SRT Division Architectures and Implementations

David L. Harris, Stuart F. Oberman, and Mark A. Horowitz

Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{harrisd, oberman, horowitz}@leland.stanford.edu

Abstract

SRT dividers are common in modern floating point units. Higher division performance is achieved by retiring more quotient bits in each cycle. Previous research has shown that realistic stages are limited to radix-2 and radix-4. Higher radix dividers are therefore formed by a combination of low-radix stages. In this paper, we present an analysis of the effects of radix-2 and radix-4 SRT divider architectures and circuit families on divider area and performance. We show the performance and area results for a wide variety of divider architectures and implementations. We conclude that divider performance is only weakly sensitive to reasonable choices of architecture but significantly improved by aggressive circuit techniques.

1 Introduction

A simple and widely implemented class of division algorithm is digit recurrence. The most common implementation of digit recurrence division in modern microprocessors is *SRT* division, taking its name from the initials of Sweeney, Robertson [1] and Tocher [2], who developed the algorithm independently at approximately the same time. SRT division uses subtraction as the fundamental operator to retire a fixed number of quotient bits in every iteration. Two fundamental works on SRT division are those of Atkins [3], the first major analysis of SRT algorithms, and Tan [4], a derivation of high-radix SRT division and an analytic method of implementing SRT look-up tables. Ercegovac and Lang [5] provide a comprehensive treatment of the theory of SRT division and square root. Although division is typically an infrequent operation, ignoring its implementation significantly degrades system performance for many applications [6].

Various techniques have been proposed for increasing division performance, including staging of simple low-radix stages, overlapping sections of one stage with another stage, and prescaling the input operands [7]. All of these methods introduce area-performance tradeoffs. Ercegovac and Lang [5] analyze the tradeoffs of using several of these optimizations in the context of static CMOS standard-cells. Williams [8] presents a self-timed dynamic CMOS divider comprising a ring of five radix-2 stages that incorporates several of these techniques, and he also presents an analysis of the performance and area effects of the architectural components. Prabhu [9] presents the tradeoffs encountered when designing the Sun UltraSparc radix-8 divider.

In contrast to previous works, this paper analyzes in detail the effects of both circuit style and divider architecture on the area and performance of divider implementations. We present the performance results using the technology-independent metric of fanout-of-4 inverter delay. We are therefore able to extrapolate our results to future process technologies. While the discussion here is devoted to division, the theory of square root computation is an extension of the theory of division. Accordingly, most of the analyses presented here can also be applied to the design of square root units.

We survey the fundamental design parameters of SRT division and present the most common techniques for achieving higher performance in Section 2. Realizing the performance impact of advanced circuit techniques, we examine circuit issues relating to dual-rail domino divider implementations in Section 3. We present performance and area results for a wide variety of divider architectures and circuit styles in Section 4. We analyze the data and draw conclusions about the importance of architecture and circuit style in Section 5.

2 SRT Division

2.1 Definitions

In this analysis, the input operands are assumed to be represented in a normalized floating point format with n bit significands in sign-and-magnitude representation. The algorithms presented here are applied only to the magnitudes of the significands of the input operands. Techniques for computing the resulting exponent and sign are straightforward. The most common format found in modern computers is the IEEE 754 standard for binary floating point arithmetic. This standard defines single and double precision formats, where n=24 for single precision and n=53 for double precision. The significand consists of a normalized quantity, with an explicit or implicit leading bit to the left of the implied binary point, and the magnitude of the significand is in the range [1,2). However, to simplify the presentation, this analysis assumes fractional quotients normalized to the range [0.5,1).

The quotient is defined to comprise k radix-r digits with

$$r = 2^b \tag{1}$$
$$k = \frac{n}{b} \tag{2}$$

where a division algorithm that retires b bits of quotient in each iteration is said to be a radix-r algorithm. Such an algorithm requires k iterations to compute the final n bit result and thus has a *latency* of k cycles. The *cycle time* of the divider is defined as the maximum time to compute one iteration of the algorithm. Depending upon the implementation, this may or may not be the same as the cycle time of the processor.

