Design of the IBM RISC System/6000 floating-point execution unit

by R. K. Montoye
E. Hokenek
S. L. Runyon

The IBM RISC System/6000* (RS/6000) floating-point unit (FPU) exemplifies a second-generation RISC CPU architecture and an implementation which greatly increases floating-point performance and accuracy. The key feature of the FPU is a unified floating-point multiply-add-fused unit (MAF) which performs the accumulate operation $(A \times B) + C$ as an indivisible operation. This single functional unit reduces the latency for chained floating-point operations, as well as rounding errors and chip busing. It also reduces the number of adders/normalizers by combining the addition required for fast multiplication with accumulation. The MAF unit is made practical by a unique fast-shifter, which eases the overlap of multiplication and addition, and a leading-zero/one anticipator, which eases overlap of normalization and addition. The accumulate instruction required by this architecture reduces the instruction path length by combining two instructions into one. Additionally, the RS/6000 FPU is tightly coupled to the rest of the CPU, unlike typical floating-point coprocessor chips. As a result, floating-point and fixed-point instructions can be executed simultaneously. Load/store operations are performed using register renaming and store buffering to allow completely independent operation of load/store with arithmetic operations. Thus, data-cache accesses can occur in parallel with independent arithmetic operations. 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.

*RISC System/6000 is a trademark of International Business Machines Corporation.

Introduction

In the ten years since it first appeared, RISC technology has evolved significantly. An important possibility in the application of RISC architecture to technical workstations is the potential it offers for greatly improved performance in floating-point operations. In the design of the RISC System/6000* (RS/6000) processor, achieving that potential was a major goal. The RS/6000 floating-point implementation has in fact improved the state of the art of floating-point architecture for RISC machines in two distinct ways: First, it made the FPU a *tightly coupled* processor, with a parallel set of registers with direct cache access, using additional hardware to avoid conflicts. Second, and more significantly, the architecture was optimized around a single unit which combined the two required operations of multiplication and addition,

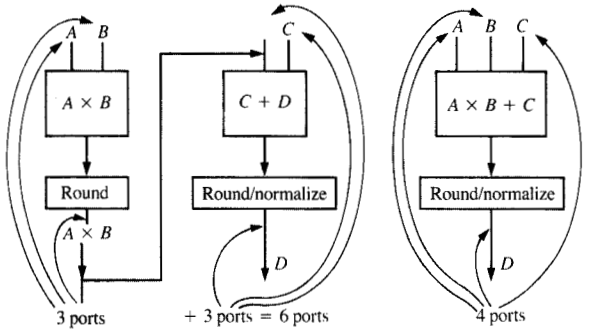


Figure 1

(a) Classical multiplier and adder; (b) MAF implementation.

increasing floating-point performance while reducing the required ports, distinct functional units, and rounding errors. We believe that this unit, called the MAF (multiply-add-fused) unit, will be the basis of future RISC floating-point architectures.

This paper is organized as follows: The next section discusses the motivation for MAF, explaining in some detail why it is an appropriate addition to the floating-point architecture in VLSI. The third section is a summary of floating-point operations, with discussion to demonstrate the parallelism that is possible when the multiply and add are fused. This is followed by a description of the two-stage pipeline used for the version of IEEE double-precision floating-point arithmetic used in the RS/6000 processor, with delays consistent with its over-all superscalar second-generation RISC architecture. The fourth section describes the interaction of logical and physical design required to incorporate several advances in VLSI arithmetic while accommodating required delay and technological (physical) constraints. The results are summarized in the final section.

Multiply-add motivation

In VLSI, connectivity is a very important factor in both cost and performance. Since the most common use of floating-point multiplication is for "dot-product" operations, a single unit which forms the multiply-accumulation operation $D = (A \times B) + C$ would produce a significant reduction in internal busing requirements, as shown in **Figures 1(a)** (classical scheme) and **1(b)** (MAF). This reduction from six connections to four is consistent with the RISC philosophy of producing heavily optimized units to attack the most frequently required functions. In the MAF implementation, only a single adder and normalizer are required, thus eliminating an output port from the multiplier and one of the input ports of the adder.

In addition to reducing latency and eliminating subunits and ports, the removal of a normalizer increases accuracy, while the single unit forces the architecture to produce more efficient code by forcing the single multiply-add instruction. The penalty for this reduction of units and ports is an additional gate delay in the adder section, caused by a 50 percent increase in the width of the adder. This factor is more than offset by the reduction in multiplexing of this single pipeline. Additional physical optimization is allowed by a large self-contained data path which reduces any overhead associated with this atomic unit. In particular, the adder can be laid out on a more compact grid, since the only connections to the adder are from the multiplier and shifters. Also, this single unit provides a superior target for compiler optimization, by providing a single operator with one-cycle pipeline time, which, in conjunction with overlapped loads, completely utilizes the bandwidth of the five-port register file. This eliminates the uncertainty in bandwidth availability required of commonly "chained" multipliers and adders. In a companion paper, Markstein [1] shows that the additional accuracy provided by the single rounding operation *is required* in order to produce correct results using Newton–Raphson-like techniques. In summary, there is significant motivation for a new floating-point operation based on the MAF primitive, provided that the unit can be designed so that it does not impact the cost or cycle time significantly.

Floating-point operations

Floating-point operations which deal with data in "scientific notation" are more complex than fixed-point or integer operations. In double-precision IEEE floating-point arithmetic [2], the basic representation is (S, E, M): S is a bit representing the sign of the number, E is 11 bits representing the exponent of the number biased by 1023, and M is the 52-bit quantity which is the mantissa, stripped of its leading 1. Let us represent two quantities in this notation (we show only a 9-bit mantissa for ease of reading):

$A = (S = 0, \quad E = 1000000010, M = 010000)$

has a value of $+2^3 \times (1/2 + 1/8)$, or 5,

$B = (S = 1, \quad E = 1000000000, M = 100000)$

has a value of $-2^1 \times (1/2 + 1/4)$, or -1.5.

