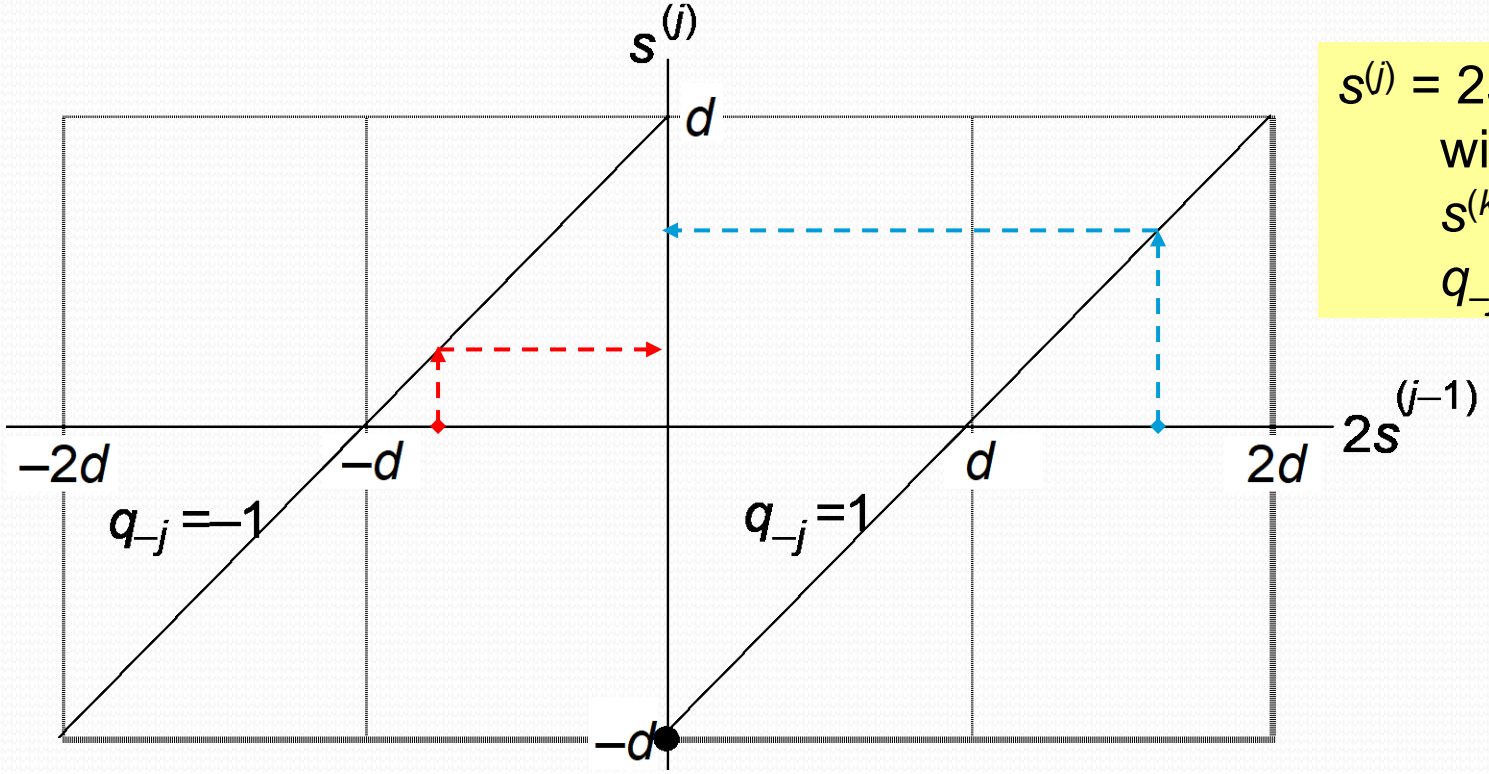
SRT Division Algorithm

13.6 Radix-2 SRT Division

SRT division takes its name from Sweeney, Robertson, and Tocher, who independently discovered the method

