A 160nS 54bit CMOS Division Implementation Using Self-Timing and Symmetrically Overlapped SRT Stages

Ted E. Williams
HaL Computer Systems
1315 Dell Avenue
Campbell, CA, 95008

Mark A. Horowitz
Center for Integrated Systems
Stanford University
Stanford, CA, 94305

Abstract

A full-custom VLSI chip demonstrates an arithmetic implementation for computing the mantissa of a 54bit (floating-point double-precision) division operation in 45nS to 160nS, depending on the data. The design uses self-timing to avoid the need to partition logic into clock cycles and the need for high-speed clocks. Self-timing allows the circuits to iterate with no overhead over the pure combinational logic delays. It also allows a greater efficiency symmetric overlapped execution of the SRT stages because of "dynamic" path ordering. The design has several other performance enhancements, and this paper tabulates their effect on the performance.

1 Introduction

Previous division implementations have generally tried to attain high-performance by increasing the complexity of the logic function performed in each "cycle." Higher radix arithmetic can utilize this complexity to reduce the number of clock cycles required in clocked designs [1], [2]. But as technology improves, it is increasingly more difficult to fully utilize all clock cycles. An alternate approach is to avoid clock cycle limitations altogether by using self-timed logic. This paper describes several implementation issues in the design of a self-timed CMOS divider, including the methodology used to eliminate latch delays between stages in an iterating ring, a modification to the quotient selection logic that narrows the remainder datapath, immediate done indication for repeating quotients, and dynamic overlapping of action between a stage and its neighbors *without* a pre-defined grouping into pairs. As in RISC processor design [3], each improvement can be judged by a version of Amdahl's law:

$$\frac{100}{T} = \frac{1}{f}\left(1 + \frac{100}{S}\right) - 1 \qquad (1)$$

where S is the percentage speedup of one part, f is the fraction of the total delay attributable to that part, and T is the percentage improvement in the total performance.

Estimates for self-timed SRT division designs suggest a radix 2 approach obtains the best performance because its simplicity allows fast stages in a reasonable area. Measurements from fabricated and tested CMOS VLSI parts verify the ideas in this paper and demonstrate high-performance without special technology.

2 Self-Timed Methodology

Self-timed components avoid the need of distributing global clocks and the need of allowing for clock-skew in synchronous systems. Variances and data dependencies in delays can be used advantageously because each component can begin when its required operands actually arrive rather than always waiting for worst-case timing. The performance is also the best possible for the actual environmental conditions, without needing to de-rate specifications to allow additional margins for the conceived ranges of power supply voltage, die temperature, and fabrication spread.

Circuits can achieve self-timed operation by carefully matching delays between components or by encoding completion information within data signals. An example of the former method is a self-timed multiplier [6] chip, which uses a matched on-chip clock generator to provide a clock for the internal blocks. In contrast, our chip demonstrates the latter method by using local completion detectors and handshaking between fully asynchronous blocks and operates correctly for any values of gate delays. Completion-information is embedded in the data throughout the design by using a pair of wires for each bit. Called a "dual-monotonic pair," the wires transmit both a value and a timing signal by using the protocol in Table 1.

Precharged function blocks use merged n-channel pull-down networks to choose which of the wires in each pair to set high when input data arrives. Completion detector NOR gates examine the dual-monotonic pairs to generate local done signals used to control the precharge signals. If

Wire A^T	Wire A^F	Signal A
0	0	Reset = Not Ready
0	1	Evaluated FALSE
1	0	Evaluated TRUE
1	1	Not used = Never occurs

Table 1: Encodings on a Dual-monotonic Wire Pair

the circuit is embedded in a synchronous system, the chip Done signal can be used to stretch clock cycles as in [12], or to indicate on which clock cycle the system may take the outputs from the self-timed chip.

Because the iterative steps of SRT division need a repetitive structure, the core of our chip is a ring of five stages that iterates completely under self-timed control. The ring achieves minimal latency operation because it has directly concatenated precharged logic blocks in a looped domino chain, as shown in Figure 1, without any explicit latches. Each stage "falls like a domino" because it evaluates as soon as its predecessor provides valid data, without waiting for control or clocking. The looped chain is thus like a ring-oscillator that computes. This is possible because the precharge (reset) signals for each block are controlled separately so each block can be used as an implicit latch without adding any additional transistors. The self-timed control precharges each block after data has passed it, and removes the precharge signal (thereby enabling its evaluation), before data has looped around to its inputs again. The critical path goes solely through the combinational data elements. Thus, the data flows continually at the same rate it would flow through an "unwrapped" combinational array implementing the same functions. While previous asynchronous approaches [5], [9] have suffered delays due to handshaking control, this method of self-timing adds zero control overhead [11] to the latency of the raw function computation.