Quantities can be summed only when their exponents agree, thus:

$A + B = 2^3 \times (1/2 + 1/4) - 2^1 \times (1/2 + 1/4) =$

$A + B = 2^3 \times (1/2 + 1/4) - 2^3 \times (1/8 + 1/16)$:

$A + B = 1100,\ 2^3 \times 0.11000000 - 2^3 \times 0.00110000,$

$A + B = 2^3 \times 0.01110000,$

$A + B = S = 0, \quad E = 1000000010,$

$M = 1100000 = 3.5,$

and

$A \times B = 2^1 \times (1/2 + 1/4) \times [-2^3 \times (1/2 + 1/8)],$

$A \times B = S = 1, \quad E = 1000000011,$

$M = 11100000 = -7.5.$

From this representation and the basic operations required (multiplication, addition), the base integer operations required for floating-point arithmetic are immediately clear:

1. Multiplication requires a parallel effort of multiplication of the true mantissas and addition of the exponents.
2. Addition requires a prenormalization stage to align the bit-weighting of the values to be summed, followed by addition.

In all cases, normalization (detection of the first 1 and shifting out 0 results), followed by rounding to conform to the number of bits in the result register, is required in each operation. Since multiplication is a costly and time-consuming operation involving multiple stages of summation and a final addition, it seems only reasonable that the prenormalization can be completely overlapped with the early phases of multiplication. Under such a scheme, the addend can be shifted in either direction, while the product is fixed in location (**Figure 2**). This allows full parallelism of the exponent operation and shifting operation. However, if the quantity to be added has a larger exponent than the product, there is a shift "overflow" which has the potential of incrementation, as shown in Figure 2.

If the "overflow" is as much as the number of bits in the mantissa, no bits from the product interact with the bits from the added quantity. If the quantity to be added has an exponent greater than the number of bits in the mantissa, there is a set of "underflow" bits which must be accumulated according to the rules of the IEEE. When the difference exceeds $2M$, all bits of the addend become "underflow" bits, and as such play no part in the computation other than the final rounding. Thus, the penalty to be paid for combining the operations is an incrementer of the length of the mantissa. Since the result of a multiply-add operation has $3m$ bits, and the m most precise bits must be returned as the result mantissa, the normalization operation is considerably more complex than a traditional multiplier and adder. A significant

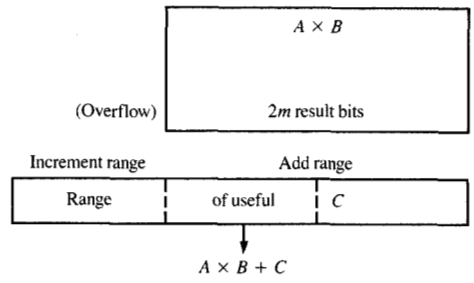


Figure 2

Schematic of MAF normalization.

amount of innovation was added to the design to hide the normalization delay completely and make this addition/normalization path comparable in time to the multiplication/shifting path. Thus, a natural two-stage pipeline arose, in which multiplication and partial compression occurred in parallel with the prenormalization, and addition occurred in parallel with postnormalization. The pipeline is shown in **Figure 3**.

• *Sign-magnitude capability*

Since all numbers and operations are represented in sign-magnitude form, it is natural to make the conversion from the adder-convenient one's-complement form to the representation-convenient sign-magnitude form as simple as possible. While the exponent values are being processed into a shift distance, the signs are also being processed. If the sign of the addend is different from the sign of the product, all the bits of the shifted result are complemented. The add/increment is performed in one's-complement form, which involves an end-around carry. The buffering scheme (to be described later) makes this end-around carry require only a single gate delay for all bits. Finally, if the sign of the result is negative, the bits of the word are complemented, so that the sign-magnitude result is correctly output.

• *IEEE compatibility*

In order to be usable in the workstation marketplace, the MAF unit had to be compatible with the IEEE binary standard. As Figure 3 shows, this was accomplished by building units which operate on hexadecimal data, and adding binary converters which operate in certain dead periods in the pipeline. For example, the alignment shifter was chosen to be hexadecimal because of the vast reduction in number of shift positions, which allowed a

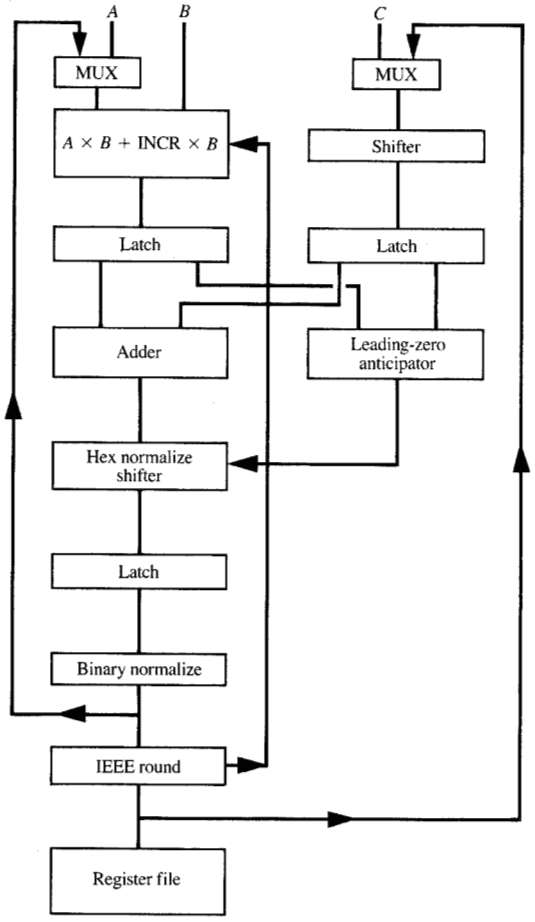


Figure 3

MAF pipeline.

very efficient layout. The binary shifter, which operates on much smaller fields, uses data available earlier (the low-order bit positions) and can be "hidden" before the larger hexadecimal shifter/complementer. Thus, there is no cycle-time impact on the binary device. In the case of the normalization shifter, the binary part of this shift is the first operation in the "cleanup" cycle and is sufficiently small that it does not add to either the rounding phase or the return to the pipeline. In order to maintain two-cycle pipeline operation for rounding operations, a dual scheme was used:

1. If the number was to return through the addend pipeline, the full increment, conditional to the proper IEEE scheme, was performed at the same time as the addition and preparation for rounding. This is possible because the exponent addition and the incrementation take about the same time.
2. If the number was to return through the multiplicand pipeline, the function $B \times (A + \text{increment})$ is performed in the multiplier, allowing the information that B is to be incremented to arrive as late as possible, and forming an "explicit" increment of A by adding another copy of B to $A \times B$ in the multiplier.

