THE DESIGN AND IMPLEMENTATION OF A HIGH-PERFORMANCE FLOATING-POINT DIVIDER

Stuart Oberman, Nhon Quach, and Michael Flynn

Technical Report: CSL-TR-94-599

January 1994

This work was supported by NSF under contract MIP88-22961, using facilities supported by NASA contract NAG2-842.

THE DESIGN AND IMPLEMENTATION OF A HIGH-PERFORMANCE FLOATING-POINT DIVIDER

by

Stuart Oberman, Nhon Quach, and Michael Flynn

Technical Report: CSL-TR-94-599

January 1994

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

Abstract

The increasing computation requirements of modern computer applications have stimulated a large interest in developing extremely high-performance floating-point dividers. A variety of division algorithms are available, with SRT being utilized in many computer systems. A careful analysis of SRT divider topologies has demonstrated that a relatively simple divider designed in an aggressive circuit style can achieve extremely high performance. Further, an aggressive circuit implementation can minimize many of the performance advantages of more complex divider algorithms. This paper presents the tradeoffs of the different divider topologies, the design of the divider, and performance results.

Key Words and Phrases: Floating-point, SRT division, domino logic

Contents

1 **Introduction** 1

2 Division 1

2.1 SRT Overview 1

2.2 SRT Topologies 2

3 Mapping to Domino Logic 4

4 Divider Design 6

4.1 Logic and Circuit Design 6

4.2 Layout/Area 9

5 Results 11

5.1 Single Stage Analysis 11

5.2 Multiple Stage Analysis 11

6 Comparison 13

7 Conclusions 13

List of Figures

1 Basic SRT Scheme 3
2 Two-Bank Scheme 3
3 Simplified Operand Scaling Scheme 4
4 Domino CSA 5
5 Allowable quotient digits in each SRT region 7
6 Layout of a CSA 10
7 Multiple stage analysis 12

List of Tables

1 Critical path comparison 6
2 Look-up table comparison 8
3 Complexity effects of optimizations 9
4 Evaluation Times 11
5 Latencies for varying number of stages 11
6 Performance comparison 13

1 Introduction

Modern computer applications have increased in their computation complexity in recent years. The development of high-speed floating-point arithmetic is a requirement to meet the computation demands of modern applications which have increased in their computation complexity in recent years. The emphasis on high performance graphics in workstations has placed further demands on the computation abilities of processors. Furthermore, the industry-wide usage of performance benchmarks, such as SPECMarks, forces processor designers to pay particular attention to floating-point computation. With die sizes of processors also increasing, processor designers can realistically include aggressive floating-point units directly on-chip to achieve even higher performance.

Floating-point intensive applications typically exhibit a distribution of operations similar to the following: 45% addition, 45% multiplication, 5% division, and 3% square- root. Past research and development efforts have placed a large emphasis on designing high-performance floating-point adders and multipliers. Thus, these two areas have many fast and efficient solutions. Division, though, has not seen a similar research effort. Being the third largest contributor to floating-point arithmetic, an increasing emphasis on high-performance division is vital to producing fast and competitive processors. Many division algorithms exist which can provide high-performance. Moreover, in a standard static CMOS implementation, the more complex algorithms often exhibit markedly higher performance than simpler algorithms. For example, Newton-Raphson implementations can achieve extremely low latency, as Newton-Raphson division converges to a result quadratically. This performance comes at the price of additional hardware, complexity, and accuracy.

In this paper, a variety of SRT division schemes will be compared. SRT is often used in workstations because it provides reasonable performance and it is relatively simple [1, 2]. The tradeoffs of these schemes will be analyzed, and the design and performance of a new divider will be presented. It will be shown that by mapping a simple topology to an aggressive circuit style in conjunction with some logic optimization, a high-performance CMOS divider can be implemented. Analysis of the latencies achievable in a dynamic CMOS circuit implementation demonstrates that the more complex algorithms do not show a distinct performance advantage. Rather, sufficiently high-performance can be obtained at a lower complexity cost by using a simple divider implementation.

2 Division

2.1 SRT Overview

SRT is a non-restoring division algorithm that is basically a trial and error process. It utilizes the following relationship:

$$P_{j+1} = rP_j - q_{j+1}D$$

This equation models the paper-and-pencil method of division. To calculate a next partial remainder, the divisor is multiplied by the next quotient digit, and the result is subtracted

from the product of the last partial remainder, or dividend for the first iteration, and a radix r. The next quotient digit is obtained by supplying a fixed number of bits from the last partial remainder, approximately 8 bits for a radix-4 divider, to a look-up table. By choosing a radix to be a power of 2, the product of the radix and the last partial remainder can be formed by shifting. Similarly, the various products of the divisor multiplied by the next quotient digit can be formed by multiplexing different multiples of the divisor. However, the problem with this basic scheme is that it requires a full-width subtractor. Consequently, this scheme can be very slow.