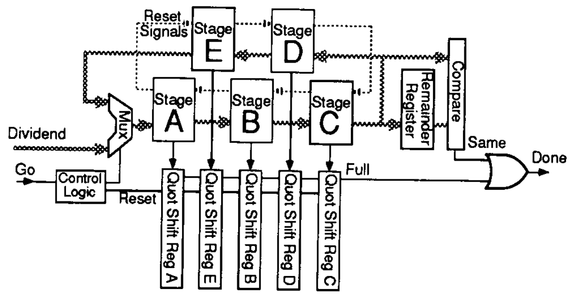

Figure 1: Block diagram for the division circuit using a ring of domino stages which iterates using self-timing. Dotted lines show control signals; shaded lines are dual-monotonic datapaths.

3 Quotient Digit Selection

The blocks of each stage in the iterative ring implement one step of a modified radix 2 SRT division algorithm:

$$P_{i+1} = 2P_i - Dq_i \qquad (2)$$

where D is the divisor, P_i is the partial remainder at stage i, and q_i is the quotient digit selected by that stage. The probabilistic distribution of quotient digits is not uniform due to the asymmetry of the two's complement number system. We wrote a computer program to collect statistics by enumerating all possible data patterns in the most significant bit positions. Because the SRT algorithm deals only with the most significant bits, the collected statistics became asymptotic for input data patterns exceeding about 9 bits in significance. The asymptotic values are therefore the correct statistics for uniformly distributed inputs of all larger mantissa widths. Table 2 shows the asymptotic distribution of the values of the partial remainder approximation and the quotient digits selected for radix 2 division.

The asymmetric distribution of quotient digits can be used to speed the more frequently used circuit paths since the self-timed implementation can take advantage of the improvement. For example, our circuit used different sized transistors in the replicated short adders (replicated because of the overlapped execution feature to be discussed in Section 5) preceding the next quotient digit selection logic. We designed the $q_i = -1$ adder arm with larger transistors because the critical path goes through it most frequently. We did not use larger transistors in the less frequently chosen $q_i = +1$ arm because the additional loading on the wires also going to the $q_i = -1$ arm would have actually decreased the total performance. The effect of this transistor optimization decreases the expected value of the delay of driving these adder inputs by about 30%, resulting in a 4% improvement in the total divider performance.

When the most negative or two most positive possible values for the partial remainder approximation occur, the selected quotient digit does not change the sign of the next remainder. So, in these cases the next quotient digit can be chosen in advance to be the same as the current quotient

Remainder	- 4	-3	-2	-1	0	+1	+2	+3
Probability	3.2%	11%	28%	35%	18%	4.6%	.40%	.05%
Quot Digit		-1		0		+1		
Probability		42%		35%		23%		

Table 2: Statistics for Radix 2 SRT division.

digit. Table 2 shows the most negative approximated remainder value occurs with a 3% probability and the two most positive values occur with less than a 1% probability. Based on these statistics, we chose to implement a force-ahead for only the negative case with the quotient selection logic equations:

$$q_i = +1 \quad \text{if} \quad \hat{P}_i \geq 0 \quad \text{and} \quad F_{i-1} = 0 \tag{3}$$

$$q_i = 0 \quad \text{if} \quad \hat{P}_i = -1 \quad \text{and} \quad F_{i-1} = 0 \tag{4}$$

$$q_i = -1 \quad \text{if} \quad \hat{P}_i \leq -2 \quad \text{or} \quad F_{i-1} = 1 \tag{5}$$

$$F_i = 1 \text{ if } \hat{P}_i = -4 \text{ or } \left(F_{i-1}=1 \text{ and } \left[\hat{P}_{i-1}\right]_{\text{msb}} = 0\right) \tag{6}$$

where $\hat{P}_i$ is the approximated partial remainder at stage i, and F_i is the flag set to force the next quotient digit. Allowing the quotient digit selection logic to leap ahead one digit in 3% of all selections improves the total performance by about 1% in itself, but it also enables the additional optimization described in the next paragraph by reducing the required size of the remainder datapath and the adder widths.

