A Quadruple Precision and Dual Double Precision Floating-Point Multiplier

Ahmet Akkaş
Computer Engineering Department
Koç University
34450 Sarıyer, İstanbul, Turkey
ahakkas@ku.edu.tr

Michael J. Schulte
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706, USA
schulte@engr.wisc.edu

Abstract

Double precision floating-point arithmetic is inadequate for many scientific computations. This paper presents the design of a quadruple precision floating-point multiplier that also supports two parallel double precision multiplications. Since hardware support for quadruple precision arithmetic is expensive, a new technique is presented that requires much less hardware than a fully parallel quadruple precision multiplier. With this implementation, quadruple precision multiplication has a latency of three cycles and two parallel double precision multiplications have a latency of only two cycles. The design is pipelined so that two double precision multiplications can be started every cycle or a quadruple precision multiplication can be started every other cycle.

Key words*: Quadruple precision, double precision, multiplier, floating-point, computer arithmetic, rounding, normalization.*

1. Introduction

Floating-point computations suffer from errors due to rounding and quantization. Fast computers let programmers write numerically intensive programs, but computed results can be far from the true results due to the accumulation of errors in arithmetic operations [2]. Quadruple precision arithmetic increases the accuracy and reliability of numerical computations by providing floating-point numbers that have more than twice the precision of double precision numbers. This is important in applications, such as computational fluid dynamics and physical modeling, which require accurate numerical computations.

Most modern processors have hardware support for double precision (64-bit) or double-extended precision (typically 80-bit) floating-point multiplication, but not for quadruple precision (128-bit) floating-point multiplication. It is also true, however, that double precision and double-extended precision are not enough for many scientific applications including climate modeling [6], computational physics [7], and computational geometry [7]. The use of quadruple precision arithmetic can greatly improve the numerical stability and reproducibility of many of these applications [6]. Due to the advantages quadruple precision arithmetic can provide in scientific computing applications, specifications for quadruple precision numbers are being added to the revised version of the IEEE 754 Standard for Floating-Point Arithmetic [9].

Since hardware support for quadruple precision arithmetic is expensive, software packages have been developed to support quadruple precision arithmetic. This approach, however, has performance problems for numerically intensive applications. For example, simulation results using the Cephes software package on a high-performance superscalar processor, indicate that quadruple precision multiplication implemented in software is more than 1,000 times slower than double precision multiplication in hardware.

As illustrated in [12], it is possible to have quadruple precision hardware support using a reasonable amount of hardware compared to double precision. The S/390 G5 floating point unit described in [12] implements quadruple precision multiplication in 27 cycles using a 56-bit by 56-bit significand multiplier, a 120-bit adder, microcode control, and additional logic. One double precision multiplication on the S/390 G5 processor has a latency of three cycles.

A current trend in modern microprocessors is to provide multiple identical functional units to speed up numerical computations [13]. For example, high-performance processors often have two double precision multipliers, which can operate in parallel. Another trend is to have wide 128-bit internal datapaths [13], [14], which can support quadruple precision operands.

The multiplier presented in this paper uses two double precision multipliers plus additional hardware to perform one quadruple precision multiplication or two parallel double precision multiplications. This approach requires much

Proceedings of the Euromicro Symposium on Digital System Design (DSD'03)
0-7695-2003-0/03 $17.00 © 2003 IEEE

less hardware than a fully parallel quadruple precision multiplier. It is faster, but requires more hardware, than the technique used on the S/390 G5 processor. With the design presented in this paper, quadruple precision multiplication has a latency of three cycles and a maximum throughput of one result every other cycle. Double precision multiplication has a latency of two cycles and a maximum throughput of two results per cycle. The multiplier presented in this paper is designed to be incorporated as part of future high-performance processors.

The remainder of this paper is organized as follows: Section 2 gives an overview of our technique for quadruple precision multiplication. Section 3 presents the design of our quadruple precision multiplier. Section 4 gives area and delay estimates for our quadruple precision multiplier and for a double precision multiplier. Section 5 presents our conclusions.

