Second-Generation RISC Floating Point with Multiply–Add Fused

ERDEM HOKENEK, MEMBER, IEEE, ROBERT K. MONTOYE, MEMBER, IEEE, AND PETER W. COOK, MEMBER, IEEE

Abstract—**A 440 000-transistor second-generation RISC floating-point chip is described. The pipeline latency is only two cycles, and a double-precision result is produced every cycle. System throughput and accuracy is increased by using a floating-point multiply–add-fused (MAF) unit, which carries out a double-precision accumulate $D=(A\times B)+C$ as a two-cycle pipelined execution with only one rounding error. While the cycle time (40 ns) is competitive with other CMOS RISC systems, the floating-point performance stretches to the range of bipolar RISC systems (7.4–13 MFLOPS LINPACK).**

I. Introduction

WITH the second-generation RISC floating-point unit (FPU), the essential RISC concept of attacking the most frequently used functions by building simple self-contained low-latency hardware was extended to floating-point operations [1]–[3]. It greatly increases the floating-point performance and accuracy by including as the key feature of the FPU a unified floating-point multiply–add-fused (MAF) unit, which executes the double-precision accumulate instruction $D=(A\times B)+C$ as an indivisible operation, with no intermediate rounding [1], [4]. By *fusing* multiplication with addition, the MAF building block allows one-cycle throughput and two-cycle latency, producing a floating-point accumulate with one rounding error. This single functional unit requiring an instruction set which others only emulate also reduces the hardware overhead associated with adders/normalizers by combining various operations necessary for fast multiplication with accumulation. In other words, the MAF primitive provides major improvements for the floating-point executions, provided that the unit can be designed without any substantial impact on the cost, cycle time, or other VLSI requirements.

This paper describes the MAF data flow implemented using high-speed CMOS circuits. It attains a peak double-precision execution of one accumulate instruction per cycle. The floating-point performance approaches that of bipolar RISC systems, and is achieved with a cycle time comparable to CMOS RISC systems.

The next section discusses the MAF concept, explaining in some detail why it is an appropriate addition to the floating-point architecture in VLSI. Section III describes the interaction of logical and physical design required to incorporate several advances in VLSI arithmetic while accommodating various delay and technological (physical) constraints. The chip data are presented in Section IV followed by the conclusions in the final section.

Manuscript received April 11, 1990; revised June 4, 1990.
The authors are with the Thomas J. Watson Research Center, IBM Research Division, Yorktown Heights, NY 10598.
IEEE Log Number 903770.

II. The Multiply–Add-Fused (MAF) Concept

A short introduction to the MAF concept will be given in this section. The MAF implementation should be consistent with the basic RISC philosophy [3] of heavily optimizing units in order to rapidly carry out the most frequently expected functions as fast as possible. In a significant majority of the SPEC floating-point benchmarks, most of the floating-point multiply and add operations were subsumed by the MAF, i.e., concatenating multiply and add. Therefore a single self-contained unit which forms the multiply–accumulate operation $D=(A\times B)+C$ would produce a considerable enhancement in the floating-point performance. The individual operations necessary would be:

1) a parallel effort of the true mantissa and exponent calculations for multiplication;
2) a prenormalization stage for the bit alignment of the values to be added;
3) addition, post-normalization followed by rounding is needed for all calculations.

Overlapping the data alignment with the early phases of multiplication would be the first step since the partial-product compression is a time-consuming operation involving multiple stages of summation and a final addition. This approach leads to a first cycle of the pipeline shared by the multiplication, the exponent calculation, and product alignment, i.e., shifting the addend in either direction.

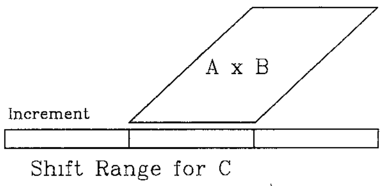

Fig. 1. Shift range with multiply–add.

