On the Design of Fast IEEE Floating-Point Adders
(Extended Abstract)

Peter-Michael Seidel
Southern Methodist University
Computer Sci&Eng Department
Dallas, TX, 75275
seidel@seas.smu.edu

Guy Even
Tel-Aviv University
Electrical Engineering Department
69978 Tel-Aviv, Israel
guy@eng.tau.ac.il

Abstract

We present an IEEE floating-point adder (FP-adder) design. The adder accepts normalized numbers, supports all four IEEE rounding modes, and outputs the correctly normalized rounded sum/difference in the format required by the IEEE Standard. The latency of the design for double precision is roughly 24 *logic levels, not including delays of latches between pipeline stages. Moreover, the design can be easily partitioned into* 2 *stages consisting of* 12 *logic levels each, and hence, can be used with clock periods that allow for* 12 *logic levels between latches. The FP-adder design achieves a low latency by combining various optimization techniques such as: a non-standard separation into two paths, a simple rounding algorithm, unifying rounding cases for addition and subtraction, sign-magnitude computation of a difference based on one's complement subtraction, compound adders, and fast circuits for approximate counting of leading zeros from borrow-save representation. A comparison of our design with other implementations suggests a reduction in the latency by at least two logic levels as well as simplified rounding implementation. A reduced precision version of our algorithm has been verified by exhaustive testing.*

1 Introduction and Summary

Floating-point addition and subtraction are the most frequent floating-point operations. Both operations use a floating-point adder (FP-adder). Therefore, a lot of effort has been spent on reducing the latency of FP-adders (see [3, 8, 16, 18, 19, 20, 21, 22, 23] and the references that appear there). Many patents deal with FP-adder design (see [6, 9, 10, 12, 13, 14, 15, 17, 24, 25, 27]).

We present an FP-adder design that accepts normalized double precision significands, supports all IEEE rounding modes, and outputs the normalized sum/difference that is rounded according to the IEEE FP standard 754 [11]. The latency of our design is analyzed in technology-independent terms (i.e. logic levels) to facilitate comparisons with other designs. The latency of the design for double precision is roughly 24 logic levels, not including delays of latches between pipeline stages. This design is amenable to pipelining with short clock periods; in particular, it can be easily partitioned into 2 stages consisting of 12 logic levels each. Extensions of the algorithm that deal with denormal inputs and outputs are discussed in [1, 22]. It is shown that the delay overhead can be reduced to 1-2 logic levels.

We employ several optimization techniques in our algorithm. A detailed examination of these techniques enables us to demonstrate how these techniques can be combined to achieve an overall fast FP-adder design. In particular, effective reduction of latency by parallel paths requires balancing the delay of the paths. We achieve such a balance by a gate-level consideration of the design. The optimization techniques that we use include the following techniques: (a) A two path design with a non-standard separation criterion. Instead of separation based on the magnitude of the exponent difference [9], we define a separation criterion that also considers whether the operation is effective subtraction and the value of the significand difference. This separation criterion maintains the advantages of the standard two-path designs, namely, alignment shift and normalization shift take place only in one of the paths and the full exponent difference is computed only in one path. In addition, this separation technique requires rounding to take place only in one path. (b) Reduction of rounding modes and injection based rounding. Following Quach *et al.* [21], the IEEE rounding modes are reduced to 3 modes, and following [7], injection based rounding is employed to design the rounding circuitry. (c) A simpler design is obtained by using unconditional pre-shifts for effective subtractions to reduce to 2 the number of binades that the significand sum and difference could belong to. (d) One's complement representation is used to compute the sign-magnitude representation of the difference of the exponents and the significands. (e) A parallel-prefix adder is used to compute the

0-7695-1150-3/01 $10.00 © 2001 IEEE

sum and the incremented sum of the significands [26]. (f) Recodings are used to estimate the number of leading zeros in the non-redundant representation of a number represented as a borrow-save number [16]. (h) Due to the latency of the rounding decision signal, the computation of the post-normalization is advanced and takes place before the rounding decision is ready.

To relate the proposed implementation to previous FP-adder designs, we give an overview about other FP-adder implementations from technical papers or patents, and summarize the optimization techniques that are used in each of these designs. We analyze two particular implementations from literature in some more detail [10, 17]. To allow for a "fair"comparison, the functionality of these designs were adopted to match the functionality of our design. A comparison of these designs with our design suggests that our design is faster by at least 2 logic levels. In addition, our design uses simpler rounding circuitry and is more amenable to partitioning into two pipeline stages of equal latency.

This paper focuses on double precision FP-adder implementations. Many FP-adders support multiple precisions (e.g. x86 architectures support single, double, and extended double precision). In [22] it is shown that by aligning the rounding position (i.e. 23 positions to the right of the binary point in single precision and 52 positions to the right of the binary point in double precision) of the significands before they are input to the design and post-aligning the outcome of the FP-adder, it is possible to use the FP-adder presented in this paper for multiple precisions. Hence, the FP-addition algorithm presented in this paper can be used to support multiple precisions.

The correctness of our FP-adder design was verified by conducting exhaustive testing on a reduced precision version of our design [2].

Many details are omitted from this extended abstract and appear in the full version.

2 Notation

We denote binary strings in upper case letters (e.g. S,E,F). The value represented by a binary string is represented in italics (e.g. s, e, f).

We consider normalized IEEE FP-numbers. In double precision IEEE FP-numbers are represented by three fields $(\mathrm{S}, \mathrm{E}[10:0], \mathrm{F}[0:52])$ with *sign bit* $\mathrm{S} \in \{0,1\}$, exponent string $\mathrm{E}[10:0] \in \{0,1\}^{11}$ and significand string $\mathrm{F}[0:53] \in \{0,1\}^{53}$. The values of exponent and significand are defined by:

$$e = \sum_{i=0}^{10} \mathrm{E}[i] \cdot 2^i - 1023, \qquad f = \sum_{i=0}^{52} \mathrm{F}[i] \cdot 2^{-i}.$$

Since we only consider normalized FP-numbers, we have $f \in [1,2)$. A FP-number $(\mathrm{S}, \mathrm{E}[10:0], \mathrm{F}[0:52])$ represents the value: $fp\_val(\mathrm{S}, \mathrm{E}, \mathrm{F}) = (-1)^{\mathrm{S}} \cdot 2^e \cdot f$.