2. Technique for Quadruple Precision Multiplication

The multiplier presented in this paper multiplies IEEE-754 double precision numbers [8] and quadruple precision numbers in the format specified in the draft of the revised IEEE Standard for Floating-Point Arithmetic [9]. An IEEE double precision floating-point number consists of three fields: a 1-bit sign, *s*, an 11-bit biased exponent, *e*, and a 52-bit significand, *f*. These fields are shown in Figure 1. A quadruple precision number consists of a 1-bit sign, a 15-bit biased exponent, and a 112-bit significand [9]. The quadruple precision number format is shown in Figure 2. A hidden-one is used to provide 53 bits of precision in normalized double precision significands, and 113 bits of precision in normalized quadruple precision significands.

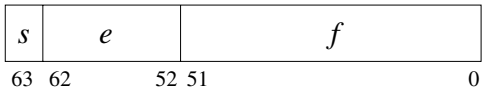

Figure 1. Double Precision Format.

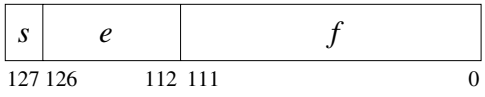

Figure 2. Quadruple Precision Format.

In our design, each 128-bit quadruple precision number is stored in two 64-bit double precision registers. Figure 3 shows how the sign, exponent, and significand bits of a quadruple precision number are stored in two 64-bit registers, R1 and R2.

The significand bits of each quadruple precision number are divided into two 57-bit parts; high and low. If X and

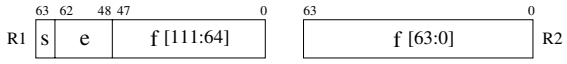

Figure 3. A Quadruple Precision Number Stored in Two 64-Bit Registers.

Y are two quadruple precision numbers stored in the 128-bit R1-R2 register pair and the 128-bit R3-R4 register pair, respectively, then the extractions of the significand bits of these two numbers are:

X_high = ['1', R1(47:0), R2(63:56)]

X_low = [R2(55:0), '0']

Y_high = ['1', R3(47:0), R4(63:56)]

Y_low = [R4(55:0), '0']

where $[a, b]$ denotes a and b are concatenated. The high parts, X_high and Y_high, are each extended with a leading hidden-one. The low parts, X_low and Y_low, are each extended with a trailing zero. The trailing zero is added to make the high and low parts the same size, which simplifies the hardware design.

The basic technique used for the quadruple precision significand multiplication is shown in Figure 4. In the first cycle of quadruple precision multiplication, X_low is multiplied by Y_low and Y_high using two parallel 57-bit by 57-bit multipliers that produce results in carry-save format. The overlapping portions of the two products are combined using 4-to-2 compressors, which keep the result in carry-save format. In the second cycle, the intermediate results computed in the first cycle, except for the 57 least significant bits, are fed back as inputs to a multiplier that adds these bits to X_high times Y_low. In parallel with this, the second multiplier computes X_high times Y_high. The overlapping portions of the two carry-save products are then compressed using the 4-to-2 compressors. In the third cycle, the carry-save product is added together, normalized, and rounded.

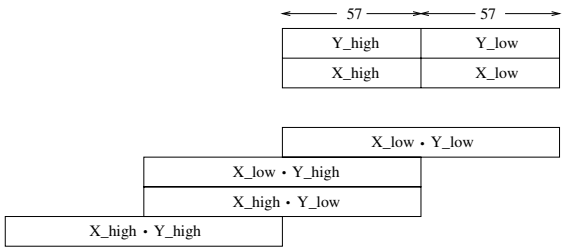

Figure 4. Quadruple Precision Multiplication.

