Radix 16 SRT Dividers With Overlapped Quotient Selection Stages[1]

A 225 Nanosecond Double Precision Divider for the S-1 Mark IIB

George S. Taylor

Computer Science Division, EECS
University of California
Berkeley, California 94720

ABSTRACT

This paper compares the three simplest SRT division methods by using them to design a divider that produces four quotient bits per cycle (radix 16). The three methods are distinguished by the number of bits found per stage of quotient selection logic:

(a) one bit per stage (radix 2) with quotient digits chosen from the set {-1,0,1},

(b) two bits per stage (radix 4) with quotient digits {-2,-1,0,1,2}, or

(c) two bits per stage (radix 4) with quotient digits {-3,-2,-1,0,1,2,3}.

For each method, we compare several ways to overlap multiple stages of quotient selection logic and we consider both irredundant and redundant (carry-save) representations for the remainder.

The cost and performance of each alternative is evaluated in terms a specific ECL gate array technology. We find that we can build a 15% faster divider with radix four stages than with radix two stages, for about the same amount of hardware. Between the two radix 4 alternatives, method (c) offers 5% more speed than method (b) at the cost of 20% more hardware.

A radix 16 divider using method (b) has been built for the S-1 Mark IIB computer under development at Lawrence Livermore Laboratory. This divider consists of eight ECL gate arrays and has a 12.5 nanosecond cycle time. It performs IEEE single and double precision floating point division in 150 and 225 nanoseconds, respectively, the shortest times reported for any general purpose computer.

1. Introduction

How might we build new dividers as more hardware becomes available in gate arrays and VLSI chips? One approach is to increase the performance of dividers that use subtraction as their iterative operation and produce a fixed number of quotient bits per cycle. In this paper, we compare several designs based on the three simplest SRT division methods [Robe58] [Toch58] [Atki67]. These methods can be as fast as more costly ones that are based on finding a multiplicative inverse [Cray82].

Subtractive normalization methods have the advantage of producing correctly rounded quotients and exact remainders. In contrast, finding a multiplicative inverse leads to correctly rounded quotients and exact remainders only after time-consuming fixup steps that have not been implemented in practice. Methods based on prescaling the dividend and divisor [Erce83] do not readily produce unscaled remainders, thus making it difficult to use the same hardware for both integer and floating point arithmetic.

Algorithm designers will avoid division if its time becomes too long relative to multiplication. The implementation described in section 9 is evidence that SRT division can stay within a factor of four of multiplication using a modest amount of hardware.

2. Factor of Four Speedup

The hardware cost for the radix 16 divison methods that we consider below is two to three times greater than the cost of a radix two non-restoring divider. However, the higher-radix methods run about four times faster in the same technology. We can attribute the factor of four speedup to three separate improvements:

(1) radix four SRT division with an irredundant remainder is almost twice as fast as radix two non-restoring division,

(2) two-stage overlapped quotient selection logic (going from radix 4 to radix 16, for example) is almost twice as fast as a single stage,

(3) division using redundant remainders is about 30% faster than with irredundant remainders.

Factor number (3) is an estimate for designs that fit on a single chip. Redundant remainders have a much larger impact on the performance of multiple chip designs because they save two on-chip to off-chip delays in every cycle.

Radix four SRT dividers have been implemented in several machines [Atki70b] [Tan78] [Tayl83]. The primary focus of this paper is on the other two enhancements, overlapped stages of quotient selection logic and redundant remainders.

[1] Research support by DARPA under contract N00034-K-0251.

3. An Experiment

Our experiment was to compare various ways to build a radix 16 divider. The cost of quotient selection (QS) logic grows exponentially with the radix, so our approach was to build radix 16 QS logic from multiple stages of either radix 2 or radix 4 QS logic. Radix 2 QS logic is smaller and faster than radix 4 logic, but this does not necessarily mean that radix 2 is the best building block for a radix 16 divider.

To determine costs and delays in a standard way, we took the "macrocells" of the Motorola MCA II ECL gate array [Prio84] as fundamental building blocks. An MCA II chip contains 220 principal cells (not counting output cells), each of which may be a full adder, a flip-flop or another function of similar complexity (see Table 3). Eight to ten of these gate arrays are necessary to build a radix 16 divider for 64-bit operands (as required for IEEE extended precision floating-point numbers [IEEE83] and for 64-bit integers).

The time to communicate between MCA II chips equals about four internal cell delays, a factor that strongly influences design tradeoffs. Future designs built with custom VLSI and larger gate arrays will not have these chip-to-chip delays. Consequently, we assume in this study that up to 1800 cells are available on a single chip, so that we can compare division methods without the temporary constraints imposed by multiple-chip designs.

