The SNAP Project: Towards Sub-Nanosecond Arithmetic

M. J. Flynn K. Nowka G. Bewick E. Schwarz N. Quach

Computer Systems Laboratory
Stanford University
Stanford, CA 94305-4055

Abstract

SNAP — the Stanford subnanosecond arithmetic processor — is an interdisciplinary effort to develop theory, tools, and technology for realizing an arithmetic processor with execution rates under 1 ns. Specific improvements in clocking methods, floating-point addition algorithms, floating-point multiplication algorithms, division and higher-level function algorithms, design tools, and packaging technology were studied. These improvements have been demonstrated in the implementation of several VLSI designs.

1 Background

Arithmetic operations including both integer and floating point add, subtract, multiply, and divide form the basic building blocks of scientific computation and signal processing. Yet usually the optimization of their performance is relegated to design exercises in which the building blocks, the circuits/package/technology have been predefined and only the algorithm itself and its implementation can be altered to achieve performance.

The Stanford subnanosecond arithmetic processor (SNAP) research effort has targeted the full spectrum of tradeoffs, from devices, materials, interconnections and circuits to algorithms and organization. The SNAP project has made significant progress toward the development of a gigaflop computing element.

Fundamental to the SNAP objectives is the development of very high speed clocking techniques, which we call "wave pipelining." These latchless pipelines allow functional units to be clocked at significantly higher repetition rates (2-3 times) than could be done with conventional designs.

In the area of algorithm improvement, SNAP work covers all of the basic operations. We briefly report on some of the highlights of this work in this paper. In floating point addition our work improves delay by reducing the delay associated with rounding. In floating point multiply delay reduction is achieved by using redundant encoding of a higher order Booth algorithm. This reduces the height of the partial product tree without incurring delay due to the generation of "hard multiples" of the multiplicand (eg. $\pm 3M$.)

With the realization that performance of arithmetic processors is not simply dependent upon the most advanced circuit design techniques and algorithms, SNAP has advanced packaging and interconnection technology. By using MCM packaging technology, we are able to separate the data processing and communication functions in a subnanosecond processor and then optimize the technology for each. Figure 1 is a block diagram of the SNAP processor. It consists of a MCM substrate with multiple functional unit die and high-speed interconnection.

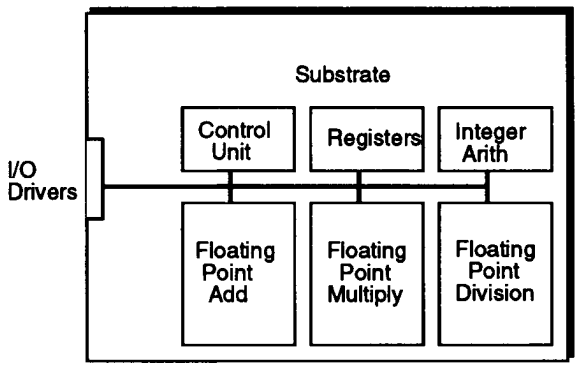


Figure 1: SNAP Processor.

In most of our work we validated our techniques by realizing implementations. Table 1 is a summary of our chip fabrication activities.

2 Algorithms and Systems

In the remainder of this paper we summarize some of the techniques and algorithms which have been developed.

2.1 Wave Pipelining

In an effort to improve the operating frequency of arithmetic processors using conventional technologies, we have employed wave pipelined design techniques for standard bipolar and CMOS fabrication technology.

Wave pipelining, or maximum rate pipelining, is a circuit design technique which allows digital synchronous systems to be clocked at rates higher than can be achieved with conventional pipelining techniques by relying on the predictable finite delay through combinational logic for virtual data storage. Conventional pipelined systems allow data to propagate from a register through the combinational network to another register prior to initiating the subsequent data transfer. Thus, the maximum operating frequency is determined by the maximum propagation delay through the

1063-6889/95 $4.00 © 1995 IEEE

Table 1: SNAP Die Fab Status

Die	Company*	Technology	Status	Result
Integer adder	Signetics	ECL	F.T.	2ns 32-bit integer [P3]
F.P. adder (64*b* IEEE)	HP Research	CMOS	I.T.	15ns latency [P10]
Multiplier (pp) slice	Signetics	BiCMOS	F.T.	Under 10ns pp tree only [P16]
Multiplier (pp) slice	Signetics	BiCMOS	F.T.	Under 7ns pp tree only [P16]
F.P. multiplier (53*b* IEEE)	Sun Micro.	ECL	I.T.	Full FMPY in 5ns
W.p. multiplier (pp) slice	Signetics	ECL	F.T.	3.8ns cycle time [P21]
W.p. multiplier 16 × 16*b*	NSF	CMOS	F.T.	3.3ns cycle time [P6]
W.p. vector unit (16*b*)	NSF	CMOS	F.T.	3.3ns cycle time
Divider	—	CMOS	NYS	Still under study