The penalty to be paid for combining these operations, however, is an increment of the mantissa length resulting in a 1/2 increase of the add/normalize range compared to a conventional multiplier/adder pair (see Fig. 1).

Hiding the normalization delay completely and making the add/normalize path comparable in time to the multiplication/shifter path requires some novel techniques. A leading-zero anticipator was added to the second pipeline in order to predict the shift amount for post-normalization before the addition is finished [5].

It should be pointed out that the two-stage pipeline MAF unit would be feasible only under the conditions of building a very fast shifter, which eases overlap of the multiplication and prenormalization, and a leading zero anticipator (LZA), which allows post-normalization and addition in the same cycle by running in parallel with a carry lookahead adder incrementer. The resulting pipeline structure is shown in Fig. 2.

The architectural trade-offs made during MAF design were focused on keeping the balance between overall performance and circuits put into a chip. That was achieved by eliminating some excess hardware needs from the conventional designs, in particular I/O ports, buses, MUX's, etc., and replacing them, if necessary, by the new circuitry (like LZA) which improves system throughput and decreases control complexity.

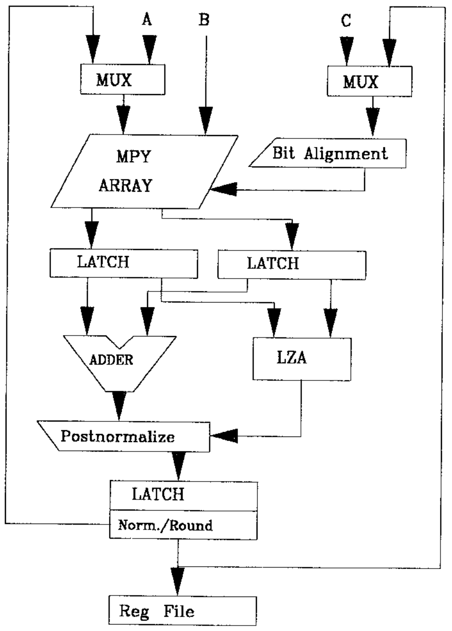

Fig. 2. Schematic of MAF pipeline organization.

III. MAF Data-Flow Implementation

As described in the preceding section, the two-cycle MAF concept is a very ambitious project. Since various operations of the dot product share a common time slot the desired logical realization of the MAF subsections and their interaction need special attention. Therefore, this section is devoted to a detailed description of the logical and physical implementation for the critical segments of this activity. We will discuss various design aspects in detail for the separate subunits of the data flow:

- multiplier,
- shifters,
- adder,
- leading 0/1 anticipator.

Having the logical design of the MAF data flow performed in parallel with the physical implementation has been the primary factor for the success, namely producing a well-designed very efficient custom chip. For all crucial trade-offs made during the data-flow implementation, particular emphasis is on two critical aspects: 1) throughput and 2) area occupied using the two levels of metal available at the time. A key objective in VLSI is to achieve an efficient design within the space allowed by the top layer of wiring for the circuit [6]. For simplicity and uniformity, a tool/design discipline called macro layout generator (MLG) was developed and used throughout this effort [7]. MLG is a grid-based symbolic design tool that conveniently aligns available metal layers with the gates, including drain/source and gate contacts, in order to accommodate the capabilities of the technology, i.e., one metal 2 wire for every contacted gate, and the gates are parallel to metal 2, allowing easy contact to metal 1. An image file for MLG defines all restrictions in terms of integration layers shared by various design parts.

Since the largest share of the MAF circuitry is in the multiplier, we will begin our discussion with that unit.