Given an IEEE FP-number $(\mathrm{S}, \mathrm{E}, \mathrm{F})$, we refer to the triple (s, e, f) as the *factoring* of the FP-number. Note that $s = \mathrm{S}$ since S is a single bit. The advantage of using factorings is the ability to ignore representation details and focus on values. The inputs of a FP-addition/subtraction are:

1. operands denoted by $(\mathrm{SA}, \mathrm{EA}[10:0], \mathrm{FA}[0:52])$ and $(\mathrm{SB}, \mathrm{EB}[10:0], \mathrm{FB}[0:52])$;
2. an operation $\mathrm{SOP} \in \{0,1\}$ where $\mathrm{SOP} = 0$ denotes an addition and $\mathrm{SOP} = 1$ denotes a subtraction;
3. IEEE rounding mode.

The output is a FP-number $(\mathrm{S}, \mathrm{E}[10:0], \mathrm{F}[0:52])$. The value represented by the output equals the IEEE rounded value of

$$\begin{aligned} fpsum \;=\; & fp\_val(\mathrm{SA}, \mathrm{EA}[10:0], \mathrm{FA}[0:52]) + \\ & (-1)^{\mathrm{SOP}} fp\_val(\mathrm{SB}, \mathrm{EB}[10:0], \mathrm{FB}[0:52]) \end{aligned}$$

3 Naive FP-adder Algorithm

In this section we overview the "vanilla" FP-addition algorithm. To simplify notation, we ignore representation and deal only with the values of the inputs, outputs, and intermediate results. Throughout the paper we use the notation defined for the naive algorithm.

Let (sa, ea, fa) and (sb, eb, fb) denote the factorings of the operands with a sign-bit, an exponent, and a significand and let SOP indicate whether the operation is an addition or a subtraction. The requested computation is the IEEE FP representation of the rounded sum:

$$rnd(sum) = rnd((-1)^{sa} \cdot 2^{ea} \cdot fa + (-1)^{\mathrm{SOP}+sb} \cdot 2^{eb} \cdot fb).$$

Let $\mathrm{S.EFF} = sa \oplus sb \oplus \mathrm{SOP}$. The case that $\mathrm{S.EFF} = 0$ is called *effective addition* and the case that $\mathrm{S.EFF} = 1$ is called *effective subtraction*.

We define the exponent difference $\delta = ea - eb$. The "large" operand, (sl, el, fl), and the "small" operand, (ss, es, fs), are defined as follows:

$$(sl, el, fl) = \begin{cases} (sa, ea, fa) & \text{if } \delta \geq 0 \\ (\mathrm{SOP} \oplus sb, eb, fb) & \text{otherwise} \end{cases}$$

$$(ss, es, fs) = \begin{cases} (\mathrm{SOP} \oplus sb, eb, fb) & \text{if } \delta \geq 0 \\ (sa, ea, fa) & \text{otherwise.} \end{cases}$$

The sum can be written as

$$sum = (-1)^{sl} \cdot 2^{el} \cdot (fl + (-1)^{\mathrm{S.EFF}} (fs \cdot 2^{-|\delta|})).$$

To simplify the description of the datapaths, we focus on the computation of the result's significand, which is assumed to be normalized (i.e. in the range $[1,2)$). The significand sum is defined by

$$fsum = fl + (-1)^{\mathrm{S.EFF}} (fs \cdot 2^{-|\delta|}).$$

The significand sum is computed, normalized, and rounded as follows:

1. exponent subtraction $\delta = ea - eb$,
2. operand swapping (compute sl,el,fl and fs),
3. limitation of the alignment shift amount: $\delta\_lim = \min\{\alpha, abs(\delta)\}$, where α is a constant greater than or equal to 55,
4. alignment shift of fs: $fsa = fs \cdot 2^{-\delta\_lim}$,
5. significand negation $fsan = (-1)^{\text{S.EFF}} fsa$,
6. significand addition $fsum = fl + fsan$,
7. conversion $abs\_fsum = abs(fsum)$, $\text{S} = sl \oplus (fsum < 0)$,
8. normalization $n\_fsum = norm(abs\_fsum)$,
9. rounding and post-normalization of n_fsum.

The naive FP-adder implements the 9 steps from above sequentially, where the delay of steps $4, 6, 7, 8, 9$ is logarithmic in the significand's length. Therefore, this is a slow FP-adder implementation.

4 Optimization Techniques

In this section we outline optimization techniques that were employed in the design of our FP-adder.

4.1 Separation of FP-adder into two parallel paths

The FP-adder pipeline is separated into two parallel paths that work under different assumptions. The partitioning into two parallel paths enables one to optimize each path separately by simplifying and skipping some of the steps of the naive addition algorithm (Sec. 3). Such a dual path approach for FP-addition was first described by Farmwald [8]. Since Farmwald's dual path FP-addition algorithm, the common criterion for partitioning the computation into two paths has been the exponent difference. The exponent difference criterion is defined as follows: the *near path* is defined for small exponent differences (i.e. $-1, 0, +1$), and the *far path* is defined for the remaining cases.

We use a different partitioning criterion for partitioning the algorithm into two paths: we define the *N-path* for the computation of all effective subtractions with small significand sums $fsum \in (-1, 1)$ and small exponent differences $|\delta| \leq 1$, and we define the *R-path* for all the remaining cases. We define the path selection signal IS_R as follows:

$$\text{IS\_R} \iff \overline{\text{S.EFF}} \text{ OR } |\delta| \geq 2 \text{ OR } fsum \in [1, 2). \quad (1)$$

The outcome of the R-path is selected for the final result if IS_R $= 1$, otherwise the outcome of the N-path is selected. This partitioning has the following advantages:

1. In the R-path, the normalization shift is limited to a shift by one position (in Sec. 4.2 we show how the normalization shift may be restricted to one direction). Moreover, the addition or subtraction of the significands in the R-path always results with a positive significand, and therefore, the conversion step can be skipped.
2. In the N-path, the alignment shift is limited to a shift by one position to the right. Under the assumptions of the N-path, the exponent difference is in the range $\{-1, 0, 1\}$. Therefore, a 2-bit subtraction suffices for extracting the exponent difference. Moreover, in the N-path, the significand difference can be exactly represented with 53 bits, hence, no rounding is required.

Note that the N-path applies only to effective subtractions in which the significand difference $fsum$ is less than 1. Thus, in the N-path it is assumed that $fsum \in (-1, 1)$.

The advantages of our partitioning criterion compared to the exponent difference criterion stem from the following two observations: (a) a conventional implementation of a far path can be used to implement also the R-path; and (b) the N-path is simpler than the near path since no rounding is required and the N-path applies only to effective subtractions. Hence, we were able to implement the N-path simpler and faster than the near-path presented in [23].