In the next section, we summarize SRT division methods with two equations and a block diagram of the hardware that implements them. Then we focus on quotient selection hardware, since that is the limiting factor on performance.

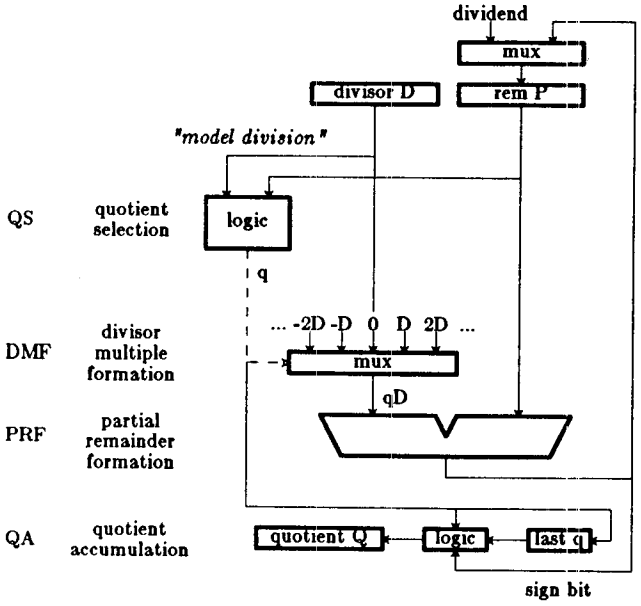

Figure 1. SRT dividers use subtraction as their fundamental iterative operation. The distinguishing feature of an SRT divider is the logic that performs "model division" in order to select the next quotient bits before they are used to compute the next remainder. The model division depends on just a few high-order bits of the remainder and the divisor.

4. SRT Division

SRT division methods produce a fixed number of quotient bits per step. Each step consists of subtracting a multiple of the divisor from the product of the radix and the remainder after the previous step (equation 1a). The quotient is accumulated in a parallel iteration (equation 1b). If the quotient digits were irredundant, as they are in restoring division methods, equation 1b would represent merely a shift register. However, SRT division methods use redundant quotient digits which must resolved into irredundant ones.

$$P_i = rP_{i-1} - q_i D \quad \text{for } i = 1, \cdots, m \tag{1a}$$

where

P_i = remainder after the ith iteration

rP_0 = dividend

r = radix

q_i = ith quotient digit

D = divisor.

$$Q_i = rQ_{i-1} + q_i \quad \text{for } i = 1, \cdots, m \tag{1b}$$

where

Q_i = accumulated quotient after the ith iteration

$Q_0 = 0$.

Q_m , the intermediate quotient before rounding, has the form $q_1\ q_2 \cdots q_m$.

Figure 1 shows the design of a simple SRT divider. Quotient digits are found in advance rather than after "trial subtraction," as they would be in a restoring divider. Knowing the quotient digits before the remainder iteration makes it possible to use only one full-precision subtractor and still produce more than one quotient bit per cycle. The QS logic observes a few high-order bits of the remainder and divisor in order to choose the next quotient digit. By necessity, quotient digits chosen this way incorporate redundancy so that later digits can correct small errors made in previous ones.

The four components of one division step are labeled in Figure 1, where the notation follows [Atki74]:

(1) quotient selection (QS),

(2) divisor multiple formation (DMF),

(3) partial remainder formation (PRF),

(4) quotient accumulation (QA).

Division hardware can be divided into three parts execute in parallel: QS, DMF and PRF lumped together, and QA. Three possible degrees of overlap between successive iterations are illustrated in Figure 2. The cycle time is not necessarily the maximum of t_{QS}, $t_{DMF} + t_{PRF}$ and t_{QA} because we can shift the boundary between parts to take advantage of one that is faster than the others.

To a first approximation, the cycle time is limited by t_{QS}, while the cost is dominated by DMF, PRF and QA. t_{QS} is the limiting factor because $t_{DMF} + t_{PRF}$ can be made smaller by holding the remainder in redundant form and t_{QA} is merely the time to clock a shift register. The costs of DMF, PRF and QA are proportional to the length of the operands, while the size of the quotient selection logic is independent of the length of the operands. The cost of the QS logic for a given radix depends entirely on its speed.