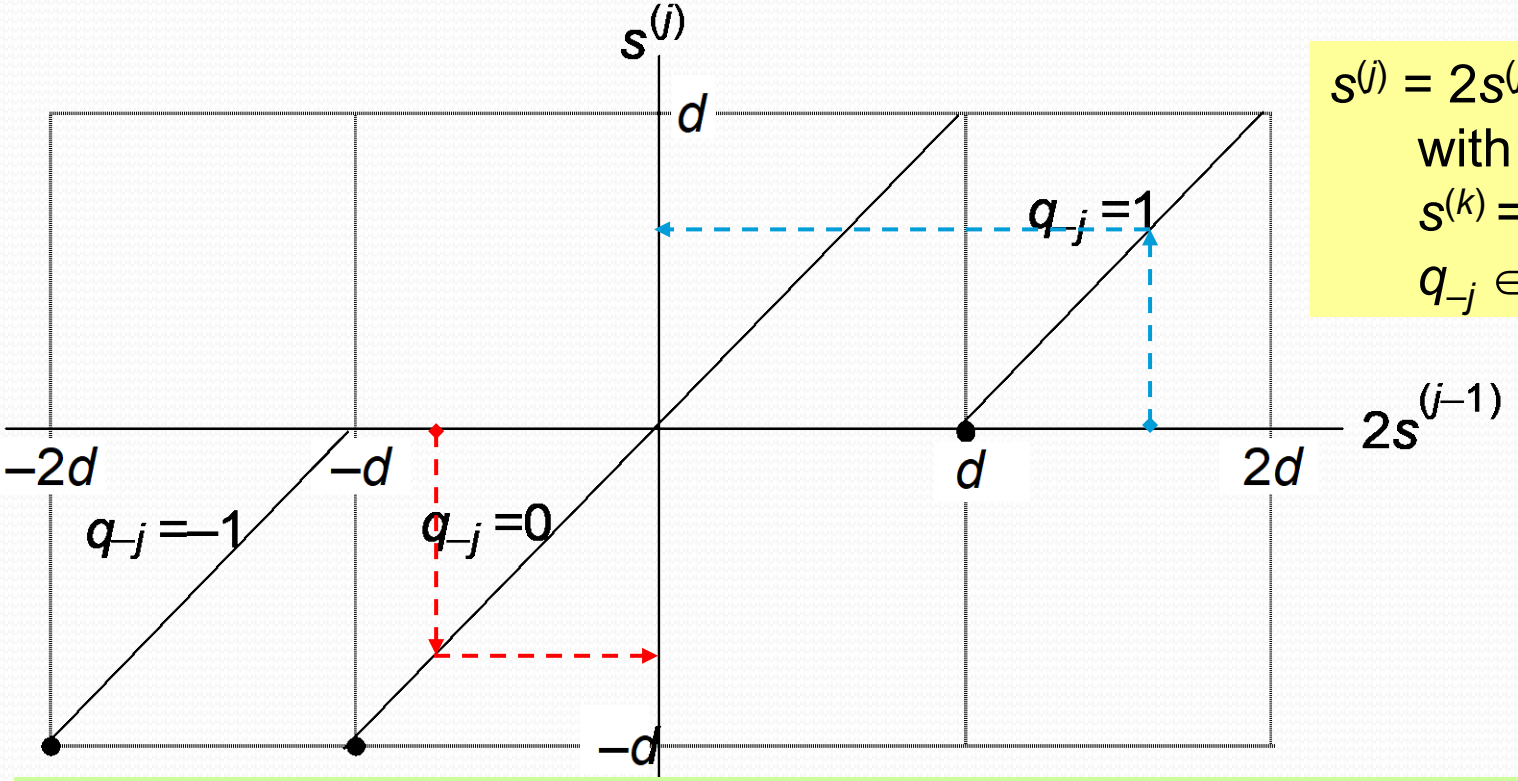


$$s^{(j)} = 2s^{(j-1)} - q_{-j}\, d$$
with $s^{(0)} = z$
$s^{(k)} = 2^k s$
$q_{-j} \in \{-1, 1\}$

Fig. 13.11 The new partial remainder, $s^{(j)}$, as a function of the shifted old partial remainder, $2s^{(j-1)}$, in radix-2 nonrestoring division.