These techniques allowed two-cycle behavior with minimal cycle-time impact.

Dataflow implementation

As is the case for all well-designed custom chips, the logical design of the MAF dataflow was performed in parallel with the physical design, producing a very efficient chip. In this section we discuss the logical and physical design in detail for the separate sections of the dataflow: the multiplier, the shifters, the adder, and the leading-zero/one anticipator [3].

Since the key to an efficient chip is significant interaction between the desired logical operation of the subsections and their efficient interaction, and since the most aggressive logical operation in the RS/6000 processor is the previously described two-cycle MAF, this section is devoted to a detailed description of the logical and physical implementation of the critical segments of this activity. Particular emphasis is on two critical aspects, 1) true performance and 2) space required using the two levels of metal available at the time. For simplicity, a tool/design discipline called Macro Layout Generator (MLG) was used throughout this effort [4]. It forced all connections onto the metal-1/metal-2 grid, which is conveniently aligned with the gates, including drain/source and gate contacts. In other words, the technology allows one metal-2 wire for every contacted gate, and the gates are *parallel* to metal 2, allowing easy contact to metal 1. See **Figure 4** for an example of MLG output.

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

- *Multiplier*

Fast-multiplication techniques are well known, and are based on two results: In 1951, Booth discovered that signed numbers could be multiplied using a recoding technique [5] which resulted in $N/2$ additions or subtractions to multiply two N-bit two's-complement numbers. Booth's method was extended in 1961 by MacSorley [6], and in 1964, at the University of Illinois, Wallace [7] showed that multiplication could be performed in $\log_{(2/3)} N$ plus an addition of two N-bit

numbers, using a construct called a "Wallace tree." Over the years, these techniques were exploited and proved optimal. We use Booth's method and an extension of Wallace trees more usable in VLSI to produce a VLSI CMOS multiplier that is highly efficient.

An explanation of Wallace multiplication is a good starting place to describe the MAF multiplier. Since multiplication in its simplest form is the addition of N N-bit words (or, using Booth's invention, $N/2$), the objective is to add together as quickly as possible all N words. Using an N-bit carry-save adder, three words can be input and two words output. If this technique is simultaneously applied to all N words of the multiply (using $N/3$ adders of N bits each), a single delay of a full adder can reduce the problem from a summation of N words to a summation of $2N/3$ words. This technique can be applied iteratively until there are only two words left to be summed, and the summation can then be done using the carry-lookahead techniques discussed later. Since each stage reduces by one third the number of terms to be added, the number of stages is $\log_{(2/3)} N$; that is,

- Three terms or fewer can be reduced to two in one stage.
- Four terms or fewer can be reduced to two in two stages.
- Six terms or fewer can be reduced to two in three stages.
- Nine terms or fewer can be reduced to two in four stages.

Additionally, it requires about $N^2 - 2N$ full adders to sum the N terms to two. Using Booth's technique to produce $N/2$ terms of length N reduces the number of carry-save adders by two, but only removes $\log_{(2/3)} N - \log_{(2/3)} (N/2)$, or one to two full adder delays.

To be more efficient, the number of stages must be reduced. This is because the VLSI technology used, CMOS, has significant delay due to the long wiring required in a Wallace tree. Thus, a reduction in the number of long-wire stages is significant in terms of performance. The major extension of the multiplier beyond what appears in the literature is that of (7, 3) counting. Traditional Wallace multiplication uses carry-save adders to count the value of three operands into two results, one which has a binary weight of 1 and one which has a binary weight of 2. The logical extension adds seven objects to a set of results, one result which has a binary weight of 1, one result which has a binary weight of 2, and one result which has a binary weight of 4. The VLSI advantages are quite apparent, since the most expensive feature of VLSI is connectivity. The (3, 2) adder has five connections and removes one bit from the problem. The (7, 3) adder has ten connections and removes four bits from the problem. Thus, the (7, 3) adder requires only half as many connections as the (3, 2) adder to produce a final result. If a circuit with little upper-level wiring can be produced efficiently to perform this operation, this structure will create fewer wiring demands than a Wallace adder. Additionally, the (7, 3) adder reduces by a factor of 2.5 the number of stages needed, since each stage reduces the number of remaining terms to be summed by a factor of (7/3), in comparison to (3/2). While the cell is more complicated and slow, the number of long wires that must be driven is reduced by that factor of 2.5, resulting in a timing-efficient design as well. Thus, a combination of Booth encoding and (7, 3) addition produces a very efficient multiplier, but it still must be implemented in CMOS VLSI.

A key objective in VLSI is to build the design efficiently within the space required for the top layer of wiring for the circuit. Since our objective was the design of a 56-bit multiplier [56 bits to allow the construction of either a 53-bit (IEEE) or 56-bit (HEX) double-precision unit], the wiring requirements for 7/3 counters were

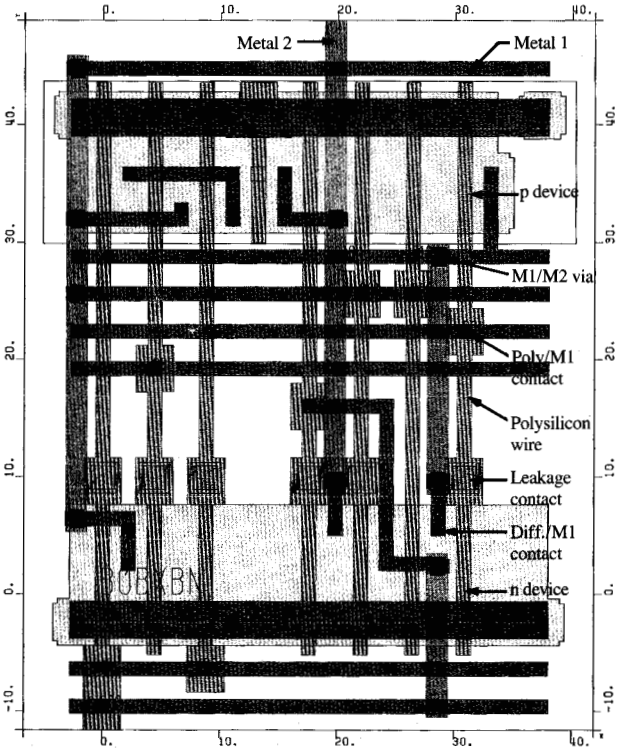


Figure 4

Example of MLG output.