* Company or organization that sponsored die fab.
Abbreviations: F.T. = Fully tested I.T. = Incompletely tested due to fab difficulties NYS = Not yet submitted
W.p. = wave pipelined, F.P. = floating-point

longest pipeline stage. Figure 2 contrasts the timing of a conventional pipeline and a wave pipeline. In the wave pipeline, data 2 is output from the source register to the combinational logic prior to data 1's arrival at the sink register.

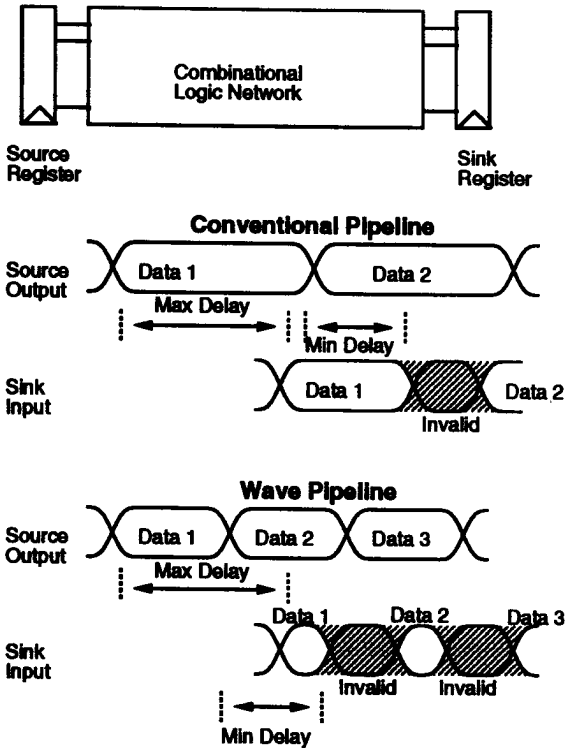


Figure 2: Conventional and Wave Pipeline Timing.

The clock period T_{clk} of a conventional pipelined circuit is limited by the maximum propagation delay through a logic network P_{max}, the uncontrollable clock skew ΔC, and register overhead T_{reg} by:

$$T_{clk} > P_{max} + \Delta C + T_{reg}$$

Wave pipelined systems apply the subsequent data to the network as soon as it can be guaranteed that it will not interfere with the current data wave. The maximum operating frequency of a wave pipeline is therefore determined by the difference between the maximum propagation delay and the minimum propagation delay in the combinational logic.

The clock period T_{clk} of a wave pipelined circuit is limited by the variation in propagation delay through a logic network ($P_{max} - P_{min}$), uncontrollable clock skew ΔC, and register overhead T'_{reg} by:

$$T_{clk} > P_{max} - P_{min} + 2\Delta C + T'_{reg}$$

The primary causes of data wave interference are the variation in the propagation delay due to differences the delay along separate paths, differences in the rate of propagation due to the state of the network inputs and intermediate nodes, and difference in fabrication and environmental conditions.

2.1.1 Wave Pipelining Tools and Techniques

To maximize the performance of wave pipelined arithmetic circuits, the variations in delay through the combinational logic must be minimized. Automated delay balancing tools have been developed as part of the SNAP research effort. These tools balance the propagation delay of all paths by inserting delay buffers and by modifying the individual bipolar [P20] and CMOS gates to increase the delays along the paths which are faster than the critical paths.

Run-to-run process variation can have a significant impact on propagation delay. Maximum to minimum delay along the same path for seven different runs of a one-micron feature size CMOS fabrication process was found to be 1.35x [P7]. Unless this variation is controlled, all wave pipelined logic will be limited to speedups of two to three to ensure that devices from any of these runs will operate over a specified clock interval. To counteract the effects of process variation, we have developed an adaptive supply

voltage method. An on-chip detector circuit determines if the propagation delays are faster than the nominal delays and the power supply is lowered until the delay approaches the anticipated delay. In this manner, ICs fabricated with fast processes are run at a lower supply voltage to ensure correct operation over the frequency range for which they were designed.

2.1.2 Wave Pipelined VLSI designs

Wave pipelining has been used in the development of ECL and CMOS arithmetic VLSI designs. A wave pipelined ECL population counter which can be clocked at 2.5 to 3-times the rate that the circuit could be operated without wave pipelining was developed [P21]. The worst-case delay of this circuit is 10 ns and the minimum clock period for wave pipelined operation is 4 ns.

