s_{i-1_4} \oplus \overline{u}_{i-1}))\overline{u}_i + u_i\overline{s}_{i+1_0}\overline{s}_{i_4} \end{aligned}$$

The leading non-zero digit is then determined by using the $\overline{z}$ bits out of the recodings. A tree network can be used to encode the position of the first non-zero digit and this amount is forwarded to the left shifters to normalize the result. Obviously, the n bits out of the recodings should be shifted as well. The final sign of the number is that of the leading digit as determined by its n bit. Based on this either the result or its negation is chosen and the sign of the whole floating point result is affected.

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

As mentioned earlier, the fine adjustment is performed in the rounding stage. This is the other piece of challenging logic in the design at hand. In the proposed format the MSD has four bits. The rounding stage must determine the leading one among those four bits in order to decide on the approximate bit location for the rounding. The fine adjustment is then when another circuit determines if the remaining part of the number below the leading bit of the MSD is positive or negative. Those two indicators allow for the decision on the correct bit location to apply the IEEE rounding. 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. Any such rounding of the proposed format does not propagate a carry through the whole number as the rounding in conventional adders do. SD addition is used instead and the addition of a ± 1 digit representing the rounding decision is easily handled. 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 our 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

Table 2. Time delay of various components in terms of number of *FO4* delays.

Part	Delay
Adder	$5 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$
$[4:2]$ compressors	3
$(3,2)$ counters	2
Mux, in to out	1
Mux, select to out	$\lceil \log_4(n) \rceil + 1$
Signed digit adder	$8 + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil$ $+ \lceil \log_4(r+1) \rceil$
Shifter	$\lceil \log_2(n) \rceil$
Other (no design details)	$\lceil \log_f(n) \rceil$

is chosen.

4. Time delay modeling and rationale for postponed rounding

The current authors had previously proposed [6] a parametric time delay model to compare floating point unit implementations. In that analytical model, the operand width n, the fan-in f of the logic gates and the radix r of the redundant format are used as parameters. The model gives an estimate of the number of equivalent elementary delay units in the critical path of the floating point hardware. The floating point unit delay is presented in "fanout of 4"(*FO4*) delays, or the delay of an inverter driving a load that is four times its own size. The model is validated through transistor simulation of different circuits. The different parts of the model are summarized in Table 2. Using units of *FO4* delays makes the model independent of the technology scaling to a large degree since this elementary gate scales almost linearly with the technology [11]. However, the model does not include any assumptions about long wires across the chip and the time delay associated with them.

This simple model allows us to have a sense of the complexity of some parts of the floating point addition algorithms. We see that 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 through the whole number and it is also an $\mathcal{O}(\log n)$ operation. It is usually combined with the addition step [17] so that both time delays overlap. Another part that is of $\mathcal{O}(\log n)$ is the leading one prediction (if fine adjustment is performed) as mentioned in section 3. Simply using a redundant format that makes the significand addition independent of n will not enhance the speed by much if nothing is done to all those other parts. That is why postponed rounding and the other features in the proposed two-path adder are integral to the design and are as important as the redundant format. To quantify this argument let us use the assumptions of the analytical model to estimate the time delay of the adder design described above.

The critical path of the design starts with the exponent difference. This is a 15 bit adder and not an 11 bit one as in conventional adders using double precision because of the special format used in this design. In fact, the exponent width in this format $expWF$ is equal to the conventional exponent width $expW$ (which is dependent on n as specified by the IEEE standard) expanded to allow for the normalization of denormalized numbers minus $\log_2 r$ when the radix is 2^r. The significand in this format is also larger than the corresponding significand for the conventional designs because of the redundancy. The significand width is $\lceil \frac{n}{r} \rceil \times (r+1) - 1$. The swapping multiplexers must be as wide as the significand and the output of the exponent difference is used to drive the select lines. Up to this point, the delay is estimated to be $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{expWF}{f} \rceil - 1) \rceil + \lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil + 1$ *FO4* delays. The operand then passes through a $\lceil \frac{n}{r} \rceil$-way shifter which adds $\lceil \log_2(\lceil \frac{n}{r} \rceil) \rceil$ *FO4* delays. The following multiplexer adds one more *FO4* delay. The signed digit adder takes $8 + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil + \lceil \log_4(r+1) \rceil$ *FO4* delays. The select lines of the last multiplexer partially depend on the output of the adder in order to determine if there is a need to adjust to the right by one bit. Hence, there is a delay from the select lines to the output equal to $\lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil + 1$ *FO4* delays. The total delay for this design is thus:

$$
\begin{aligned}
\tau \;=\; & 16 \\
& + 2 \times \lceil \log_{f-1}(\lceil \frac{expWF}{f} \rceil - 1) \rceil \\
& + 2 \times \lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil \\
& + \lceil \log_2(\lceil \frac{n}{r} \rceil) \rceil \\
& + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil + \lceil \log_4(r+1) \rceil
\end{aligned}
$$

From this derivation we can evaluate the benefit coming from each of the novel ideas in this design: the postponed rounding, the use of a higher radix base for the exponents and the use of an SD adder.