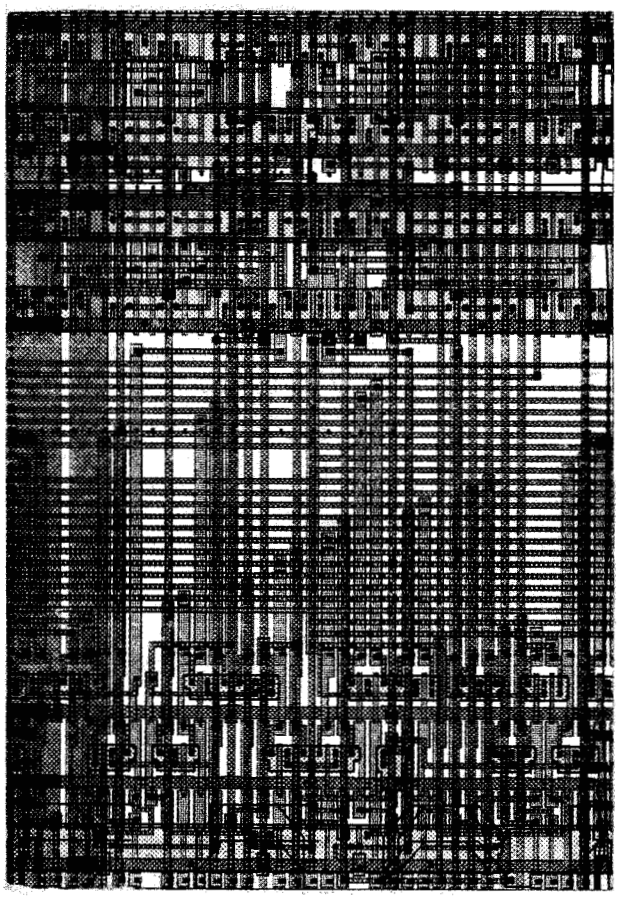


Figure 5

Multiplier cell.

shown to require a width of 18 transistor locations or M2 (metal-2) wires per cell. The most efficient cell organization was to produce a mirrored pair of cells with twice that width which shared a common control bay, as shown in **Figure 5**.

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 thus fit into the 36-track-wide cell. The second stage was a 7-input, 3-output cell, using full adders to save area while retaining the low wiring/driving requirements of a (7, 3) adder. As shown in **Figure 6**, the complete multiplier uses all the M2 and all the polysilicon available, and requires only 4 mm × 5 mm of area in a 1.2-μm technology.

• *Shifters*

The problem of shifting/rotating data in an efficient manner is important in VLSI, and MAF puts considerable pressure on both the performance and the cost of very large shifters. We adopted a particularly novel solution, which provided a (probably) minimum-height configuration (in terms of metal-1 wiring tracks required) and an extremely fast shift time which was extended to include both sticky-bit computation (required for IEEE) and a novel 32-bit rotator (required for fixed-point computations). We first describe the 32-bit left/right rotator, because it provides the most illustrative case for the technique of partial-decode shifters. The objective is to rotate a 32-bit quantity, either left or right, using a minimum of silicon area and a minimum of delay from the control word ($C_0, \cdots, C_4$), the direction signal (D), and the data inputs ($C_0, \cdots, C_{31}$). Notice that left rotation is simply right rotation of the two's complement of the control word. The two classical approaches are as follows:

1. Decode the control lines to 32, one of which will be high, and control thirty-two 32-wide pass-device multiplexors [8]. This technique requires 32 data and 32 control lines, plus significant space for the large