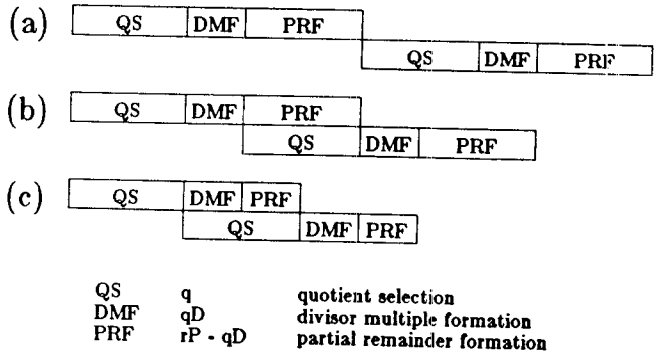

Figure 2. Three ways to overlap successive SRT division steps. (a) shows no overlap, corresponding to the hardware in Figure 1. (b) shows overlap dominated by PRF time, as for the hardware with an irredundant remainder in Figure 3. (c) shows overlap dominated by QS time, as is likely for hardware with a redundant remainder, such as in Figure 4. Quotient accumulation (QA) is not shown since it can run in parallel with the other operations regardless of the amount of overlap.

5. Considerations for a fast divider

In a previous paper [Tayl81], we showed that the radix four divider in Figure 3 has nearly the same cycle time as any radix two divider that also uses an irredundant remainder. The radix four divider needs only about 25% more hardware to achieve this two-to-one speedup, making a radix two divider look unattractive in comparison.

Since the cycle time in Figure 3 is limited by $t_{DMF} + t_{PRF}$, quotient digits {-2,-1,0,1,2} are used in order to make DMF as fast and inexpensive as possible. While quotient digits with more redundancy {-3,-2,-1,0,1,2,3} would make the QS logic simpler, they would also require DMF to produce three times the divisor.

To improve this radix four divider, we can relax any of three constraints.

(1) Use **multiple stages** of quotient selection logic and overlap them. The primary speedup is due to the overlap, but this approach also has the advantage of less overhead per stage. DMF and PRF take only slightly longer to handle multiple quotient digits per step rather than one.

(2) Make the **remainder redundant** so that PRF consists of a carry-save addition rather than a carry-propagate addition. This changes the bottleneck to quotient digit selection. As a result, radix two methods would once again be attractive if radix two QS logic took less than half the time of radix four QS logic.

(3) Use **more redundancy** in the radix four quotient digits now that QS is the bottleneck. (Radix two digits {-1,0,1} already have maximal redundancy.)

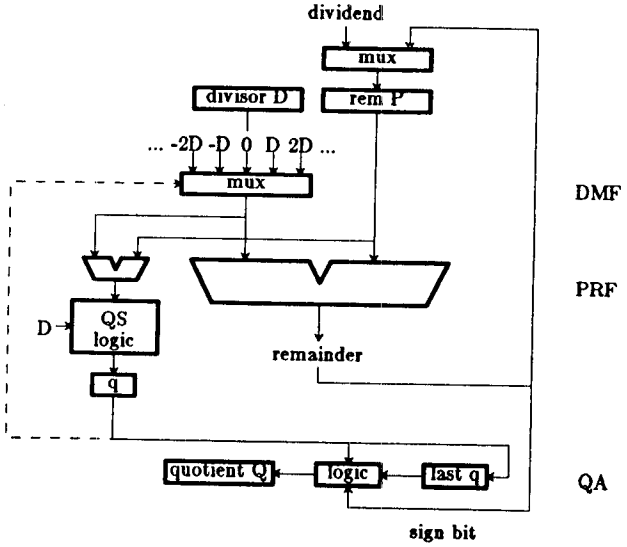

Figure 3. Radix four SRT divider with pipelined QS logic. The QS logic uses the narrow adder on the left to estimate the leading bits of the next remainder. By the time the full precision next remainder has been computed, the next quotient digit will also have been found. The full-precision quotient is accumulated on the fly by looking at consecutive pairs of quotient digits and the sign of the partial remainder.

Figure 4 shows a divider with the remainder represented redundantly. The comparisons in section 8 will use this model along with the one shown in Figure 3. Figure 4 has one multiplexer to form divisor multiples, but more could be added to handle additional quotient digits per cycle. The number of inputs to the multiplexer varies from three to seven in our examples.

The cost of redundant remainders can be seen by comparing Figure 3 with Figure 4. Figure 3 contains three full-precision registers or shift registers, while Figure 4 contains five of them, plus a carry-save adder and another full-precision carry-propagate adder.

With the organization in Figure 4, the QS and PRF hardware are almost independent. The QS logic operates one step ahead of the rest of the divider, leaving a trail of quotient digits. Because the QS logic observes only the leading bits of the remainder, its estimates of future remainders are slightly inaccurate. To keep the inaccuracies from compounding, the PRF hardware continuously refreshes the remainder bits that QS observes. The PRF hardware generates exact remainders because it uses a full-precision carry-save adder.