A. Multiplier

The multiply array utilizes two basic techniques: modified Booth encoding and extended Wallace (carry-save) tree [8], [9]. The major extension from the literature of the carry save or (3,2) addition is that of (7,3) counting (seven input bits reduced to their 3-b binary sum) [1], [2]. In order to be efficient, the number of significantly loaded stages must be kept to a minimum since the CMOS technology used has substantial delay because of the long wiring that occurs in a Wallace tree implementation. This is due to the high sensitivity of the CMOS technology to capacitive loading. A reduction in stages with long wires yields a considerable improvement in performance. The seven-input three-output cell built using carry-save adders

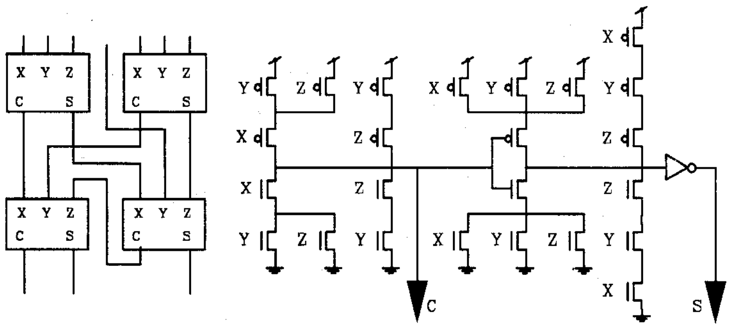

Fig. 3. Implementation of carry-save adder and (7/3) counter.

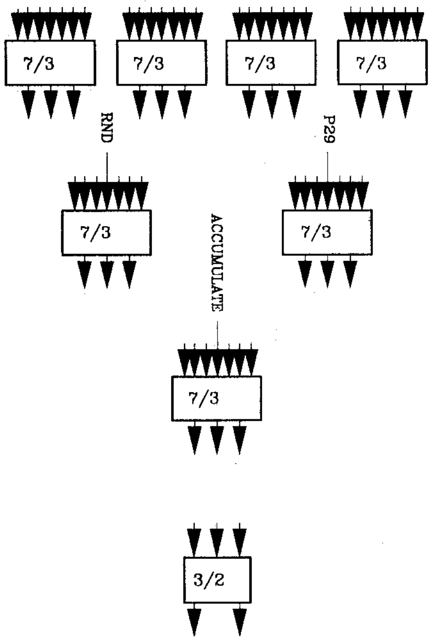

Fig. 4. Partial-product array organization with (7/3) counters.

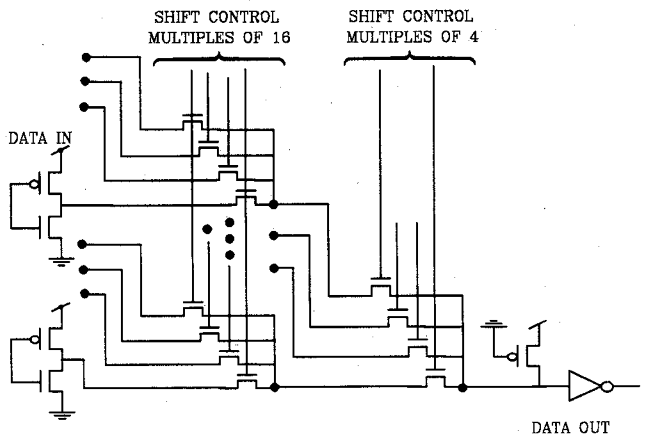

Fig. 5. A typical shifter stage.

depicted in Fig. 3 is more complicated and slow. With a careful physical design, however, the number of long wires that must be driven is reduced by a factor of 2.5, resulting in a timing-efficient design.

Another important factor in VLSI design is connectivity defining both cost and performance. The (7,3) adder eliminating 4 b from the partial-product compression requires only half as many connections as the (3,2) adder to produce a final result. Also, this cell organization reduces by a factor of 1.6 the number of stages needed, since each stage removes a factor of (7,3) in comparison to (3,2) from the number of remaining terms to be summed. In other words, the (7,3) counters enable us to optimize the wire lengths by shifting the placement and wiring demands onto a coarser grain.