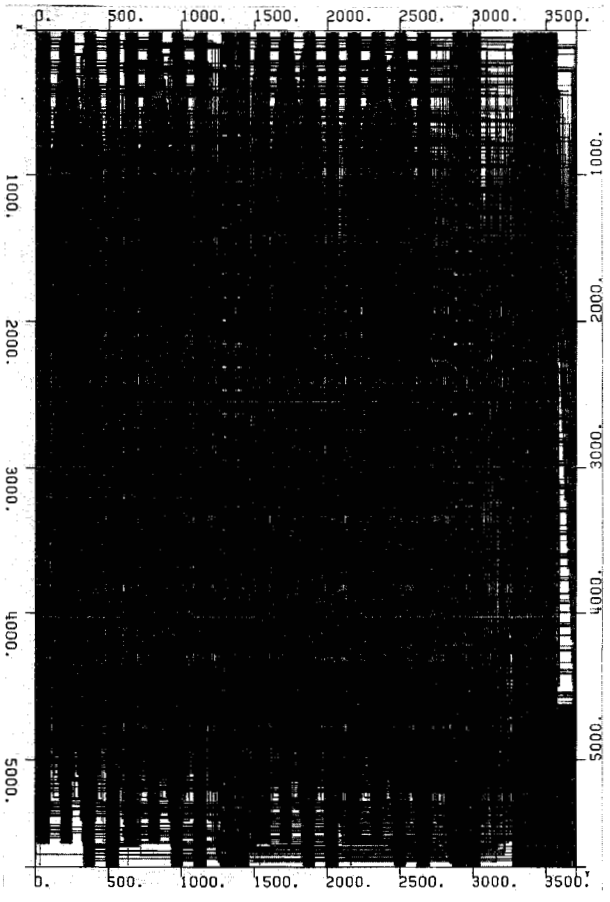
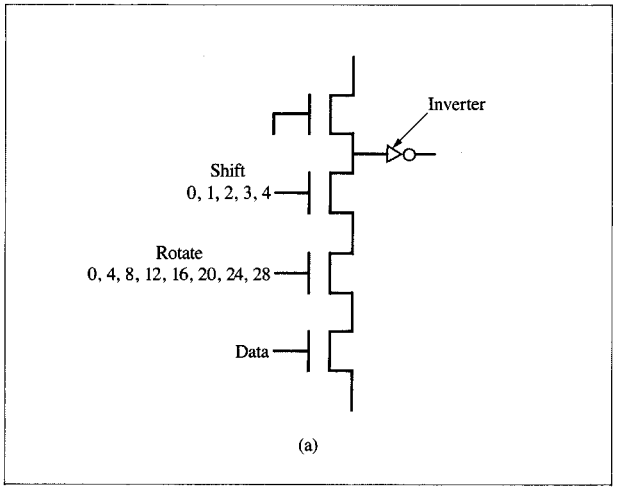
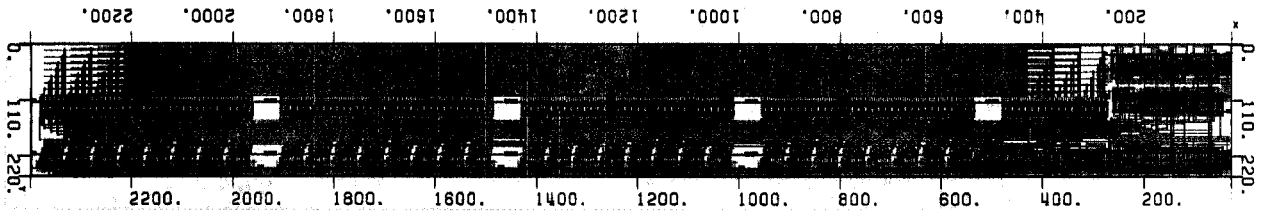


Figure 6

Multiplier unit.



(a)



(Lower) shift of 0, 1, 2, 3, 4

(Upper) rotate of 0, 4, 8, 12, 16, 20, 24, 28

(b)

Figure 7

(a) 32-bit rotator circuit; (b) completed unit.

(and therefore slow) pass-device multiplexor. The main virtue of this "fully decoded" technique is simplicity of design.

2. Use the undecoded control lines (XOR D, the direction) as controls for five 2-way switches, rotating 16, 8, 4, 2, 1, 1 positions left with C_4, C_3, C_2, C_1, C_0, and D. This requires 64 wires for data, since the first section connects both i to $16 + i$ and $16 + i$ to i, requiring 32 wires. Each subsequent stage requires twice as many wires as its rotation distance. This "fully encoded" technique offers ease of control and is particularly suitable if two-way switches are extremely fast.

Clearly, there must be an intermediate solution which decodes into more than one stage and requires considerably fewer control wires than the first approach and fewer data wires than the second. Choosing to produce an eight-way multiplex on the first-stage shift and rotating (0, 4, 8, 12, 16, 20, 24, 28) bit positions, and a five-way multiplex on the second stage, shifting (0, 1, 2, 3, 4) bit positions, has sufficient range for all rotation amounts. This "partial-decode" scheme, so named because it decodes partial groups, allows for significant improvement in both area and delay by requiring only 8 + 4 or 13 control wires and only 32 + 5 (with an additional optimization) or 32 + 9 (without the additional optimization) data wires by consuming all but the last five bits of shift in the first stage. Additionally, the data path is composed of only two (relatively) small multiplexors in series, allowing the use of an NMOS circuit to avoid the classical problems of pass-device design. See **Figure 7(a)** for the circuit used for the data path. Additionally, the control path consists of an inverter, a three-input NAND, and a two-way pass-device multiplexor. In order to provide both left and right rotation, the two's complement of the control word must be performed. This is simply accomplished by complementing the control bits and adding 1. The complementation is accomplished in the first (8-way) multiplexor by a two-way pass-device switch, which produces the output of the complemented bits of the control word if left is selected, and in the second (5-way) multiplexor by a two-way pass-device switch which adds

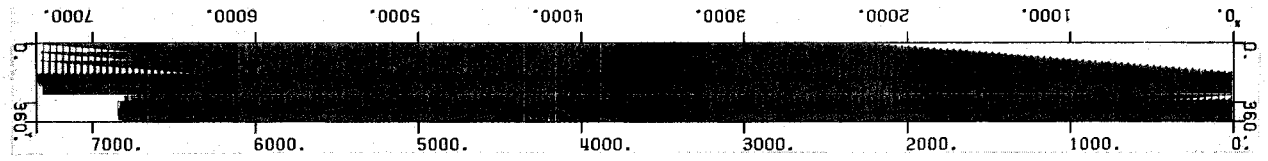


Figure 8

160-bit shifter.

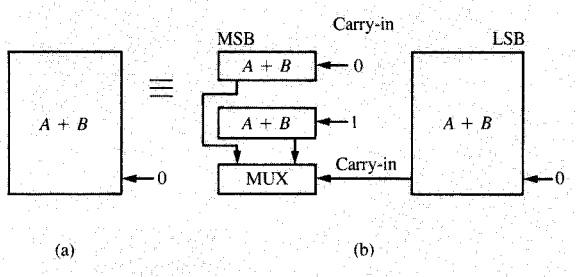


Figure 9

(a) Addition; (b) binary partitioning for carry-selection addition.

1 to the complemented result if the rotator is in left mode. Finally, the last four bits of the first stage are duplicated to avoid the long wiring of bits 28–32 to bits 0–4. Conveniently, these bits fit in the control section with the decoding circuitry. In summary, a 32-bit rotator is built requiring a minimum of control and data wires, and two rows of multiplexor circuitry. It has a single-stage delay in the data path (2 ns in 1-μm CMOS) and a four-stage control path (6 ns). Its mask-level data is shown in **Figure 7(b)**.