Allowing 0 as a Quotient Digit in Nonrestoring Division

This method was useful in early computers, because the choice $q_{-j} = 0$ requires shifting only, which was faster than shift-and-subtract



$s^{(j)} = 2s^{(j-1)} - q_{-j}\, d$
with $s^{(0)} = z$
$s^{(k)} = 2^k s$
$q_{-j} \in \{-1, 0, 1\}$

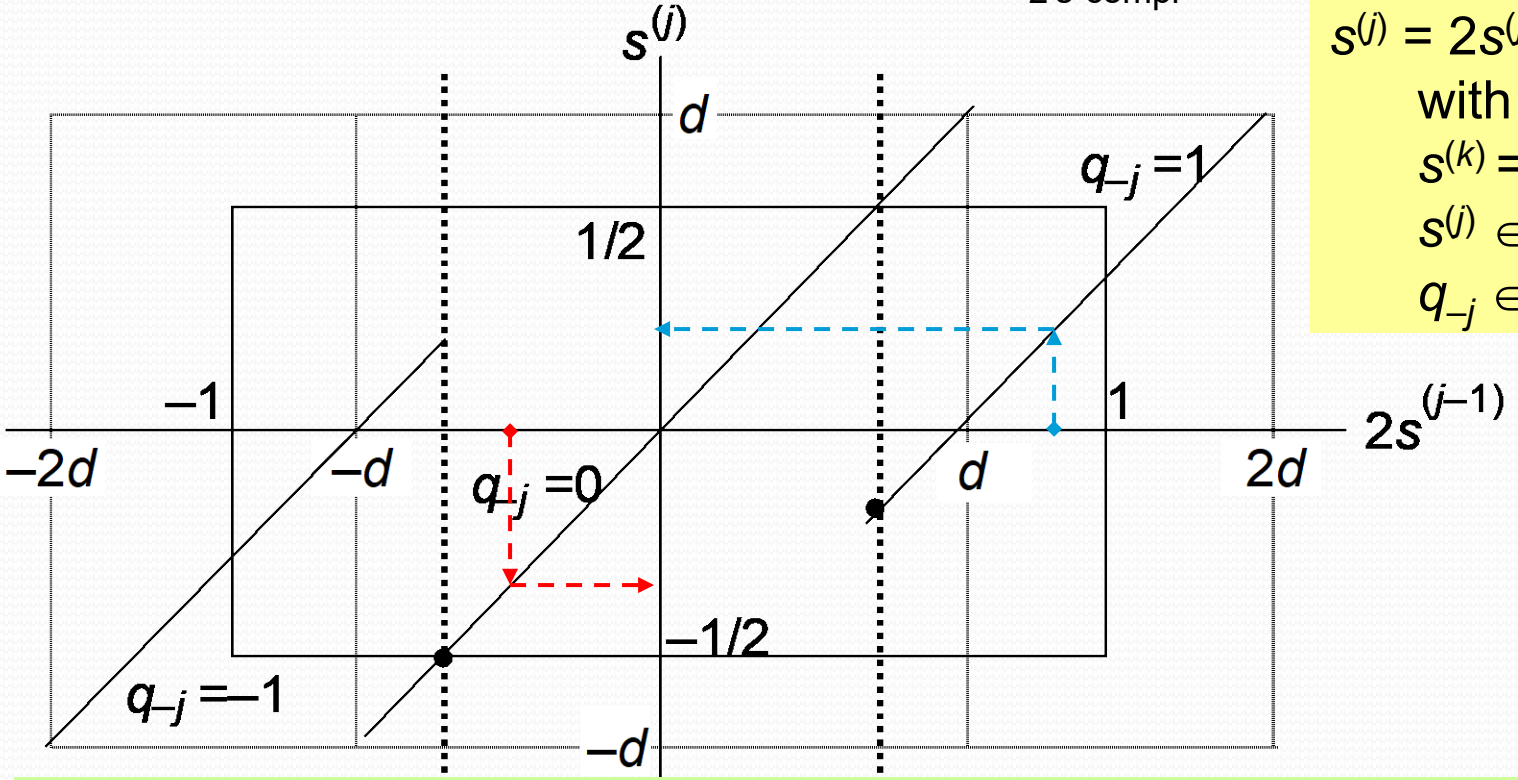
Fig. 13.12 The new partial remainder, $s^{(j)}$, as a function of the shifted old partial remainder, $2s^{(j-1)}$, with q_{-j} in {–1, 0, 1}.