It should be pointed out that our major objective was the design of 56-b multiplier (56 b to allow the construction of either a 53-b (IEEE) or 56-b HEX double-precision unit). Another important leverage offered by 7/3 counters is that the 56-b organization shown in Fig. 4 provides a free pin for an extra operand. Hence the addend can be summed during the partial-product compression without sacrificing area or cycle time.

The first stage consisted of a cell with seven 4-input multiplexors (for the Booth method), requiring 28 control and data inputs, all using polysilicon, which fit into the 36-track-wide cell. Thus, a combination of Booth encoding and (7,3) compression produces a very efficient multiplier, but it still must be implemented in CMOS VLSI. The wiring requirements for 7/3 counters were shown to occupy a width of 18 transistor locations or metal 2 (M2) wires per cell. The most efficient cell organization produced a mirrored pair of cells with twice that width, which shared a common control bay.

B. Shifters

Parallel execution of various fundamental operations of dot product is based on the assumption that wide-range shifting/rotating the data can be carried out without dramatically impairing the cycle time and area constraints. This is achieved by applying a particularly novel approach, called partial decode or modulo shifter. It is designated as partial decode because of its unique multistage structure with partial-shift shift groups, or modulo shifter since each nested shift amount can be calculated by modulo arithmetics. At this step, an illustrative example would clarify how this technique works. A 160+ bit shifter, for instance, is accomplished by shifting multiples of 16 bit positions (0,16,32,48,64,80,96,112,128,144,160), then a shift of multiples of four bit positions (0,4,8,12), and a final binary shifter (0,1,2,3). Note that, although the input data have to be moved in a broad range of first-stage shift positions, only a four-way multiplexor is enough because there are less than 64 data bits. A typical shifter stage is shown in Fig. 5. Its NMOS-like shift circuits also allow very wide ORing within the function. The wide OR required for IEEE sticky bit calculation is carried out by ORing the control signals. This simple but powerful technique minimized the cell height due to less wires for data and control and attained extremely fast shift time (for our example above, 6 ns after the shift amount is calculated). It should be noted that the partial-shift steps, their sequences, and control signals can be optimized according to each application. Also, having the shift groups subdivided facilitates other bit manipulations.

To avoid performance degradation during the two-cycle accumulation, a similar implementation with comparable performance is used for post-normalization to shift the adder output by the amount of completely overlapped zero detection.

C. Logarithmic Adders

For the very large adder (114 b) concatenated with an incrementer (55 b), one of the fastest implementations is known to be logarithmic [10], [11]. This operation is performed in a one's complement form which involves an end-around carry. Its classical carry lookahead block is built using a binary tree of the true followed by complementary propagate/generate stages. In a lookahead scheme only the most significant bit groups are being evaluated while doubling the number of carry signals at each stage to be computed. The well-known fan-out problem can be bound by using the least-significant-bit locations to buffer the increasing load, leaving a fan-out of about 3 for every stage. This logarithmic buffering scheme makes the end-around carry be generated by only a single gate delay. Also, if the sign of the result is negative, the bits of the word are complemented so that the sign magnitude result is correctly output in one more gate delay.

D. Leading Zero / One Anticipation

A minimum of the two-cycle latency and a second pipeline delay comparable in time to the multiplication/shifting path of the MAF could be achieved only by overlapping the normalization and addition. This novel approach, called leading 0/1 anticipation, requires that the shift amount for the post-normalization be computed by using the two operands.

The implementation of the LZA inputs the generate ($G =$ both inputs are on), propagate ($P =$ exactly one input is on), and zero ($Z =$ no inputs are on) signals that are already created for the carry lookahead adder and outputs the normalize shift amount. Since the post-normalization is based on counting the consecutive bits matching the sign of the final result, the first instance of a nonzero element can be determined by examining the bits