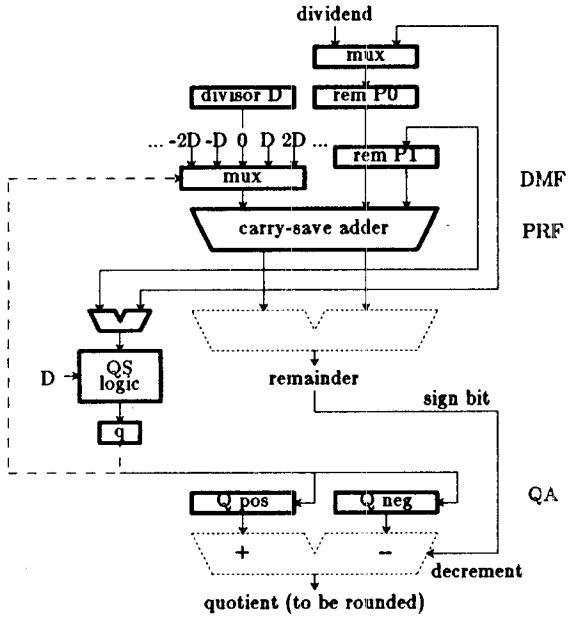

Figure 4. SRT divider with redundant remainder. The QS logic has now become the bottleneck in place of the remainder formation hardware. The functional units drawn in bold are used in every iteration. The remainder adder and the quotient subtractor shown with dashed lines are used only after the iteration ends. The quotient digits cannot be resolved until the end of the division because the sign of the remainder is never known explicitly during the iteration (contrast with Figure 3).

6. Basic QS Hardware

Now let us turn to the building blocks of a radix 16 divider. Our emphasis in the quotient selection logic is on speed rather than cost, since QS will account for only 10% to 20% of the hardware. Figure 5 shows the quotient selection hardware for (a) a radix 2 stage, (b) a radix 4 stage with mimimal redundancy, and (c) a radix 4 stage with maximal redundancy. In the radix 4 stages, the sign of the next quotient digit is determined by the adder, but the magnitude of the digit requires a logic table. The radix 2 stage is significantly faster because it determines the sign and the magnitude of a digit in parallel.

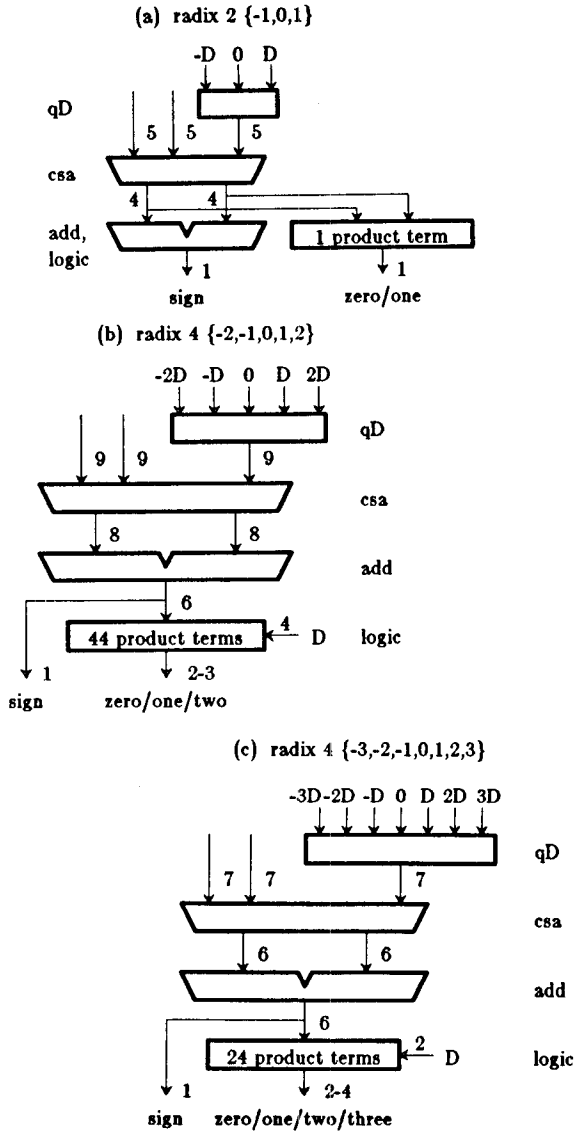

Figure 5. Single stages of quotient selection hardware. (a) radix 2, (b) radix 4 with minimal redundancy, and (c) radix 4 with maximal redundancy. Radix 2 takes less time than radix 4 because its adder is only four bits wide and its one product term can be computed in parallel with the addition. Radix 4 version (b) has more product terms, but (c) has more inputs to the *qd* multiplexer that selects a multiple of the divisor.