The Radix-2 SRT Division Algorithm

We use the comparison constants –½ and ½ for quotient digit selection

$2s \geq +½$ means $2s = (0.1xxxxxxxx)_{2\text{'s-compl}}$

$2s < -½$ means $2s = (1.0xxxxxxxx)_{2\text{'s-compl}}$



$s^{(j)} = 2s^{(j-1)} - q_{-j}\, d$

with $s^{(0)} = z$

$s^{(k)} = 2^k s$

$s^{(j)} \in [-½, ½)$

$q_{-j} \in \{-1, 0, 1\}$

Fig. 13.13 The relationship between new and old partial remainders in radix-2 SRT division.

Example Unsigned Radix-2 SRT Division

In $[-½, ½)$, so okay

z	.0100 0101
d	0.1010
$-d$	1.0110
$s^{(0)}$	0.0100 0101
$2s^{(0)}$	0.1000 101
$+(-d)$	1.0110
$s^{(1)}$	1.1110 101
$2s^{(1)}$	1.1101 01
$s^{(2)} = 2s^{(1)}$	1.1101 01
$2s^{(2)}$	1.1010 1
$s^{(3)} = 2s^{(2)}$	0.1010 1
$2s^{(3)}$	1.0101
$+d$	0.1010
$s^{(4)}$	1.1111
$+d$	0.1010
$s^{(4)}$	0.1001
s	0.0000 0101
q	$0.100^{-}1$
q	0.0110

$\geq ½$, so set $q_{-1} = 1$ and subtract

In $[-½, ½)$, so set $q_{-2} = 0$

In $[-½, ½)$, so set $q_{-3} = 0$

$< -½$, so set $q_{-4} = {}^{-}1$ and add

Negative, so add to correct

Uncorrected BSD quotient
Convert and subtract *ulp*

0.1 Choose 1
1.0 Choose $^{-}1$
0.0/1.1 Choose 0

Fig. 13.14 Example of unsigned radix-2 SRT division.

SRT Division Diagrams and Their Usage in Designing Custom Integrated Circuits for Division

T. E. WILLIAMS and M. HOROWITZ

TECHNICAL REPORT: CSL-TR-87-328

NOVEMBER 1986

This paper has been supported by a National Science Foundation Fellowship.

SRT Division Diagrams and their Usage in Designing Custom Integrated Circuits for Division

T.E. Williams and M. Horowitz

Technical Report No. 87-326

November 1986

Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Abstract

This paper describes the construction and analysis of several diagrams which depict SRT division algorithms. These diagrams yield insight into the operation of the algorithms and the many implementation tradeoffs available in custom circuit design. Examples of simple low radix diagrams are shown, as well as tables for higher radices. The tables were generated by a program which can create and verify the diagrams for different division schemes. Also discussed is a custom CMOS integrated circuit designed which performs SRT division using self-timed circuit techniques. This chip implements an intermediate approach between a fully combinational array and a fully iterative in time method in order to get both speed and small silicon area.