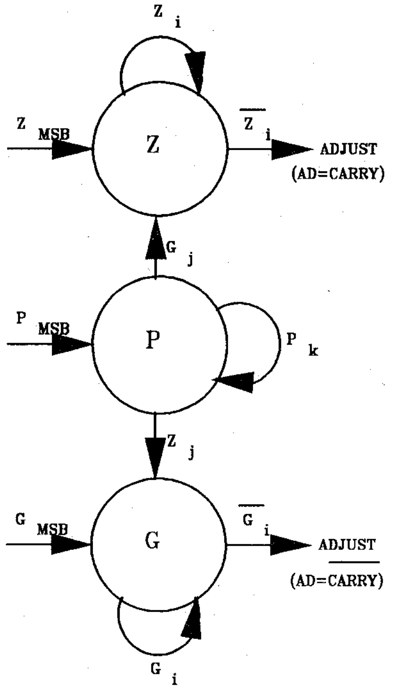

Fig. 6. General state diagram for LZA.

from the high-order end of the addition until the first one (or, in the case of a negative result, the first zero) is predicted. In fact, the sign token carries enough information to start with the leading 0/1 anticipation, namely: zero (Z)—adding two positive numbers, positive result; generate (G)—adding two negative numbers, negative result; and propagate (P)—subtraction. Note that a string of propagate tokens implies that the sign of the word is unresolved; the first Z labels the result negative, while the first G labels the result positive. Subsequently, after the word becomes negative, the next non-G signals the first bit of significance, since a string of G symbols produces all "1" symbols. Similarly, in the case of a positive result, the next non-Z represents the first nonzero in the word, and thus stops the shift count. The undecided state (P) will always go back either to the Z or G case that is defined entirely. Since the low-order bits are not monitored, a carry can cause a single bit position overshift, which is corrected during the binary normalization. Fig. 6 represents the general state diagram along with the adjustments for each output case.

The carry propagation is known to take $\log(n)$ time, therefore it is desired that the leading zero count be computed in the same period. This would mean processing the string of P, G, and Z inputs as fast as the adder, e.g., by using a parallel algorithm similar to the carry lookahead structure. The leading 0/1 lookahead construction is more complicated than carry calculation. Hence the first stage is a four-step process, to allow the more complex LZA parallel cell to fit in 4 b of the adder cell. Therefore, for logarithmic leading 0/1 anticipation, the state diagram of Fig. 6 should be generated for all hexadecimal subgroups of the P, G, and Z tokens. It can easily be shown that, in general, all possible segment

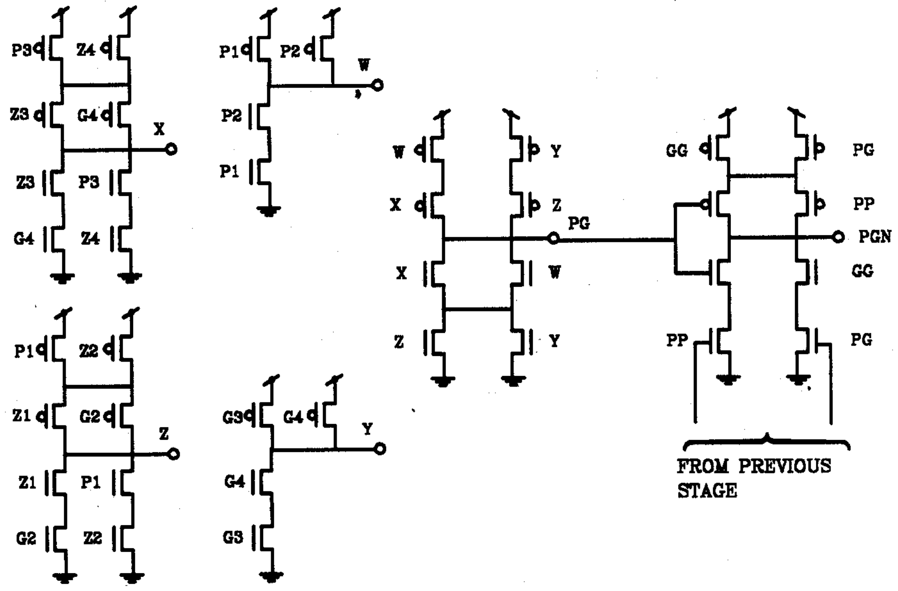

Fig. 7. *PG* state generation along with recursion circuit.