Wave pipelining was used in the development of a CMOS 16-bit multiplier which operates at 300-350 MHz [P6]. This VLSI chip is clocked at a rate 3.7-times a non-pipelined design could achieve.

To demonstrate that this technology can be applied to arithmetic system design, we have developed a CMOS wave pipelined vector unit. Extensive use of wave pipelining was employed to achieve high clock rates in the functional units. The VLSI processor consists of a wave pipelined vector register file, a wave pipelined adder, a wave pipelined multiplier, load and store units, an instruction buffer, and a scoreboard and control logic. The register file, adder, and multiplier were designed to operate with less than a 2 ns variation in propagation delay.

The adder takes 16-bit unsigned operands and produces a 16-bit result with a maximum delay of 5.5 ns. Register add operations have a 3 cycle latency.

The multiplier takes 8-bit unsigned operands and produces a 16-bit result in a maximum of 10.8 ns. The multiplier is constructed with (4,2) counters and employs an additional parallel adder. Booth encoding is not used. Register multiply operations have a 5 cycle latency.

The vector register file consists of 5 vector registers. Each vector register has sixteen 16-bit elements. The registers have separate read and write ports and local address generation to optimize for cycle time. Read access time is less than 3.4 ns and minimum cycle time is 2 ns for the register file.

The VLSI vector unit contains approximately 47000 transistors and occupies an area of 43 sq mm. It has been fabricated in a 1 micron CMOS technology. Correct operation at 300 MHz has been verified. Figure 3 is a die photo of the wave pipelined vector unit. Figure 4 details the vector register-to-register wave pipelined multiply operation at 300 MHz.

We have developed bipolar and CMOS circuit design techniques and tools which enable VLSI system design using wave pipelining. We have demonstrated the benefits of wave pipelining for arithmetic processors via an ECL population counter, a CMOS multiplier, and a CMOS arithmetic vector unit designs. Using standard device fabrication technology these VLSI units achieve operating frequencies of 250 to 350 MHz through the use of wave pipelining.

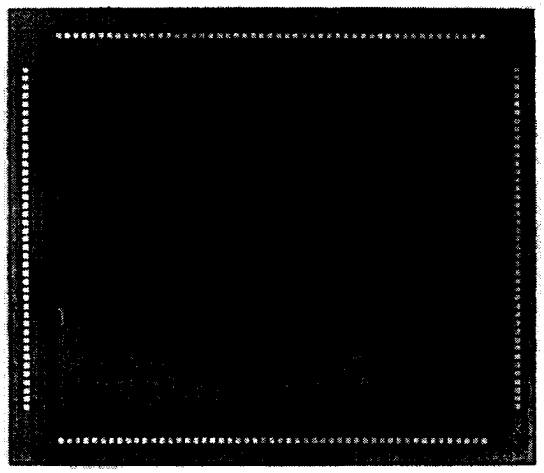


Figure 3: WP Vector Unit Die Photo.
300MHz, 43 sq mm, 1 micron CMOS

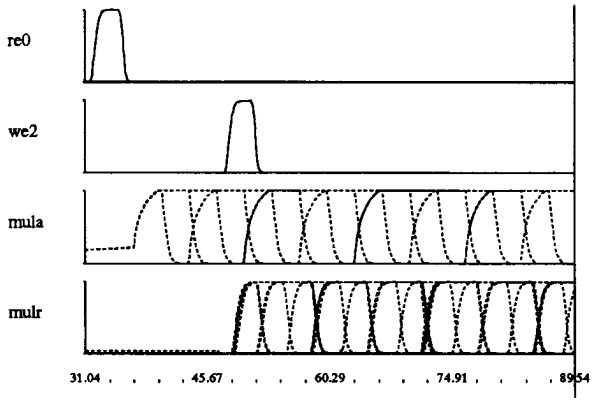


Figure 4: WP Vector Unit Multiply.

2.2 Addition and Floating-point Addition

We have been able to optimize earlier work by Ling on integer adders [2] to avoid excessive fanout, yet preserve the speed advantage. We have also mapped the Ling approach over to CMOS technology [P12].

We have made basic improvements to the floating-point addition algorithm. Conventionally, floating-point addition consists of a subtraction of exponents, a shifting of fractions by an amount equal to the exponent difference, addition or subtraction of the fractions, a shifting of the result (on subtraction) to leave it in normalized form, and a rounding of the result. These steps are generally sequential, and require two shifts and three additions (the exponent subtraction, fraction addition, and rounding). Farmwald, in earlier work [3], showed that one of the shifts could be eliminated by creating two simultaneous paths, since a long preshift could not be accompanied by a long postshift following the fraction addition/multiplication.