The possible shifted remainder values in any maximally redundant radix r SRT division step (just before reduction by a quotient digit) range from $-rD$ to $+rD$. When the remainder is represented in carry-save form as the sum of two numbers each in two's complement form, it is possible for each of these numbers to be in the range $-2rD$ to $+2rD$ and previous implementations have required the datapath be wide enough to hold the bits necessary to represent those numbers. However, the approximation of the remainder, formed by a short CPA combining the sum and carry bits, must still have its actual value in the range $-rD$-$2U$ to $+rD$ where U is the maximum value of the unpropagated bits in either the sum or the carry terms. Since the unpropagated bits are always positive, U is always positive. If the minimum representable number were only $-rD$, then values in the range $-rD$-$2U$ to $-rD$ would be aliased as positive numbers in the range $+rD$-$2U$ to $+rD$ because of the sign bit "falling off." The incorrect interpretation of the approximation would, of course, cause the wrong quotient digit to be chosen. However, for radix 2, this can only happen in the cases in which the remainder is at the extreme of its possible valid range where it was possible to predict two quotient digits in advance. The quotient selection logic in equations (3)-(5) never misinterprets an aliased remainder as a positive number even when the sign bit falls off. Equation (6) sets the force-next-digit flag when the most negative quotient digit occurs or if the flag was set before and the remainder is already aliased. The partial remainder need only be able to represent numbers in the range $-2D$ to $+2D$, thus narrowing the datapath and shortening by one bit the adders used to form the irredundant approximation.

The trimming of the datapath is only applicable in maximally redundant SRT algorithms because the minimum approximated remainder $-rD\frac{\rho}{r-1}$ $-2U$ of lower redundancy choices (such as ρ=2, $q_i \in \{-2,-1,0,1,2\}$ in radix 4) can be represented in the same number of bits as $-rD\frac{\rho}{r-1}$. For radix 2, the only choice of redundancy, ρ=1, is maximally redundant, and therefore the trimming can be applied to make the adders need only 3 bits instead of the 4 bits used in previous implementations [4]. Since the logic to force both the current and next quotient digits has about 40% less delay than the delay of an extra bit in the adders and this accounts for a sixth of the total delay through a stage, the net total performance is improved by 5%.

4 Quotient Accumulation and Early Done Detection

As the quotient selection logic in the five stages of the self-timed ring output quotient bits, they are collected by five separate asynchronous shift registers composed of the cells shown in Figure 2. Between each valid quotient digit sent into a shift register, a reset spacer is sent to separate the digits. The ring loops a maximum of 11 times to fill the five shift registers with a total of 55 bits for a double-precision result. On each iteration, the remainder comparison on the right side of Figure 1 determines if the partial remainder has remained unchanged during the last iteration:

$$P_{i+5} = P_i \tag{7}$$

If the remainder repeats, then subsequent remainders and quotient digits will also repeat:

$$P_{i+10} = P_{i+5} = P_i, \tag{8}$$
$$q_{i+5} = q_i,$$
$$q_{i+6} = q_{i+1},$$
$$q_{i+7} = q_{i+2},$$
$$q_{i+8} = q_{i+3},$$
$$q_{i+9} = q_{i+4}$$

Since there is no need to compute the repeating digits again, the iterations terminate and the division done signal is generated early. Even when the iterations terminate early, the full quotient is immediately available from the shift registers because the asynchronous design using C-elements correctly ripples the quotient digits to their final positions as they arrive rather than waiting for a fixed number of clocks as would a synchronous shift register. The repeated quotient digits are also immediately available because they fill all of the positions behind each new quotient digit as it ripples through a shift register. Only

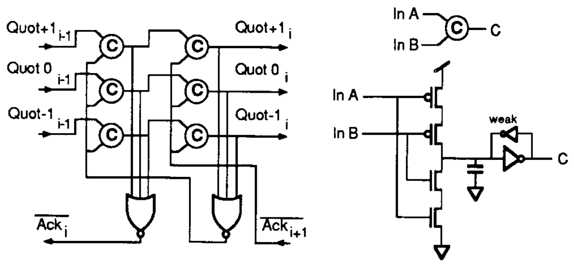

Figure 2: A cell of the asynchronous shift registers for capturing quotient digits on a triple-monotonic wire set. Each quotient digit arrives when one of the three input wires is set high, and is followed by a "spacer," where all three are again low. A static C-element is defined at right.

after the remainder comparison determines more iterations will be needed are reset spacers sent into the shift registers to wipe out the repeated digits and prepare the shift registers to accept the next digits. This interlocking of the reset spacers does not add any delay to the overall computation because it occurs in parallel with the evaluation of other quotient digits.