Key Words and Phrases: division, SRT, self-timed, floating-point, VLSI

by

T.E. Williams and M. Horowitz

Contents

List of Figures

List of Tables

1 Introduction

There has been much emphasis placed on providing hardware support for floating-point addition, subtraction, and multiplication because these operations have seemed easier to enhance than division. Accordingly, for high performance, division has been avoided in algorithms whenever possible. Whereas hardware circuits have been designed using multi-stage carry look ahead and Wallace trees to do addition, subtraction and multiplication in $O[\log(n)]$ time where n is the word length in bits, the direct algorithms for doing division require a full addition or subtraction operation for every output bit requiring $O[n \log(n)]$ time and hence are undesirable for modern computers.

There are many approaches beyond the direct ones for building computer arithmetic hardware dedicated to division. Since high-speed multipliers are part of all floating-point designs, many of the methods for division are really algorithms using multiplication iteratively to perform division. The very popular Newton/Raphson iteration technique, and the Divisor reciprocal technique are examples of this approach[3]. In circuits which are designed specifically for division, there exist iterative division algorithms which, although requiring quotient digit selection between arithmetic operations, can base this selection on approximations of the true operands. These algorithms, which are a form of the SRT algorithms[5][9], allow the intermediate results to be calculated in carry-save form. By avoiding complete carry propagation, they can achieve intermediate results in a time independent of word length, and hence the overall division can be computed in $O[n]$ time.

In parallel with the design of a radix 2 chip, we examined the tradeoffs of SRT division in general. The algorithms are characterized by a series of diagrams which show graphically the required constraints and help provide an intuitive understanding of the design tradeoffs involved, such as those affecting the precision of the approximations. We developed a set of tools to generate and analyze these diagrams which enabled us to evaluate the choices within higher radix division schemes and possible modifications such as using prescaling.

The next section gives a brief review of SRT division, and then section 3 describes the construction and use of the Robertson, Taylor, and colored Taylor diagrams. On the basis of these diagrams, some different division schemes are examined in sections 4 and 5, and hardware implementations are discussed in section 6. A summary of our findings is given in section 7.

2 SRT division

Performing division requires making a choice of quotient digits starting with the most significant, and progressing to the least significant digits. The quotient digit decision is made as a part of each iteration which recomputes the partial remainder based on the last partial remainder and quotient digit. The complete quotient is accumulated from the equation:

$$Q = \sum_{i=0}^{n-1} q_i r^{-i} \tag{1}$$

where

- r is the radix
- n is the number of quotient digits calculated
- Q is the accumulated quotient result with a precision of $r^{-(n-1)}$

q_i is the quotient digit determined from stage i

Since in binary hardware the full quotient result is easiest to form if it is merely the concatenation of the bits of the individual digits, we set the radix $r = 2^m$ where m is the number of quotient bits determined at each stage.

In irredundant division, the quotient digits are in the set $\{0, \ldots . r-1\}$, and the full quotient has only a single valid representation since each digit position in the quotient has only a single correct possibility. Unfortunately, determining the correct digit at each position requires comparison of the entire partial remainder, and this means that the entire partial remainder must be computed before making each quotient digit selection. This computation requires a complete carry propagation along the length of the partial remainder before each quotient digit may be selected. These irredundant division schemes are much slower than multiplication because multiplication does not require such a carry propagation in order to compute partial results.

A complete carry propagation in each iteration can be avoided by making the set of valid quotient digits redundant by including both positive and negative integers. In this method, the divisor and dividend must be normalized to the same binary range, and the valid quotient digits for a maximum quotient digit p are in the set $\{-p, \ldots . 0, \ldots . p\}$ whichis symmetric about, and includes, zero. The quotient digit chosen at each stage in the division determines the operation computing the next partial remainder according to the equation:

$$R_{i+1} = rR_i - Dq_i \tag{2}$$

where

R_i is the partial remainder output from stage i
D is the Divisor

and the sequence is initialized with

rR_0 = the Dividend

With redundant quotient digit sets, the final quotient result can be represented in several different ways giving a choice of quotient digits for each position. Any valid representation can always, of course, be converted to the original irredundant representation containing no negative digits by subtracting the positionally weighted negative quotient digits from the positionally weighted positive digits. This subtraction requires a carry propagation, but it is a single operation which needs only to be performed once for the whole division operation rather than once per stage. Further, in a floating-point chip, this full-length carry-propagate operation could be performed by shipping the quotient results to the part of the chip used for other floating-point additions.