As part of the SNAP effort, we have shown that it is possible to further improve the floating-point addition by integrating the rounding step with the final fraction addition [P13]. The traditional, two path, and SNAP algorithms are shown in figure 5. The improvements are the result of creating up to four different results and selecting the correct result.

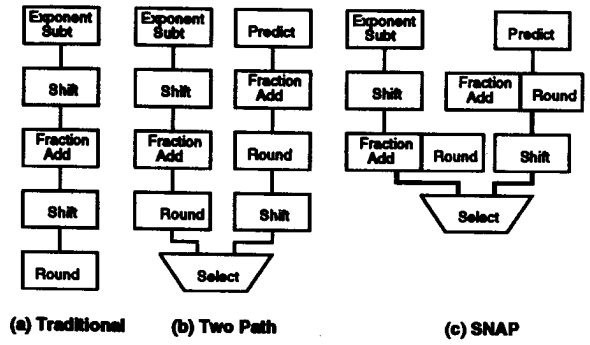


Figure 5: Floating Point Addition Algorithms.

Existing addition algorithms perform rounding by first computing the guard bit, round bit, and sticky bit. These bits then determine if the result has to be rounded and if so, the result is incremented. The SNAP algorithm sought to merge the increment step with the mantissa addition step by having the mantissa adder computing multiple results in advance and then selecting the final rounded result.

Table 2 shows the possible events that can happen in an addition and the events that the rounding algorithm has to consider. In the table, Ex and Mx stand for the exponent and mantissa of operand x. When the effective operation is addition, the mantissa adder computes $Ma + Mb$ and $Ma + Mb + 2$ and select the proper result. To compute $Ma + Mb + 2$, there is a row of half adder before the mantissa adder. The result $Ma + Mb + 2$ is needed when a right shift and a round is needed at the same time. The result $Ma + Mb + 1$ is needed when there is no right shift and a round is needed. $Ma + Mb + 1$ is computed by inverting the lsb of the result $Ma + Mb + 2$ or the result $Ma + Mb$. In either case, a simply inversion of the lsb suffices and no carry propagation is needed.

For effective subtraction, two cases need to be considered: $Ea > Eb$ and $Ea = Eb$. The $Ea > Eb$ case is quite straightforward because computing $Ma - Mb$ and $Ma - Mb + 1$ is enough to cover all the cases. A complication arises in the $Ea = Eb$ case and the result is negative because IEEE standard calls for a positive mantissa. This case can be handled as follows. Mb is first inverted and added to Ma. If the result is negative, the result itself is bit-inverted to obtain a possible mantissa. Again, no carry propagation is needed.

By careful management and integration of the paths, the overall hardware cost is only modestly increased over the state-of-the-art approach. A floating-point adder based upon this idea has been fabricated. Using a conventional 1-micron CMOS process, the floating-point addition delay is about 15–16 nanoseconds (IEEE standard floating-point addition).

2.3 Floating-point Multiplication

In a multiplication study, we developed a scheme based upon a redundant 3-bit Booth encoding of the multiplier [P1]. This redundant encoding reduces the number of partial products to be summed by one-third over a full (53 bit) tree partial product summation, but it avoids the requirement of forming plus or minus three times the multiplicand ($\pm 3M$) using a wide carry propagate adder.

The conventional Booth 3 algorithm assumes that the $3M$ multiple is available in non-redundant form. Before the partial products can be summed, a time consuming carry propagate addition is needed to produce this multiple. The Booth 3 algorithm with fully redundant partial products avoids the carry propagate addition, but has the equivalent of twice the number of partial products to sum. The new scheme tries to combine the small number of partial products from the conventional Booth 3 algorithm, with the ease of the hard multiple generation of the fully redundant Booth 3 algorithm.

The idea is to form the $3M$ multiple in a *partially redundant* form by using a series of small length adders, with no carry propagation between the adders (figure 6). If the adders are of sufficient length, the number of bits per partial product can approach the number in the non-redundant representation. This reduces the number of bits needing summation. If the adders are small enough, carries will not be propagated across large distances, and the small adders will be much faster than a full carry propagate adder. Also, less hardware is required due to the elimination of the logic which propagates carries between the small adders. The new algorithm can achieve performance parity while reducing the total area of the multiplier by 15-20%.