The following recurrence is used in every iteration of the SRT algorithm:

$$rP_0 = dividend \tag{3}$$
$$P_{j+1} = rP_j - q_{j+1} divisor \tag{4}$$

where P_j is the partial remainder, or residual, at iteration j. In each iteration, one digit of the quotient is determined by the quotient-digit selection function:

$$q_{j+1} = SEL(rP_j, divisor) \tag{5}$$

The final quotient after k iterations is then

$$q = \sum_{j=1}^{k} q_j r^{-j} \tag{6}$$

2.2 Divider Parameters

2.2.1 Choice of Radix

The fundamental method of decreasing the overall latency (in machine cycles) of the algorithm is to increase the radix r of the algorithm, typically chosen to be a power of 2. However, this latency reduction does not come for free. As the radix increases, the quotient-digit selection becomes more complicated, which may increase the cycle time. Moreover, the generation of all required divisor multiples may become impractical for higher radices. Oberman [10] shows that the delay of quotient selection tables increases linearly with increasing radix, while the area increases quadratically. While prescaling of the input operands [11] reduces table complexity at the expense of additional latency, nevertheless the difficulty in generating all of the required divisor multiples for radix 8 and higher limits practical divider implementations to radix 2 and radix 4.

2.2.2 Choice of Quotient Digit Set

For a given choice of radix r, some range of digits is decided upon for the allowed values of the quotient in each iteration. The simplest case is where, for radix r, there are exactly r allowed values of the quotient. However, to increase the performance of the algorithm, a *redundant digit set* is used. This allows a quotient digit to be selected based upon an approximation of the partial remainder, permitting the use of a *redundant* remainder representation as discussed in the next section. Small errors in the quotient due to the remainder approximation are corrected in later iterations. Such a digit set is composed of symmetric signed-digit consecutive integers, where the maximum digit is a. The digit set is made redundant by having more than r digits in the set. By using a larger number of allowed quotient digits, the complexity and latency of the quotient selection function is reduced. However, choosing a smaller number of allowed digits for the quotient simplifies the generation of the multiple of the divisor. Specifically, for radix 2, the digit set is $\{-1, 0, 1\}$. For radix 4, there are two typical choices for the digit set: minimally redundant $\{-2, -1, 0, 1, 2\}$ and maximally redundant $\{-3, -2, -1, 0, 1, 2, 3\}$. The quotient selection logic for a maximally-redundant radix 4 digit set is about 20% faster and 50% smaller than for a minimally-redundant digit set [10]. However, maximally-redundant radix 4 requires the computation of the 3x divisor multiple, which typically requires extra initial delay and area.

2.2.3 Choice of Remainder Representation

The partial remainder also can be represented in two different forms, either *redundant* or *nonredundant*. Each iteration of the algorithm requires a subtraction to compute the next partial remainder. If this partial remainder is in a nonredundant form, then this operation requires a time-consuming full-width carry-propagate-adder, increasing the cycle time. Therefore, the partial remainder is typically stored in redundant form so that a fast carry-free adder, such as a carry-save adder (CSA), can be used in the partial remainder calculation.

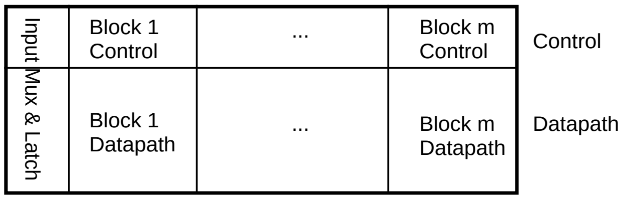

Figure 1. SRT divider block diagram

2.3 Higher Performance

Several techniques have been proposed for improving the performance of SRT division, but most involve cascading low-radix stages to form a higher radix divider. The primary problem with a cascade of stages is the corresponding increase in cycle time. To avoid this increase, some additional computation can proceed in parallel at the expense of area. Taylor [12] proposes overlapping the quotient-digit selection of consecutive stages. Oberman [10] and Quach [13] discuss overlapping remainder computation. Fandrianto [14] discusses a cascade of lower radix segments in which there is no shifting of the partial remainder between the segments through the use of range reduction. In this study, we analyze the effects of five possible overlap schemes: no overlap, overlapped quotient selection, overlapped remainder computation, overlapped quotient and remainder computation, and a hybrid overlap scheme.

A general divider organization using overlapping is shown in Fig. 1. This floorplan is divided into a control section for quotient digit computation that typically operates on a small number of the most significant bits (about 4 for radix 2 and 8 for radix 4), and a datapath section for partial remainder formation of the remaining significand bits.