With the redundancy in the quotient digits of SRT division, the quotient selection for a given position need only use an approximation of the divisor and partial remainder, because small errors may be corrected with less significant quotient bits of the opposite sign. Since only an approximation of the divisor and partial remainder is required at each stage for the selection of quotient digits, only a small number of the most significant bits need to be examined. The number of bits not examined determines the maximum error in assessing the values of the divisor and partial remainder. The

unexamined bits of lower significance in the partial remainder may be kept in carry-save form, thus avoiding a full word carry propagation.

The radix r bounds the maximum digit p in the set of quotient digits which is representable in m bits in the range: [1]

$$\frac{r}{2} \leq p \leq r - 1 \tag{3}$$

Quotient digit sets where $p = \frac{r}{2}$ have minimal redundancy, whereas quotient digit sets at the other end of the range where $p = r - 1$ have maximal redundancy. More redundancy lessens the complexity of the quotient selection logic and allows fewer partial remainder bits to be examined, but also means that more multiples of the divisor must be formed. These multiples of the divisor must be either precomputed and shipped around the chip, sacrificing wiring area, or computed at each stage, costing time. Thus, choosing the value of p trades quotient selection logic complexity with divisor multiple formation complexity.

3 SRT Constraint Diagrams

SRT division chooses valid quotient digits based upon approximations obtained by examining only the top few bits of the divisor and partial remainder at each stage. In order to determine how many bits of divisor and partial remainder need to be examined to determine the correct quotient digit, one can graphically depict the required constraints with a series of diagrams. These diagrams, called the Robertson, Taylor, and colored Taylor diagrams yield insight helpful in making implementation tradeoffs setting the quotient digit set and the precision of the approximations. The diagrams for the three simplest cases of SRT division are shown as figures 1, 2, and 3. Figure 1 examines the case of radix 2 division where the quotient digit set is $\{-1, 0, 1\}$, figure 2 examines the case of maximally redundant radix 4 division where the quotient digit set is (-3, -2, $-1, 0, 1, 2, 3\}$, and figure 3 examines the case of minimally redundant radix 4 division where the quotient digit set is $(-2, -1, 0, 1, 2\}$.

3.1 Robertson Diagram

In each set of figures, the Robertson diagram[1][3], shown in part a, is a plot of the next partial remainder scaled by the divisor, R_{i+1}/D, versus the radix times the current partial remainder scaled by the divisor, rR_i/D. The diagonal lines correspond to the different possible choices of quotient digits in equation (2). The bold horizontal lines on the diagram at $\left|\frac{R_{i+1}}{D}\right| = \frac{p}{r-1}$ restrict the choice of quotient digits so that

$$\left|\frac{R_{i+1}}{D}\right| \leq \frac{p}{r-1} \tag{4}$$

This is required to keep $\left|r\frac{R_{i+1}}{D}\right| \leq r\frac{p}{r-1}$ since $r\frac{p}{r-1}$ is the maximum remainder which can be correctly reduced by subsequent quotient digits. For values on the abscissa where there is only one diagonal line between the bold horizontal bounds, the quotient digit corresponding to this line is the one which must be selected. For values where there is more than one diagonal line, the quotient digits corresponding to either of the lines may be selected. It is apparent that the bounds for choosing

quotient digits are looser for quotient digit sets with more redundancy, and tighter for those with less redundancy.