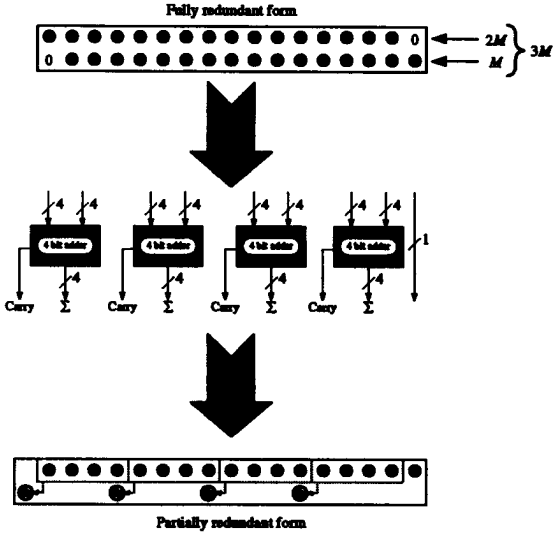


Figure 6: Computing 3M in a partially redundant form.

Coupled to the multiplier encoding, we have developed techniques that allow us to handle irregular counter implementations of the partial product tree. State-of-the-art

Operation	Exponent	Rounding need to consider
Effective add	don't care	- mantissa computes $Ma + Mb$ and $Ma + Mb + 2$ - rounding needs to account for a potential right shift.
Effective subtract	a. $Ea > Eb$	- mantissa inverts Mb and computes $Ma + Mb$ and $Ma + Mb + 1$. - rounding needs to account for a potential left shift.
	b. $Ea = Eb$	- mantissa inverts Mb and computes $Ma + Mb$ and $Ma + Mb + 1$ - mantissa inverts the results if the result is negative.
	c. $Ea < Eb$	- same as case (a) with Ea and Eb swapped.

Table 1: SNAP Addition Rounding Operations.

implementations of partial product trees have focused on 4–2 and similar approaches that provide a regularity in wiring of the partial product tree. We have shown that it is possible to use irregular implementations based upon 3–2 counters with the assistance of CAD tools, which automatically optimize the routing and path length through the partial product tree. Using the combined techniques of the redundant multiplier encoding, and the CAD-assisted layout of the partial product tree, we have implemented a full IEEE floating-point multiplier. This has been implemented in ECL with the sponsorship of SUN Microsystems. Figure 7 is a die photo of the multiplier. The delay of the multiplier is shown in figure 8.

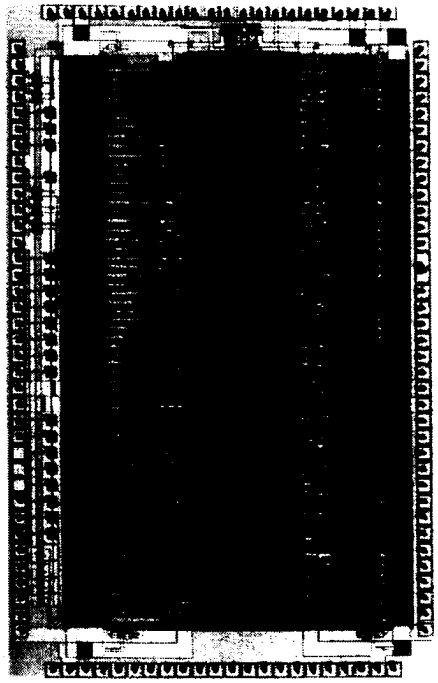


Figure 7: SNAP Multiplier Die Photo.
5 mm x 3 mm, 0.6 micron BiCMOS

2.4 Division and Higher Level Functions

Advances in the performance and correctness of division and higher level functions have been made in the SNAP research effort.

2.4.1 Approximation Theory

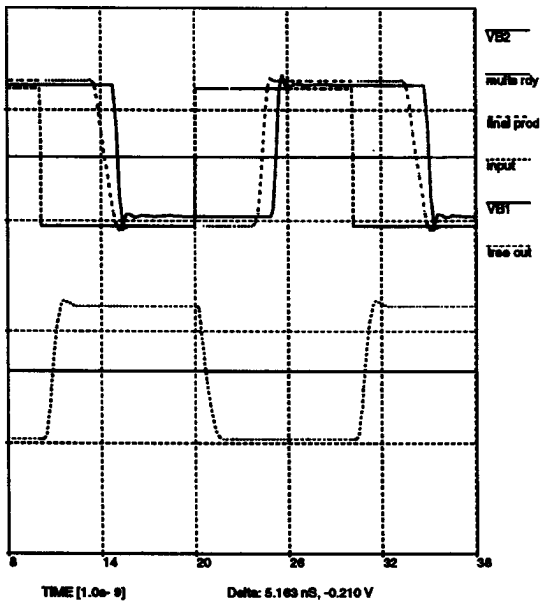
Significant accomplishments have been made in SNAP in the field of approximation theory and its application to starting approximations. Traditionally look-up tables have been used to accelerate division and other high-order arithmetic operations. In our study [P15], a non-traditional approach of back-solving logic equations is used. We further prior research in this area [1, 4], by creating a generic method for describing approximations and suggesting a unique implementation of reusing the hardware of a multiplier. In this manner, division and other high level functions can be accelerated with high-precision approximations at little cost.