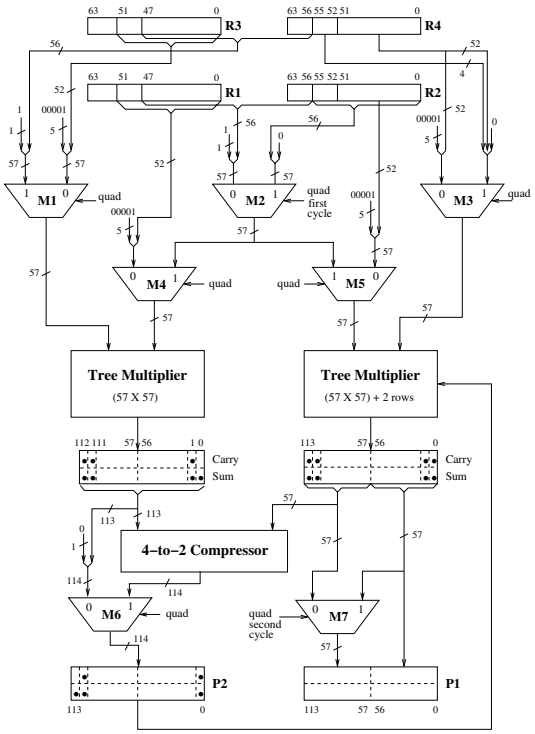

Figure 5. Quadruple Precision Multiplier: Significand Multiplication.

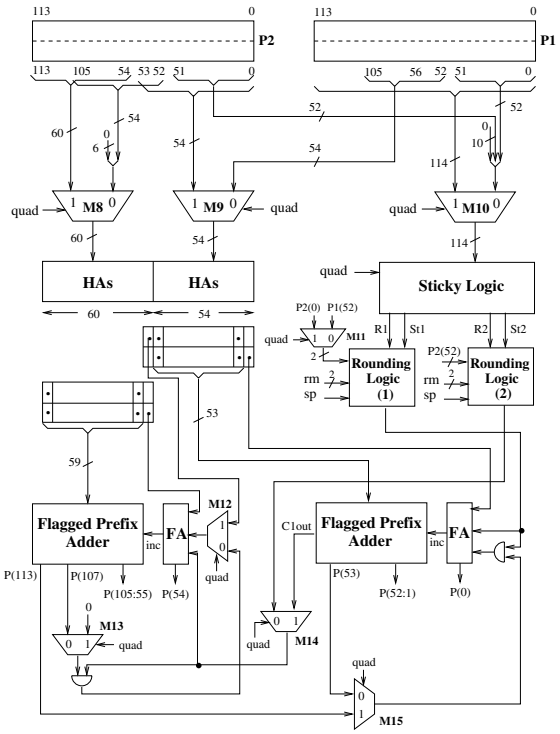

Figure 6. Quadruple Precision Multiplier: Addition and Rounding.

3. Quadruple Precision Multiplier

Our hardware design, which provides support for one quadruple precision multiplication or two parallel double precision multiplications, is shown in Figures 5 and 6. To simplify these figures, exponent and sign computations, exception and special case handling logic, and the registers that store the final product are not shown.

The idea of performing one quadruple precision multiplication or two parallel double precision multiplications is similar to extending a double precision floating-point unit to support two single precision operations in parallel [10]. To limit the hardware requirements, however, the significand multiplier portion of our design is used twice to perform a quadruple precision multiplication. This approach uses approximately half as much hardware and has less delay than a quadruple precision multiplier that is implemented with a full 113-bit by 113-bit multiplier tree. It is similar to the approach used in [5] to perform one double precision multiplication or two parallel single precision multiplications, where each type of multiplication takes three clock cycles and a new multiplication can be started every other clock cycle.

When the multiplier shown in Figures 5 and 6 is used to implement quadruple precision multiplication, registers R1 and R2 store the first quadruple precision input operand, and registers R3 and R4 store the second quadruple precision input operand. The first five multiplexors, M1 through M5, are used to extract the significand bits of the input operands based on their precision and the operation cycle. All multiplexors in this design are controlled by three signals; *quad*, *quad first cycle*, and *quad second cycle*. For quadruple precision multiplication, the *quad* signal is set. In the first and second cycles of quadruple multiplication, the *quad first cycle* and the *quad second cycle* signals are also set, respectively. The *quad*, *quad first cycle*, and *quad second cycle* signals are set to zero when performing double precision multiplication.