The divider is defined to comprise a cascade of m blocks, where each block has a delay of t_{block}. Each block is composed of s overlapped radix-r stages. The divider therefore retires $b' = m \times s \times b$ bits in each cycle, with a total cycle time of $m \times t_{block} + t_{overhead}$, where $t_{overhead}$ includes latch delay, clock skew, and the input multiplexor for injecting new operands. Such an overlapped scheme produces k' radix-r' digits with

$$r' = 2^{b'} \tag{7}$$

$$k' = \frac{n}{b'} \tag{8}$$

The overlap schemes we consider are illustrated in Figs. 2 through 6. The critical path(s) are indicated by the heavy black lines. While all partial remainders are stored in carry-save form, they are drawn with a single wire for simplicity. The architectures are shown overlapping two stages, but can be generalized to higher overlap [9].

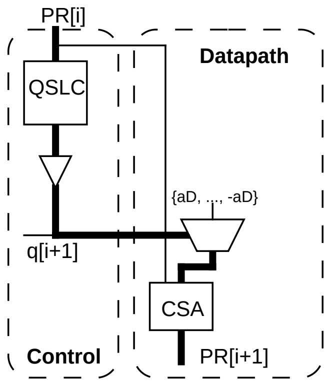

Figure 2. Non-overlapped design

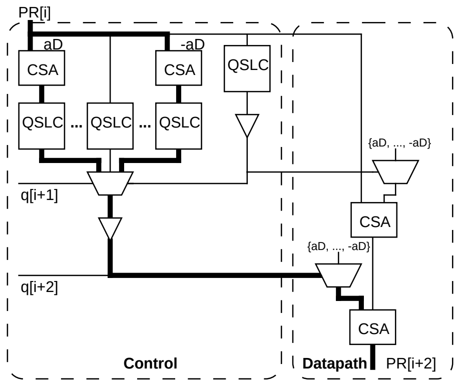

Figure 3. Overlapped quotient selection

2.3.1 Non-overlapped

A simple non-overlapped design is common in low-cost applications, such as the Intel Pentium Processor [15]. A block diagram is shown in Fig. 2.

The critical path involves generating the next quotient digit using high order bits from the current partial remainder and the divisor as input to the quotient selection logic (QSLC), driving the quotient across the datapath to select the proper divisor multiple, and subtracting the divisor multiple from the partial remainder with a carry-save adder to compute the next partial remainder. The critical path delay is equal to:

$$t_{block} = t_{qslc} + t_{buf} + t_{mux} + t_{CSA} \tag{9}$$

2.3.2 Overlapping Quotient Selection

Overlapping quotient selection (Fig. 3) requires additional control logic but no additional datapath elements. This technique was demonstrated by Taylor [12]. The critical path involves speculatively generating all possible second quotient digits, then choosing among them given the first quotient digit. The results are used to select the appropriate divisor

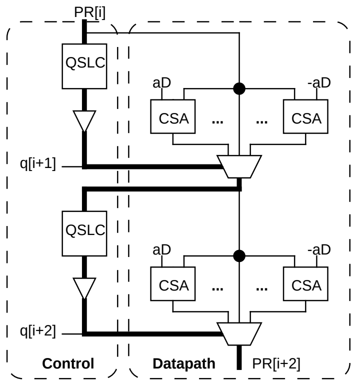

Figure 4. Overlapped remainder formation

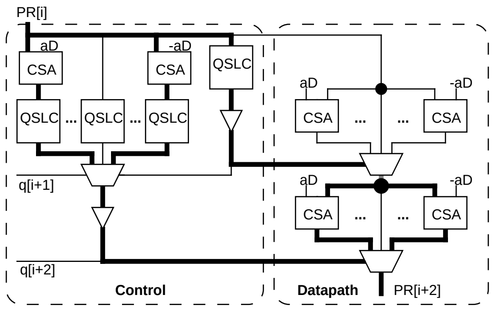

Figure 5. Overlapped remainder and quotient selection

multiple to subtract from the partial remainder. The datapath CSAs may be optimized for a single late input. The critical path for $s = 2$ is equal to:

$$t_{block} \quad = \quad t_{qslc} + t_{buf} + 2t_{mux} + 2t_{CSA} \qquad (10)$$

2.3.3 Overlapping Remainder Formation

Overlapping partial remainder computation (Fig. 4) speculatively computes all of the next partial remainders, then selects the appropriate one based on the actual quotient digit. The critical path delay for $s = 2$ is:

$$t_{block} \quad = \quad 2(t_{qslc} + t_{buf} + t_{mux}) \qquad (11)$$

2.3.4 Overlapping Quotient Selection and Remainder Formation

The previous two schemes can be combined (Fig. 5) such that both the quotient selection and partial remainder formation are overlapped. The Sun UltraSparc [9] implements such a combination. This design has two equally critical paths, one through the quotient digit selection logic, and the other through the speculative partial remainder formation.