These approximations are described in terms of a general Partial Product Array (PPA). A binary multiplication's PPA consists of the bits of the multiplier logically ANDed with the bits of the multiplicand. Each element in the array is a 2 input logical AND of a multiplier bit and a multiplicand bit. These elements are summed by a large counter tree and a carry propagate adder to form the product. Our work focused on creating the optimal PPA which describes an approximation to a function such as the reciprocal function. Instead of using array elements that are 2 way ANDs they were replaced with any type of Boolean logic gate. By using a similar array to the multiplier's PPA, the counter tree and adder could be reused. The use of a PPA to describe an approximation of a function and reusing an existing multiplier was founded by the SNAP project.

An example of deriving a PPA approximation for the reciprocal function is shown. First, the PPA of a multiplication ($B * Q = 0.111\cdots$) is back-solved in terms of the multiplier (Q).

$$
\begin{array}{cccccccccc|c}
 & & & & 0. & 1 & b_2 & b_3 & b_4 & b_5\cdots & = B \\
 & & & X & q_0. & q_1 & q_2 & q_3 & q_4 & \cdots & = Q \\
\hline
 & & & & & & \vdots & \vdots & \vdots & \vdots\cdots & \\
 & & & & & q_4 & b_2q_4 & b_3q_4 & b_4q_4 & b_5q_4\cdots & \\
 & & & & q_3 & b_2q_3 & b_3q_3 & b_4q_3 & b_5q_3 & \cdots & \\
 & & & q_2 & b_2q_2 & b_3q_2 & b_4q_2 & b_5q_2 & \cdots & & \\
 & & q_1 & b_2q_1 & b_3q_1 & b_4q_1 & b_5q_1 & \cdots & & & \\
 & q_0 & b_2q_0 & b_3q_0 & b_4q_0 & b_5q_0 & \cdots & & & & \\
\hline
0. & 1 & 1 & 1 & 1 & 1 & \cdots & & & & \approx 1.0
\end{array}
$$

By choosing the quotient digits appropriately in a redundant notation, each column of the PPA forms an indepen-



input ($t = 0$) final product ($t = 5.2$ns)

tree output ($t = 4.3$ns) $3x$, $1x$ multiples ready ($t = 0.6$ns)

Figure 8: An ECL 53×53^b (IEEE standard) multiply in 5.2 ns (latency). The 106^b addition at the multiply end takes 850 ps.

dent equation.

$$\begin{aligned} q_0 &= 1 \\ q_1 + b_2 q_0 &= 1 \\ q_2 + b_2 q_1 + b_3 q_0 &= 1 \\ q_3 + b_2 q_2 + b_3 q_1 + b_4 q_0 &= 1 \\ q_4 + b_2 q_3 + b_3 q_2 + b_4 q_1 + b_5 q_0 &= 1. \end{aligned}$$

These equations are solved to yield the following formulations for five digits of the quotient:

$$\begin{aligned} q_0 &= 1 \\ q_1 &= 1 - b_2 \\ q_2 &= 1 - b_3 \\ q_3 &= 1 - b_2 + 2b_2b_3 - b_3 - b_4 \\ q_4 &= 1 - b_2b_3 - b_4 + 2b_2b_4 - b_5. \end{aligned}$$

These equations are used to form the PPA of the approximation.

q_0	q_1	q_2	q_3	q_4
			$-b_4$	$-b_5$
			$-b_3$	$2b_2b_4$
			$2b_2b_3$	$-b_4$
	$-b_2$	$-b_3$	$-b_2$	$-b_2b_3$
1	1	1	1	1

Reduction of the Boolean elements is performed, such as applying the equivalency $1 - b_i = \overline{b_i}$ or representing $2b_j$ by b_j in the next more significant column. If a large multiplier is available such as one having 53 rows, a PPA can be developed which has a minimum of 12 bits correct for the reciprocal function. This high-precision approximation can be used as a starting approximation to accelerate a multiplicative division algorithm.

This technique has been developed for many operations such as the square root, logarithm, exponential, and several trigonometric functions. For the square root operation, a PPA with a minimum of 16 bits was determined. This enables a square root algorithm to require one less iteration than the typical 8 bit ROM implementations. Thus, several cycles can be saved by using the PPA approximation technique.