It is not obvious how much faster (c) is than (b) across a range of technologies. (c)'s logic has fewer product terms (24 vs. 44) and its adder is narrower (6 bits vs. 8), but its *qD* multiplexer has more inputs (7 vs. 5). When implemented with MCA II macrocells, the delay of a 6-bit adder is nearly the same as the delay of an 8-bit one, but there is a difference in the delay for the random logic. (c)'s output terms can be computed in two macrocell delays, while (b)'s output terms require three delays.

7. Overlapped QS Stages

Two stages of quotient logic can be overlapped by replicating the second stage once for each value that the first stage might produce. Each of the second stages assumes a particular outcome for the first stage, and the correct second stage is chosen based on the eventual outcome of the first stage. Figure 6 shows how this works for radix 2. The overlap is more dramatic with radix 4 QS logic, but the costs are higher, as well, because the first stage has more possible outcomes.

There are many ways to overlap stages of QS logic. To write these options in a simple form, we first convert the information from Figure 5 into the following chart:

logic steps	radix 2 -1,..,1		radix 4 -2,...,2	radix 4 -3,...,3
1	qD		qD	qD
2	csa		csa	csa
3	4-bit add,	xor	8-bit add	6-bit add
4	4-bit add,	and	8-bit add	6-bit add
5			8-bit add	6-bit add
6			logic	logic
7			logic	logic
8			logic	

csa	carry-save-adder
qD	quotient digit * Divisor

The logic steps refer to roughly equal amounts of time in an implementation.

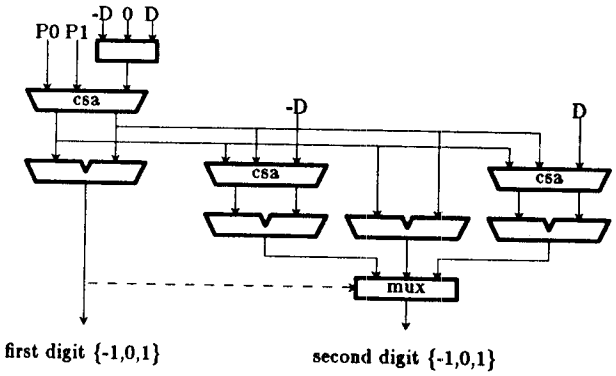

Figure 6. Overlapped stages of radix 2 quotient selection logic. The second stage has three copies, so that the second quotient digit can be found assuming any value (-1, 0 or 1) for the first quotient digit. The adder for the first digit and the middle adder for the second digit are offset from each other by one bit position.

This notation is used in Table 1 to show four ways to overlap four stages of radix 2 QS logic. The first two options show QS stages without overlap. In Option B, rows of redundant remainder PRF hardware alternate with stages of QS logic, forming a combinational divider array of the type described by [Will81]. Quotient digits are used as soon as they are formed. In Option A, on the other hand, the four quotient digits found during one cycle are consumed by the remainder hardware in the following cycle (logic steps 2 through 5). This allows the remainder to be held in irredundant form, which reduces the cost of the PRF hardware. QS runs ahead of PRF during a cycle, but does not keep any information from one cycle to the next.

Logic Steps	Option A	Option B	Option C		
1	read	read	read		
2	qD,qD,qD,qD	qD	qD,qD		
3	csa	csa	csa		
4	csa	add/logic	add/logic	csa	...
5	csa	"	"	add/logic	...
6	add/logic	qD	qD	"	...
7	"	csa	csa	mux	
8	qD	add/logic	qD		
9	csa	"	csa		
10	add/logic	qD	add/logic	csa	...
11	"	csa	"	add/logic	...
12	qD	add/logic		"	...
13	csa	"		mux	
14	add/logic	qD	setup	setup	
15	"	csa			
16	qD	add/logic			
17	csa	"			
18	add/logic	setup			
19	"				
20	setup				
Time (ns)	16.9	15.2	11.6		
Cost (cells)	150	70	160		

Logic Steps	Option D						
1	read						
2	qD,qD						
3	csa						
4	csa						
5	add	csa	...				
6	"	add	...	csa	...		
7	buf	"	...	add	...	csa	...
8	qD	mux		"	...	add	...
9	csa	buf		mux		"	...
10		qD		"		mux	
11		csa		buf		"	
12						"	
13				setup		setup	
Time (ns)	11.3						
Cost (cells)	400						

Table 1. Four ways to overlap four stages of radix 2 quotient selection logic. Logic steps represent roughly equal amounts of time. Options A and B show stages without overlap. Option B assumes that the remainder is represented redundantly in the PRF hardware, while Option A assumes an irredundant remainder. Option C shows two pairs of overlapped stages. The functions in the middle column of Option C are replicated three times on all rows marked by the three dots in the right-hand column. Option D shows maximal overlap, with functions replicated three times, then nine times, then twenty-seven times.