In the first cycle of quadruple precision multiplication, X_low is multiplied by Y_low and Y_high. In Figure 5, the tree multiplier on the right multiplies X_low by Y_low and the tree multiplier on the left multiplies X_low by Y_high. The tree multiplier on the right has two additional rows, since part of the first cycle result needs to be fed back to be used in the second cycle. In the first cycle, the P2 register is reset to clear these additional rows. The results of the tree multipliers are kept in carry-save format to avoid carry-propagate addition, which would increase the delay. The upper part of the tree multiplier result on the right (X_low · Y_low) is combined with the lower part of the tree multi-

plier result on the left (X_low · Y_high) using 4-to-2 compressors. The M6 multiplexor selects the correct significand bits to go to P2 based on the precision of the multiplication. At the end of the first cycle, the partial products in carry-save format are stored into the lower part of P1 and all of P2.

In the second cycle of quadruple precision multiplication, the values in P2 are fed back into the tree multiplier on the right as two additional rows. These additional rows do not increase the number of stages of the tree multiplier, which uses nine stages of (3,2) and (2,2) counters [1]. In the second cycle, X_high is multiplied by Y_low and Y_high. The tree multiplier on the right multiplies X_high by Y_low, and the tree multiplier on the left multiplies X_high by Y_high. The lower part of the tree multiplier result on the right goes through M7 to the upper part of P1. The lower part of P1, computed in the first cycle, keeps its first cycle content in the second cycle. The upper part of the tree multiplier result on the right and the lower part of the tree multiplier result on the left are combined using the 4-to-2 compressor and the compressor output is stored in P2.

In the third cycle, the final addition, rounding, and normalization are performed. As shown in Figure 6, multiplexors M8 and M9 select the most significand parts of the product in carry-save format. This corresponds to the quadruple precision result before it is added, rounded and normalized. The outputs of multiplexors M8 and M9 are inputs to a row of 114 half adders (HAs). The half adders are used to create an empty slot(s) in the least significand bit(s), so that there are two entries available in the least significant bit position(s), as discussed in [11].

Multiplexor M10 selects the least significant parts of the product to determine the carry, round, and sticky bits [11]. In this design, there are two sets of Rounding Logic. Only Rounding Logic (1) is used for quadruple precision multiplication, and both sets are used for parallel double precision multiplications. The Sticky Logic computes the round bit (R1) and sticky bit (St1) using the 114 least significant product bits, and sends them to Rounding Logic (1). Rounding Logic (1) takes these bits, the 2-bit rounding mode, *rm*, the sign of the product, *sp*, and guard bit, *P2(0)*, and determines if a one needs to be added for correct rounding. If it does, then the output of Round Logic (1) is set. This output is one of the inputs to the full adder (FA) on the right.

Flagged prefix adders are used for the final addition, because they can produce both $A+B$ and $A+B+1$ [3]. This is accomplished by computing n flag bits ($F = f_{n-1}...f_1 f_0$). The i^{th} flag bit, f_i, is defined as:

$$f_i = (a_{i-1} \oplus b_{i-1}) \cdot (a_{i-2} \oplus b_{i-2}) \cdot ... \cdot (a_0 \oplus b_0)$$
$$= (a_{i-1} \oplus b_{i-1}) \cdot f_{i-1} \text{ with } f_0 = 1$$

Thus, f_i can be computed using parallel prefix methods [3]. Table 1 gives an example that shows how $A+B+1$ is obtained from $A+B$ using the flag bits. A high-speed parallel prefix adder is used to compute $S = A+B$. In parallel with this addition, the flag bits, F, are computed. To obtain $R = A+B+1$, the bits of S and F are exclusive-ORed, such that $r_i = s_i \oplus f_i$. A multiplexor is used to select R when a one is added to $A+B$ and S when no one is added, as shown in Figure 7. In the actual design, the adder and F-logic share hardware to reduce the overall area.