combinations can be expressed by five logical equations given as follows:

$$ZZ = Z_1Z_2Z_3Z_4 \tag{1}$$

$$GG = G_1G_2G_3G_4 \tag{2}$$

$$PP = P_1P_2P_3P_4 \tag{3}$$

$$PZ = P_1P_2(P_3G_4 + G_3Z_4) + (P_1G_2 + G_1Z_2)Z_3Z_4 \tag{4}$$

$$PG = P_1P_2(P_3Z_4 + Z_3G_4) + (P_1Z_2 + Z_1G_2)G_3G_4. \tag{5}$$

The subsequent prediction step is recursively combining the individual lookahead blocks. At the nth stage, the logical equations for general state anticipation can be given as follows:

$$ZZ_{(n+1)m} = ZZ_{n(m-1)}ZZ_{nm} \tag{6}$$

$$GG_{(n+1)m} = GG_{n(m-1)}GG_{nm} \tag{7}$$

$$PP_{(n+1)m} = PP_{n(m-1)}PP_{nm} \tag{8}$$

$$PZ_{(n+1)m} = PP_{n(m-1)}PZ_{nm} + PZ_{n(m-1)}ZZ_{nm} \tag{9}$$

$$PG_{(n+1)m} = PP_{n(m-1)}PG_{nm} + PG_{n(m-1)}GG_{nm}. \tag{10}$$

A typical circuit for state generation (PG) and recursion is depicted in Fig. 7. Each cell at the following iterations will have the intermediate prediction signals as inputs (to receive the information from its neighbors on the left and the top) and outputs (to supply the anticipated states to its neighbors on the right and the bottom as well as to the network outputs). The doubling and buffering procedure mentioned above is abstracted in Fig. 8 for the LZA implementation. The conclusion of the LZA is an OR of all states representing ZZ, PP, PZ, PG, and GG signals for each hexadecimal position. In order to accommodate the partial decode scheme used in the shifters [2], the shift signals are further coded into shift positions $(0 \cdots 7) \times 16 + (0 \cdots 4) \times 4$, to feed the shifter.

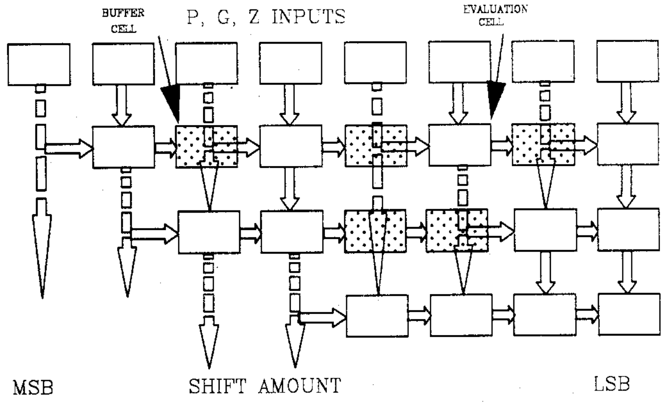

Fig. 8. LZA lookahead doubling and buffering scheme.

In this implementation, the delays for hexadecimal shifting tuned to the delays for add/complement output, thus allowing the control and data to arrive nearly simultaneously at the shifter.

IV. Chip Physical Data

The FPU was fabricated on a 12.7-mm × 12.7-mm die using triple-level metal, single-polysilicon, 1-μm minimum feature technology [12]. Fig. 9 shows a photomicrograph of the RISC FPU chip. This unit attains a peak execution rate of 50 MFLOPS with a 25-MHz clock frequency and is capable of sustaining nearly that rate in complex programs such as graphics and Livermore loops. It operates at 40 ns under worst-case conditions and dissipates 4 W of power.