Once the significand addition is independent of n by using redundancy, the rounding delay must be masked by the delay of another part that is as long or longer. Otherwise, it will add to the overall delay of the adder. The exponent difference and multiplexers time delay (second line and half of the third line of the equation) are both $\mathcal{O}(\log n)$ operations that are essential and that are already on the critical path. Performing the rounding in parallel with them seems then to be the best choice. Hence, the rounding delay does not appear at all in the equation and is effectively hidden.

The higher radix base for the exponent has an effect on the alignment shifter which only shifts then to digit bound-

aries. The fourth line of the multi-line equation above captures this as a delay of $\lceil \log_2(\lceil \frac{n}{r} \rceil) \rceil$ rather than $\lceil \log_2(n) \rceil$ in conventional adders. Shifting is still an $\mathcal{O}(\log n)$ operation but its time delay is reduced by about $\log_2 r$ when using a higher radix.

The effect of the SD adder is shown in the fourth line (and part of the constant of the first line) where the terms $8+2\times\lceil\log_{f-1}(\lceil\frac{r+1}{f}\rceil-1)\rceil+\lceil\log_4(r+1)\rceil$ appear instead of $5+2\times\lceil\log_{f-1}(\lceil\frac{n}{f}\rceil-1)\rceil$ in a conventional adder.

It is clear that the effect of r on the shifter delay is the opposite of its effect on the SD adder delay. The amount of redundancy (reflected by r) also has an effect on the area of the circuit and the required increase in the register file as noted in section 1. Hence, depending on the choice of n and f for the implementation, the benefit from using a different radix for the exponent may be more than the benefit from the redundancy or vice versa. To compare this design to other designs some assumptions regarding those parameters are needed. For practical CMOS designs, the fan-in is usually limited to 3 or 4. The majority of the floating point adders are currently designed to handle double precision numbers ($n = 53$) or larger. For this range, the design proposed with r set to 4 or 8 provides the best performance as presented in our previous work [6].

5. The multiplication, division and elementary functions

The design of the proposed multiplier is shown in Fig. 3. The least significant digit and the rounding bits of each operand are used to determine the rounding values r_x and r_y. Simultaneously, the multiplier Y operand is forwarded to a modified Booth recoding block and the multiplicand X is used to generate the partial products. The multiplication proceeds as $(X+r_x)(Y+r_y) = XY+r_xY+r_yX+r_xr_y$. The rounding correction block produces the last three terms of this equation and delivers them to the reduction tree with the rest of the partial products.

As noted in section 1, the extra bits in the format are negatively valued. Hence, for the multiplier operand Y, the Booth 2 recoding scheme is modified to take into consideration those extra negative bits. The extra negative bits of the multiplicand, X, are dealt with in a slightly different way. The significand of X is taken as having two components: P the positively valued bits and E the negatively valued extra bits. The output of the Booth recoders are used as select lines in multiplexers to generate the required partial products. The positive vectors are then summed by a tree of $[4:2]$ compressors while the negative vectors are summed by a separate tree of $[4:2]$ compressors. The output of each tree is in carry save format. The positive and negative vectors are then added using a $[4:2]$ compressor followed by a signed digit adder to form the final result.

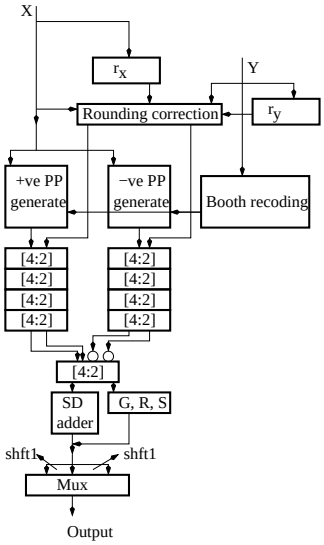

Figure 3. General block diagram of the multiplier.

To perform division and other elementary functions, a design from the literature is adapted [8, 9]. This arithmetic unit provides rapid convergence based on higher-order Newton-Raphson and series expansion techniques. To adapt the original design to the format proposed here, a short adder is used to eliminate the redundancy from the most significant part of the divisor operand by subtracting the extra bits. This non-redundant part is used to access the lookup table while the rest of the operand is fully transformed into a non-redundant form. In parallel, another adder is used to convert the dividend into a non-redundant form as well. The unit then works on those two operands as in the original design and at the end a signed digit adder is used instead of the regular carry propagate adder. The delay of the proposed unit is not much different from the original design since the extra delay of the short adder at the start is compensated by the reduced delay in the final addition.

6. Simulation results

A scalable CMOS technology was used to design the adder and the multiplier at the transistor level with $n = 53$, $f = 3$ and $r = 4$ implementing all the IEEE rounding modes.

Both circuits mostly use static CMOS technology gates with only few parts using NMOS pass transistors (namely the shifters). The designs were simulated for functionality at the logic level using *verilog* and for speed at the transistor level using the switch level simulator *irsim*.