In order to improve upon this basic scheme, some redundancy is introduced into the algorithm. An extra constraint is added to provide redundancy:

$$|P_{j+1}| < kD$$

where k = n / (r - 1) and n is the number of positive allowed digits for the next quotient digit. A design tradeoff can be noted in this relationship. By using a large number of allowed digits for the next quotient digit, and thus a large value for k, a smaller look-up table is required, and thus the complexity and latency of the table look-up can be reduced. However, choosing a smaller number of allowed digits for the quotient simplifies the generation of the multiple of the divisor. Multiples that are powers of two can be formed by simply shifting. If a multiple is required that is not a power of two (e.g. three), an additional operation such as addition may also be required, which can add to the complexity and latency of the divisor multiple generating process. Thus, the complexity of the look-up table and that of generating multiples of the divisor must be balanced.

To increase the performance of the subtraction process, the partial remainders themselves are kept in a redundant form. Instead of using full-width adders that require carry propagation to compute partial remainders, a series of carry-save adders are used to compute the next partial remainder in the delay of a single full-adder. In this way, the conversion to a non-redundant form only needs to be done after the final iteration using a full width adder.

2.2 SRT Topologies

In this study, three different SRT topologies were analyzed for cost and performance. Figure 1 shows the topology of a Basic SRT implementation [3]. In this scheme, the critical path, shown by the dotted line, includes delays through the carry-lookahead adder, an XOR gate, an OR gate, a look-up table, driving 56 multiplexors, and finally a CSA. The XOR gates and OR gates are necessary to convert from a two's complement form to sign-magnitude before the look-up table can be accessed.

Figure 2 is the topology of a Two-Bank SRT implementation as proposed by Quach [4]. This design removes some of the latency of the basic SRT scheme by duplicating hardware. The CSA's compute two sums of $rP_j - q_{j+1} D$ in parallel: one with q_{j+1} equal to either 0 or 1, and the other with q_{j+1} equal to 2. Because it is known early whether q_{j+1} will be either 0 or 1, this value can simply be multiplexed into one of the CSAs. From the figure, it can be seen that the critical path includes delays through a CSA, a multiplexor, a carry-lookahead adder, and a big multiplexor.

