A 17 × 69 Bit Multiply and Add Unit with Redundant Binary Feedback and Single Cycle Latency

W.S. Briggs

Cyrix Corporation
Richardson, TX 75080

D. W. Matula

Southern Methodist University
Dallas, TX 75275

Abstract

We describe a numeric processor whose kernel is a tree of redundant binary adders effecting either a 17 × 69 bit multiply-and-add or a 19 ×69 bit multiply with exact redundant binary output and single cycle latency. Feedback paths selectively allow a high-order or low-order part of the adder tree output to be fed back in redundant binary form to the multiplicand and/or addend inputs to the adder tree. We describe algorithms iteratively employing this adder tree kernel for: IEEE double extended multiplication, division, and square root; conversions between 18 digit BCD integers and 64 bit binary integers; and transcendental function evaluation. The multiplier design described was implemented in the Cyrix 83D87 numeric coprocessor (typically 33 MHz). Comparative results of this coprocessor with competitive x87 units are included.

I. Introduction and summary

Two important and related questions in the design of a floating-point numeric processor are:

i) How much hardware should be devoted to the multiply/add unit?,

ii) What algorithms can be employed to realize auxiliary arithmetic functions (e.g. divide, square root, transcendentals) sharing the multiply/add hardware?

If the emphasis is minimal hardware, then a single adder employing traditional shift and add/subtract steps for multiply, divide, square root and CORDIC transcendentals is sufficient at a heavy cost in execution time. For IEEE standard double extended floating-point operations, such a single adder solution will likely lead to most auxiliary arithmetic functions having execution times measured in the 100's of machine cycles. If the emphasis is on fastest multiply time, then n × n multipliers formed by a tree of redundant binary adders have been known [Wa 64, Da 65] for some time. They also support division and square root by Newton Raphson or convergence iterative methods and transcendental approximations by polynomial evaluation. Such functions are likely to be realized in the 10's of machine cycles, but generally will not utilize the full hardware multiplier capability very efficiently. The heavy cost in hardware for this full n × n fast multiplier must then be largely justified by the enhanced multiply performance.

If the emphasis in our original questions is in obtaining improved performance of the multiply and auxiliary functions all in rough proportion to the increased hardware cost of the multiply/add unit, then we claim a good solution is provided by an enhanced k× n multiplier. k ×n multipliers effecting reasonably efficient n ×n multiplication at reduced hardware cost have been investigated in the literature [BT 87, SH 89]. We note that k ×n multipliers may be formed employing a proportionally smaller number of adders than n × n multipliers, and organized as a shallow adder tree. Typically k is a fraction of n, say $n/8 \le k \le n/2$. Employing ripple-free redundant binary adders [Av 61] allows us to set the long dimension n to the target precision support of double extended without additional time penalty. Importantly, the smaller dimension is then effectively scalable to the hardware chip area available, with the desirable additional provision that the adder tree height be shallow enough to allow single cycle latency for the k × n multiplication.

Our initial purpose in this paper is to describe such a 17 ×69 bit multiply and add unit allowing redundant feedback with single cycle latency. Our principal results are then the description of auxillary function algorithms, some of which are new and interesting in their own right, and their implementations in this multiply and add unit. The algorithms and their implementations support the IEEE 754 floating point

1063-6889/93 $03.00 © 1993 IEEE

standard, and further match or exceed previous x86 IBM compatible norms for efficiency and/or accuracy of the auxiliary functions. The design we describe has been implemented in the Cyrix 83D87 shown in Figure 1, released in October 1989.

The multiplier unit comprises about one-third of the chip area of the 83D87. This unit handles the "number crunching" fixed point component of multiply/divide/square root with sufficient additional information bits to allow a separate unit to complete the normalization and infinitely precise roundings required by IEEE754. The multiplier also performs BCD-to-binary and binary-to-BCD integer conversions, and transcendental function evaluation. Our focus is on the multiplier with particular regard to the performance enhancement provided by the adder tree kernel and its feedback options to a variety of distinct arithmetic operations. We do not discuss exponent handling, addition/subtraction, roundings or normalization, which are handled in two other units of the 83D87.