Further research is aimed at overcoming the irregularity of the PPAs in physical design. Also other techniques for deriving PPAs have been suggested such as using other approximation techniques expressed in PPA form. The optimal PPA for a function has yet to be determined, though PPAs which provide a minimum of 8 to 16 bits correct have been shown. PPA implementations create high precision approximations at very low cost with the latency of a multiplication.

2.4.2 Correctness of Division Algorithms

One currently hot topic in computer arithmetic is how to insure the correctness of high order arithmetic functions such as division. Rather than addressing one of simpliest algorithms for division such as non-restoring radix-4 division, we have focused our research on one the most difficult algorithms, the Goldschmidt algorithm. Non-restoring division algorithms and even the Newton-Raphson algorithm are self-correcting algorithms and can be proven correct by analyzing a generic iteration. The Goldschmidt algorithm accumulates errors each iteration and thus needs to be analyzed for all iterations. This algorithm is very attractive due to its quadratic convergence. The Newton-Raphson algorithm also is quadratically converging but the Goldschmidt algorithm has the advantage that some of the operations can be computed in parallel. Thus, our task has been to prove the correctness of implementations of the Goldschmidt algorithm.

We intend our research in this area to provide guidelines for designers that will insure the correctness of their implementations. We have tried to generalize the derivation for any length operands and any multiplier type. Currently we have finished the derivation for a rectangular multiplier of N by $N + G$ bits where N is the number of bits desired and G is the necessary guard bits. The relationship of G to table size and N has been determined. The proof of the

algorithm has paralleled the development of a S/390 CMOS microprocessor which will hopefully serve as an example application of this research. Though, this research will provide guidelines for the correctness of any implementation of the Goldschmidt algorithm.

2.5 MCM Technology [P4]

Considerable progress has been made in the area of high-performance packaging for SNAP. In particular, a technology for attaching and connecting chips onto a silicon membrane multichip module (MCM) has been demonstrated by S. Wong and his colleagues [P4]. This package offers a high density (over 500 per chip) of small (as small as $10 \times 10\ \mu m^2$), low resistance (less than 0.024 Ω/contact) and low parasitic interconnections between a chip and the substrate (see figure 9). In addition, controlled impedance striplines have been fabricated on the MCM substrate.

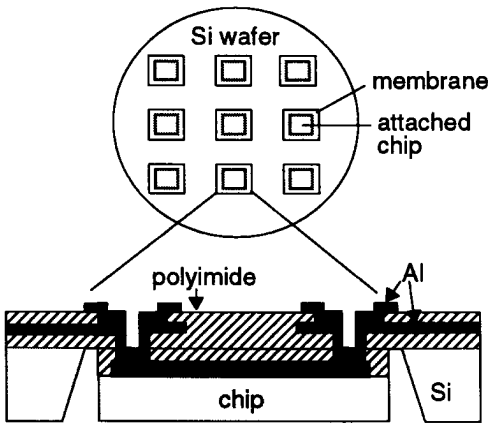


Figure 9: Membrane multi-chip module in Si. [P4]

The initial experimental striplines have been implemented in a three-layer ground-signal-ground configuration with aluminum conductors and polyimide dielectric. The sandwiched structure ensures a constant characteristic impedance as the stripline crosses the boundary between the MCM substrate and the chip. The stripline delay has been measured at 68 psec/cm which corresponds to a propagation velocity of 1.5×10^{10} cm/sec. A pulse rise time of 72 psec has been measured on a 1-cm long line.

3 Summary

By employing a unified, multidisciplinary approach to high-performance arithmetic research, SNAP has made significant progress toward the development of a gigaflop computing element. Improvements in clocking methods, algorithms, tools, and packaging technology have been demonstrated in implementations of high-performance VLSI adders, multipliers, and a vector unit.

Acknowledgments

This work was supported by NSF under contract MIP93-13701. Support for K. Nowka was provided by an ARPA Fellowship in High Performance Computing administered by the Institute for Advanced Computer Studies, University of Maryland. Support for E. Schwarz was provided by the IBM Resident Study Program.

Partial List of SNAP Publications

The following is a partial list of publications related to the SNAP project. Recent publications and information on the SNAP project and researchers may be obtained through the World Wide Web using the URL http://umunhum.stanford.edu.

[P1] G. Bewick. *Fast Multiplication: Algorithms and Implementation.* PhD thesis, Stanford University, 1994.

[P2] G. Bewick and M. Flynn. Binary Multiplication Using Partially Redundant Multiples. Technical report CSL-TR-92-528, Stanford University, June 1992.