Very fast wide-shifters are a necessity in the multiply-add. By using this partial-decode structure and NMOS-like circuits to allow large "dotting" within the function, a 160-bit shifter with sticky-bit accumulation and post-complementation can be performed within 6 ns of the time the shift amount is calculated. This is accomplished by using a binary preshifter, a stage-one shift of 16 bit positions (0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160), and a final shift of four bit positions. The wide OR required for sticky-bit calculation is accomplished by ORing the control signals. Despite the apparent large number of first-stage shift inputs, only a four-way multiplexor is required in stage one, since there are fewer than 64 input bits. The mask-level design appears in **Figure 8**. Additionally, a comparable-performance normalization shifter is used in conjunction with completely overlapped zero detection to allow two-cycle accumulation without cycle-time degradation.

- *Logarithmic adders*

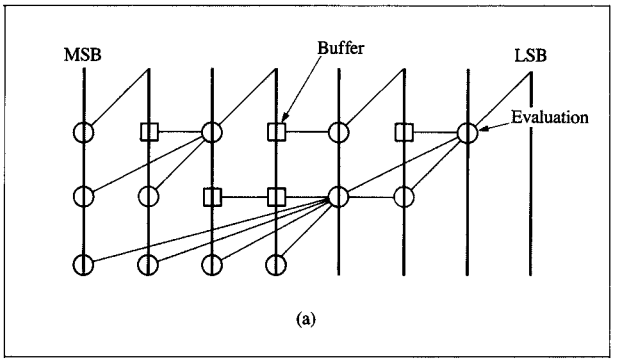
To execute a multiply-add, a very large adder (106+ bits) followed by a 53-bit incrementer is required. For N as large as 160, logarithmic solutions are required, because even less costly carry-skip schemes, whose delay is approximately $\sqrt{n}$, are significantly less competitive than they were at 32 bits. Note that $\sqrt{32} = 6$, while $\log 32 = 5$, but $\sqrt{160} = 13$, while $\log 160 = 8$. It was shown as early as 1965 by Winograd [9] that addition requires a minimum of $\log(n)$ time, where n is the number of bits to be added. We give a very simple proof of the $\log(N)$ delay, and use it to derive both the function and the layout scheme which holds this bound, including fan-out and wiring delays, while requiring $N \log(N)$ area.

In **Figure 9(a)**, two N-bit numbers are being added. In **Figure 9(b)**, the problem is partitioned into two problems: one $N/2$ addition for the least precise operands, and two $N/2$-bit additions (one with a carry-in signal and one without) for the most precise $N/2$ bits. The result for the most precise $N/2$ bits is selected in this *carry-select* scheme [10], since the carry input selects the correct addition. Clearly, this process can be applied to each of the two sub-additions, and the recursion is terminated with a one-bit addition, producing 2^N bits of correct carries in N stages of carry selection.

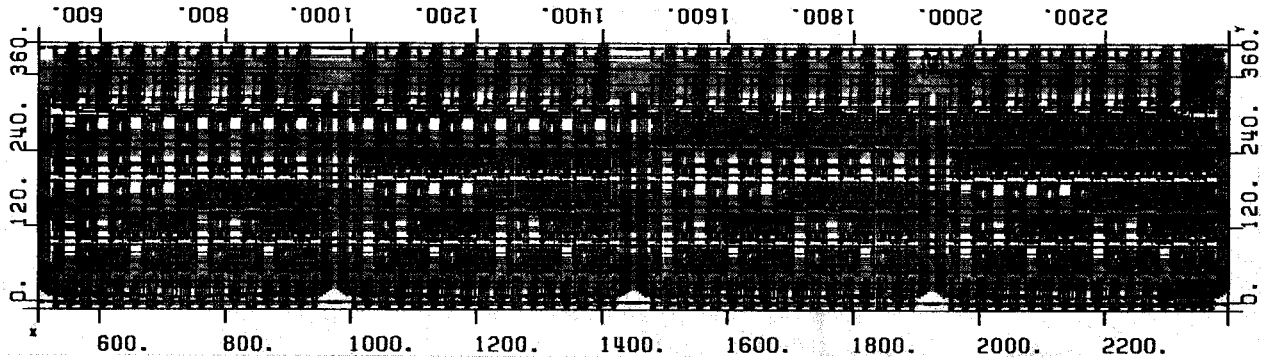
To simplify the carry equation derivation, let us note the fact that addition can be partitioned into two tasks: carry generation and sum generation based on the half-sum being XORed with the carry-in to the bit position. Define G as the signal which is on if a carry results independently of carry-in, and P as the signal which is on if a carry results only in the presence of an input carry. The merged generate is due either to a high-group generate or to a low-group generate with a high-group propagate; the merged propagate is formed when both the low group and the high group have a propagate. Algebraically,

$$G = G + Pg,$$

$$P = Pp,$$



(a)



(b)

Figure 10

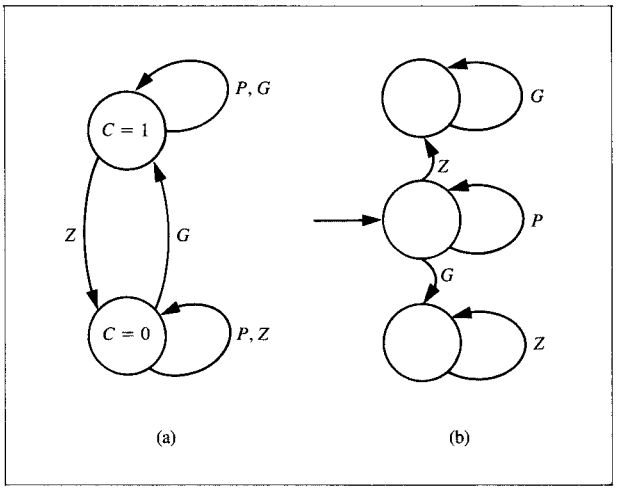
(a) Abstract 8-bit adder; (b) actual 32-bit adder unit.