Exhaustive testing of the functionality is obviously not practical for such designs. However, with the multiplier

Table 3. Circuit statistics and simulation results for the adder and multiplier.

	Adder	Multiplier
number of nodes	46845	76523
NMOS transistors	63589	104695
PMOS transistors	61649	105037
Test vectors	5000	10000
Model delay	$34FO4$	$35FO4$
Sim delay($0.6\mu m$)	$14ns$	$14.8ns$
	($33.35FO4$)	($35.25FO4$)
Sim delay($0.3\mu m$)	$6ns$	$6.4ns$
	($32.40FO4$)	($34.60FO4$)

for example successfully passing more than 2.4 million random test vectors (with a random rounding mode selected for each test), we are confident enough in the implementation. Our focus then changed to the speed simulation and sizing the different gates and transistors. Technology files ranging from $0.6\mu m$ down to $0.3\mu m$ were used. On that range of scaling factors, the adder and multiplier perform as predicted by the analytical model when compared to the delay of *FO4* inverters at the same scaling factor. At that level fewer test vectors are used due to the longer time the simulation takes. So, only $5\,000$ and $10\,000$ random test vectors are used for the adder and the multiplier respectively. The multiplier being a larger circuit it is tested more thoroughly. The results are given in Table 3.

7. Conclusions

The proposed internal format with the proposed algorithms and arithmetic units provide a complete arithmetic system allowing all the IEEE rounding modes. The elimination of carry propagation from the arithmetic operations enhances the performance of the functional units. The proposed arithmetic unit architecture includes further enhancements that increase the floating point performance such as a hexadecimal based number format and postponed rounding techniques. The proposed system pushes the performance boundary of the design space and provides a means to achieve the computational demands of numerically intensive applications. For the double (and specially larger) precision units using standard CMOS technologies, the designs presented here are predicted to yield the highest performance.

References

[1] IEEE standard for binary floating-point arithmetic, Aug. 1985. (ANSI/IEEE Std 754-1985).

[2] J. D. Bruguera and T. Lang. Leading-one prediction with concurrent position correction. *IEEE Transactions on Computers*, 48(10):1083–1097, Oct. 1999.

[3] M. Daumas and D. Matula. Recoders for partial compression and rounding. Research Report No. RR97-01, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Jan. 1997. Available at http://www.ens-lyon.fr/LIP/Pub/rr1997.html.

[4] M. Daumas and D. Matula. Further reducing the redundancy of a notation over a minimally redundant digit set. Research Report No. RR2000-09, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Mar. 2000. Available at http://www.ens-lyon.fr/LIP/Pub/rr2000.html.

[5] H. A. H. Fahmy, A. A. Liddicoat, and M. J. Flynn. Improving the effectiveness of floating point arithmetic. In *Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA*, volume 1, pages 875–879, Nov. 2001.

[6] 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.

[7] P. M. Farmwald. *On the Design of High Performance Digital Arithmetic Units*. PhD thesis, Stanford University, Aug. 1981.

[8] A. A. Liddicoat. *High-Performance Arithmetic for Division and The Elementary Functions*. PhD thesis, Stanford University, Feb. 2002.

[9] A. A. Liddicoat and M. J. Flynn. High-performance floating point divide. In *Proceedings of the Euromicro Symposium on Digital System Design*, pages 354–361, Sept. 2001.

[10] D. W. Matula and A. M. Nielsen. Pipelined packet-forwarding floating point: I. foundations and a rounder. In *Proceedings of the 13th IEEE Sympsoium on Computer Arithmetic, Asilomar, California, USA*, pages 140–147, July 1997.

[11] G. W. McFarland. *CMOS Technology Scaling and Its Impact on Cache Delay*. PhD thesis, Stanford University, June 1997.

[12] A. M. Nielsen, D. W. Matula, C. N. Lyu, and G. Even. An IEEE compliant floating-point adder that conforms with the pipelined packet-forwarding paradigm. *IEEE Transactions on Computers*, 49(1):33–47, Jan. 2000.

[13] B. Parhami. Generalized signed-digit number systems: A unifying framework for redundant number representations. *IEEE Transactions on Computers*, 39(1):89–98, Jan. 1990.

[14] B. Parhami. On the implementation of arithmetic support functions for generalized signed-digit number systems. *IEEE Transactions on Computers*, 42(3):379–384, Mar. 1993.

[15] D. S. Phatak and I. Koren. Hybrid signed-digit number systems: A unified framework for redundant number representations with bounded carry propagation chains. *IEEE Transactions on Computers*, 43(8):880–891, Aug. 1994.

[16] N. Quach and M. J. Flynn. Leading one prediction—implementation, generalization, and application. Technical Report No. CSL-TR-91-463, Computer Systems Laboratory, Stanford University, Mar. 1991.

[17] N. T. Quach. *Reducing the latency of floating-point arithmetic operations*. PhD thesis, Stanford University, Dec. 1993.