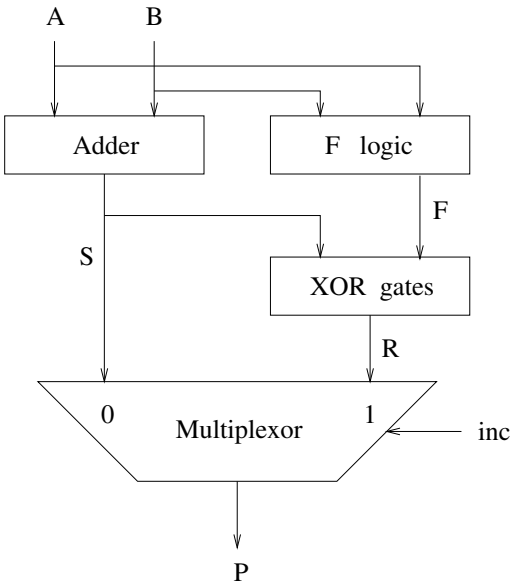

Figure 7. Flagged Prefix Adder.

00001001	$A = a_7 a_6 ... a_0$
01001110	$B = b_7 b_6 ... b_0$
01010111	$S = A + B$
00001111	$F = f_7 f_6 ... f_0$
01011000	$R = A + B + 1 = S \oplus F$

Table 1. Obtaining $A+B+1$ from $A+B$.

Details of the flagged prefix adder are presented in [3]. As explained in [4], this adder provides an efficient rounding implementation for double precision multiplication. The main difference between the design given in [4] and the design given here is that one of the FA inputs comes from the least significant bit of the HA. The second FA input comes from the Rounding Logic. The output of the Rounding Logic is set, if a one needs to be added to the least significant bit of the truncated product.

When the result of the significand addition is greater than or equal to two, a one is added to the second least significant bit, because there is a 1-bit normalizing right shift [11]. To accomplish this, the most significant product bit, P(113), is checked when performing quadruple precision multiplication. If it is one and the output of Rounding Logic (1) is set,

a second one is added to the least significant bit to produce the correct result. Adding two ones in the least significant bit is the same as adding one to the second least significant bit. Hence, the third input to the FA is set if the output of the Rounding Logic (1) is one and the pre-normalized result is at least two.

The carry outputs of the FAs, *inc*, in Figure 6 are used as selection bits for the multiplexors used in the flagged prefix adders, shown in Figure 7. The sum outputs of the FAs are P(0) and P(54). The product is the output of the FAs and flagged prefix adders. A 1-bit normalizing right shift, which is needed when the result is at least two, is not shown here. The use of flagged prefix adders allows the 114-bit addition, which is needed for quadruple precision multiplication, to be divided into a 60-bit addition and a 54-bit addition that operate in parallel to reduce the overall delay.

When this unit performs two double precision multiplications in parallel, the first pair of input operands goes to registers R1 and R3, and the second pair of input operands goes to registers R2 and R4. In the first cycle, multiplexors M1 through M5 are used to choose the correct significand bits for the tree multipliers. The tree multiplier on the right multiplies the values in R2 and R4, while the tree multiplier on the left multiplies the values in R1 and R3. Since the IEEE-754 double precision format is used, the 52-bit significands of the input operands are extended to 57 bits with a hidden-one and four leading zeros. The outputs from the tree multipliers go into P1 and P2 in carry-save format, without any change. The 4-to-2 compressors are not used for double precision multiplication.

In the second cycle of double precision multiplication, the lower 52 bits of P1 and P2 are chosen by the M10 multiplexor and sent to the Sticky Logic. The Sticky Logic provides round and sticky bits to both Rounding Logic units. Rounding Logic (1) and Rounding Logic (2) are used to the compute the rounding bits for the results in P1 and P2, respectively. The pre-normalized result is available from P(53:0) for the multiplication of the double precision numbers in R2 and R4 and the other pre-normalized result is available from P(107:54) for the multiplication of the double precision numbers in R1 and R3.