The effect on performance of detecting repeating quotients and finishing early is dependent upon the distribution of input operands. Data from some algorithmic applications may be likely to have more round numbers, and data from an external input, like a sensor, may be uniformly distributed only within a limited precision. For example, the early done detection will speedup 12% of the cases in a uniform distribution of 8bit input operands, for a total performance improvement of 9%. A typical division instruction operand mix might get half that speedup.

The final quotient can be rounded correctly even when the iterations terminate early. In both the early done and the normal case requiring all of the iterations, the remainder at the stage where the iterations stopped can be sent through a carry-look-ahead adder (CLA) to determine its sign. If the remainder is negative, the quotient must be decremented at the least-significant bit position to which the remainder corresponds. This operation and the conversion of the redundant quotient into a standard binary form can be performed by a carry-select-adder with multiple carry chains operating in parallel. The different rounding possibilities and the remainder sign select the correct CLA output. Thus, after the iterations terminate, only a single CLA delay is required to resolve both the final remainder and rounded quotient.

5 Dynamic Overlapping of Stages

Since the carry-save adders (CSA) used for the computation of partial remainders provide only a redundant result, short 3bit carry-propagate adders (CPA) are needed to form an irredundant remainder approximation for input to the quotient digit selection logic. While the traditional SRT algorithm is purely sequential, the implementation here, shown in Figure 3, overlaps the execution of adjacent stages by replicating the CPAs for each possible quotient digit so they can begin operation before the actual quotient digit arrives and chooses the correct branch. Two of the three CPAs are also preceded by CSAs to combine the remainder respectively with the divisor and the negation of the divisor. Actually, the logic for the sum terms in these two CSAs is shared because the dual-monotonic data convention already provides both the true and complement of each bit. The carry terms cannot be shared. While the arm for a zero quotient digit still requires a CPA to combine the sum and carry terms of the redundant representation, the zero arm requires no preceding CSA.

The self-timed control for each stage is also shown in Figure 3 and uses C-elements to combine the completion-detector signals to produce the signals that reset each block as soon as its outputs have been consumed. Since these C-elements are never in the overall critical path they introduce zero overhead.

Figure 4 shows the concatenation of the data elements for any two adjoining stages. The overlapping of execution allows the average delay through a stage to be the average rather than the sum of the propagation delays through the remainder and quotient digit selection paths. The

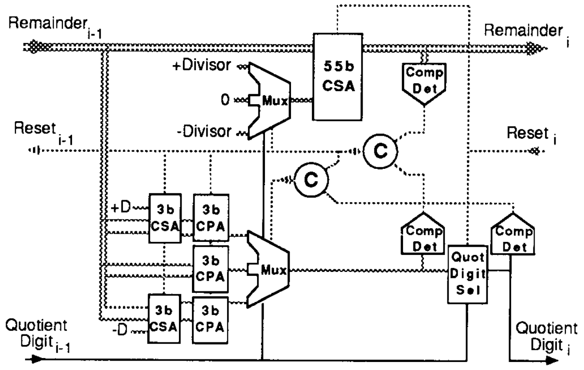

Figure 3: Internal structure of each stage in the ring implementing an SRT division step with overlapped execution and self-timed reset (precharge) control.