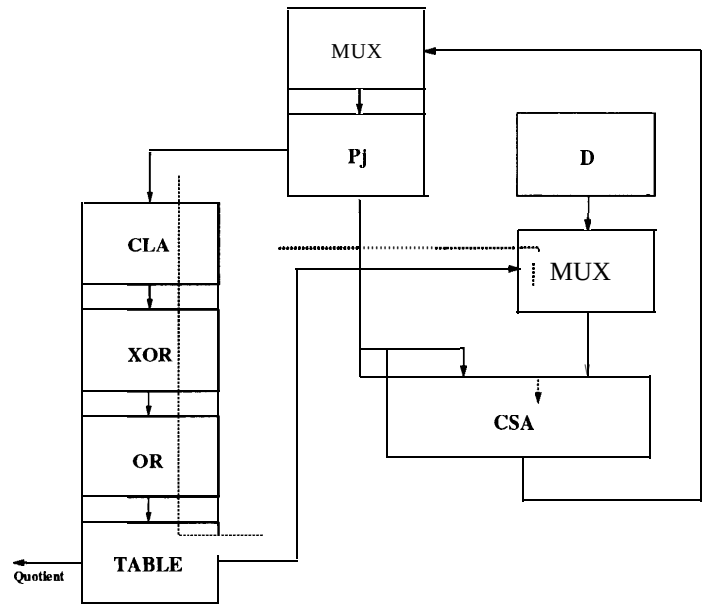


Figure 1: Basic SRT Scheme

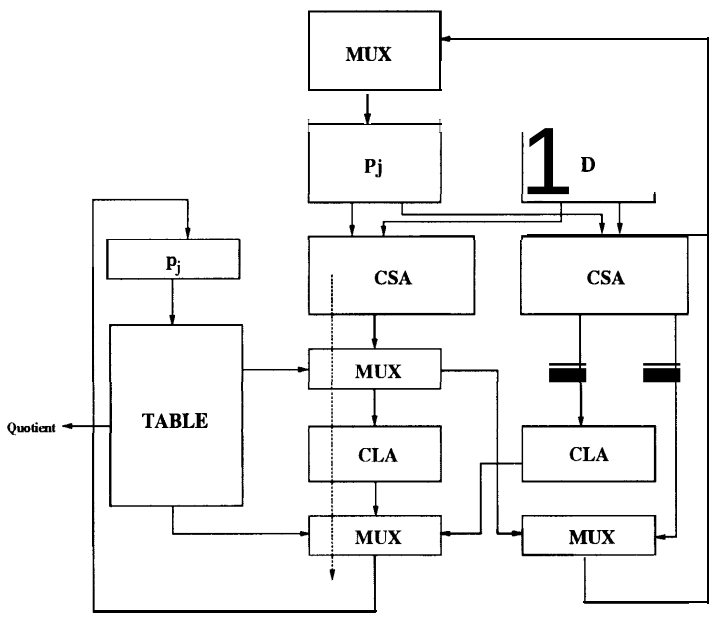


Figure 2: Two-Bank Scheme

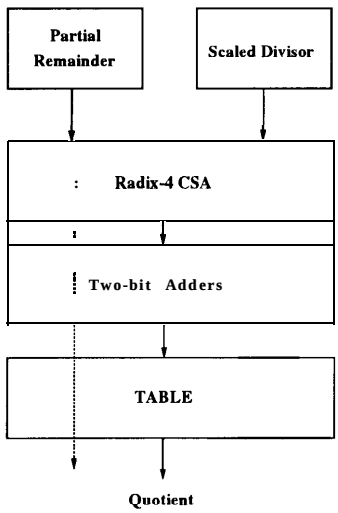


Figure 3: Simplified Operand Scaling Scheme

A topology based upon operand scaling as proposed by Ercegovac et al [5] is displayed in figure 3. This design attempts to speed-up the critical path of the conventional SRT scheme. To reduce the quotient-selection complexity, the divisor is scaled to be around unity. This scaling procedure makes the quotient-selection function independent of the divisor value. Accordingly, the table has fewer terms and thus becomes simpler to implement. The overall algorithm consists of three steps: scaling of the divisor and the dividend, division recurrence, and quotient conversion. Thus, the critical path is faster due to the decreased look-up table complexity. However, the algorithm does require an extra cycle for scaling.

3 Mapping to Domino Logic

In order to achieve maximum performance from the divider, circuit styles besides conventional static CMOS were considered. Specifically, the use of a precharged logic style was decided upon due to its inherent performance advantages. Precharged logic is fast, as latency is only dependent upon one logic level transition. Latency in static gates is typically a function of the rise times of the p-transistors in the pull-up network. By devoting a fixed precharge phase in every clock cycle in which all gates have their outputs precharged high, the delay of the gates become more a function of the fall-times of the n transistors, which are inherently faster due to their higher mobility. Additionally, in general, precharged logic is denser and provides lower input loading than static logic, as there is only one transistor per input.

The limitation of precharged logic is that precharged gates can not be easily cascaded.