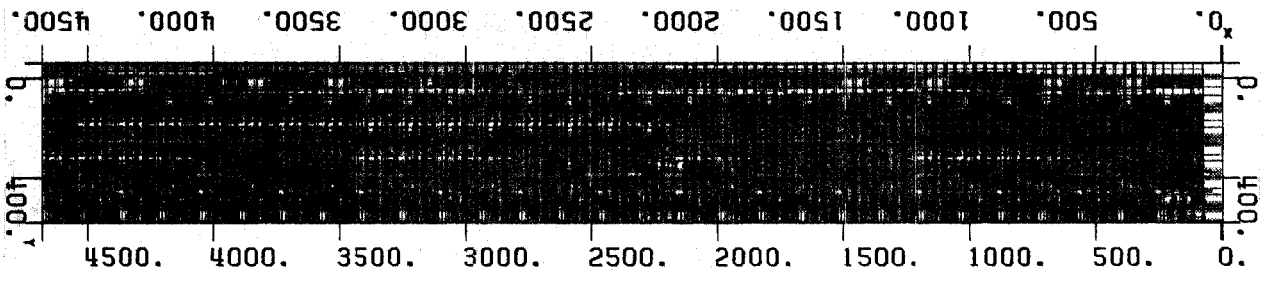
leading to the classical carry-lookahead formula. However, fan-out is a significant penalty, and the carry-select scheme doubles its maximum fan-out every section (as it must to completely double the number of correct carry signals in each stage). The principle that makes the system workable is the fact that in the carry-select scheme, only half the signals are being evaluated each cycle, since the less significant bits are not under evaluation. The empty locations created by this half usage of evaluation circuits allow for significant buffering, where both the number of gates in parallel for drive and the load are doubled at every stage, leaving a fan-out of about three for every section. **Figure 10(a)** shows a schematic representation of the adder, while **Figure 10(b)** shows the mask-level design for a 32-bit version of the adder. Notice that the wiring for paralleling gates shows the carry-select nature of the system.

• *Leading-zero/one anticipation*

To reduce the latency of the MAF unit to a minimum of two cycles, the normalization function is overlapped with the addition function. The unit required is a *leading-zero/one anticipator* [3], which inputs the generate and propagate signals from the adder and outputs the shift distance. Since the carry propagation is known to take log (N) time, it would be very profitable if the 0/1 count could be computed in this same period. In a manner similar to adder carry propagation, the first instance of a nonzero element can be determined, subject to a single bit correction due to carry propagation from the low-order end of the word. We describe the serial solution to the carry propagation, and make this solution parallel by the same method as that used for the adder.

Examine the inputs to the adder from the most significant bits to the least. Label each input set of bits as G (both inputs are on), P (exactly one input is on), and Z (no inputs are on). Notice that a string of propagate tokens signals that the sign of the word is unresolved; the first Z labels the word negative, while the first G labels the word positive. Additionally, after the word is negative, the next non-G signals the first digit of significance, since a string of G symbols produces all "1" symbols. Similarly, in the case of a positive word, the next non-Z represents the first non-zero in the word, and





(c)

Figure 11

(a) Adder state diagram; (b) LZA state diagram; (c) completed LZA unit.

thus determines the shift amount. **Figure 11** represents this state diagram.

Notice that the adder was made parallel in the previous section, and operated in log (N) time. The leading-zero/one anticipator (LZA) operates over a similar doubling process with the following two exceptions: The first stage is a four-step process, to allow the more complex LZA parallel cell to fit in four bits of the adder cell. The conclusion of the LZA is an OR of the three states representing P, G, and Z for each hexadecimal position. This OR is further coded into shift positions $(0 \cdots 7) \times 16 + (0 \cdots 4) \times 4$, to feed the shifter. In the RS/6000 implementation, the delays for hexadecimal shifting match the delays for add/complement output, thus allowing the control and data of the shifter to arrive simultaneously.

- *Chip physical data*

The RS/6000 floating-point unit was fabricated on a 12.7-mm × 12.7-mm die using triple-level metal and single polysilicon with a gate length of 1.2 μm (drawn). It operates at 40 ns under worst-case conditions and dissipates 4 W of power. **Figure 12** shows a microphotograph of the RS/6000 FPU chip.

Summary

Experimental first-generation RISC machines such as the 801 [11] were built at IBM as long as ten years ago. However, the floating-point unit described in this paper is the component which stresses the second-generation-RISC nature of the RISC System/6000 processor. Its single MAF (multiply-add-fused) unit increases the throughput and accuracy of floating-point arithmetic while requiring an instruction set which others only emulate. Its reduced complexity has allowed for the design of a completely overlapped zero/one detector to reduce overall system latency. Its careful logical/physical correlation allows for reduced wiring area and reduced delay in the logarithmic multiplication and provably logarithmic addition required for the massive 168-bit