4.2 Unification of significand result ranges

In the R-path, the range of the resulting significand is different in effective addition and effective subtraction. Using the notation of Sec. 3, in effective addition, $fl \in [1, 2)$ and $fsan \in [0, 2)$. Therefore, $fsum \in [1, 4)$. It follows from the definition of the path selection condition that in effective subtractions $fsum \in (\frac{1}{2}, 2)$ in the R-path. We unify the ranges of $fsum$ in these two cases to $[1, 4)$ by multiplying the significands by 2 in the case of effective subtraction (i.e. pre-shifting by one position to the left). The unification of the range of the significand sum in effective subtraction and effective addition simplifies the rounding circuitry. To simplify the notation and the implementation of the path selection condition we also pre-shift the operands for effective subtractions in the N-path. Note, that in this way the pre-shift is computed in the N-path unconditionally, because in the N-path all operations are effective subtractions. In the following we give a few examples of values that include the conditional pre-shift (note that an additional 'p' is included in the names of the pre-shifted versions):

$$flp = \begin{cases} 2 \cdot fl & \text{if S.EFF} \\ fl & \text{otherwise} \end{cases}$$

$$fspan = \begin{cases} 2 \cdot fsan & \text{if S.EFF} \\ fsan & \text{otherwise} \end{cases}$$

$$fpsum = \begin{cases} 2 \cdot fsum & \text{if S.EFF} \\ fsum & \text{otherwise.} \end{cases}$$

Note, that based on the significand sum $fpsum$, which includes the conditional pre-shift, the path selection condition can be rewritten as

$$\text{IS\_R} \iff \overline{\text{S.EFF}} \text{ OR } |\delta| \geq 2 \text{ OR } fpsum \in [2, 4). \quad (2)$$

4.3 Reduction of IEEE rounding modes

The IEEE-754-1985 Standard defines four rounding modes: round toward 0, round toward $+\infty$, round toward $-\infty$, and round to nearest (even) [11]. Following Quach *et al.* [21], we reduce the four IEEE rounding modes to three rounding modes: round-to-zero RZ, round-to-infinity RI, and round-to-nearest-up RNU. The discrepancy between round-to-nearest-even and RNU is fixed by pulling down the LSB of the fraction (see [7] for more details).

In the rounding implementation in the R-path, the three rounding modes RZ, RNU and RI are further reduced to truncation using injection based rounding [7]. The reduction is based on adding an injection that depends only on the rounding mode. Let $\mathrm{X} = \mathrm{x}_0.\mathrm{x}_1\mathrm{x}_2\ldots\mathrm{x}_k$ denote the binary representation of a significand with the value $x = |\mathrm{X}| \in [1,2)$ for which $k \geq 53$ (double precision rounding is trivial for $k < 53$), then the injection is defined by:

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

For double precision and $mode \in \{RZ, RNU, RI\}$, the effect of adding INJ is summarized in the following equation:

$$|\mathrm{X}| \in [1,2) \;\Rightarrow\; rnd_{mode}(|\mathrm{X}|) = rnd_{RZ}(|\mathrm{X}| + \mathrm{INJ}). \quad (3)$$

4.4 Sign-magnitude computation of a difference

In this technique the sign-magnitude computation of a difference is computed using one's complement representation [18]. This technique is applied in two situations:

1. Exponent difference. The sign-magnitude representation of the exponent difference is used for two purposes: (a) the sign determines which operand is selected as the "large" operand; and (b) the magnitude determines the amount of the alignment shift.

2. Significand difference. In case the exponent difference is zero and an effective subtraction takes place, the significand difference might be negative. The sign of the significand difference is used to update the sign of the result and the magnitude is normalized to become the result's significand.

Let A and B denote binary strings and let $|\mathrm{A}|$ denote the value represented by A (i.e. $|\mathrm{A}| = \sum_i \mathrm{A}[i] \cdot 2^i$). The technique is based on the following observation:

$$abs(|\mathrm{A}| - |\mathrm{B}|) = \begin{cases} |\mathrm{A}| + |\overline{\mathrm{B}}| + 1 & \text{if } |\mathrm{A}| - |\mathrm{B}| > 0 \\ \overline{|\mathrm{A}| + |\overline{\mathrm{B}}|} & \text{if } |\mathrm{A}| - |\mathrm{B}| \leq 0 \end{cases}$$

The actual computation proceeds as follows: The binary string D is computed such that $|\mathrm{D}| = |\mathrm{A}| + |\overline{\mathrm{B}}|$. We refer to D as the *one's complement lazy* difference of A and B. We consider two cases:

1. If the difference is positive, then $|\mathrm{D}|$ is off by an ulp and we need to increment $|\mathrm{D}|$. However, to save delay, we avoid the increment as follows: (a) In the case of the exponent difference that determines the amount of the alignment shift, the significands are pre-shifted by one position to compensate for the error. (b) In the case of the significand difference, the missing ulp is provided by computing the incremented sum of $|\mathrm{A}|$ and $|\overline{\mathrm{B}}|$ using a compound adder.

2. If the exponent difference is negative, then the bits of D are negated to obtain an exact representation of the magnitude of the difference.

4.5 Compound addition

The technique of computing in parallel the sum of the significands as well as the incremented sum is well known. The rounding decision controls which of the sums is selected for the final result, thus enabling the computation of the sum and the rounding decision in parallel.

Technique. We follow the technique suggested by Tyagi [26] for implementing a compound adder. This technique is based on a parallel prefix adder in which the carry-generate and carry-propagate strings, denoted by *Gen_C* and *Prop_C*, are computed [4]. Let $Gen\_C[i]$ equal the carry bit that is fed to position i. The bits of the sum S of the addends A and B are obtained as usual by:

$$S[i] = xor(A[i], B[i], Gen\_C[i]).$$

The bits of the incremented sum SI are obtained by:

$$SI[i] = xor(A[i], B[i], or(Gen\_C[i], Prop\_C[i])).$$

Application. There are two instances of a compound adder in our FP-addition algorithm. One instance appears in the second pipeline stage of the R-path where our delay analysis relies on the assumption that the MSB of the sum is valid one logic level prior to the slowest sum bit.

The second instance of a compound adder appears in the N-path. In this case we also address the problem that the compound adder does not "fit" in the first pipeline stage according to our delay analysis. We break this critical path by partitioning the compound adder between the first and second pipeline stages as follows: A parallel prefix adder placed in the first pipeline stage computes the carry-generate and carry-propagate signals as well as the bitwise *xor* of the addends. From these three binary strings the sum and incremented sum are computed within two logic levels

as described above. However, these two logic levels must belong to different pipeline stages. We therefore compute first the three binary strings $S[i]$, $P[i] = A[i]\ xor\ B[i]$ and $GP\_C[i] = or(Gen\_C[i], Prop\_C[i])$ which are passed to the second pipeline stage. In this way the computation of the sum is already completed in the first pipeline stage and only an *xor*-line is required in the second pipeline stage to compute also the incremented sum.