Options C and D show overlapped QS stages. The first two columns under Option C show hardware executing in parallel. Hardware elements in the second column are replicated on rows marked by three dots in the third column. While the four QS stages are overlapped in two pairs in Option C, all of the stages are overlapped at the same time in Option D. Hardware elements are replicated three times in columns 2 and 3, nine times in columns 4 and 5, and twenty-seven times in columns 6 and 7. Unfortunately, the overhead costs of the wide multiplexers in columns 4 and 6 cancels most of the advantage in speed that option D gains from its large degree of overlap.

Table 2 shows three options for overlapping two stages of radix 4 QS logic. The time steps are essentially the same for either set of quotient digits: {-2,-1,0,1,2} or {-3,-2,-1,0,1,2,3}. In Option A, two stages of QS logic are cascaded without overlap. Option C corresponds to the idea for maximal overlap introduced in Figure 6. Options B is an overlap that saves half of the time between Options A and C, at a cost only a little higher than Option A's.

Logic Steps	Digit Values	Option A	Option B			Option C		
1		read	read			read		
2		qD,qD	qD,qD			qD,qD		
3		csa	csa			csa		
4		add	add			add	csa	...
5		"	"			"	add	...
6		"	"			"	"	...
7		logic	logic	xor	...	logic	"	...
8		"	"	add	...	"	logic	...
9		"	"	"	...	"	"	...
10		qD		"	...		"	...
11		csa		mux			mux	
12		add		"			"	
13		"		logic		setup	setup	
14		"		"				
15		logic		"				
16		"	setup	setup				
17		"						
18		setup						
Time (ns)	-2,...,2	14.8	12.4			10.2		
Time (ns)	-3,...,3	14.0	12.0			9.8		
Cost (cells)	-2,...,2	130	180			350		
Cost (cells)	-3,...,3	110	170			370		

Table 2. Three ways to overlap two stages of radix 4 quotient selection logic. Option A has minimal overlap and assumes an irredundant remainder. Option C has maximum overlap, with the hardware in its middle column replicated either five times or seven times. Option B is about halfway between the others in speed, but much smaller in size than Option C. The hardware in its middle column is replicated either two times or three. Options B and C assume a redundant remainder in the PRF hardware and also that the high-order bits of the remainder are summed to a single operand just before the end of each cycle.

8. Performance and Costs

Table 4 is a summary of the speeds and costs for the quotient selection options given in Tables 1 and 2. Table 5 lists the cost and performance for the corresponding DMF, PRF and QA hardware. The costs in Table 5 include all of the hardware necessary to produce a quotient rounded according to the requirements of the IEEE standard. Combinations of quotient and remainder hardware to form complete division units are shown in Table 6. Detailed breakdowns of each cost and speed estimate were derived are given in [Tayl85].

The cost numbers are based on the costs for individual macrocell components listed in Table 3. The times shown in each table are maximum delays including fanout and long paths. Extra buffering required by signals with large fanout has also been taken into account. Divisor multiple formation is an example of large fanout and a long path. The QS hardware is at the most significant end of the word, so quotient digits must travel the entire length of the word during each cycle. and they must control a gate for each bit of the divisor.

Column six of Table 6 shows the number of cycles for a double precision divide. For all of the examples except the radix two non-restoring divider listed on the last line, the cycle count is broken down into the following parts:

load operands	1
load quotient pipeline	1
accumulate 55-bit quotient	14
rem P0 + rem P1, pos quo -- neg quo	1
normalize and round	1

We assume that the dividend and the divisor are loaded in parallel in the first cycle. The next step is to load the two-stage pipeline by finding the first quotient digits. This is followed by

component		cost per bit (cells)	notes
	flip-flop	1	
qD	quo * divisor	0.5	q = {-1,0,1}
		1	q = {-2,-1,0,1,2}
		1.5	q = {-3,-2,-1,0,1,2,3}
csa	carry-save-adder	1	slow
	carry-save-adder	2	fast
	4-bit adder	1.5	
	8-bit adder	2	
	64-bit adder	2.5	
	2:1 mux	0.5	
	2-input xor	0.5	
	4-input or/nor	0.5	
	3 3 or-and gate	0.5	
	radix 4 QS logic	12 total	{-2,-1,0,1,2}
	radix 4 QS logic	8 total	{-3,-2,-1,0,1,2,3}

Table 3. Cost of Divider Components. Based on the macrocells in the Motorola MCA II ECL gate array.