4 Area and Delay Estimates

For comparison purposes, we also designed a double precision multiplier, shown in Figure 8. This multiplier uses the same performance-enhancement techniques as the quadruple precision multiplier, which include the high-speed significand tree multiplier and the flagged prefix adder. The multiplier is pipelined for a latency of two cycles and a maximum throughput of one result per cycle. Thus, two of these double precision multipliers operating in parallel have the same throughput per cycle as our quadruple precision multiplier when performing double precision multiplication, but they cannot directly perform quadruple precision multiplication.

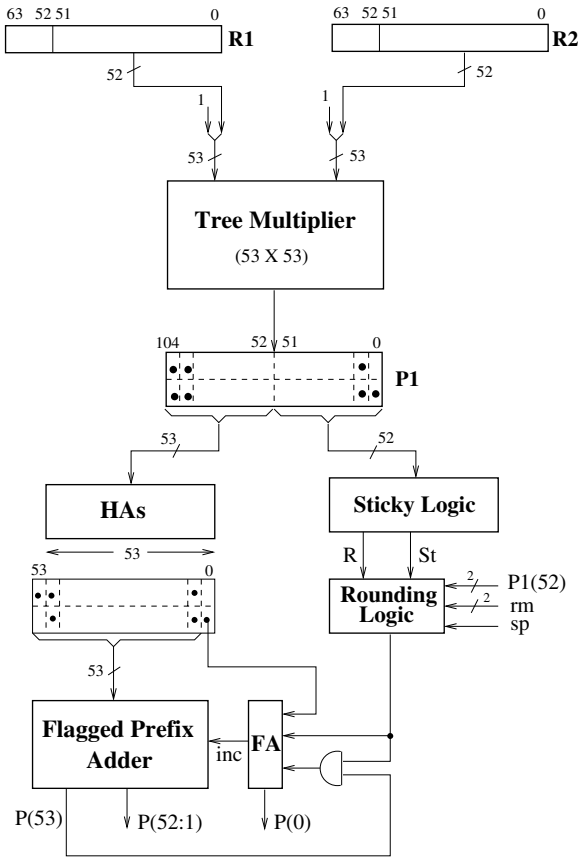

Figure 8. Double Precision Multiplier.

Compared to two double precision multipliers, our quadruple precision multiplier requires an additional seven 57-bit multiplexors, six 1-bit multiplexors, two 60-bit multiplexors, two 54-bit multiplexors, four 114-bit multiplexors, one 57-bit 4-to-2 compressor, and two additional rows in the significand tree multiplier. The multiplexors M6, M7, M8, M9, M10, and M11 are each counted as two multiplexors, since they select values that are in carry-save format. The amount of hardware used for the multiplier trees, sticky logic, registers, half adders and parallel prefix adders also increases slightly to handle 57-bit values, instead of 53-bit values.

Structural level models of the double precision multiplier and quadruple precision multiplier were written in VHDL. The designs for both multipliers were synthesized using Mentor Graphics' LeonardoSpectrum synthesis tool and the TSMC 0.25 micron CMOS standard cell library. The designs were optimized for speed with a target frequency of 250 MHz. Tables 2 and 3 give area and worst case delay estimates for both multipliers based on our synthesis results,

Pipeline Stage	Multiplier Precision	
	Quadruple	Double
Significand Multiply	131532	63934
Add and Round	15034	6118
Total Area	146566	70052

Table 2. Multiplier Area Estimates (Gates).

which use an operating voltage of 2.5 Volts and a temperature of 25 degrees centigrade. In Table 2, area is given in terms of equivalent gates and results are normalized, such that an equivalent gate corresponds to the area of a single minimum-size inverter. Area and worst delay estimates are given for each pipeline stage, along with the total area and overall worst case delay. For both designs, the two pipeline stages are fairly well balanced, but the delay of the significant multiplication stage is slightly greater than the delay of the addition and rounding stage.