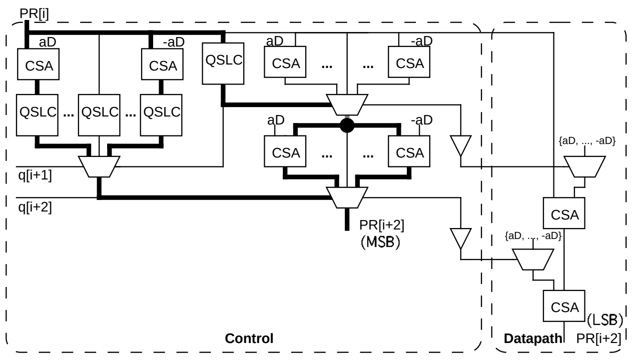

Figure 6. Hybrid overlapping

The critical path delay for $s = 2$ is:

$$t_{block} \quad = \quad t_{qslc} + t_{buf} + 2t_{mux} + t_{CSA} \qquad (12)$$

2.3.5 Hybrid Overlap

Closer examination shows that only the most significant bits of the next partial remainder are critical. We can exploit this fact in two ways. One is to buffer the quotient digits before driving the low-order mux selects. This optimization is applicable to all architectures. Another way is to only overlap partial remainder of the critical high-order bits to save area. These techniques are combined in Fig. 6.

This architecture is a hybrid of the overlapped partial remainder design for the critical bits and just overlapped quotient selection for the non-critical bits. The quotient digits are buffered before driving the datapath with the non-critical least significant bits. This eliminates the buffer delay from the critical path to the high order bits. The critical path delay for $s = 2$ is:

$$t_{block} \quad = \quad t_{qslc} + 2t_{mux} + t_{CSA} \qquad (13)$$

Hardware is saved relative to the overlapped partial remainder formation because no speculative adders are needed in the datapath for partial remainder formation. The non-critical partial remainder bits lag behind the critical bits by the delay of a buffer plus CSA. This may slightly increase the time required for rounding and normalization at the end of a divide. More importantly, the non-critical bits must catch up in time before they are required by quotient selection. Therefore, the speculative partial remainder formation must operate on $s \times b$ extra bits to catch these bits up before they become critical. The lag of the non-critical bits plus the delay through the partial remainder formation must be less than the delay of the critical quotient selection path in order for the non-critical bits to catch up as they are shifted into the critical portion. Specifically, the critical path delay from q[i+2] to PR[i+4] through the quotient digit logic to generate the next set of most-significant partial remainder bits is:

$$t_{crit} = t_{mux} + (t_{CSA} + t_{qslc} + t_{mux} + t_{mux}) \quad (14)$$

while the sum of the lagging path delay and partial remainder formation delay is:

$$\begin{aligned} t_{noncrit} = & (t_{buf} + t_{mux} + t_{CSA}) + \\ & (t_{CSA} + t_{mux} + t_{CSA} + t_{mux}) \end{aligned} \quad (15)$$

Simplifying, to keep $t_{noncrit} < t_{crit}$,

$$t_{buf} + 2t_{CSA} < t_{qslc} \quad (16)$$

As we show later, quotient selection is the most time consuming component, especially for radix-4 designs. Therefore, it is reasonable to expect this constraint to be satisfied, especially since the datapath adder may be optimized for a single late input. If quotient selection is too fast, the final partial remainder formation can be overlapped in the datapath, adding more CSAs, but relaxing the timing constraint to $t_{buf} + t_{CSA} < t_{qslc}$.