In Section II we describe the adder tree kernel of our multiplier unit. We note how the unit is employed as either a 17 × 69 multiply and add unit or a 19 × 69 multiplier providing enhanced capability to the short side of the multiplier. In Section III we describe how the exact product-sum can be partitioned so that a high or low order 69 signed bit portion may be fed back to either the multiplicand or adder input to the adder tree kernel with single cycle latency.

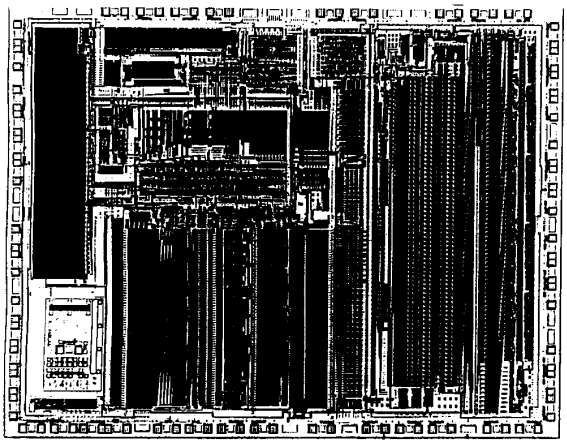

Figure 1: Cyrix 83D87 math coprocessor. Implementation in 1μ CMOS has about 200K transistors with die size 380 by 480 mils. Typical power at 33 MHz is 0.5 watts.

Finally, in Section IV, we outline the algorithms and highlights of their implementation for multiply, divide, square root, binary-to-BCD integer conversion, BCD-to-binary integer conversion and transcendental function evaluation. To measure and compare the "end user visible" overall efficiency of these instructions and accuracy of the transcendentals, we then cite from Juffa [Ju 91], cycle times and transcendental accuracy evaluations for the Cyrix 83D87 in comparison with three commercially available competitive 387 coprocessors.

II. Adder tree kernel

The largest arithmetic unit of the Cyrix 83D87 numeric coprocessor is a multiply and add component whose kernel is a tree of signed binary adders. This adder tree illustrated in Figure 2 has five inputs A, B, C, L, S and a single output P, some of which may transmit data in redundant format. Input and internal data paths shown as single lines designate that values transmitted are in binary, by which we mean the tradition non-redundant binary format with digit values {0, 1}. Data paths shown as double lines designate that values transmitted are in signed binary format, i.e. employing for digits the signed bit values {-1, 0, 1}. Each signed bit is then composed of a pair of bits, one for the sign and one for the magnitude of the signed bit.

The multiplier input S accepts a "short" 18 bit binary operand, the multiplicand input L accepts a "long" 70 bit signed binary operand, and the adder input B accepts a 70 bit signed binary operand which may be left or right adjusted within an 88 bit field width. The exact product-sum is provided as an 88 bit signed binary result at output P. Also, input C accepts a 1 bit binary carry signal and input A accepts a 4 bit binary input for accumulation into the product-sum. These inputs are utilized respectively, for example, when the adder tree is employed iteratively to generate and simultaneously accumulate 18 × 69 bit partial products of a 71 × 69 bit multiplication and when BCD digits are input for BCD-to-binary conversion.

Single pass 17 × 69 bit multiply and add

The functioning of the adder tree is conveniently described by considering the execution of a concurrent 17 × 69 bit multiply and 70 bit add operation. This product-sum operation yields an exact 88 signed bit result in one pass through the adder tree. The short operand (17 bit multiplier) is input through S and