Pipeline Stage	Multiplier Precision	
	Quadruple	Double
Significant Multiply	6.11	4.68
Add and Round	5.94	4.37
Overall Delay	6.11	4.68

Table 3. Multiplier Delay Estimates (ns.).

Compared to an implementation that uses two parallel double precision multipliers, our quadruple precision multiplier has roughly 5% more area and a worst case delay that is roughly 30% longer. Due to the fairly large increase in worst case delay, our quadruple precision multiplier is well suited for implementations in which the delay of the floating point multiplier is not likely to limit the clock cycle time (e.g., in systems where the delay of the cache is significantly longer than the delay of the multiplier). To achieve higher clock frequencies, our quadruple precision multiplier could be more deeply pipelined to use two stages for significand multiplication and two stages for addition and rounding.

5 Conclusions

The quadruple precision multiplier presented in this paper takes three cycles to perform quadruple precision multiplication and can produce a quadruple precision product every other cycle. It takes two cycles to perform two parallel double precision multiplications and can produce two double precision products every cycle. Compared to two double precision multipliers operating in parallel, the design presented in this paper has 5% more area and 30% more delay, but provides the ability to quickly perform quadruple precision multiplication.

6 Acknowledgments

This material is based upon work supported by the Scientific and Technical Research Council of Turkey (TÜBİTAK) under the project number 102E002 and by the National Science Foundation under Grant No. CCR-9703421.

References

[1] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr. Parallel Reduced Area Multiplier. *Journal of VLSI Signal Processing*, 9:181–191, 1995.

[2] G. Bohlender. What Do We Need Beyond IEEE Arithmetic? In C. Ullrich, editor, *Computer Arithmetic and Self-Validating Numerical Methods*, pages 1–32. Academic Press, 1990.

[3] N. Burgess. The Flagged Prefix Adder for Dual Addition. In *Proceedings of the SPIE Conference: Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*, pages 567–575, July 1998.

[4] N. Burgess and S. Knowles. Efficient Implementation of Rounding Units. In *Proceedings of the 1999 Asilomar Conference on Signals, Systems, and Computers*, pages 1489–1493, October 1999.

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

[6] Y. He and C. Ding. Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications. *Journal of Supercomputing*, 18:259–277, 2001.

[7] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for Quad-Double Precision Floating Point Arithmetic. In *Proceedings of 15th IEEE Symposium on Computer Arithmetic*, pages 155–162, 2001.

[8] *ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic*, 1985.

[9] *DRAFT IEEE Standard for Binary Floating-Point Arithmetic*, 2003. Available from: http://754r.ucbtest.org/.

[10] R. Kolla, A. Vodopivec, and J. Wolff von Gudenberg. The IAX Architecture : Interval Arithmetic Extension. Technical Report 225, Universitat Wurzburg, 1999.

[11] M. R. Santoro, G. Bewick, and M. A. Horowitz. Rounding Algorithms for IEEE Multipliers. *Proceedings of 9th Symposium on Computer Arithmetic*, pages 176–183, 1989.

[12] E. M. Schwarz, R. M. Smith, and C. A. Krygowski. The S/390 G5 Floating Point Unit Supporting Hex and Binary Architecture. In *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, pages 258–265, 1999.

[13] M. Suzuoki et al. A Microprocessor with a 128-bit CPU, Ten Floating-point MAC's, Four Floating-point Dividers, and an MPEG-2 Decoder. *IEEE Journal of Solid-State Circuits*, 34(11):1608–1618, November 1999.

[14] J. Tyler, J. Lent, A. Mather, and H. Nguyen. AltiVec: Bringing Vector Technology to the PowerPC Processor Family. In *IEEE International Performance, Computing and Communications Conference*, pages 437–444, November 1999.