4.6 Approximate counting of leading zeros

In the N-path a resulting significand in the range $(-1, 1)$ must be normalized. The amount of the normalization shift is determined by approximating the number of leading zeros. Following Nielsen *et al.* [16], we approximate the number of leading zeros so that a normalization shift by this amount yields a significand in the range $[1, 4)$. The final normalization is then performed by post-normalization. There are various other implementations for the leading-zero approximation in literature. The input used for counting leading zeros in our design is a borrow-save representation of the difference. This design is amenable to partitioning into pipeline stages, and admits an elegant correctness proof that avoids a tedious case analysis.

Nielsen *et al.* [16] presented the following technique for approximately counting the number of leading zeros. The input consists of a borrow-save encoded digit string $\mathrm{F}[-1:52] \in \{-1, 0, 1\}^{54}$. We compute the borrow-save encoded string $\mathrm{F}'[-2:52] = P(N(\mathrm{F}[-1:52]))$, where $P()$ and $N()$ denote P-recoding and N-recoding [5, 16]. (P-recoding is like a "signed half-adder" in which the carry output has a positive sign, N-recoding is similar but has an output carry with a negative sign). The correctness of the technique is based on the following claim.

Claim 1 *[16] Suppose the borrow-save encoded string* $\mathrm{F}'[-2:52]$ *is of the form* $\mathrm{F}'[-2:52] = 0^k \cdot \sigma \cdot t[1:54-k]$, *where* $\cdot$ *denotes concatenation of strings,* 0^k *denotes a block of* k *zeros,* $\sigma \in \{-1, 1\}$, *and* $t \in \{-1, 0, 1\}^{54-k}$. *Then the following holds:*

(1.) If $\sigma = 1$, *then the value represented by the borrow encoded string* $\sigma.t$ *satisfies:* $\sigma + \sum_{i=1}^{54-k} t[i] \cdot 2^{-i} \in (\frac{1}{4}, 1)$.

(2.) If $\sigma = -1$, *then the value represented by the borrow encoded string* $\sigma.t$ *satisfies:* $\sigma + \sum_{i=1}^{54-k} t[i] \cdot 2^{-i} \in (-\frac{3}{2}, -\frac{1}{2})$.

The implication of Claim 1 is that after PN-recoding, the number of leading zeros in the borrow-save encoded string $\mathrm{F}'[-2 : 53]$ (denoted by k in the claim) can be used as the normalization shift amount to bring the normalized result into one of two binades (i.e. in the positive case either $(\frac{1}{4}, \frac{1}{2})$ or $[\frac{1}{2}, 1)$, and in the negative case after negation either $(\frac{1}{2}, 1)$ or $[1, \frac{3}{2})$).

We implemented this technique so that the normalized significand is in the range $[1, 4)$ as follows:

(1.) In the positive case, the shift amount is $lz2 = k = lzero(\mathrm{F}'[-2:52])$. (See signal $LZP2[5:0]$ in Fig. 4).

(2.) In the negative case, the shift amount is $lz1 = k\text{–}1 = lzero(\mathrm{F}'[-1:52])$. (See signal $LZP1[5:0]$ in Fig. 4).

4.7 Pre-computation of post-normalization shift

In the R-path two choices for the rounded significand sum are computed by the compound adder (see section 4.5). Either the 'sum' or the 'incremented sum' output of the compound adder is chosen for the rounded result. Because the significand after the rounding selection is in the range $[1, 4)$ (due to the pre-shifts from section 4.2 only these two binades have to be considered for rounding and for the post-normalization shift), post-normalization requires at most a right-shift by one bit position. Because the outputs of the compound adder have to wait for the computation of the rounding decision (selection based on the range of the sum output), we pre-compute the post-normalization shift on both outputs of the compound adder before the rounding selection, so that the rounding selection already outputs the normalized significand result of the R-path.

5 Our FP-adder Implementation

In this section we give an overview of our FP-adder implementation. We describe the partitioning of the design implementing the optimization techniques from the previous section. The algorithm is a dual path two-staged pipeline. The final result is selected between the outcomes of the two paths based on the signal IS_R (see equation 2). A high-level block diagram of algorithm is depicted in Figure 1. Details about the implementation of the stages are provided by detailed block diagrams. A detailed description of the implementation can be found in the full version of this paper.

R-Path. The R-path works under the assumption that (a) an effective addition takes place; or (b) an effective subtraction with a significand difference (after pre-shifting) greater than or equal to 2 takes place; or (c) the absolute value of the exponent difference $|\delta|$ is larger than or equal to 2. Note that these assumptions imply that the sign-bit of the sum equals SL.

The R-path is divided into two pipeline stages. Loosely speaking, in the first pipeline stage, the exponent difference is computed, the significands are swapped and pre-shifted if an effective subtraction takes place, and the subtrahend is negated and aligned. In the *Significand One's Complement* box, the significand to become the subtrahend is negated (recall that one's complement representation is used). In the *Align 1* box, the significand to become the subtrahend is (a) pre-shifted to the right if an effective subtraction takes

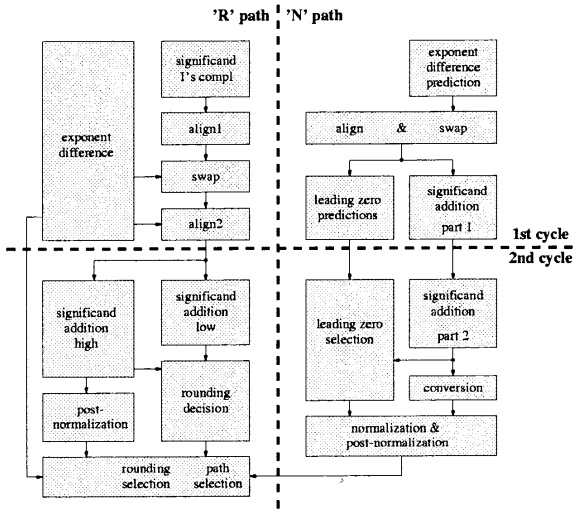

Figure 1. High level structure of the new FP-adder algorithm. Vertical dashed line separates two pipelines: R-path and N-path. Horizontal dashed line separates pipeline stages.