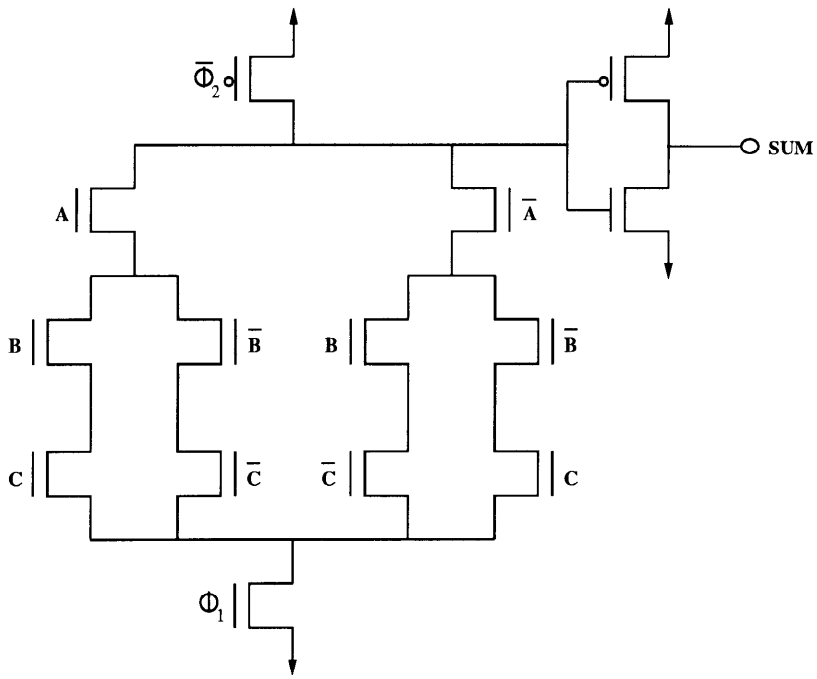


Figure 4: Domino CSA

Utilizing a standard timing convention for input and output signals, inputs to a precharged gate must be stable during the evaluation period, but the outputs of a precharged gates are only valid during the evaluation period. Thus, this delay between the timings of the inputs and outputs precludes the usage of the output of a precharged gate to directly drive the input of another precharged gate.

Accordingly, it was decided to utilize domino logic for the circuit level implementation. By placing a static inverter in-between any two precharge gates, one can take advantage of the fact that during the evaluation period, the output can only fall. Utilizing a cascaded inverter, the output of the inverter can only rise. Thus, this output can be used to drive another domino gate. However, because all outputs are monotonically rising, domino logic forms a non-inverting logic family. Given that arithmetic circuits are being designed in this study, it is vital that inverting gates be allowed. By providing the true and complements of all signals, it is possible to use DeMorgan's law to produce inverted outputs. In the worst case, this requires approximately two times the number of precharged transistors, or about the number of transistors in a static implementation. Figure 4 is an implementation of a CSA in domino logic.

To determine the relative performance of the three aforementioned divider topologies, the components of the critical paths of each of the implementations were simulated in HSPICE as domino circuits in 1μ CMOS under typical conditions. Table 1 contains the results of this analysis.

These results demonstrated that the difference between the schemes was not large. Thus,

Implementation	Critical Path	Latency
Conventional	CLA + XOR + OR + Table(6) + BigMux + CSA	2.26ns
Two-bank	CSA + MUX + CLA + BigMux	1.51ns
Operand Scaling	4-2 CSA + 2 bit-adder + Table(3)	1.45ns
Improved Conv.	CLA + XOR + Table(4) + BigMux + fastCSA	1.86ns

Table 1: Critical path comparison

the choice of which topology to utilize or modify was not clear. The operand scaling scheme requires more complex CSAs and an extra scaling cycle. The two-bank scheme essentially doubles the number of CSAs in each stage, and thus requires nearly double the area. Given the lowest cost of the conventional scheme in terms of area and complexity, an analysis of the effects of various optimizations among the different topologies compared to an improved conventional scheme (Oberman-Quach) was to be performed.

4 Divider Design

4.1 Logic and Circuit Design

It was decided to implement a divider based upon the conventional scheme based on its inherently good cost/performance. However, the basis of this research was the methods by which this divider could be optimized to yield even higher performance. The two means of optimization were: 1) logic encoding for the look-up table and 2) taking advantage of the domino logic technology mapping.

The equations for the three different schemes is presented in table 2. The first optimization, logic encoding, was made possible by noting some optimizations that could be made in the logic equations for the look-up table in the conventional scheme. Figure 5 shows an SRT table with the allowable quotient digits. This is actually a folded table so that only the positive half of the original table needs to be implemented. This provided an initial reduction in the size of the look-up table as reported in [3]. To further reduce the size of the look-up table, the quotients digits were encoded as follows:

$$q(-2) = Sq_2$$

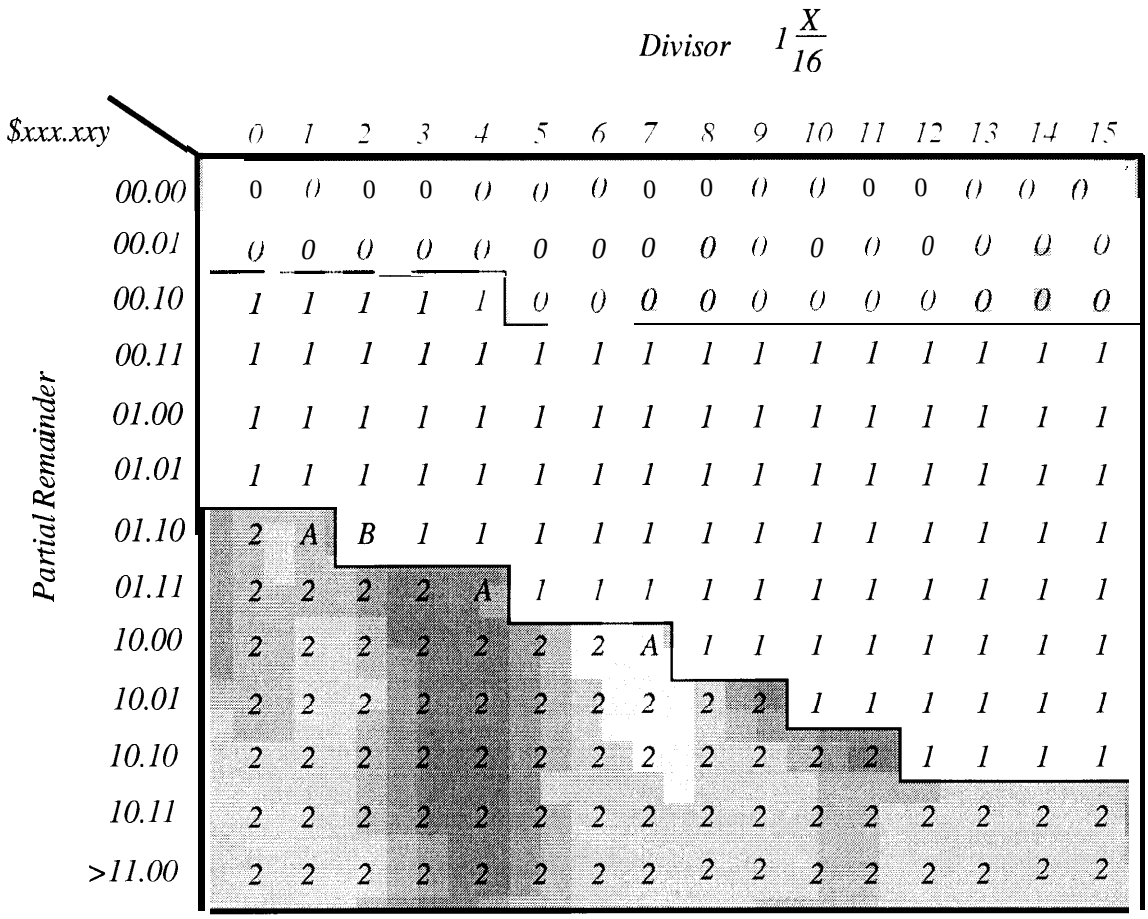
$$q(-1) = Sq_1q_2'$$

$$0 = q_1'q_2'$$

$$q(1) = S'q_1q_2'$$

$$q(2) = S'q_2$$

This encoding reduces the complexity of the logic to form q_1, as it only has to follow the top edge of the q = 1 region of figure 5. The bottom edge is followed by ANDing with q_2'. This encoding also allows for the removal of the OR gates that were required in converting



A = 1 if negative and y bit = 1; A = 2 otherwise;

B = 2 if negative and y bit = 1: B = 1 otherwise.

Figure 5: Allowable quotient digits in each SRT region

Scheme	Equations
Operand Scaling	qm2 = a'bce' + a'bcd' + a'bcf' + a'bcg' + a'bch' + ab'c'f' + ab'c'e' + ab'c'd' + ac'd'e'f 'h' + ab'd'e'f'h' + ab'c'h' + ac'd'e'f 'g' + ab'd'e'f 'g' + ab'c'g' qm1 = ac'd'f'gh + ab'd'f 'gh + a'cdefgh + ac'd'e'f + ab'd'e'f + abc'd'e + ab'cd'e + abc'de'f' + ab'cde'f' + ab'c'defgh q0 = b'c'd'e'f' + bcd'e'f' + bc'de + b'cde + bc'df + b'cdf q1 = b'c'd'e'f + bcd'e'f + a'b'c'd'e + abcd'e + a'b'c'de'f' + abcde'f' q2 = a'c'de + a'c'df + a'bc' + a'b'c + abcde + abcdf
Optimized Baseline	q1 = a'c'd'u + a'b'u + uv + y + x q2 = a'b'd's'yuw + a'b'c'yuw' + a'c'd'yuv + a'b'c's'yu + a'b'c'd'yu + a'b'yuv + b'c'xv + a'xw' + xuv + a's'x + a'd'x + a'xv + b'xu + a'c'x + a'b'x + a'xu + xy + t
Baseline	q1 = cdt'x'yv' + bst'x'yw + cst'x'yv' + dst'x'yv'w + ct'x'yv'w' + a'c'd't'x'y'u + bct'x'uv + t'x'y'uv + bdt'x'uv + bt'x'yv' + a'b't 'x'y'u + at 'xy'u'v' + t'x'yu' + bcdst'xy'u'v'w + abt'xy'u' + act 'xy'u' + abt 'xy'v' + at 'x'y q2 = t + a'xw' + a's'x + a'd'x + a'c'x + a'b'x + b'c'xv + a'xv + b'xu + a'xu + a'b'c'yuw' + a'b'd's'yuw + a'b'c's'yu + a'b'c'd'yu + xuv + a'c'd'yuvw' + a'c'd's'yuv + a'b'yuv + xy