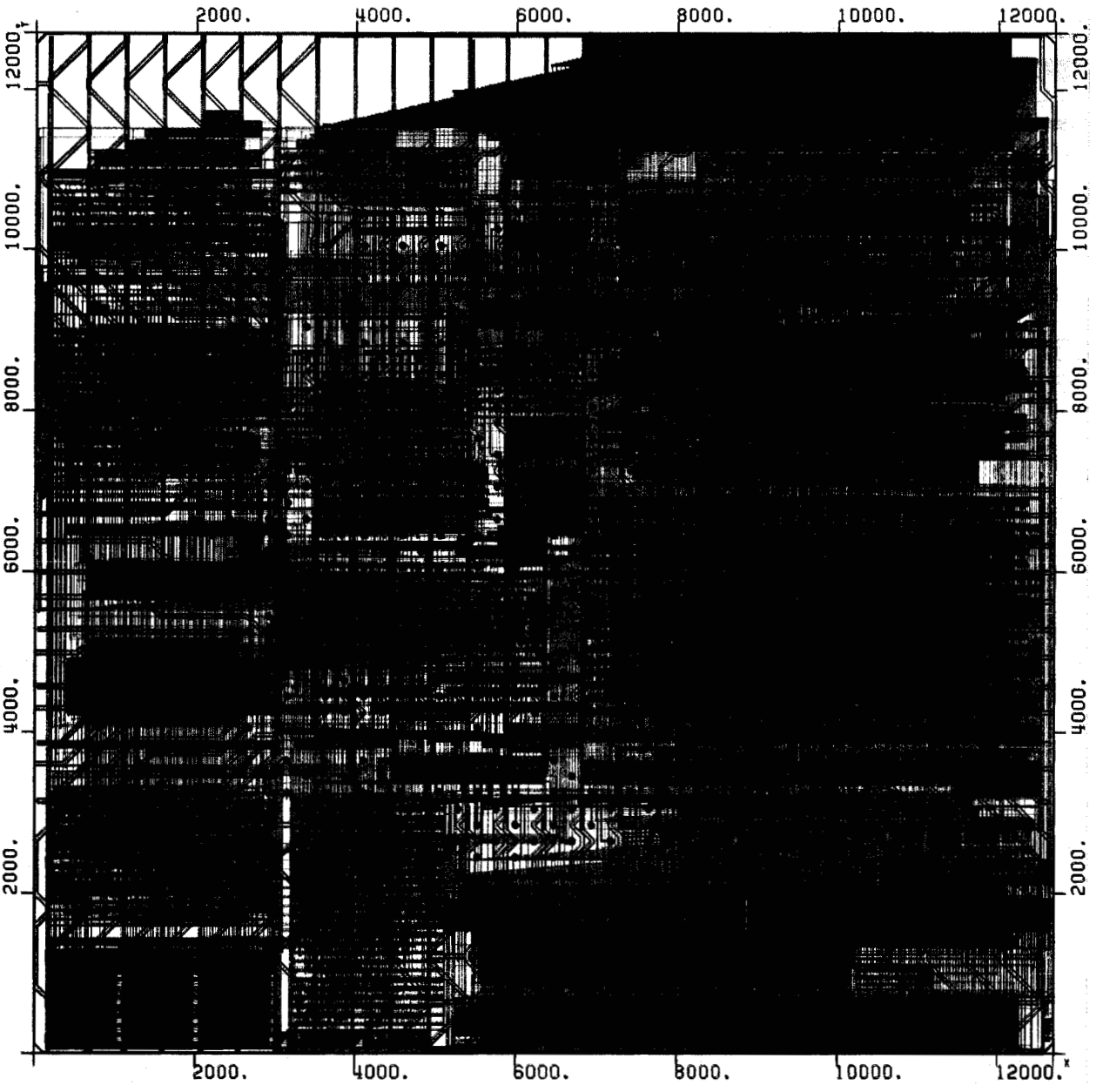


Figure 12

Mask-level diagram of completed FPU chip.

adder/accumulator. The state of the art in floating-point and VLSI arithmetic has been enhanced in many ways by the RS/6000 FPU; companion papers [12, 13] describe its integration with other innovations in the architecture and hardware implementation of the RS/6000 processor.

Acknowledgments

First of all, there would be no second-generation RISC machine without John Cocke, whose ideas and pioneering work established the substance of IBM RISC technology. Ravi Nair provided extensive support for MLG. Acknowledgment is also due the entire RISC

System/6000 development team, with particular thanks to M. P. Patel and P. T. Patel for their support of the work described in this paper.

References

1. P. W. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor," *IBM J. Res. Develop.* **34,** 111-119 (1990, this issue).
2. "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Standard No. 754*, American National Standards Institute, Washington, DC, 1988.
3. E. Hokenek and R. K. Montoye, "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating-Point Execution Unit," *IBM J. Res. Develop.* **34,** 71–77 (1990, this issue).
4. R. Nair, "MLG—A Case for Virtual Grid Symbolic Layout Without Compaction," *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD-87)*, Santa Clara, CA, November 1987, pp. 180–183.
5. A. D. Booth, "A Signed Multiplication Technique" (Part 2), *Quart. J. Mech. & Appl. Math.* **4,** 236–240 (1951).
6. O. L. MacSorley, "High Speed Arithmetic in Binary Computers," *Proc. IRE* **49,** 67–91 (January 1961).
7. C. S. Wallace, "A Suggestion for Parallel Multipliers," *IEEE Trans. Electron. Comput.* **EC-13,** 14–17 (1964).
8. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., Reading, MA, 1980.
9. S. Winograd, "On the Time Required for Binary Addition," *J. ACM* **12,** 277–285 (1965).
10. J. Slansky, "Conditional-Sum Addition Logic," *IEEE Trans. Electron. Comput.* **EC-9,** 226–231 (1960).
11. G. Radin, "The 801 Minicomputer," *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, in *ACM SIGARCH Computer Architecture News* **10,** No. 2, 39–47 (1982).
12. R. R. Oehler and R. D. Groves, "IBM RISC System/6000 Processor Architecture," *IBM J. Res. Develop.* **34,** 23–36 (1990, this issue).
13. G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. Res. Develop.* **34,** 37–58 (1990, this issue).

Received March 6, 1989; accepted for publication October 30, 1989

Robert K. Montoye *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Montoye received his B.S. in 1977 in physics and his M.S. in 1981 and Ph.D. in 1983 in computer science from the University of Illinois. He joined IBM in 1983 and began research into high-performance CMOS design, including the MAF floating-point unit. Dr. Montoye is the author of numerous articles and holds patents in parallel processing, VLSI architectures, and design automation.

Erdem Hokenek *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Hokenek 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 postdoctoral World Trade Visiting Scientist, he was assigned in 1985 to the Thomas J. Watson Research Center, Yorktown Heights, New York, where he joined the VLSI Department as a Research Staff Member in 1986.

Stephen L. Runyon *IBM Advanced Workstations Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Runyon received the B.E.E. degree from the Georgia Institute of Technology in 1980, and the M.S.E. degree in electrical engineering from the University of Texas in 1985. He worked for IBM during the summers of 1976 through 1980 while attending college, and joined IBM in Austin as a full-time employee in 1981. Since that time he has worked in Austin on the design of NMOS and CMOS VLSI chips.