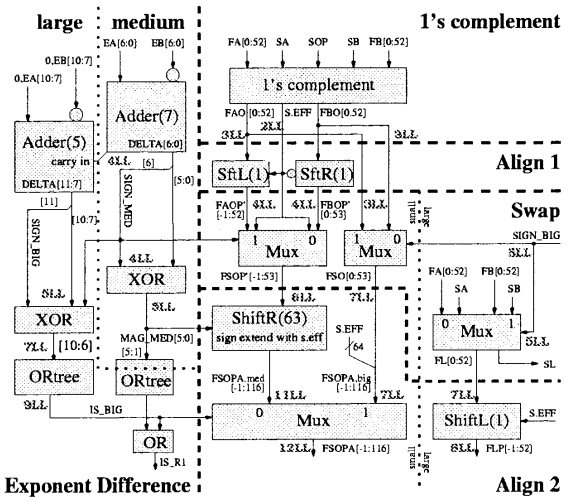

Figure 2. Detailed block diagram of the 1st clock cycle of the R-path annotated with timing estimates ("5LL" next to a signal means that the signal is valid after 5 logic levels).

place; and (b) aligned to the left by one position if the exponent difference is positive. This alignment by one position compensates for the error in the computation of the exponent difference when the difference is positive due to the one's complement representation (see Sec. 4.4). In the *Swap* box, the significands are swapped according to the sign of the exponent difference. In the *Align2* box, the subtrahend is aligned according to the computed exponent difference. The *exponent difference* box computes the swap decision and signals for the alignment shift. This box is further partitioned into two paths for medium and large exponent differences. A detailed block diagram for the implementation of the first cycle of the R-path is depicted in Figure 2.

The input to the second pipeline stage consists of the significand of the 'larger' operand and the aligned significand of the 'smaller' operand which is inverted for effective subtractions. The goal is to compute their sum and round it while taking into account the error due to the one's complement representation for effective subtractions. The second pipeline stage is very similar to the rounding algorithm presented in our companion paper [7]. The significands are divided into a low part and a high part that are processed in parallel. The low part computes the LSB of the final result based on the low part and the range of the sum. The high part computes the rest of the final result (which is either the sum or the incremented sum of the high part). The outputs of the compound adder are post-normalized before the rounding selection is performed. A detailed block diagram for the implementation of the second cycle of the R-path is depicted in Figure 3.

N-Path. The N-path works under the assumption that an effective subtraction takes place, the significand difference (after the swapping of the addends and pre-shifting) is less that 2 and the absolute value of the exponent difference $|\delta|$ is less than 2. The N-path has the following properties:

1. The exponent difference must be in the set $\{-1, 0, 1\}$. Hence, the exponent difference can be computed by subtracting the two LSBs of the exponent strings. The alignment shift is by at most one position. This is implemented in the *exponent difference prediction* box.
2. An effective subtraction takes place, hence, the significand corresponding to the subtrahend is always negated. We use one's complement representation for the negated subtrahend.
3. The significand difference (after swapping and pre-shifting) is in the range $(-2, 2)$ and can be exactly represented using 52 bits to the right of the binary point. Hence, no rounding is required.

Based on the exponent difference prediction the significands are swapped and aligned by at most one bit position in the *align and swap* box. The leading zero approximation and the significand difference are then computed in parallel. The result of the leading zero approximation is selected based on the sign of the significand difference according to Sec. 4.6 in the *leading zero selection* box. The *conversion* box computes the absolute value of the difference (Sec. 4.4) and the *normalization & post-normalization* box normalizes the absolute significand difference as a result of the N-path. Figure 4 depicts a detailed block diagram of the N-path.

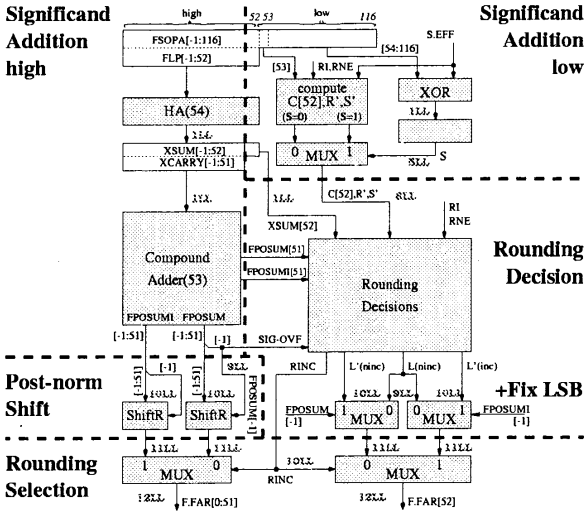

Figure 3. Detailed block diagram of the 2nd clock cycle of the R-path annotated with timing estimates.

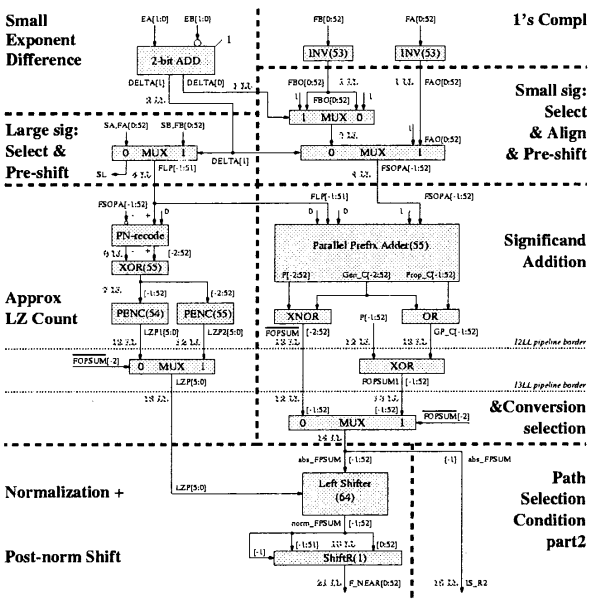

Figure 4. Detailed block diagram of the N-path annotated with timing estimates.

Path Selection We select between the R-path and the N-path result depending on the signal IS_R. The implementation of this condition is based on the three signals IS_R1, IS_R2 and S.EFF, where IS_R1 $\iff$ $(abs(delta) \geq 2)$ is the part of the path selection condition that is computed in the R-path and IS_R2 is the part of the path selection condition that is computed in the N-path. With the definition of IS_R1 we get according to Eq. 1:

$$\begin{aligned} \text{IS\_R} &= \overline{\text{S.EFF}} \vee \text{IS\_R1} \vee (fpsum \in [2,4)) \\ &= \overline{\text{S.EFF}} \vee \text{IS\_R1} \vee \\ &\quad ((fpsum \in [2,4)) \text{ AND S.EFF AND } \overline{\text{IS\_R1}}) . \end{aligned}$$

We define $\text{IS\_R2} = (fpsum \in [2,4)) \wedge \text{S.EFF} \wedge \overline{\text{IS\_R1}}$, so that

$$\text{IS\_R} = \overline{\text{S.EFF}} \vee \text{IS\_R1} \vee \text{IS\_R2}. \quad (4)$$

Because the assumptions S.EFF $= 1$ and $\overline{\text{IS\_R1}}$ are exactly the assumptions that we use during the computation of $fpsum$ in the N-path, the condition IS_R2 is easily implemented in the N-path by the bit at position $[-1]$ of the absolute significand difference. The condition IS_R1 and the signal S.EFF are computed in the R-path. After IS_R is computed from the three components according to equation 4, the valid result is selected either from the R-path or the N-path accordingly. Because the N-path result is valid a few logic levels before the R-path result, the path selection can be integrated with the final rounding selection in the R-path.