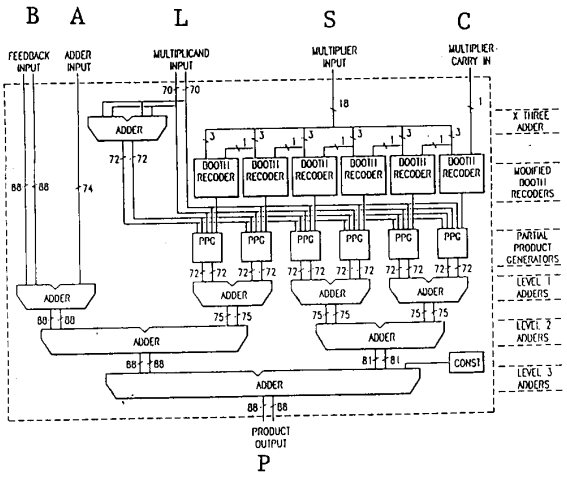

Figure 2: Tree of signed binary adders

converted by the Booth recoders shown to six octal digits in the minimally redundant digit set {-4, -3, -2, -1, 0, 1, 2, 3, 4}. The partial product generators of Figure 2 each generate a digit multiple of the long operand (69 signed bit multiplicand) input through L by either a selective shift and/or sign complement to effect the digit multiplies for digits {-4, -2, -1, 1, 2, 4}, or a selective complement of the 3-times adder output to obtain the digit multiplies for {-3, 3}. The six digit multiplies thus selected along with the third operand (70 signed bit addend) input through B are accumulated in the three levels of signed binary adders shown in Figure 2. Signed digit addition has been extensively discussed in the literature [Av61, Ma76, TY85]. Successively larger data path widths are shown in the figure to accommodate computation of the exact product-sum in the resulting signed binary format at output P. The regularity of the three levels of adders in Figure 2 provides more convenience for VLSI design than the adder arrangement of a traditional Wallace tree [Wa64] of 3-2 adders.

Single pass 19 × 69 bit multiply

The adder tree of Figure 2 may alternatively be utilized to provide a 19 × 69 bit multiply with exact output in a single pass. To achieve this result first consider that any 19 bit integer $b_{18}\ b_{17} \ldots b_0$ may be written as a 7 digit octal integer $d_6\ d_5 \ldots d_0$, where $0 \le d_i \le 7$ for $0 \le i \le 5$, and specifically the leading digit d_6 is either 0 or 1. The Booth recoding we employ converts any digit in the range $4 \le d_i \le 7$ to the value $d_i - 8$ with propagation of a plus one carry that is absorbed in the next higher position. It follows after conversion to $d_6'\ d_5' \ldots d_0'$ that $-4 \le d_0' \le 3$, $-4 \le d_i' \le 4$ for $1 \le i \le 5$, and specifically that the leading digit d_6' is now either 0, 1, or 2.

The 69 signed bit multiplicand is input as before through the long multiplicand input L. From the 19 bit multiplier $b_{18}\ b_{17} \ldots b_0$ the low order 18 bits $b_{17}\ b_{16} \ldots b_0$ are input through the short multiplier input S. The leading multiplier bit b_{18} and implicit Booth recoding leading carry bit b_{17} are inspected outside the adder tree kernel to determine if the leading recoded octal digit would be 0, 1, or 2. This is determined simply by adding the bit values b_{17} and b_{18}. If the leading digit is 1 or 2, the 69 bit multiplicand is first left-shifted one or two places and then also input through the signed binary feedback input B.

The importance of this simple facility to extend the range of the short side input to the multiply operation will become apparent in subsequent discussion of the iterative algorithms for the more complicated operations such as divide and square root. Suffice it here to say that when the short side input must necessarily accept an approximation of the desired variable, the facility to effectively extend the short side multiplier by 2 "guard" bits provides valuable options in combination with the same multiplier being employed for exact product-sum 17 × 69 bit operations. In summary:

Observation 1: With single cycle latentcy the adder tree of Figure 2 is capable of outputting either:

i) an 88 signed bit product-sum being a concurrent 17 × 69 bit multiply and 70 signed bit add, or

ii) an 88 signed bit product being a 19 × 69 bit multiply.

III. Multiplier feedback options

The adder tree of Figure 2 provides a single cycle product-sum capability that can be employed iteratively to provide efficient hardware implementation of a wide range of arithmetic functions. Figure 3 illustrates the multiplier unit embodied in the Cyrix 83D87 numeric coprocessor. The adder tree kernel of Figure 2 is supplemented in Figure 3 by various feedback paths, shifters, and other components. This unit allows efficient microcoded iterative "algorithms on silicon" for many functions required by the x87 instruction set including:

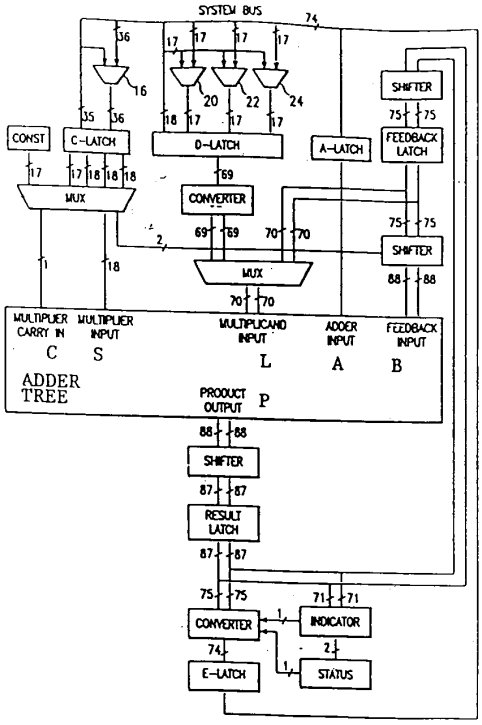

Figure 3: Multiplier unit illustrating feedback paths

i) Multiply/Divide/Square Root: This unit supports the normalized fraction (fixed-point) computations for IEEE 754 standard floating point multiply, divide, and square root for precision levels up to double extended (the exponent arithmetic and final rounding are handled elsewhere);

ii) Transcendental Functions: This unit supports argument reduction and function evaluation over standard ranges for various transcendental functions with error over the standard ranges of less than one unit in the last place (ulp) at the double extended precision format length with preservation of monotonicity of the evaluated function [FB91];

iii) Binary-Decimal Integer Conversion: This unit supports exact binary-to-BCD and BCD-to-binary integer conversion for integers up to 18 decimal digits efficiently with minimal additional hardware components beyond those otherwise required for the other arithmetic capabilities.

Fundamental to achieving these arithmetic capabilities with maximum efficiency is the ability to feed back a selected portion of the exact product-sum of value p from output P of the adder tree to an appropriate input (or inputs) of the adder tree for operation on the immediately next cycle. This design avoids the delay inherent in a more traditional multiple cycle latency pipelined multiply-add unit implementation of our iterative algorithms.

Two options for partitioning (without conversion to non-redundant form) the product-sum p output by the adder tree for the purpose of feedback of an appropriate portion are:

Long-high-back: $p = h_L + l_S$

This designates that high order part h_L of "long" length 69 or 70 signed bits is determined by simple truncation, where h_L is then available in signed bit format to be fed back as input to either L, B, or both, for use on the next cycle. The low order part l_S is available for concurrent inspection by other components of the multiplier unit.

Long-low-back: $p = h_S + l_L$

This designates that a low order part l_L of "long" length 69 signed bits is determined by truncation supplemented to 70 signed bits by inclusion of an overflow bit responsive to appropriate conversion of the high order part h_S, where here the "short" length quantity h_S is available in non-redundant form for output as required. The low order part l_L is available in signed bit form for feedback to either L or B. Its use on the next cycle, or delay to a subsequent cycle, is dependent on the particular conversion operation applied to the high order part.

Important features of the components illustrated in Figure 3 useful in realizing the arithmetic capabilities required and feedback options cited should be noted. Data paths and latches wider by several guard bits than the target 64 bit results are carefully chosen to support the IEEE754 requirements to provide 64 bit precision infinitely precisely rounded multiply, divide and square root results, and our requirements of one ulp accuracy at 64 bits for transcendental results.

The 2-bit data path from the multiplexor below the C-latch to the shifter between the feedback latch and the feedback input B is provided to effect the leading digit multiple selection for the 19×69 bit multiply capability of the adder tree previously discussed. The multiplicand for this operation is provided from the feedback latch in 70 signed bit format (including an overflow signed bit) and is sent to both the L and B inputs of the adder tree.