Table 2: Look-up table comparison

the high order 8 bits from a redundant form to the signed-magnitude form required by the look-up table. Thus, approximately one gate delay can be removed from the critical path of the conventional scheme.

Second, many of the terms in the new table's equations are heavily dependent upon the divisor, which is constant throughout the entire computational process. Synthesis tools can reduce the overall gate delay of the table by noting this behavior. Specifically, in this design, Synopsys was utilized to synthesize the improved baseline look-up table. It generated combinational logic to implement the table, with the constraints for the design optimization being to minimize the critical path for any of the inputs which are from the partial product (i.e. variables t, x, y, u, v, and w). A library of standard-cells was used as input to Synopsys, with fan-in of each gate limited to a maximum of three. The optimized implementation yielded a critical path of 5 two-input NOR gates.

Third, because two of the three inputs to the partial remainder CSA's are known early, as they are the result of the previous partial remainder, there is only one late signal. Thus, the CSA is similar to a mux. This can be implemented in domino logic by placing the latest arriving signal, the multiple of the divisor driven by the table output, close to the CSA output. In this way, the latency of the CSA can be further reduced.

By utilizing HSPICE, the CSA's, muxes, latches, and CLA's were sized to provide maximum performance. There is always a tradeoff in domino logic between evaluation time and precharge time. Typically, in domino logic, the designer designs to make one edge, the rising edge of the inverter, to be very fast. This can be accomplished by appropriate sizing: increasing the size of the p-transistor in the inverter, and decreasing the n-transistor. This can bring two problems: 1) the falling/precharge edge can become very slow, and 2) noise margins can become a significant problem.

The inverters were designed to have large noise margins. The pull-ups were designed three times as large as the pull-downs, which almost exactly compensates for the mobility differences in the MOSIS process, placing the DC switching point very near Vdd/2. Thus, noise margins, area, and power were considerations in this design.

Further analysis demonstrated that the n-transistor in the evaluation path of the domino gates is not necessarily needed. As long as all of the inputs to the domino gate are guaranteed to be low during the precharge phase, an explicit evaluation transistor is not required. However, this guarantee becomes difficult to achieve, as without the evaluation transistor, any domino gate can not precharge until its previous gate has precharged, and so on. Thus, while this scheme can decrease the evaluation times in the circuit, it can lead to an increase in precharge times.

A summary of the effects of the various optimizations on the different dividers' complexity is shown in table 3.

Unit	Method	Scaling	2-banks	Oberman-Quach
Adder	Scaling	down		
Converter	Encoding		down	down
Table	Radix, Encoding	down	down	down
CSA	Circuit	up		down

Table 3: Complexity effects of optimizations

This table demonstrates how the Oberman-Quach design compares in terms of reducing the complexity, and thus the latency, of the various divider components in relation to the other designs.

4.2 Layout/Area

Layout has been done for the radix-256 divider utilizing a two-level metal technology file for Magic. Figure 6 shows a domino CSA extracted from the magic layout.

Current cell area for a CSA is 11,120 μm2. Thus, the total area for the four-levels of 56-wide CSAs is approximately 2.5mm2. Total area, including control logic and wire routing, should be under 5mm2.