6 Delay Analysis of Our FP-adder

In this section we analyze the delay of our FP-adder implementation in technology-independent terms (logic levels). Our delay analysis is based on the detailed description of our algorithm in the full version of this paper and on the assumptions on delays of basic boxes used in [7, 23]. We discuss the delay analysis separately for the stages that are depicted in Figures 2,3 and 4.

For the first stage of the R-path the timing estimates are annotated in the block diagram in Figure 2, where "5LL" next to a signal means that the signal is valid after 5 logic levels. Our delay analysis suggests that the latest signals in this stage are valid after 12 logic levels. For the second stage of the R-path the timing estimates are annotated in the block diagram in Figure 3. Because the implementation of this stage is almost identical to the implementation of multiplication rounding from [7], also the timing estimates are almost identical and we get the latest signals in this stage after 12 logic levels, so that the whole R-path has a delay of 24 logic levels.

For the N-path the timing estimates are annotated in the block diagram in Figure 4. Corresponding to this delay analysis the latest signals in the whole N-path are valid after 21 logic levels, so that this path is not time critical. The delay analysis depicted in Fig. 4 suggests two pipeline borders: one after 12 logic levels and one after 13 logic lev-

els. As already discussed in section 4.5, a partitioning after 12 logic levels requires to partition the implementation of the compound adder between two stages. This can be done with the implementation of the compound adder from section 4.5, so that we get a first stage of the N-path that is valid after 12 logic levels and a second stage of the N-path where the signals are valid after 9 logic levels. This leaves some time in the second stage for routing the N-path result to the path selection mux in the R-path.

Hence, our FP-adder can be realized in 24 logic levels that can be partitioned into two pipeline stages with 12 logic levels between latches.

7 Verification and Testing of Our Algorithm

The FP addition and subtraction algorithm presented in this paper was verified and tested by Bar-Or *et al.* [2]. They used the following novel methodology. Two parametric algorithms for FP-addition were designed, each with p bits for the significand string and n bits for the exponent string. One algorithm is the naive algorithm, and the other algorithm is the FP-addition algorithm presented in this paper. Small values of p and n enable exhaustive testing (i.e. input all $2 \cdot 2^{p+n+1}$ binary strings). This exhaustive set of inputs was simulated on both algorithms. Mismatches between the results indicated mistakes in the design. The mismatches were analyzed using assertions specified in the description of the algorithm, and the mistakes were located. Interestingly, most of the mistakes were due to omissions of fill bits in alignment shifts. The algorithm presented in this paper passed this verification without any errors. The algorithm was also extended to deal with denormal inputs and outputs [1, 22].

8 Other FP-adder Implementations

In this section, we are looking at several other FP-adder implementations that are described in literature. In addition to the technical papers [3, 8, 16, 18, 19, 20, 21, 22, 23] there are also several patents [6, 9, 10, 12, 13, 14, 15, 17, 24, 25, 27] that deal with the implementation of an FP-adder. To overview the designs from all of these publications, we summarize the optimization techniques that were used in each of the implementations in Table 1. The entries in this table are ordered from top to bottom corresponding to the year of publication.

The last two entries in this list correspond to our proposed FP-adder implementation, where the bottom-most entry is assumed to use an additional optimization of the alignment shift in the R-Path to be implemented by duplicating the shifter hardware (like also used in [17]) and use one shifter for the case that $\delta > 0$ and the other shifter for the case that $\delta < 0$. On the one hand this optimization has the additional cost of more than a 53-bit shifter, on the other hand it can save one logic level in the latency of our implementation to reduce it to 23 logic levels. Even with this optimization our algorithm is to be partitioned into two pipeline stages with 12 logic levels between latches, although the first stage then only requires 11 logic levels.

Although many of the designs use two paths for the computations, not in every case these two paths really refer to one path with a simplified alignment shift and the other path with a simplified normalization shift and without the need to complement the significand sum like originally suggested by [8]. In some cases the two paths are just used for different rounding cases. In other cases rounding is not dealt with in the two paths at all, but computed in a separate rounding step that is combined for both paths after the sum is normalized. These implementations can be recognized in Table 1 by the fact that they do not pre-compute the possible rounding results and only have to consider one result binade to be rounded.

Among the 'two-path' implementations from literature there are mainly three different path selection conditions:

- The first group uses the 'original' path selection condition from [8] which is only based on the absolute value of the exponent difference. A 'far'-path is then selected for $|\delta| > 1$ and a 'near'-path is selected for $|\delta| \leq 1$. This path selection condition is used by the implementations from [3, 14, 18, 21, 23]. All of them have to consider four different result binades for rounding.

- A second version of the path selection condition is used by [17]. In this case the far path is additionally used for all effective additions. This allows to unconditionally negate the smaller operand in the 'near'-path. Also this implementation has to consider four different result binades for rounding.

- In the implementation of [10] a third version of the path selection condition is used. In this case additionally the cases where only a normalization shift by at most one position to the right or one position to the left are computed in the 'far'-path. In this way, the design could get rid of the rounding in the 'near'-path. Still there are three different result binades to be considered for rounding and normalization in the 'far'-path of this implementation.

Our path selection condition is different from these three and was developed independently from the later two. Its advantages are described in section 4.1. Not only that by our path selection condition no additions and no rounding have to be considered in the 'near' path. We were also able