A variety of redundant binary feedback options are employed in the algorithms for the arithmetic operations supported by this arithmetic unit. The implementation of these algorithms will be described in some detail in the next section. We here preview a few of the individual steps to illustrate the alternative feedback options employed. All but one of these provide that the fed back signed bit value is available for immediate use on the next machine cycle.

i) In a step of full precision multiplication, an output partial product p is split to h_L and l_S, with h_L fed back to the feedback input B for accumulation into the next partial product, and with l_S sent to the indicator unit for updating the determination of the positive, negative, or zero status of the truncated portion.

ii) In a step of Newton Raphson iteration for refining an approximate reciprocal y′ of y, a product-sum p = 2-y′y is split to h_L and l_S, with h_L fed back to the multiplicand input L for computing the next product y′ (2-y′ y), with l_S discarded.

(iii) In a step of BCD-to-binary integer conversion evaluating a radix polynomial in the Horner method order, $(...(((d_{17} \times 10) + d_{16}) \times 10 + d_{15}) \times 10 + ... + d_1) \times 10 + d_0$, the product-sum p = (previous product sum) $\times 10$ + (next digit), is split to h_L and l_S. Here l_S is assured to be zero by the scaling employed, with the exact result value then fed back as h_L to the multiplicand input L for immediate use on the next product-sum operation.

iv) In a step of remainder updating for division, the product-sum p is the new remainder determined by subtracting a 17-bit-digit divisor multiple from the previous remainder. This new remainder is then split into h_S and l_L, with h_S implicitly converted to binary zero and discarded, whereby an appropriate leading overflow signed bit is appended (without any conversion time delay) to l_L which is then fed back to the feedback latch.

v) In a step of exact binary-to-BCD integer conversion, the product p is split to h_S and l_L with h_S a converted high order part that is known to be a BCD digit, and l_L converted to be a positive value by appending the appropriate overflow signed bit. This converted value for l_L is fed back to the multiplicand input L over the signed binary feedback path, with a one cycle time delay imposed in this case to accommodate the conversion of l_L to a positive value.

IV. Arithmetic operation implementation

The hardware organization of the iterative multiplier of Figure 3 was designed interactively with the design of algorithms for multiplication, division, square root, BCD-to-binary and binary-to-BCD integer conversion, and transcendental function evaluation by rational polynomial approximation. Some of these algorithms are new and described in more detail elsewhere [FB91, Ma 92]. Here we shall present an outline of these procedures and their implementation. We indicate the data flow through the components in each case to illustrate the extent of hardware sharing. This process also allows us to provide a measure of the number of hardware cycles of each of these operations attributable to the multiplier.

For the multiplication, division, and square root operations, the multiplier operates in a fixed point manner on the fraction portions of the operands, with the exponent arithmetic and rounding handled elsewhere. We do not include the initial loadings of the latches from the bus in our cycle counts here.

Multiply operation

A full precision multiplication herein denotes a 71 × 69 bit multiply. This is accomplished by four passes through the adder tree kernel followed by a conversion cycle. In the first pass the low order 18 bits of the multiplier are multiplied by the 69 bit multiplicand. With reference to Figure 3, the high order 69 bits of this 87 bit partial product are sent to the feedback latch and the low order 18 bits are sent to the indicator for positive, zero, or negative status determination. Then 18 more bits on the second and third passes and 17 more bits on the fourth pass of the multiplier along with the carry in from the Booth recoding of the lower order bits are input to S and C to form the next multiplier which is multiplied by the 69 bit multiplicand input from the D-latch. The resulting next 18 × 69 bit partial product is accumulated with the previous sum of partial products input from the feedback latch in each single pass. Each pass the new accumulated sum of partial products is split to h_L and l_S with low order 18 bits sent to the indicator. The high order 69 signed bits h_L are sent back to the feedback latch after each of the first three passes. After the fourth pass the 69 signed bits h_L are sent to the converter along with information from the indicator and status units to yield the desired output in the E-latch after the fourth pass. Since a pass through the adder tree kernel takes only one cycle, the 71 × 69

bit multiply described here with conversion to non-redundant form and status information on the truncated portion consumes only five machine cycles.

Divide and square root operations

The divide and square root operations are implemented by new algorithms finely tuned to the multiplier termed "short reciprocal" divide and square root. The short reciprocal divide algorithm is described in detail elsewhere [Ma 93]. Its principal features are outlined here.