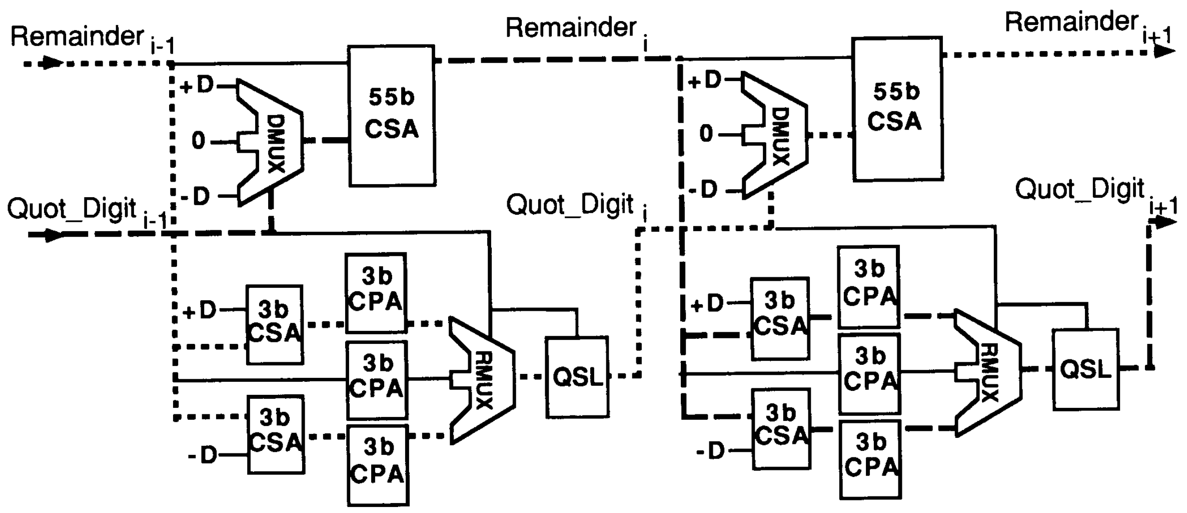

Figure 4: Dataflow through a pair of stages in the present overlapped execution scheme.

overlapping of these paths can be abstracted to the arrangement of overlapping blocks shown in Figure 5.

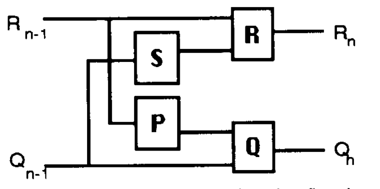

Figure 5: Model for overlapped execution dataflow in each stage

When these blocks are self-timed and therefore operate as soon as their required operands arrive, the average delay per stage[1] in a chain of identical stages of the overlapped arrangement is

$$\frac{1}{2}\left\{ P+Q+R+S + \max\left[0, \text{abs}(R-Q)-(P+S)\right] \right\} \quad (9)$$

The last term is usually negative and drops out, giving a performance increase due to the factor of $\frac{1}{2}$ in front. In the overlapping of the stages for SRT division, the delay of block P in the quotient selection path is the largest of the delays because it contains the CPAs. The overlapping reduces by one-half the effect of the delay in block P on the total delay. When the added mux delays are taken into account, the overlapping of the stages in a radix 2 design increases performance by 35% over a standard sequential arrangement of the same blocks.

The structure of our overlapping scheme results in a data wavefront that leapfrogs down the succession of stages. If the critical path goes through the quotient selection path in one stage, it will likely go through the partial remainder path in the next stage, and vice-versa. However, data-dependent variances in delays make it possible for the overall minimal critical path to go through the same path in two adjacent stages. Delay variances arise because of the varying number of bits propagated in the carry chains, the occurrence of some zero quotient digits, and the cases in which a negative quotient digit can be selected in advance when a single stage can determine two quotient digits. The self-timing of the datapath ensures data always flows through the minimal critical path. Previous overlapped execution schemes such as the one from [7] have pre-grouped stages into pairs. Whereas our approach makes all the stages symmetric, the scheme in Figure 6 does not replicate the first CPA. This lack of symmetry

[1] An exact analysis of the delays of Figure 5 shows that the time the nth R block finishes in a chain of identical stages is

$$R_n = \frac{n}{2}(P+Q+R+S) + \max\left[S+\frac{n}{2}(R-S-P-Q),\ \left(\frac{n}{2} - 1\right)(Q-P-R-S),\ \frac{e}{2}(Q-P-R-S),\ R-P-Q+\frac{e}{2}(Q+P-R+S)\right]$$

when the chain's inputs start at $R_0 = Q_0 = 0$ and where $e \equiv \left\{ \begin{array}{l} 1 \text{ if } n \text{ odd} \\ 0 \text{ if } n \text{ even} \end{array} \right\}$.

Symmetrically, the nth Q block finishes at time

$$Q_n = \frac{n}{2}(P+Q+R+S) + \max\left[P+\frac{n}{2}(Q-P-R-S),\ \left(\frac{n}{2} - 1\right)(R-S-P-Q),\ \frac{e}{2}(R-S-P-Q),\ Q-R-S+\frac{e}{2}(R+S+P-Q)\right].$$