3.2 Taylor Diagram

The second useful type of diagram, which we will name the Taylor diagram based upon the work in [7], graphs the shifted partial remainder rR_i versus the divisor D, and is shown in part b of figures 1, 2, and 3. These graphs are drawn for divisors and dividends normalized to the range $[1,2)$ to conform with the IEEE floating-point standard. The graphs are shaded to show the range of partial remainder and dividend for which each quotient digit is valid. When the restriction of equation (4) is applied to equation (2), it is clear that q_i must be selected so that:

$$q_i - \frac{p}{r-1} \leq r\frac{R_i}{D} < q_i + \frac{p}{r-1} \tag{5}$$

Each stipple pattern on the Taylor diagram indicates the valid region for a quotient digit according to equation (5). The unshaded regions on the extreme right and left of the inverted trapezoid correspond to the values of partial remainder outside of the range generated by a correct SRT algorithm, and hence can be considered "don't care" regions. For those ranges of partial remainder for which the Robertson diagram indicates that there is more than one valid quotient digit, the Taylor diagram will contain more than one stipple pattern. Hence, singly shaded regions in the Taylor diagram are regions where there is only a single valid quotient digit, and overlapping shading indicates regions where there is a choice of valid quotient digits.

3.3 Colored Taylor Diagram

At each stage in a division sequence the quotient selection logic must choose the quotient digit that will be used to form the partial remainder in the next stage. This selection of a valid quotient digit must be performed by choosing one of the valid stipple patterns covering the corresponding point in the Taylor diagram. A third type of diagram, which we call the colored Taylor diagram, can be drawn to show which stipple pattern will be chosen for every point in the Taylor diagram. The colored Taylor diagram is constructed by painting every region of the Taylor diagram with a color which corresponds to one of the valid stipple patterns for that region. The painting task is made difficult because the divisor and partial remainder are only known to a limited precision. Graphically, this corresponds to painting with a rectangular paint brush which has a vertical size corresponding to the maximum value of the unexamined bits of the divisor, and a horizontal size corresponding to the sum of the maximum value of the unexamined sum bits and the unexamined or unpropagated carry bits of the partial remainder. The grid spacing on which the paint brush can be positioned corresponds to the tolerances to which the divisor and partial remainder are examined. The brush and grid sizes for each case are then determined:

$$\begin{aligned}
\text{grid height} &= 2^{(-\text{DivisorBits})} \\
\text{brush height} &= 2^{(-\text{DivisorBits})} \\
\text{LeftBits} &= 2 + \left\lceil \log_2 \left(\frac{pr}{r-1} \right) \right\rceil \\
\text{grid width} &= 2^{(\text{LeftBits}-\text{RBits})}
\end{aligned}$$

$$\text{brush width} = 2^{(\text{LeftBits}-\text{RBits})} + 2^{(\text{LeftBits}-\text{CBits})}$$

where

r	is the radix
p	is the maximum quotient digit in the quotient digit set $\{-p, \ldots 0, \ldots p\}$
LeftBits	is the number of partial remainder bits to the left of the binary point
DivisorBits	is the number of divisor bits examined
RBits	is the number of partial remainder sum bits examined
CBits	is the number of partial remainder carry bits examined

Observe that the vertical spacing of the grid is always the same as the vertical size of the brush, but the horizontal sizes will differ because the brush size shows the additional uncertainty of the unpropagated carry bits in the partial remainder. In order to achieve the simplest and fastest quotient selection logic, the goal of coloring is to find the largest valid brush and grid sizes possible in order to use the lowest precision approximations of the divisor and partial remainder.

Each splotch of paint laid by the paint brush in the colored Taylor diagram selects a particular quotient digit for the grid point of divisor and partial remainder value on which it is aligned. Because the unexamined sum and carry bits can only make the actual value of the divisor and partial remainder greater than the examined value, the brush lays a splotch of' paint aligned to the grid on its lower left corner. When a splotch is laid, the paint brush must fit wholly inside the region on the Taylor diagram for which the particular quotient digit is valid (the region which has the same color as the brush); the paint brush must stay within the lines of the original Taylor diagram. Since the paint brush size is larger than the grid spacing, there will be some overlap of the splotches. Where there is an overlap of different color paint, this denotes that the points within the overlap region could be represented, due to redundancy, by two different combinations of most significant bits. The quotient digit chosen will be the one corresponding to the actual combination of most significant partial remainder bits which occurs at the particular stage in the division.