2.4 Comparison

Table 1 compares the latency, hardware cost, and wiring cost of each architecture. The number of QSLCs (which dominate control area) and number of CSAs and MUXes in the datapath are listed. Also, the number of metal tracks required is computed (see Fig. 7). Overlapping quotient digit selection saves $s-1$ quotient selection and buffer delays at the expense of additional quotient selection logic blocks. Overlapping partial remainder formation saves the delay of one CSA at the expense of many more CSAs performing speculative computation. Finally, the hybrid scheme eliminates the buffer delay and also avoids a large number of speculative CSAs. From Table 1, the hybrid overlapping scheme has the lowest latency and also saves hardware relative to the next fastest scheme. However, the performances of other architectures that overlap quotient selection are within a CSA and buffer delay of the hybrid scheme, and as discussed previously, the buffer may be optimized out of the critical path.

The number of quotient selection blocks increases sharply when moving from radix-2 to radix-4 and for increasing s, as shown in Table 2. Due to the exponential increase in area for increasing s, reasonable designs are limited to an overlap of $s = 2$ or possibly $s = 3$.

3 Circuits

Overlapping stages is important, but Table 1 shows that the incremental improvement of better overlap techniques is small. Execution unit designers are therefore turning to

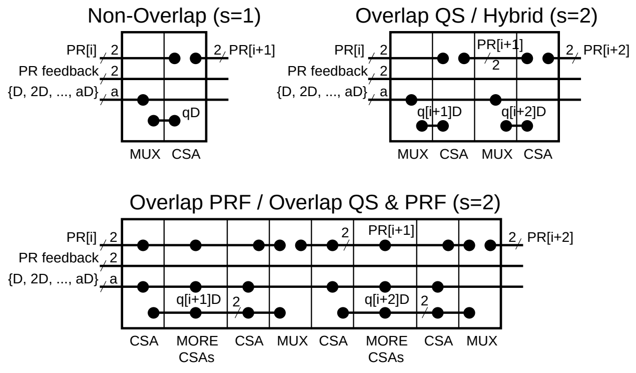

Figure 7. SRT datapath floorplan for one block

more aggressive circuit techniques, especially domino circuits, to greatly reduce latency [16].

Three key circuit issues which impact architectural choices are domino monotonicity requirements, wiring cost, and clocking overhead. Designers accustomed to static logic must remember that domino circuits require monotonically rising inputs during evaluation. Since single-rail domino circuits are not a functionally complete logic family, non-monotonic gates such as CSAs require dual-rail inputs and outputs to code both true and complementary signals. This increases the area of logic gates and the amount of interconnect. Radix-4 quotient selection logic is especially impacted because it is relatively large. Fortunately, the quotient digits produced are consumed only by multiplexors, so they can be computed in single-rail 1-hot form. Two options for high-speed QSLCs are domino gates and dynamic self-timed PLAs. We found that radix-4 minimally/maximally redundant 1-hot PLAs have 50/25 minterms respectively, while according to [10] minimally/maximally redundant Gray-encoded PLAs have only 25/14 minterms. Static designs can implement either a 1-hot PLA or an encoded PLA to save area, but domino implementations require the larger 1-hot design.

The datapath dominates the area of most dividers, so we must examine the number of bitlines running between elements within a bitslice. Dynamic designs double the wire count because dual-rail signals are needed. All floating point blocks have fixed overhead of power and ground, and three data busses (two inputs and one output). Dividers have many additional bitlines because partial remainders are kept in redundant form.

Fig. 7 illustrates wiring requirements on a floorplan of a block for various architectures. A dot on a wire over an element indicates that the signal is used in the element. On all architectures, two lines are required for driving the redundant partial remainder along the path and another two lines are required for feeding the result back for the next itera-

Architecture	Block Latency	# QSLC	# Wide CSAs	# Wide MUXes	# Bitlines
Nonoverlap	$s(t_{qsel} + t_{buf} + t_{mux} + t_{CSA})$	s	s	s	5+a
Overlap QS	$t_{qsel} + t_{buf} + st_{mux} + st_{CSA}$	see Table 2	s	s	5+a
Overlap PRF	$s(t_{qsel} + t_{buf} + t_{mux})$	s	2as	s	6+a
Overlap QS and PRF	$t_{qsel} + t_{buf} + st_{mux} + (s-1)t_{CSA}$	see Table 2	2as	s	6+a
Hybrid	$t_{qsel} + st_{mux} + (s-1)t_{CSA}$	see Table 2	s	s	5+a

Table 1. Comparison of architectures

tion. Furthermore, divisor multiples must be driven to all of the blocks. Although there are $2a+1$ divisor multiples, we must only transmit the a positive multiples and can generate the negative multiples with a local inversion. Finally, the appropriate divisor must be selected. In architectures which do not speculatively generate partial remainders, only one wire is required to drive the divisor mux output to the CSA which generates the next partial remainder. In speculative architectures, $2a$ CSAs generate two bits of output each. It would seem $4a$ wires are required as input to the divisor mux. However, by distributing the multiplexor across the adders, only two wires are needed for the redundant result.