The short reciprocal division process involves determining four successive digits in the maximally redundant digit set for the large radix 2^{17}. Given a next such digit, a new remainder can be determined and placed in the feedback latch replacing the previous remainder in one cycle as follows. The absolute value of the digit is input through the C-latch to S and multiplied by the divisor which is input to L from the D-latch and this product added or subtracted, depending on the sign of the digit, to the previous remainder input to B from the feedback latch. The result p is split to h_S and l_L where h_S is coerced to zero by a special conversion process. Note that the quotient digit is always chosen so that h_S must be either zero, or have an initial signed one somewhere in the field followed by one's of opposite sign. Examining the second lowest bit of h_S, and if a one, complementing the lowest bit of h_S and appending this signed bit as an overflow bit of l_L, effectively converts the truncated leading portion of h_S to zero without any timing delay. The resulting value of l_L is sent to the feedback latch as the new remainder.

The 17-bit signed quotient digit is obtained by multiplying the remainder (initially the dividend) by a specially chosen 19 bit short reciprocal of the dividend using the 19 × 69 bit capability of the multiplier. The product is split after conversion so h_S and l_L have the same sign with h_S then the next 17-bit digit. The correctness of this process is shown [Ma 93] to result from the short reciprocal being chosen larger than the true reciprocal and by an amount no greater than one part in 2^{18}. The quotient digit determination takes two cycles per quotient digit resulting from the multiply and subsequent conversion. We note here the importance of the 19 × 69 bit alternative multiply capability to provide sufficient accuracy for this process.

The short reciprocal is itself initially determined by table look-up followed by two Newton Raphson iterative refinements. A final adjustment and conversion to 19 bits is possible with assurance that the reciprocal approximation then falls in the required range. The iterative refinement formula $y'' = y'(2 - y'y)$ is conveniently computed in two multiplicative cycles and a conversion cycle to place the new refined valued in the C-latch, contributing a count of six cycles for the process of determining the short reciprocal. The full division process then requires 18 cycles from the iterative multiplier operations.

Square root is completed by a similar process with the only timing difference being two additional cycles beyond that of divide attributable to the extra cost in the Newton Raphson root reciprocal formula. The details may be found in [BM 91].

Binary-to-BCD conversion

For binary to BCD conversion the input is taken to be a non-negative integer k in binary format that has been verified to be less than 10^{18}. This value is multiplied by a 71 bit approximation to 10^{-18} in a four pass 71 ×69 bit multiply operation. The implicit normalization of the final output at p provides that if the result is split $p = h_S + l_L$, then $h_S = 0$ and l_L is a 69 bit approximation to $k10^{-18}$ with radix point to the left of the leading bit. Importantly, the 71 bit constant approximation of 10^{-18} is carefully chosen to assure that the resulting value of l_L falls in the range $k10^{-18} < l_L < k10^{-18}(1 + 10^{-18})$, so that subsequent infinitely precise multiplication of l_L by 10^{18} rounded down to an integer will return the value k.

To perform the conversion to BCD, the value l_L is multiplied by 10 and the new output value p is split to a new h_S and l_L. This splitting process here consumes an extra machine cycle as the new value l_L is coerced to be non-negative by appending an overflow signed bit and decrementing h_S as appropriate. This provides that h_S is the leading BCD digit which is stripped off and output, and l_L is returned to the feedback latch. This iterative step is repeated 18 times to determine the 18-digit BCD result. As all intermediate products are exact, the eighteen multiplications by 10 are equivalent to multiplication of the original approximation by 10^{18}, so the process yields an exact binary-to-BCD conversion. The original 4 pass multiply by the approximation to 10^{-18}, along with the 18 multiplications by 10 with intermediate computations to coerce l_L to be non-negative require a total of 40 multiplier unit cycles for this operation.

Note the same procedure and apparatus may be utilized for binary-to-base B conversion for any integer base $3 \le B < 2^{17}$. In fact, if more hardware to speed up binary-to-BCD conversion were justifiable, the conversion first from binary-to base 10^3 could be performed followed on-the-fly by table look-up to convert the 10 bit base 1000 digit to 3 BCD digits.