The complete multiplier array utilizes all the M2 and all the polysilicon available, and occupies only 4 mm × 5 mm of area in a 1-μm minimum feature technology. Table I summarizes the number of transistors, area, and speed properties of the subunits described so far.

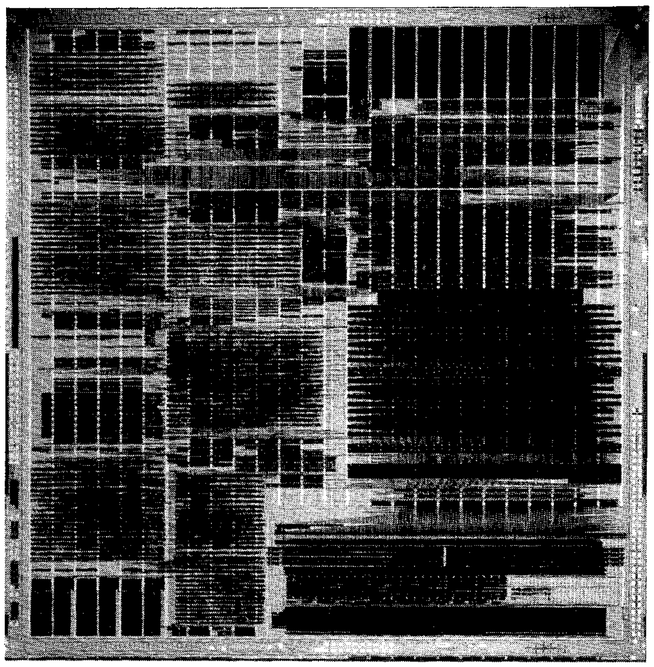
Fig. 9. Photomicrograph of the FPU chip.

TABLE I

SEGMENT	TRANSISTORS	AREA(sqmm)	SPEED
MULTIPLIER	82934	21.3	25ns
ALIGNMENT SHIFTER (Hex)	4756	3.6	7ns
(Binary)	1275	0.3	3ns
ADDER/INCREMENTER	22780	4.2	22ns
LEADING ZERO ANT.	12226	3.0	17ns
NORMALIZE SHIFTER (Hex)	2211	4.5	5ns
(Binary)	945	0.4	3ns

Note that a test site of the MAF data flow has also been manufactured using the 1-μm CMOS ($L_{\text{eff}} =$ 0.5 μm) technology [2].

V. Conclusion

The second-generation FPU chip that stretches fundamentals underlying the RISC philosophy beyond the CPU implementations has been described. Its MAF (multiply–add-fused) unit increases throughput and accuracy of floating-point computations over other existing designs. The single accumulate instruction is emulated by other designs, but does not maintain the accuracy or latency. The cycle time is competitive with other CMOS RISC systems, while the floating-point performance is considerably above other CMOS units.

Leading zero anticipation made the two-cycle pipeline possible by nearly eliminating the additional post-normalization time and allowing for reduced overall system latency. Partial decode shifters enable complete time sharing for the multiply and data alignment. Improved design techniques for logarithmic addition and higher order counters for multiplication complete this second-generation RISC FPU design. Its tightly coupled logical/physical correlation allows for reduced wiring area and delay in the logarithmic multiplication and probably logarithmic addition necessary for the massive 169-b adder/accumulator.

Finally, the state of the art in floating-point/VLSI arithmetic has been enhanced in many ways by the RISC FPU. We believe that the MAF concept will provide a solid basis for the future RISC floating-point architectures.

Acknowledgment

First of all, there would be no second-generation RISC machine without J. Cocke, who is utterly inspirational. Acknowledgment is also due to research and development teams, with particular thanks to E. Kronstadt for his encouragement and support to the work described in this paper.