A static divider requires either $5+a$ or $6+a$ bitlines, which poses little difficulty. A dual-rail domino divider requires twice as many bitlines. These bitlines can be accommodated in a bit pitch of about 120λ (λ is half of the minimum drawn transistor length), especially if some M4 tracks are allocated for routing the input and output buses over the floating point unit. Cells can be efficiently laid out at this pitch, so wire limitations are not expected to increase divider area significantly in a process with 3 or 4 metal layers. This is consistent with [8] which reports only a 15% area penalty for dual-rail domino in a 2 layer process.

Clocking overhead is an important concern in both static and dynamic designs. Static designs conventionally use a flip-flop with a multiplexor at the beginning of the cycle to capture either a new divider input or the result of the previous iteration. The flip-flop adds a delay of $t_{clk\rightarrow q} + t_{setup} + t_{skew}$ to the path, which can be large. Traditional domino designs require latches between phases of domino logic and are also sensitive to clock skew, but skew-tolerant domino techniques [17] make domino much more attractive by eliminating latch delays and clock skew from the critical path. Good domino designs must only pay the cost of one 2:1 multiplexor at the beginning of each cycle. This cost may be amortized over many bits produced in the cycle.

Since the area is proportional to the number of blocks, area can be reduced without impacting latency by clocking the divider at a higher frequency than the rest of the processor. For example, the HP-PA7100 [18] achieves higher radix by clocking a lower radix core at double frequency. This technique improves area at the expense of the complexity of generating a higher frequency clock.

s	r=2 a=1	r=4 a=2	r=4 a=3
1	1	1	1
2	4	6	8
3	11	27	39
4	26	112	166

Table 2. QSLCs for overlapped architectures

In summary, domino is an attractive approach for high-performance dividers. It greatly reduces gate delays and eliminates much of the clocking overhead found in flip-flop-based static designs. The domino wiring requirements do not significantly increase area in processes with 3 or more metal layers. Thus, the primary costs of domino designs are the extra area consumed by quotient selection logic, the increased power consumption while the divider is active, and the necessary circuit design expertise.

4 Results

We assigned a divider design project in an advanced VLSI circuit class at Stanford University. Twelve teams explored a wide variety of radix 2 and radix 4 double precision SRT divider designs using skew-tolerant domino circuits. The designs reflect a variety of skill levels and area/performance tradeoffs, but the best designs are reasonable. The delay results are from HSPICE simulation, and the area estimates are based upon total transistor count and device size. We compare these results with data on static dividers extrapolated from Ercegovac [5].

The designs are shown on a scatter plot of delay/bit vs. area/bit/divider cycle, shown in Fig. 8. The labeled points are described further in Table 3 and represent those designs with the best performance for a given area. The delay/bit is measured in fanout-of-4 inverter delays in the 1 μm (drawn) HP-CMOS26B process. Area must be normalized by the number of bits produced per cycle because a divider can simply unroll more blocks until the cycle is full. Area is estimated in mm^2 and can be converted to λ^2 by multiplying by 4×10^6. Area reflects only the divider control and datapath components, not auxiliary circuits required for normal-

Design	Architecture	r	a	s	Delay/bit (FO4)	Area/bit/cycle (mm^2)	QSLC Style	Notes
A	Non-overlapped	4	2	1	9.5	0.23	synthesized	smallest static
B	Hybrid overlap	4	2	2	5.7	0.33	synthesized	fastest static
C	Hybrid overlap	2	1	2	4.6	0.63	custom domino	most flexible
D	Overlap QS	2	1	3	4.7	0.68	custom domino	like UltraSparc
E	Hybrid overlap	4	3	2	3.7	0.86	1-hot PLA	fastest domino