BCD-to-binary-conversion

BCD-to-binary conversion is performed by evaluating the BCD radix polynomial in the Horner method order. $(\ldots (((d_{17} \times 10) + d_{16}) \times 10 + d_{15}) \times 10 + \ldots + d_1) \times 10 + d_0$. During each cycle a new product-sum, p = (previous product-sum) $\times 10$ + (next digit), is computed with the previous product-sum sent to the multiplicand input L from the feedback latch (see Figure 3), the multiplier 10 sent to input S from the C-latch, and the next digit input to the second adder input A. The new product-sum p is split to h_L and l_S each cycle. l_S is assured to be zero by the scaling, and h_L is sent to the feedback latch every cycle until the last, when instead it is converted and output employing one extra cycle. Thus 18 iterative multiplier unit cycles are used for this operation.

Transcendental functions

The procedures for evaluating the special functions sin, cos, tan, arctan, exp and log have been described in [FB 91]. The procedures involve argument reduction, function approximation, and output value construction, all of which can extensively utilize the multiplier unit. Argument reduction can involve division. Division may also be employed when the approximation is by a rational polynomial, that is, the ratio of two polynomials. Furthermore, the output value construction for selected intervals of input values can involve a square root. This occurs in the algorithm design employed when the sin is obtained by computing $\sqrt{1-\cos^2(x)}$, with the intent of such a computation being to assure the monotocity of the resulting function. Thus the transcendental computations benefit greatly from the efficiency and accuracy of the divide and square root results which are internally available in 67 signed bit form with one ulp accuracy.

A principle component of each transcendental function computation is evaluation of a polynomial in the reduced argument x determined in Horner polynomial order, e.g. $(((a_4 x + a_3) x + a_2) x + a_1) x + a_0$. This exemplary fourth order polynomial can be evaluated by computing four linear polynomials of the form p x + a, where x is the reduced argument, p is initially the highest order constant a_4 and subsequently the ongoing product-sum of prior iterations, and a is the next constant. Referring to Figure 3, x is a 69 bit value sent to the multiplicand input from the D-latch, the ongoing product-sum p is a value up to 71 bits as converted and sent over the system bus to the C-latch, and the up to 69 bit next constant a is obtained from the feedback latch. Note that a is loaded to the feedback latch while the previous product-sum is being converted for transmission to the C-latch via the system bus.

Care is taken [FB91] in the scaling and selection of the constants for accumulating the intermediate product-sums so that some of the required full product-sums px may be computed to sufficient accuracy in fewer than four cycles. The total iterative multiply cycles employed for the transcendental functions vary considerably with both the particular function and argument region, all being largely in the range of 50-125 cycles. In comparison, the bitwise CORDIC evaluation of transcendental functions to this same accuracy can take several hundred cycles.

An important aspect of the multiplier design highlighted here by the transcendental functions is that the internal data path widths and latches are all larger by appropriate numbers of guard bits than the target support level (here 64 bits) of the final output to the external system. This provides for one ulp accuracy of the resulting transcendentals without extensive increase in hardware size or total iteration cycles.

The Cyrix 83D87 has been compared with the ULSI 83C87, IIT 3C87, and Intel 387DX, in an extensive study of x87 class math coprocessors by Juffa [Ju 91]. A selection of these results [Ju 91] regarding the accuracy of transcendental function evaluation is given in Table 1. For the seven functions, 100,000 arguments uniformly distributed over the intervals cited in Table 1 were evaluated by each of the four coprocessors. For each coprocessor, the percent of results differing from the infinitely precise round-to-nearest value for that function is given, along with the maximum difference in ULPs between the coprocessor computed value and the infinitely precise round-to-nearest value for that function.

The implementations of multiply, divide, and square root operations, conversions between BCD and binary integers, and transcendental function evaluations described in this section have been so far measured

solely by citing the number of cycles attributable to the execution steps in the multiplier unit. A meaningful test of how much end value efficiency this design provides to the user of a 386 class personal computer is available in the test data developed by Juffa. In Table 2 we have extracted from Juffa's results the measured cycle times determined for the same four coprocessors for the eight instructions cited. Clearly included in these observed totals are all cycles attributable to communication with the host microprocessor, transmission and loading of arguments and results, exponent handling, and final normalization and rounding. These additional processing steps significantly dominate on all but the transcendental functions the multiplier unit steps we have previously cited.