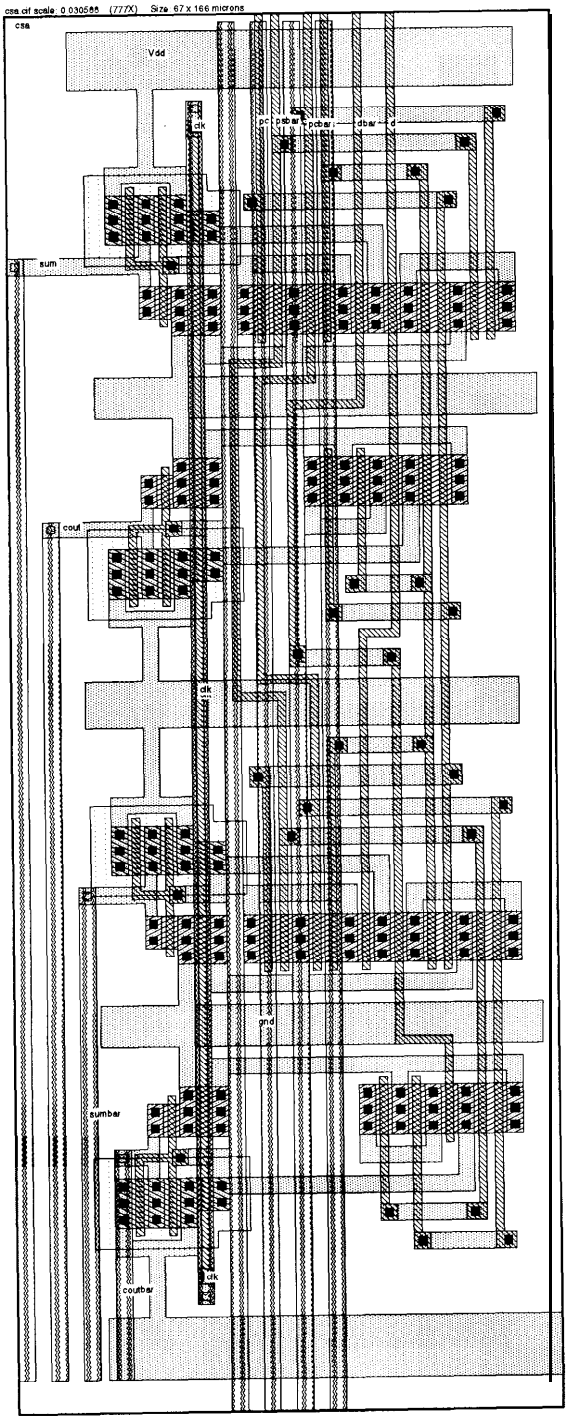


Figure 6: Layout of a CSA

5 Results

5.1 Single Stage Analysis

The HSPICE simulations using 1μ CMOS and typical conditions provided the following results:

Circuit	Evaluation Time
fast CSA	fan-out 8 = .32ns
reg CSA	fan-out 8 = .42ns
4-2 CSA	.83ns
2-bit adder	.30ns
2-input XOR	200fF = .20ns
2-Input MUX	200fF=.15ns, 2pF=.33ns
4-Input OR	200fF=.10ns
CLA	200fF=.61ns

Table 4: Evaluation Times

The worst case precharge time for all gates was a maximum of .7ns. Thus, the cycle time for a radix-4 (single stage) improved conventional divider would be 2.56ns.

5.2 Multiple Stage Analysis

Higher radix dividers can be formed by cascading single stage radix-4 dividers [4]. To calculate the cycle times and latencies for other radices, it is necessary to multiply the evaluation time for a single stage by the number of stages, add the precharge time per cycle, and multiply the result by the number of cycles. The total number of cycles is equal to $(56/(\text{bits/iteration}) + 1(\text{rounding})$. Table 5 shows the latencies of various radix dividers.

Stages	Avg. Cycle time	Num. Cycles	Total Latency
1	2.56ns	29	74.2ns
2	4.42ns	15	66.3ns
3	6.28ns	11	69. 1ns
4	8.14ns	8	65.1ns
7	13.7ns	5	68.6ns

Table 5: Latencies for varying number of stages

These numbers demonstrate that the highest performance can be achieved by staging four radix-4 dividers together to form a radix-256 divider that generates 8 quotient bits per iteration. Additionally, the corresponding cycle time is of the order of modern processor cycle times.