14 quotient accumulation cycles. After the inner loop is finished, the redundant remainder and redundant quotient are resolved. The two adds do not have to execute sequentially because we only care about the remainder's sign. The carry propagation circuitry of the two adders can be connected so that the total delay is only slightly more than one add time. The last cycle is for normalization (a one bit shift or no shift) and rounding.

The number of cycles with an redundant remainder is 18. If the remainder is irredundant, then the quotient can be accumulated on the fly, so the next to last cycle is unnecessary. The non-restoring divider takes one cycle to load the operands, 55 cycles to generate quotient bits, and one cycle for normalization and rounding.

9. An Implementation in the S-1 Mark IIB

Radix 4 Option B quotient selection logic is the basis for the divider in the S-1 Mark IIB computer under development at Lawrence Livermore Laboratory. This divider consists of eight ECL gate arrays: one chip for quotient selection and one chip (a 10-bit slice, replicated seven times) for the remainder iteration, quotient accumulation and rounding hardware. The cycle time is 12.5 nanoseconds.

A multiple chip design introduces extra delays not considered in our hypothetical study in previous sections. These delays make it necessary to use a redundant remainder and to confine the quotient selection unit to a single chip. The quotient selection chip is designed so the critical path never leaves the chip. All of the time required to propagate signals between chips is part of the remainder iteration cycle.

radix of each QS stage & option	quo digits	redundant rem?	**time per four bits (ns)**	QS cost (cells)
2 A	-1,0,1	no	**16.9**	150
2 B	-1,0,1	yes	**15.2**	70
2 C	-1,0,1	yes	**11.6**	160
2 D	-1,0,1	yes	**11.3**	400
4 A	-2,...,2	no	**14.8**	130
4 B *	-2,...,2	yes	**12.4**	180
4 C	-2,...,2	yes	**10.2**	350
4 A	-3,...,3	no	**14.0**	110
4 B	-3,...,3	yes	**12.0**	170
4 C	-3,...,3	yes	**9.8**	370

Table 4. Speed and Cost of Radix 16 Quotient Selection Options. All times are worst-case and include delays due to fanout and estimated wire lengths. Times are in bold because quotient selection delays are the limiting factor in the divider as a whole. Option 4 B * is used in the S-1 Mark IIB implementation described in section 9.

radix of each QS stage	quo digits	redundant rem?	time per four bits (ns)	**DMF, PRF, QA cost (cells)**
2 nonrestoring	-1,1	no	45.2	**600**
2	-1,0,1	no	15.3	**1050**
4	-2,...,2	no	12.7	**950**
4	-3,...,3	no	13.0	**1150**
2	-1,0,1	yes	9.6	**1400**
4 *	-2,...,2	yes	8.5	**1250**
4	-3,...,3	yes	8.8	**1500**

Table 5. Speed and Cost of Remainder Formation Options. The first line is a radix 2 restoring divider. All of the others are radix 16. Costs are based on 64-bit operands, as would be necessary for IEEE extended precision floating point or for 64-bit integer arithmetic. Costs are shown in bold because the divisor multiple formation, partial remainder formation and quotient accumulation functions account for most of the hardware in a complete divider.

QS option	quo digits	redundant rem?	time per cycle (ns)	cycles for 53-bit rounded fraction	**total time: double precision (ns)**	QS cost (cells)	DMF, PRF, QA cost (cells)	**total cost (cells)**
4 C	-3,...,3	yes	9.8	18	**176**	370	1500	**1870**
4 C	-2,...,2	yes	10.2	18	**184**	350	1250	**1600**
2 D	-1,0,1	yes	11.3	18	**203**	400	1400	**1800**
2 C	-1,0,1	yes	11.6	18	**209**	160	1400	**1560**
4 B	-3,...,3	yes	12.0	18	**216**	170	1500	**1670**
4 B *	-2,...,2	yes	12.4	18	**223**	180	1250	**1430**
4 A	-3,...,3	no	14.0	17	**238**	110	1150	**1260**
4 A	-2,...,2	no	14.8	17	**252**	130	900	**1030**
2 B	-1,0,1	yes	15.2	18	**274**	70	1400	**1470**
2 A	-1,0,1	no	16.9	17	**287**	150	1050	**1200**
non-restoring	-1,1	no	11.3	57	**644**	0	600	**600**

Table 6. Speed and cost of various dividers based on combinations from Tables 4 and 5. Times are for double precision floating point with a 53-bit fraction. Costs are based on 64-bit operands. The bottom row shows a radix two non-restoring divider. All of the other dividers are radix 16.

The Mark IIB performs integer division and remainder operations by first converting the operands to floating point format, because SRT division depends on a normalized divisor. The conversion is exact because 64-bit integers will fit into the sign and fraction fields of an IEEE extended precision floating point number. The number of integer quotient bits produced is one plus the difference between the operands' exponents.