Note, in Table 2 that the multiply time of the Cyrix 83D87 is roughly comparable to the multiply time on the other three processors. Then observe that the wide variety of floating point numeric functions realized by our algorithm designs, which designs are closely bound to the signed binary feedback capabilities of this short by long multiplier, show a significant advantage in performance.

REFERENCES

[Av 61] A. Avizienis: "Signed-digit Number Representations for Fast Parallel Arithmetic," IRE Trans. Electron. Comput., Vol. EC-10, pp. 389-400, 1961.

[BM 91] W. S. Briggs, T. B. Brightman, and D. W. Matula, "Method and Apparatus for Performing the Square Root Function Using a Rectangular Aspect Ratio Multiplier," U.S. Patent No. 5,060,182, 1991.

[BT 87] B.K. Bose, L. Pei, G.S. Taylor, and D.A. Patterson, "Fast Multiply and Divide for a VLSI Floating-Point Unit," Proc. 8th Sym. on Comp. Arith., pp. 87-94, 1987.

[Da 65] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, Vol. 34, pp. 349-356, 1965.

[FB 91] W. E. Ferguson and T. Brightman, "Accurate and Monotone Approximations of Some Transcendental Functions," Proc. 10th Sym. on Comp. Arith., pp. 237-244, 1991.

[IE 85] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE754-1985, IEEE, New York, 1985.

[Ju 91] S. Juffa, "What you Always Wanted to Know About Math Coprocessors," Karlsruhe, 1991, private communication.

[Ma 76] D. W. Matula, "Radix Arithmeitc: Digital Algorithms for Computer Architecture," in *Applied Computation Theory: Analysis, Design, Modeling*, R.T. Yeh ed., Prentice-Hall, Eng'ewood Cliffs, 1976, pp. 374-448.

[Ma 93] D. W. Matula, "Short Reciprocal Division," in preparation.

[SH 89] M. R. Santoro and M. A. Horowitz, "SPIM: A Pipelined 64 x 64-bit Iterative Multiplier," *IEEE Journal of Solid-State Circuits*, Vol. 24, pp. 487-493, 1989.

[TY 85] N. Takagi, H. Yasuura and S. Yajima: "High-speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," IEEETC, Vol. C-34, pp. 789-796, . 1985.

[Wa 64] C. S. Wallace, " A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, Vol. EC-13, pp. 14-17, 1964.

		CYRIX 83D87		INTEL 387DX		ULSI 83C87		ITT 3C87	
Funct.	Interval	% Not RN	Max ULP	% Not RN	Max ULP	% Not RN	Max ULP	% Not RN	Max ULP
SIN	0, pi/4	1.554	1	28.873	2	35.530	2	18.650	1
COS	0, pi/4	0.925	1	27.121	2	43.989	2	7.700	1
TAN	0, pi/4	4.147	1	10.711	1	48.539	3	20.973	2
ATAN	0, 1	0.656	1	7.088	2	20.858	1	19.280	1
2XM1	0, 0.5	2.628	1	32.024	1	21.257	1	25.660	1
YL2XP1	0, sqrt(2)-1	3.242	1	22.611	1	27.893	2	45.830	3
YL2X	0.1, 10	0.931	1	13.020	1	13.603	1	10.888	3

Table 1: Accuracy of the transcendental functions on four 387 class coprocessors [Ju 91].

	CYRIX 83D87	ULSI 83C87	ITT 3C87	INTEL 387DX
FMUL [QWord]	46	42	48	41
FDIV [QWord]	57	72	72	79
FSQRT (L2T)	36	87	57	102
FSIN (L2T)	121	472	217	452
FCOS (L2T)	156	467	222	507
FPTAN (L2T)	136	517	306	297
FPATAN(L2T)	141	362	422	373
FYL2X(L2T)	111	437	357	393

Table 2: Execution time in clock cycles plus or minus one cycle for the indicated instruction on the indicated coprocessor according to Juffa [Ju 91], where QWord denotes a 64 bit argument and L2T denotes the particular constant $\log_2 10$.