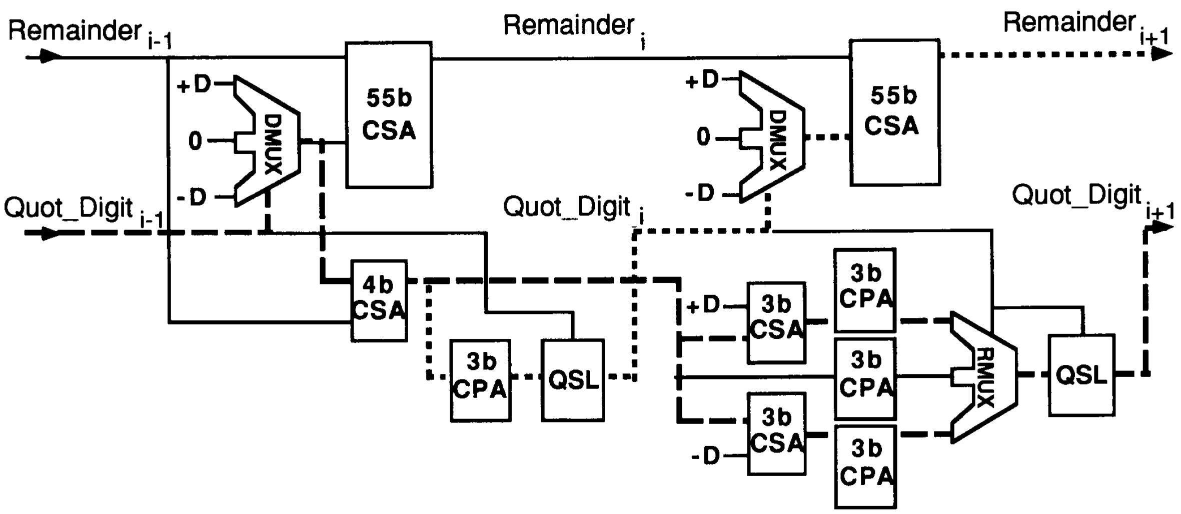

Figure 6: Other implementations used asymmetric dataflows that enforced a specific grouping of the stages into pairs.

makes the critical path go through the same blocks every time. Such a grouping loses about 5% in performance because it enforces extra waiting in some cases and does not achieve the additional path minimization possible with self-timed overlapped execution, which allows a "dynamic" adjustment of the execution order instead of a static grouping of the stages into pairs.

The latch-free methodology introduced in Section 2 can attain the full benefit of overlapped execution. Because the remainder and quotient digits passed between the stages are skewed, any imposed latch with a common clock would add the delay of that skew, in addition to the propagation delay through the latch and added margin to tolerate skew between the clock and data. The propagation delay through a latch would be about 15% of the other delays through a pair of stages; the required clock skew margin 5%; and the skew between remainder and quotient digits is about 10%. So, since adding latches would increase the delay through a pair of stages by a total of 30%, using self-timing to remove the latches results in a 23% performance improvement.

6 Choice of Radix

In an integrated circuit implementation, there are always tradeoffs between speed, power, and chip area. Self-timing provides a useful means of comparing performance without constraints from fixed clock cycles, or delays from latches or input/output considerations. In particular, VLSI implementations of different radix approaches for SRT division can tradeoff complexity amongst the various arithmetic components [8]. This section states fair relative comparisons by using the delay of a gate with unity fan-in and fan-out as the basic speed unit. Real gates have delays many times this basic unit because of stacked transistors (fan-in) and loading (fan-out). The delays stated for blocks include the delays due to buffers on inputs and due to the loading of outputs.

Estimates for SRT radix 2 and radix 4 design choices, with and without overlapped execution, are based on circuit simulations and have been updated and calibrated with measurements from the fabricated chips. Table 3 summarizes the comparisons of these implementation choices. All of the parameters are for an implementation with a minimal-latency self-timed ring. The radix 2 designs require five stages in the ring, while the radix 4 designs only require four stages since the propagation delay of each individual stage is longer [10]. None of the figures include the final 55bit CLA required for rounding and converting the redundant representations back to standard binary. The right two columns contain numbers specific to the CMOS fabrication technology available.