Table 7 shows that division times are three to five times longer than multiplication times, depending on the precision. The multiplication hardware consists of about three times as many gate arrays as the divison hardware. Table 8 compares the Mark IIB's performance with that of other machines.

10. Summary

Our study compared radix 2 and radix 4 building blocks for the quotient selection logic of a radix 16 divider. We found that radix 4 quotient selection stages are cost-effective building blocks for multiple stage dividers. At least for radix 16, it appears to be cheaper and faster to use overlapped radix 4 stages than to use overlapped radix 2 stages.

11. References

[Atki67]
D. E. Atkins, "The Theory and Implementation of SRT Division," Report No. 230, Dept. of Computer Science, University of Illinois, June, 1967.

[Atki70a]
D. E. Atkins, "A Study of Methods for Selection of Quotient Digits During Digital Division," Ph.D. Dissertation, Report No. 397, Dept. of Computer Science, University of Illinois, June, 1970.

[Atki70b]
D. E. Atkins, "Design of the Arithmetic Units of Illiac III: Use of Redundancy and Higher Radix Methods," *IEEE Transactions on Computers,* Vol. C-19, No. 8, pp. 720-723, August, 1970.

[Atki74]
D. E. Atkins and U. Kalaycioglu, "Concurrency in Generalized Radix Non-Restoring Division," *Proceedings of the Twelfth Allerton Conference on Circuit and Switching Theory,* pp. 628-640, October, 1974.

[Cray82]
"Cray X-MP Computer Systems Mainframe Reference Manual," Cray Research, Inc., 1982.

[Erce83]
M. D. Ercegovac, "A Higher-Radix Division with Simple Selection of Quotient Digits," *Proceedings of the Sixth IEEE Symposium on Computer Arithmetic,* pp. 94-98, June, 1983.

[Foss85]
T. Fossum, "Floating Point Processor for the VAX 8600," *Digest of Papers, Thirtieth IEEE Compcon,* pp. 176-180, February, 1985.

[IEEE83]
IEEE Computer Society Microprocessor Standards Committee, "A Proposed Standard for Binary Floating Point Arithmetic, Draft 10.0," January, 1983.

[Ibbe82]
R. N. Ibbett, *The Architecture of High Performance Computers,* Springer-Verlag, New York, 1982. "

[Prio84]
J. Prioste and A. Bass, MECL MCA II Macrocell Array Design Manual, Motorola Semiconductor Products, Inc., 1984.

[Robe58]
J. E. Robertson, "A New Class of Digital Division Methods," *IRE Transactions on Electronic Computers,* vol. EC-7, no. 9, pp. 218-222, September, 1958.

[Tan78]
K. Tan, "The Theory and Implementations of High-Radix Division," *Proceedings of the Fourth IEEE Symposium on Computer Arithmetic,* pp. 154-163, October, 1978.

[Tayl81]
G. Taylor, "Compatible Hardware for Division and Square Root," *Proceedings of the Fifth IEEE Symposium on Computer Arithmetic,* pp. 127-134, May, 1981.

[Tayl83]
G. Taylor, "Arithmetic on the ELXSI 6400," *Proceedings of the Sixth IEEE Symposium on Computer Arithmetic,* pp. 110-115, June, 1983.

[Tayl85]
G. Taylor, "Radix 16 SRT Division Methods With Overlapped Quotient Selection Stages," Technical Report, U. C. Berkeley Computer Science Division, 1985.

[Toch58]
T. D. Tocher, "Techniques of Multiplication and Division for Automatic Binary Computers," *Quarterly Journal Mech. Appl. Mathematics,* vol. 11, part 3, pp. 364-384, 1958.

[Will81]
J. Williams and V. C. Hamacher, "A Linear-Time Divider Array," *Canadian Electrical Engineering Journal,* Vol. 6, No. 4, pp. 14-20, 1981.

	single precision (ns)	double precision (ns)	extended precision (ns)
Add/Subtract	50	50	50
Multiply	50	50	50
Divide	150	225	275

Table 7. S-1 Mark IIB Floating Point Execution Times

Computer	Double Precision Floating Point Division Time (ns)	Method
S-1 Mark IIB	225	radix 16 SRT
Cray-1 X-MP	275	inverse & multiply
Cray-1	362	inverse & multiply
CDC 7600	550	radix 8 restoring
ELXSI 6400	1700	radix 4 SRT
VAX 8600	5360	radix 2 nonrestoring
VAX 11/780	8800	radix 2 restoring

Table 8. Register to register scalar division times. The various floating point formats have between 48 and 56 fraction bits. [Foss85][Ibbe82]