Table 3. Designs with best performance for given area

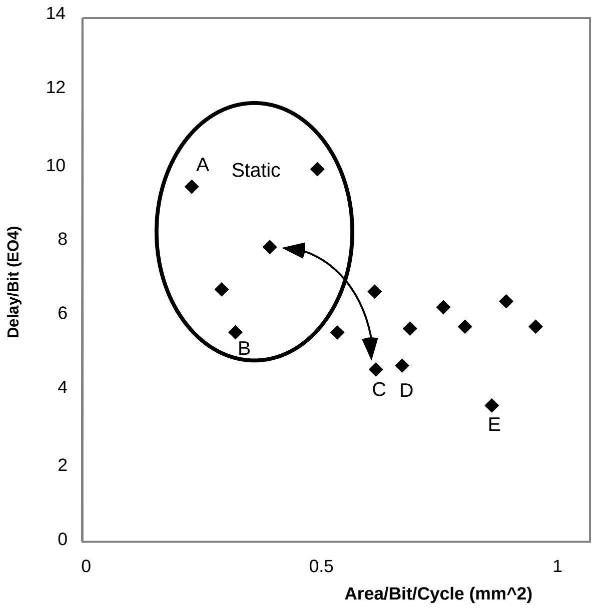

Figure 8. Scatter plot of results

izing, rounding, and exponent handling. Design E includes the area but not the additional setup latency of a fast CPA, as maximally-redundant radix 4 designs require precomputation of the 3x divisor multiple.

The arrowheads point out two hybrid overlapped radix 2 designs. The domino design is 1.7 times as fast but has 1.6 times as much area. The extra area is attributable to generating dual-rail outputs. The fastest domino design is 1.5 times as fast as the fastest static design; radix 4 domino designs have a larger area penalty because of the larger PLA required for monotonic quotient digit selection. The skew-tolerant domino designs assume an overhead of 1 FO4 delay / 4 bits for the input mux, while static designs use 4.4 FO4 delays / 4 bits for the mux and flip-flop overhead. This is a major advantage of skew-tolerant domino circuits.

The static results from [5] were normalized with the conversion that one fanout-of-3 NAND2 delay equals 1.05 FO4

Design	t_{qslc}	t_{mux}	t_{CSA}	t_{buf}
A,B	11.3	1.9	2.3	1.8
C	4.0	1.5	1.5	NA
D	3.6	0.9	1.5	1.1
E	7.8	1.7 - 2.6	1.7	NA

Table 4. Selected component delays (FO4)

delays and has an average area of 400 μm^2. Thus, the data for the static designs has more uncertainty than for the dynamic designs. Further, some of the static results were extrapolated from element delays tabulated in [5], rather than from complete designs. The designs in [5] assume iteration overhead of 8.4 FO4 delays in a conservative standard-cell methodology, but we use a more aggressive 4.4 FO4 delays as noted previously. The reported static quotient-selection logic delay is unexpectedly low compared with the domino.

Table 4 lists key component delays for several designs. As expected, quotient selection delay dominates the radix-4 critical path. The mux delay in design D is suprisingly low.

For comparison, Williams [8] reports delays of overlapped quotient selection for radix-2 and radix-4 stages in terms of FO1 inverter delay. Converting these delays using 1 FO4 inverter $\approx$ 2.5 FO1 inverters, we find a delay per bit of 4.7 FO4 for radix-2 and 4.5 FO4 for radix-4. These are consistent with the delays reported in this study. Comparing area is more difficult, as Williams' self-timed ring is constrained to overlapping of $s = 5$ stages to mask the overhead of self-timing.

5 Conclusions

The data points investigated in this study cover a wide range of the design space. However, the domino data points in Fig. 8 are tightly clustered with comparable area and performance. Therefore, we conclude that the choice of architecture, and especially the choice of either radix-2 or radix-4, makes little difference in the overall performance. Three examples of reasonable architectures which provide high performance with modest areas are hybrid overlapped radix-2 (s=2), quotient selection overlapped radix-2 (s=3), and hybrid overlapped maximally-redundant radix-4 (s=2).

Although the maximally-redundant radix-4 design achieves the lowest core delay, it requires extra time and hardware outside the iterations for 3x divisor multiple generation. Also, the designs producing more bits per block ($b \times s$) are less flexible, as typically an integral number of blocks must fit within a cycle. The reasonable architectures offer a delay/bit of 4-5 FO4 and a cost of approximately 3Mλ^2 or 6000 transistors per bit/cycle in the core.