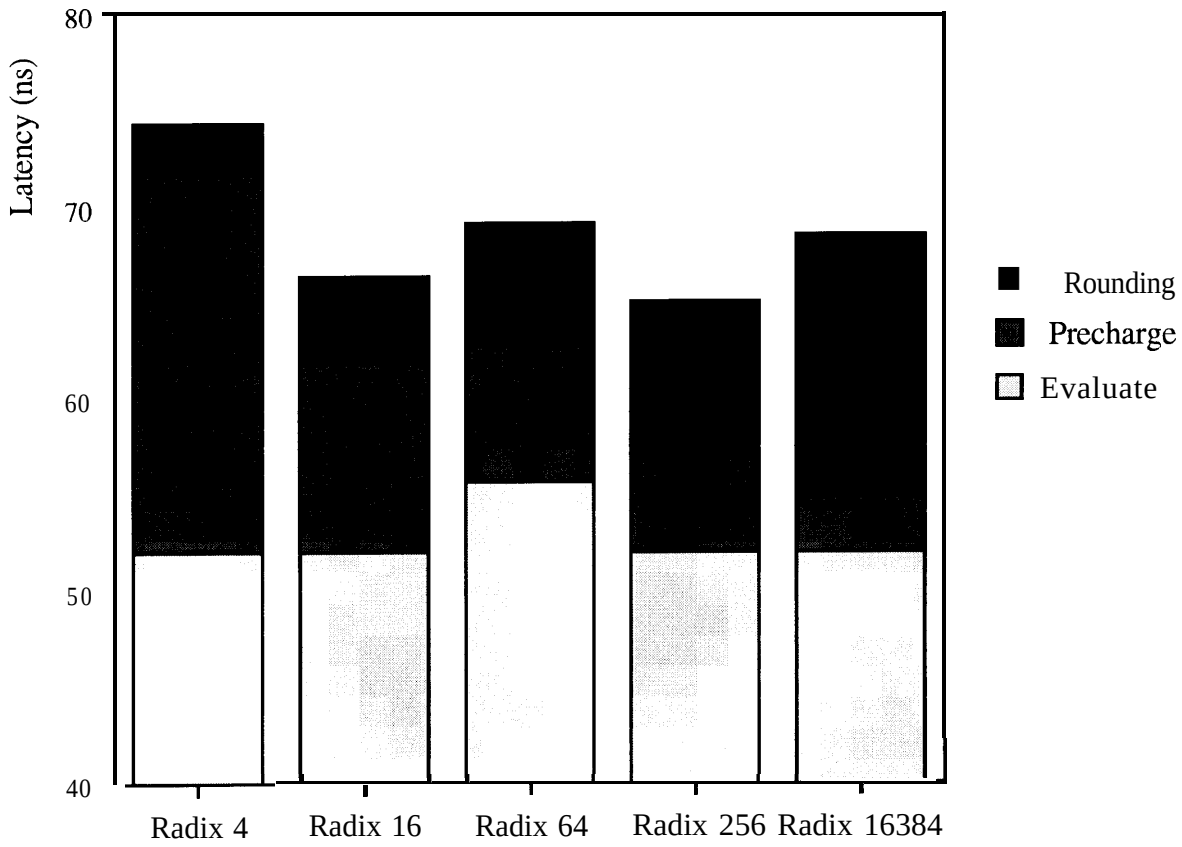


Figure 7: Multiple stage analysis

Model	Cycles	Cycle Time	Latency
SuperSparc	9	20ns	180ns
Snake	15	10ns	150ns
Oberman-Quach	8	8.14ns	65.1ns

Table 6: Performance comparison

The contributions of the various phases of the division process in the above numbers can be seen in figure 7.

6 Comparison

To provide some quantitative reference, table 6 compares the performance of the divider presented in this study with two other competitive dividers:

It should be noted that both the SuperSparc and Snake are fabricated in 0.8μ processes, while simulations presented in this study were for a typical 1μ process. If the circuit was fabricated in a 0.8μ process, it is estimated that the total latency would be reduced by about 20%, totalling 52ns. If it were fabricated in a 0.35μ process, total latency could be reduced by up to 100%, yielding a latency of under 33ns.

7 Conclusions

In this study, current divider technology has been analyzed. By using optimizations on a conventional SRT topology, a high-performance divider was designed. The combination of logic encoding to reduce the latency for conversion and table look-up with circuit style and implementation optimizations yielded a performance advantage over a conventional SRT divider. The other two divider topologies analyzed yield a slight performance advantage over the presented divider, but at nearly double the hardware and complexity. When compared with the conventional SRT scheme, the Oberman-Quach divider realizes a similar performance advantage, but at a reduced hardware cost. By staging four such radix-4 dividers to form a radix-256 divider, the precharge time of the domino circuit can be amortized in such a way that a minimum overall latency can be achieved for a full double-precision divide. Thus, when all of the topologies presented are mapped into a faster circuit style, the performance advantage of any one scheme is minimized, and the focus shifts to the desirability of increased hardware and complexity.

References

[1] D. E. Atkins. Higher-radix division using estimates of the divisor and partial remainders. *IEEE Transactions on Computers*, C-17(10):925-934, October 1968.