Table 3 shows that radix 2 is slightly better than radix 4 when neither have overlapped execution. This is because the additional complexity of the radix 4 quotient selection does not quite justify the use of radix 4 when clocking does not need to be considered. However, if a design were clocked, the difficulty of supplying a clock at twice the frequency might make radix 4 preferable. Overlapped execution in either radix 2 or radix 4 gives a significant performance increase, about 30% for radix 2 and 35% for radix 4. The key advantage of the self-timed overlapped execution style here is that the average critical path per

Radix & Style (OverExec = Overlapped Execution)	Average Critical Path per Pair of Stages in Unity fan-in, Unity fan-out gate delays	Unity fan in/out Gate Delays per Quot Bit	Latency for 54 bits with 250pS Unit gate delays	Silicon Area in 1.2μ Technology
Radix 2	2 (CPA3+QSL3+DMUX3+CSA55) 2 (5.7 + 3.8 + 2.8 + 4.5) = 33.6	16.8	225 nS	7 mm^2
Radix 2, OverExec	CSA3+CPA3+RMUX3+QSL3+DMUX3+CSA55 3.9 + 4.9 + 3.5 + 3.8 + 2.8 + 4.7 = 23.6	11.8	160 nS	10 mm^2
Radix 4	2 (CPA7+QSL5+DMUX5+CSA56) 2 (11 + 16 + 3.5 + 4.5) = 70.0	17.5	235 nS	12 mm^2
Radix 4, OverExec	CSA7+CPA7+RMUX5+QSL5+DMUX5+CSA56 3.9 + 10 + 5.0+ 16 + 3.5 + 6.2 = 44.6	11.2	150 nS	18 mm^2

Table 3: Tradeoffs in Speed and Area for different implementation approaches.

stage has a factor of $\frac{1}{2}$ times the delay from the CPAs and quotient selection logic. Since, for higher radices, these components occupy bigger proportions of the total delay, the effect of overlapped execution is even more significant for radix 4. To summarize, with overlapped execution, radix 4 is faster than radix 2, but the area cost is much higher because of the replication of the carry-propagate adders. Not only are five adders required instead of only three, but they are also larger. Still higher radices, such as radix 8, would accentuate these tradeoff effects. Overlapped execution would have an even greater percentage reduction in delay, but at a formidable cost in area.

7 Test Results

The tradeoffs discussed in the previous section led to the choice of implementing radix 2 with overlapped execution. We used MAGIC for the full-custom layout of the self-timed divider. The 45K transistor design was fabricated in 1.2μ CMOS technology through MOSIS. The ring's five stages are columns which are mirrored appropriately to weave the datapath and achieve equal path lengths. By careful cell design and over-cell routing, the area cost of using two wires for each bit in dual-monotonic pairs added only about 20% to the total area. The die photo shown in Figure 7 shows an active area of 9.7mm^2, which contains test registers surrounding a core iterating ring in the central 6.8mm^2.

The chips generate the correct data outputs over a wide range of operating conditions. The actual operating conditions determine the actual performance, and Figure 8 shows measured speeds for various voltages and temperatures. For operation at 5V and 35°C ambient temperature, quotient bits are produced internally every 2.8nS. The measured total latency for a 54bit quotient is 160nS for worst-case data, and 45nS for best-case data requiring only two ring iterations. This large data-dependency in timing shows the effect of the early-done detection.

Since exponent logic can operate in parallel, a complete floating-point division operation could be formed by adding the measured delays for the mantissa operation to

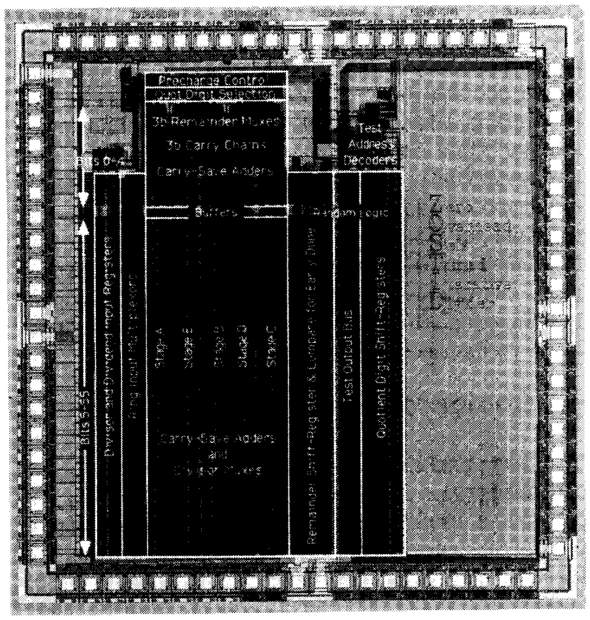

Figure 7: Micrograph of the zero-overhead self-timed 54bit divider in 1.2μ CMOS technology.