implementation	two parallel computation paths	# CP adders for significands	only subtraction in one of the two paths	no rounding required in one path	pre-computation of rounding results	one's complement significand negation	parallel approx lead0/1 count	modified adder including round decision	injection-based rounding reduction	unification of rounding cases for add/sub	pre-computation of post-normalization	one's complement exponent difference	split of δ in upper and lower half	two alignment shifters for $\delta \leq 0$ & $\delta < 0$	# binades to consider for rounding	latency (in LL) for double precision
naive design(sec3)	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	>42
Farmwald'87 [9]	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	
INTEL'91 [24]	-	2	-	-	X	X	X	-	-	-	-	-	-	-	3	
Toshiba'91 [12]	-	2	-	-	X	X	-	-	-	-	-	-	-	-	3	
Stanford Rep'91 [21]	X	1	-	-	-	-	-	-	-	-	-	-	-	-	3	
Weitek'92 [15]	X	2	-	-	-	X	-	-	-	-	-	-	-	-	1	
NEC'93 [14]	X	3	-	-	X	-	X	-	-	-	-	-	-	-	4	
Park etal'96 [19]	-	1	-	-	X	X	-	-	-	-	-	-	-	-	3	
Hitachi'97 [27]	-	1	-	-	-	X	X	-	-	-	-	-	-	-	1	
SNAP'97 [18]	X	2	-	-	X	X	X	-	-	-	-	-	-	-	4	>28
Seidel/Even'98 [23]	X	2	-	-	X	X	X	-	X	X	X	X	X	-	3	24
AMD'98 [25]	X	4	-	-	-	-	X	-	-	-	-	-	-	-	1	
IBM'98 [6]	X	2	-	-	-	X	-	-	-	-	-	-	-	-	1	
SUN'98 [10]	X	2	X	X	X	X	X	X	-	-	-	-	-	-	3	28
NEC'99 [13]	-	1	-	-	-	-	X	-	-	-	-	-	-	-	1	
Adelaide'99 [3]	X	2	-	-	X	X	X	-	-	-	-	-	-	-	4	>28
AMD'00 [17]	X	2	X	-	X	X	X	-	-	-	-	-	-	X	4	26
Seidel/Even'00(sec5)	X	2	X	X	X	X	X	-	X	X	X	X	X	-	2	24
Seidel/Even'00*	X	2	X	X	X	X	X	-	X	X	X	-	-	X	2	23

Table 1. Overview of optimization techniques used by different FP-adder implementations.

to reduce the number of binades to 2 that have to be considered for rounding and normalization in the 'far' path. As shown in section 5 there is a very simple implementation for the path selection condition in our design that only requires very few gates to be added in the R-path.

Beside the implementation in 'two paths', the optimization techniques most commonly used in previous designs are: the use of one's complement negation for the significand, the parallel pre-computation of all possible rounding results in an upper and a lower part, and the parallel approximate leading zero count for an early preparation of the normalization shift. Especially for the leading zero approximation there are many different implementations suggested in literature. The main difference of our algorithm for leading zero approximation is that it operates on a borrow-save encoding with Recodings. Its correctness can be proven very elegantly based on bounds of fraction ranges.

We pick two from the implementations in Table 1 and describe them in detail: (a) an implementation based on the 2000 patent [17] from AMD and (b) an implementation based on the 1998 patent [10] from SUN. The union of the optimization techniques used by these two implementations to reduce delay form a superset of the main optimization techniques from the previously published designs. Only our proposed designs add some additional optimization techniques to reduce delay and to simplify the design as pointed out in Table 1. Therefore, it is likely that the AMD design and the SUN design are the fastest implementations that were previously published. For this reason we have chosen to analyze and compare their latency with the latency of our design. Although some of the other designs additionally address other issues like for example [20, 24] try to reduce cost by sharing hardware between the two paths or like [16] demonstrates an implementation following the pipelined-packet forwarding paradigm, these implementations are not optimized for speed and do not belong to the fastest designs. We therefore do not further discuss them in this study.

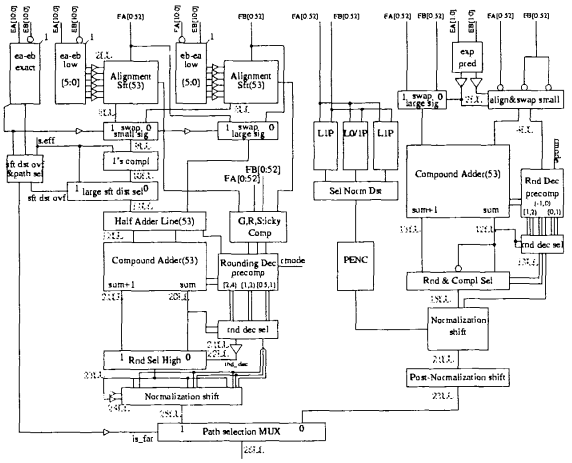

Figure 5. Block diagram of the AMD FP-adder implementation according to [17] adopted to accept double precision operands and to implement all 4 IEEE rounding modes.

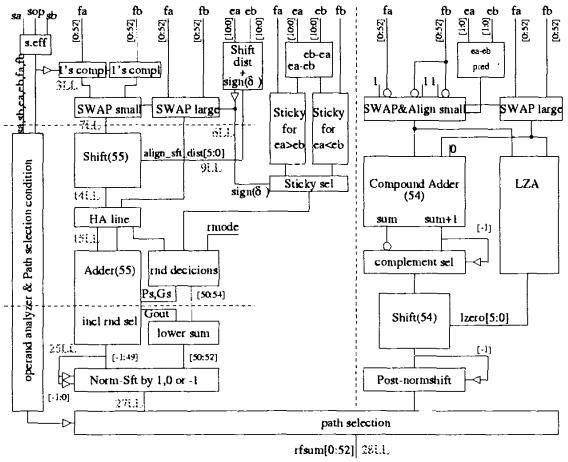

Figure 6. Block diagram of the SUN FP-adder implementation according to [10] adopted to work only on unpacked normalized double precision operands and to implement all 4 IEEE rounding modes.

8.1 FP-adder implementation corresponding to the AMD patent[17]

The patent from [17] describes an implementation of an FP-adder for single precision operands that only considers the rounding mode round to nearest up. To be able to compare this design with our implementation, we had to extend it to double precision and we had to add hardware for the implementation of the 4 IEEE rounding modes. The main changes that were required for the IEEE rounding implementation was the 'large shift distance selection'-mux in the 'far'-path to be able to deal also with exponent differences $|\delta| > 63$. Then, the half adder line in the far path before the compound adder had to be added to be able also to precompute all possible rounding results for rounding mode round-to-infinity. Moreover, some additional logic had to be used for a L-bit fix in the case of a tie in rounding mode round-to-nearest in order to implement the IEEE rounding mode RNE instead of RNU. Figure 5 shows a block diagram of the adopted FP-adder implementation based on [17]. This block diagram is annotated with timing estimates in logic levels. These timing estimates were determined along the same lines like in the delay analysis of our FP-adder implementation. In this way the analysis suggests that the adopted AMD implementation has a delay of 26 logic levels.

One main optimization technique in this design is the use of two parallel alignment shifters at the beginning of the 'far'-path. This technique makes it possible to begin with the alignment shifts very early, so that the first part of the 'far'-path is accelerated. On this basis the block diagram 5 suggests to split the first stage of the 'far'-path after 11 resp. 12 logic levels, leaving 15 resp. 14 logic levels for a second stage. Thus, the design is not very balanced for double precision and it would not be easy to partition the implementation into two clock cycles that contain 13 logic levels between latches.