The choice of circuit style has a larger effect on performance. Specifically, moving from static CMOS to dual-rail domino reduces the delay of the individual gates. Skew-tolerant domino increases performance further by eliminating clocking overhead. Comparing similar architectures, dual-rail domino provides a 1.5 - 1.7x speedup over static designs. However, static designs are generally smaller than dual-rail designs of the same architecture because only one polarity of output need be generated and because quotient selection logic can be more compactly designed with non-monotonic gates. Skew-tolerant domino circuits provide the performance advantages of self-timed circuits without the complexity of asynchronous design or the need to duplicate hardware to hide control overhead. For performance-critical designs, we recommend the use of dual-rail domino.

Acknowledgments

Thanks to all of the students in EE371 for their excellent work on the projects. We would like to note the designers of the “reasonable” architectures: **C** by Jeff Solomon and Derek Debusschere; **D** by Min Xu and Mengchen Yu; **E** by Peter Richards and Peter Verplaetse. Peter Verplaetse also contributed optimized 1-hot PLA data.

This work was supported by the NSF through a fellowship and grant MIP93-13701 and by Stanford’s Center for Integrated Systems.

References

[1] J. E. Robertson, “A new class of digital division methods,” *IRE Trans. Electronic Computers*, vol. EC-7, pp. 218–222, Sept. 1958.

[2] K. D. Tocher, “Techniques of multiplication and division for automatic binary computers,” *Quart. J. Mech. Appl. Math.*, vol. 11, pt. 3, pp. 364–384, 1958.

[3] D. E. Atkins, “Higher-radix division using estimates of the divisor and partial remainders,” *IEEE Trans. Computers*, vol. C-17, no. 10, Oct. 1968.

[4] K. G. Tan, “The theory and implementation of high-radix division,” in *Proc. 4th IEEE Symp. Computer Arithmetic*, pp. 154–163, June 1978.

[5] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.

[6] S. F. Oberman and M. J. Flynn, “Design issues in division and other floating-point operations,” *IEEE Trans. Computers*, vol. 46, no. 2, pp. 154–161, Feb. 1997.

[7] S. F. Oberman and M. J. Flynn, “Division algorithms and implementations,” *to appear in IEEE Trans. Computers*, 1997.

[8] T. E. Williams and M. A. Horowitz, “A zero-overhead self-timed 160-ns 54-b CMOS divider,” *IEEE J. Solid-State Circuits*, vol. 26, no. 11, pp. 1651–1661, Nov. 1991.

[9] J. A. Prabhu and G. B. Zyner, “167 MHz radix-8 floating point divide and square root using overlapped radix-2 stages,” in *Proc. 12th IEEE Symp. Computer Arithmetic*, pp. 155–162, July 1995.

[10] S. F. Oberman, *Design Issues in High Performance Floating Point Arithmetic Units*, Ph.D. thesis, Stanford University, Nov. 1996.

[11] M. D. Ercegovac and T. Lang, “Simple radix-4 division with operands scaling,” *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1204–1208, Sept. 1990.

[12] G. S. Taylor, “Radix 16 SRT dividers with overlapped quotient selection stages,” in *Proc. 7th IEEE Symp. Computer Arithmetic*, pp. 64–71, June 1985.

[13] N. Quach and M. Flynn, “A radix-64 floating-point divider,” Technical Report No. CSL-TR-92-529, Computer Systems Laboratory, Stanford University, June 1992.

[14] J. Fandrianto, “Algorithm for high-speed shared radix 8 division and radix 8 square root,” in *Proc. 9th IEEE Symp. Computer Arithmetic*, pp. 68–75, July 1989.

[15] H. P. Sharangpani and M. L. Barton, “Statistical analysis of floating point flaw in the pentium processor,” Intel Corporation White Paper, November 1994.

[16] P. Gronowski et al., “A 433-MHz 64-b quad-issue RISC microprocessor,” *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1687–1696, Nov. 1996.

[17] D. Harris and M. Horowitz, “Skew-tolerant domino circuits,” in *Digest of Technical Papers, IEEE Int. Solid-State Circuits Conf.*, 1997, pp. 422–423.

[18] T. Asprey, G. S. Averill, E. DeLano, R. Mason, B. Weiner, and J. Yetter, “Performance features of the PA7100 microprocessor,” *IEEE Micro*, vol. 13, no. 3, pp. 22–35, June 1993.