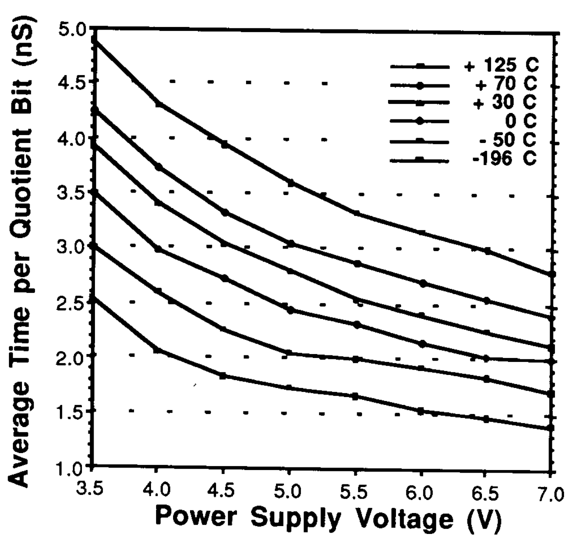

Figure 8: Measured performance per quotient digit at various voltages and temperatures

only the time required for the additional 55bit CLA to round and resolve out of a redundant representation. In the same technology, this delay would likely be 4nS to 8nS.

8 Summary

The set of enhancements in this paper and the performance gained from each of them are summarized in Table 4. The total effect of all the performance enhancements provides a factor of two increase in performance due to architectural improvements over a "straightforward" SRT approach. Moreover, the table does not quantify some advantages such as the benefit that self-timing does not require the cost of global clock distribution.

The self-timed design methods in this paper enable an iterative circuit to compute at the speed of a combinational array while only requiring a fraction of its silicon area. Since the design produces a done indication, the outputs can be used as soon as they are available without waiting for worst case margins over the ranges of possible data values, temperature, voltage, and fabrication spread. Self-timing allows an overlapping of the execution of the SRT stages to attain a dynamically adjusting data-dependent minimal critical path. Measurements on fabricated parts verified the architectural techniques achieve high-performance even with ordinary CMOS technology.

Enhancement	Speedup
Resizing CSA inputs based on statistics	4%
Forcing $q_{i+1} = -1$ when possible	1%
Shortening Remainder Approx CPA by 1 bit	5%
Early Done Detection of Repeating Quotients	4%
Overlapped Stage Execution	35%
Dynamic Overlapping	5%
Latch-Free Self-Timing	23%
Total Effect of Performance Enhancements	100%

Table 4: Summary of gains from performance enhancements

References:

[1] D. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainder," *IEEE Tran. on Computers*, vol. 17, no. 10, pp. 925-934, Oct. 1968.

[2] J. Fandrianto, "Algorithm for High Speed Shared Radix 8 Division and Radix 8 Square Root," Proc. 9th Symp. Comp. Arith., pp. 68-75, Sept. 1989.

[3] J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach*, Palo Alto: Morgan Kaufmann, pp. 5-12, 1990.

[4] W. McAllister , D. Zuras, "An nMOS 64b Floating-Point Chip Set," ISSCC Digest of Technical Papers, pp. 34-35, Feb. 1986.

[5] T. Meng, R. Brodersen, D. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Tran. on Computer-Aided Design*, vol. 8, no. 8, pp. 1185-1205, Nov. 1989.

[6] M. Santoro, M. Horowitz, "SPIM: A Pipelined 64x64-bit Iterative Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 487-493, Apr. 1989.

[7] G. Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages," Proc. 7th Symp. Comp. Arith., pp. 64-71, June 1985.

[8] T. Williams, M. Horowitz, "SRT Division Diagrams and Their Usage in Designing Custom Integrated Circuits for Division," Stanford Tech Report CSL-TR-87-326, Nov. 1986.

[9] T. Williams, M. Horowitz, et. al., "A Self-Timed Chip for Division," Proc. Stanford Conference on Advanced Research in VLSI, pp. 75-95, Mar. 1987.

[10] T. Williams, "Latency and Throughput Tradeoffs in Self-Timed Asynchronous Pipelines and Rings," Stanford Tech Report CSL-TR-90-431, Aug. 1990.

[11] T. Williams, M. Horowitz, "A Zero-Overhead Self-Timed 160nS 54b CMOS Divider," ISSCC Digest of Technical Papers, pp. 98-99, Feb. 1991.

[12] G. Wolrich, E. McLellan, et.al., "A High Performance Floating-Point Coprocessor," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 5, pp. 690-696, Oct. 1984.