In the last entry of Table 1 we also considered the technique to use two parallel alignment shifters in our implementation. Because also in this case the first stage of the R-path could be reduced to 11 logic levels, we would get a total latency of 23 logic levels for this optimized version of our implementation.

8.2 FP-adder implementation corresponding to the SUN patent[10]

The patent from [10] describes an implementation of an FP-adder for double precision operands considering all four IEEE rounding modes. This implementation also considers the unpacking of the operands, denormalized numbers, special values and overflows. The implementation targets a partitioning into three pipeline stages. For the comparison with our implementation and the adopted AMD implementation, we reduce the functionality of this implementation also to consider only normalized double precision operands. We get rid of all additional hardware that is only required for the unpacking or the special cases.

Like mentioned above the FP-adder implementation corresponding to this SUN patent uses a special path selection condition that simplifies the 'near'-path by getting rid of

effective additions and of the rounding computations. In this way the implementation of the 'near'-path in this implementation and our N-path implementation are very similar. There are only some differences regarding the implementation of the approximate leading zero count and regarding the possible ranges of the significand sum that have to be considered. Additionally, we employ unconditional pre-shifts for the significands in the N-path that do not require any additional delay.

In the 'far'-path it is the main contribution of this patent to integrate the computation of the rounding decision and the rounding selection into a special CP adder implementation. On the one hand this simplifies to partition this design into three pipeline stages like suggested in the patent, because this modified CP adder design can be easily split in the middle. In [10], the delay of the modified CP adder implementation is estimated to be the delay of a conventional CP adder plus one additional logic level. The implementation of the path-selection condition seems to be more complicated than in other design and is depicted in [10] by two large boxes to analyze the operands in both paths.

Figure 6 depicts a block diagram of this adopted design. This figure is annotated with timing estimates. For this estimate we assume the modified CP adder to have a delay of 10 logic levels as discussed above. In this way our delay analysis suggests, that the adopted FP-adder implementation corresponding to the SUN patent has a delay of 28 logic levels. In this case the implementation of the first stage is not very fast and requires 14 logic levels.

Thus, in comparison with our design, the FP-adder implementations corresponding to the AMD patent and corresponding to the SUN patent both seem to be slower by at least two logic levels. Additionally, they have a more complicated IEEE rounding implementation and can not as easy be partitioned into two balanced stages as our design. Because the two implementations were chosen to be the fastest from literature, our implementations seem to be the fastest FP-adder implementations published to date.

References

[1] S. Bar-Or, Y. Levin, and G. Even. On the delay overheads of the supporting denormal inputs and outputs in floating point adders and multipliers. in preparation.

[2] S. Bar-Or, Y. Levin, and G. Even. Verification of scalable algorithms: case study of an IEEE floating point addition algorithm. in preparation.

[3] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim. Reduced latency IEEE floating-point standard adder architectures. *Proc. 14th Symp. on Computer Arithmetic*, 14, 1999.

[4] R. Brent and H. Kung. A Regular Layout for Parallel Adders. *IEEE Trans. on Computers*, C-31(3):260–264, Mar. 1982.

[5] M. Daumas and D. Matula. Recoders for partial compression and rounding. Technical Report RR97-01, Ecole Normale Superieure de Lyon, LIP, 1996.

[6] L. Eisen, T. Elliott, R. Golla, and C. Olson. Method and system for performing a high speed floating point add operation. IBM Corporation, U.S. patent 5790445, 1998.

[7] G. Even and P.-M. Seidel. A comparison of three rounding algorithms for IEEE floating-point multiplication. *IEEE Transactions on Computers, Special Issue on Computer Arithmetic*, pages 638–650, July 2000.

[8] P. Farmwald. *On the design of high performance digital arithmetic units*. PhD thesis, Stanford Univ., Aug. 1981.

[9] P. Farmwald. Bifurcated method and apparatus for floating-point addition with decreased latency time. U.S. patent 4639887, 1987.

[10] V. Gorshtein, A. Grushin, and S. Shevtsov. Floating point addition methods and apparatus. Sun Microsystems, U.S.patent 5808926, 1998.

[11] IEEE standard for binary floating point arithmetic. ANSI/IEEE754-1985.

[12] T. Ishikawa. Method for adding/subtracting floating-point representation data and apparatus for the same. Toshiba,K.K., U.S. patent 5063530, 1991.

[13] T. Kawaguchi. Floating point addition and subtraction arithmetic circuit performing preprocessing of addition or subtraction operation rapidly. NEC, U.S. patent 5931896, 1999.

[14] T. Nakayama. Hardware arrangement for floating-point addition and subtraction. NEC, U.S. patent 5197023, 1993.

[15] K. Ng. Floating-point ALU with parallel paths. Weitek Corporation, U.S. patent 5136536, 1992.

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

[17] S. Oberman. Floating-point arithmetic unit including an efficient close data path. AMD, U.S. patent 6094668, 2000.

[18] S. Oberman, H. Al-Twaijry, and M. Flynn. The SNAP project: Design of floating point arithmetic units. In *Proc. 13th IEEE Symp. on Comp. Arith.*, pages 156–165, 1997.

[19] W.-C. Park, T.-D. Han, S.-D. Kim, and S.-B. Yang. Floating Point Adder/Subtractor Performing IEEE Rounding and Addition/Subtraction in Parallel. *IEICE Transactions on Information and Systems*, E79-D(4):297–305, 1996.

[20] N. Quach and M. Flynn. Design and implementation of the SNAP floating-point adder. Technical Report CSL-TR-91-501, Stanford University, Dec. 1991.

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

[22] P.-M. Seidel. *On The Design of IEEE Compliant Floating-Point Units and Their Quantitative Analysis*. PhD thesis, University of the Saarland, Germany, December 1999.

[23] P.-M. Seidel and G. Even. How many logic levels does floating-point addition require? In *Proceedings of the 1998 International Conference on Computer Design (ICCD'98): VLSI in Computers & Processors*, pages 142–149, Oct. 1998.

[24] H. Sit, D. Galbi, and A. Chan. Circuit for adding/subtracting two floating-point operands. Intel, U.S.patent 5027308, 1991.

[25] D. Stiles. Method and apparatus for performing floating-point addition. AMD, U.S. patent 5764556, 1998.

[26] A. Tyagi. A Reduced-Area Scheme for Carry-Select Adders. *IEEE Transactions on Computers*, C-42(10), October 1993.

[27] H. Yamada, F. Murabayashi, T. Yamauchi, T. Hotta, H. Sawamoto, T. Nishiyama, Y. Kiyoshige, and N. Ido. Floating-point addition/subtraction processing apparatus and method thereof. Hitachi, U.S. patent 5684729, 1997.