References

[1] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC system/6000 floating-point execution unit," *IBM J. Res. Develop.*, vol. 34, pp. 59–70, 1990.
[2] R. K. Montoye, P. W. Cook, E. Hokenek, and R. P. Havreluk, "An 18ns 56-bit multiply-adder circuit," in *ISSCC Dig. Tech. Papers*, Feb. 1990, pp. 46–47.
[3] G. Radin, "The 801 minicomputer," in *Proc. Symp. Architectural Support for Programming Languages and Operating Systems* (ACM SIGARCH Computer Architecture News), vol. 10, no. 2, Mar. 1–3, 1982, pp. 39–47.
[4] P. Markstein, "Computation of elementary functions in the IBM RISC system/6000," *IBM J. Res. Develop.*, vol. 34, pp. 111–119, 1990.
[5] E. Hokenek and R. K. Montoye, "Leading zero anticipator (LZA) in the IBM RISC system/6000," *IBM J. Res. Develop.*, vol. 34, pp. 70–77, 1990.
[6] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[7] R. Nair, "MLG-A case for virtual grid symbolic layout without compaction," in *Proc. ICCAD* (Santa Clara, CA), Nov. 1987, pp. 180–183.
[8] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, 1964.
[9] O. L. MacSorley, "High speed arithmetic for binary computers," *Proc. IRE*, vol. 49, pp. 67–91, 1961.
[10] S. Winograd, "On the time required for binary addition," *J. ACM.*, vol. 1, pp. 277–285, 1965.
[11] J. Slansky, "Conditional-sum addition logic," *IEEE Trans. Electron. Comput.*, vol. EC-9, pp. 226–231, 1960.
[12] K. Klein *et al.*, "Characteristics of a set of 12.7-mm processor chips," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 783–789, Oct. 1987.

Erdem Hokenek (M'84) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Technical University of Istanbul, Turkey, and the Swiss Federal Institute of Technology, Zurich, Switzerland, in 1974, 1976, and 1985, respectively.

As a World Trade Visiting Scientist, he was with the Thomas J. Watson Research Center, IBM Research Division, Yorktown Heights, NY, in 1985, where he joined the VLSI Department as a Research Staff Member in 1986. Currently, he is Manager of the Custom VLSI Design/Applications group at the Watson Research Center. His research interests include analysis/design of analog and digital networks, VLSI design and design automation, computer arithmetics, and neural networks.

Robert K. Montoye (S'81–M'89) received the B.S. degree in physics, the M.S. degree in computer science, and the Ph.D. degree in computer science from the University of Illinois.

His summer employment consisted of work at IBM in Burlington, VT, in 1980, and Yorktown Heights, NY, in 1981 and 1982. He officially began employment at IBM in 1983 and began research into high-performance CMOS design, including the MAF floating-point unit, and various sections of the second-generation RISC machine. In 1986 he moved to IBM in Austin, TX, and transferred the MAF technology, as well as significant improvements in symbolic physical design, and remained to improve the cost performance of the system dramatically by inventing a partitioning which allowed the RS/6000 320 desktop to widen the cost/performance range of the machine. Additionally, he upgraded the design library and is currently making the improved physical design capability available to the rest of the corporation. He is the author of numerous articles and patents in parallel processing, VLSI architectures, and design automation.

Peter W. Cook (S'61–M'71) was born in Cleveland, OH, in 1939. He received the B.S. degree in electrical engineering from the University of Cincinnati, Cincinnati, OH, in 1962 and the M.S. and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, PA, in 1968 and 1971, respectively.

After graduation from the University of Cincinnati in 1962, he took a position as Member of the Staff of the Laboratory of Technical Development of the National Heart Institute in Bethesda, MD, where he worked on ultrasonic techniques for fluid flow measurement. He left this position in late 1965 for a position as Research Staff Member at the IBM Thomas J. Watson Research Center in Yorktown Heights, NY. Since that time he has worked on several aspects of FET design, including artwork generation, CAD, and electrical/physical design of systems. He is Manager of the Exploratory FET Processor group of the Silicon Technology Department at the Watson Research Center.

Dr. Cook is an associate member of Sigma Xi.