[P3] G. Bewick, P. Song, G. De Micheli and M. Flynn, "Approaching a Nanosecond: A 32 Bit Adder," *ICCD, Proceedings of the International Conference on Circuit and Computer Design,* Rye, NY, pp. 221–226, October 1988.

[P4] W. Cheng, M. Beiley and S. Wong. "Membrane Multi-Chip Module Technology on Silicon," *IEEE Multi-Chip-Module Conference,* Santa Cruz, March 1993.

[P5] F. Klass. Balancing Circuits for Wave Pipelining. Technical report CSL-TR-92-549, Stanford University, October 1992.

[P6] E. F. Klass. "Wave Pipelining: Theoretical and Practical Issues in CMOS," PhD. thesis, Dept. of Elect. Engr., Delft Univ. of Technology, 1994.

[P7] K. Nowka and M. Flynn. "Environmental Limits on the Performance of CMOS Wave-Pipelined Circuits," Technical Report CSL-TR-94-600, Stanford Univ., Jan. 1994.

[P8] N. T. Quach and M. J. Flynn. High-Speed Addition in CMOS. Technical report CSL-TR-90-415, Stanford University, February, 1990.

[P9] N. T. Quach and M. J. Flynn. An Improved Algorithm for High-Speed Floating-Point Addition. Technical Report CSL-TR-90-442, Stanford University, August, 1990.

[P10] N. T. Quach and M. J. Flynn. Design and Implementation of the SNAP Floating-Point Adder. Technical Report CSL-TR-91-501, Stanford University, December 1991.

[P11] N. T. Quach and M. J. Flynn. A Radix-64 Floating-Point Divider. Technical report CSL-TR-92-529, Stanford University, June, 1992.

[P12] N. Quach and M. J. Flynn. "High-Speed Addition in CMOS." *IEEE Transactions on Computers*, Vol. 41, No. 12, December 1992.

[P13] N. Quach. *Reducing the Latency of Floating-Point Arithmetic Operations*. PhD thesis, Stanford University, 1993.

[P14] E. Schwarz and M. Flynn. Using a floating-point multiplier to sum signed Boolean elements. Technical report CSL-TR-92-540, Stanford University, August 1992.

[P15] E. M. Schwarz. *High Radix Algorithms for High-Order Arithmetic Operations*. PhD thesis, Stanford University, January 1993.

[P16] P. Song and G. De Micheli, "Circuit and Architecture Trade-offs for High-Speed Multiplication," *IEEE Journal on Solid State Circuits*, Vol. 26, No. 9, pp. 1184–1198, September 1991.

[P17] D. Wong, G. De Micheli, and M. Flynn. "Designing High-Performance Digital Circuits Using Wave Pipelining." *Proceedings of VLSI '89*, pp. 241–252, August 1989.

[P18] D. Wong, G. De Micheli, and M. Flynn. "Inserting Active Delay Elements to Achieve Wave Pipelining." *Proceedings of ICCAD '89*, pp. 270–273, November 1989.

[P19] D. Wong and M. Flynn. "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations." *Proceedings of the IEEE Symposium on Computer Arithmetic*, pp. 191–201, June 1991.

[P20] D. Wong. *Techniques for Designing High-Performance Digital Circuits Using Wave Pipelining*. PhD thesis, Stanford University, August 1991.

[P21] D. Wong, G. De Micheli, and M. Flynn. "A Bipolar Population Counter Using Wave Pipelining to Achieve 2.5x Normal Clock Frequency." *Journal of Solid-State Circuits*, Vol. 27, No. 5, pp. 745–753, May 1992.

[P22] D. Wong and M. Flynn. "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations." *IEEE Transactions on Computers*, 41(8):981–995, August 1992.

[P23] D. Wong, G. De Micheli and M. Flynn, "Algorithms for Designing High-Performance Digital Circuits Using Wave Pipelining," *IEEE Transactions on CAD/ICAS*, pp. 25–46, January 1993.

References

[1] R. Stefanelli. "A suggestion for a high-speed parallel binary divider," *IEEE Transactions on Computing*, Jan. 1972, pp. 42-55.

[2] H. Ling. "High-Speed Binary Adder." *IBM Journal of Research and Development*, Vol. 25, No. 2 and 3, pages 156–166, May 1981.

[3] M. P. Farmwald. *On the Design of High-Performance Digital Arithmetic Units*. PhD theses, Stanford University, August 1981.

[4] D. M. Mandelbaum. "A Method for Calculation of the Square Root Using Combinatorial Logic," *Journal of VLSI Signal Processing*, December 1993, pp. 233-242.