The size of the smallest splotch required to meet the constraints of the stipple patterns determines the maximum paint brush size, and this determines the required number of bits of divisor and partial remainder that must be examined. For a typical case in which the sum and carry components of the partial remainder are examined to the same precision, and the carry is only propagated up the examined bits, the horizontal paint brush size will be twice the horizontal grid spacing.

A reason why it is significant to consider exactly how many bits of divisor and partial remainder are necessary is because the number of bits will directly affect the critical path and silicon area taken by a hardware implementation. In implementing arithmetic circuits with standard packaged components, there was strong motivation in keeping data path widths a multiple of the bit slice widths available; however, in custom VLSI integrated circuit design, the number of bits in adders and other data paths can be freely chosen to equal precisely the required amounts.

There is actually much choice in coloring the Taylor diagrams for higher radices with the more redundant quotient digit sets. The desired result of the coloring task is a diagram which can be encoded in a PLA with a minimal number of terms. The number of inputs to the PLA is determined by the grid and brush sizes used to color the Taylor diagram, and the number of outputs by the

quotient digit set. In making first-order comparisons, one can assume that the complexity of the PLA is related to the number of inputs and outputs. Finding the true minimal PLA is actually an interesting logic minimization problem. Not only can the equations which generate a particular quotient digit be minimized, but there is the global minimization of choosing which quotient digit color to use at the points where there is choice. Although there is a set of heuristics for choosing the colors which works well, further work still remains in performing this minimization with standard truth table minimization packages such as ESPRESSO[6].

4 SRT Diagram Examples

For radix 2, as shown in part c of figure 1, the graph can be colored correctly with a paint brush of length 2 horizontally placed on a grid of spacing 1 horizontally. There is no restriction on the brush or grid vertically. This says that the divisor need not be examined at all to determine a valid quotient digit, and that the partial remainder need be examined only to the unity weighted bit of rR_i. Thus, the bits of the partial remainder to the left of the binary point are all that need to be examined by the quotient selection logic. Since the maximum value taken by the partial remainder is 4, and since a sign bit is required, there are three bits to the left of the binary point, and these three bits are the only ones required by the quotient selection logic.

For radix 4 with the maximum quotient digit $p = 3$, the graph can be colored correctly in the worst case regions only if the divisor is known with a tolerance of $\frac{1}{4}$, so that two bits to the right of the normalized (unity) bit must be used in the selection logic. The horizontal brush size must be 1, and so with the horizontal grid size $\frac{1}{2}$, the uncertainty in the partial remainder due to the carry bits must also be $\frac{1}{2}$, and so the sum and carry bits of the partial remainder can be examined to the same precision. Since $\frac{1}{2}$ corresponds to using 1 bit to the right of the binary point, and 4 bits to the left are also required, the quotient selection logic must examine a total of 5 bits of the partial remainder and 2 of the divisor. The colored Taylor diagram is shown in part c of figure 2.

For radix 4 with the maximum quotient digit $p = 2$, the graph can be colored correctly in two different ways. If the divisor is examined to within a precision of $\frac{1}{8}$, then the partial remainder must be known to within a tolerance of $\frac{1}{4}$. Another choice is to approximate the divisor with a precision of $\frac{1}{16}$, in which case the remainder need only be determined with a precision of $\frac{5}{16}$. Hence, one can tradeoff bits of the divisor with bits of the remainder. One choice is to use 3 bits to the right of the normalized bit of the divisor, and 3 bits to the right of the binary point of the remainder when the carry is propagated up from the same precision of 3 bits. This first coloring choice has been given in tabular form in [7]. The other choice is to use 4 bits of the divisor, and 2 bits to the right of the binary point of the remainder approximation which has had the carry propagated up from a position 4 bits to the right of the binary point. There are two reasons why the latter choice, illustrated in the colored Taylor diagram shown in part c of figure 2, is more desirable than the former for a hardware implementation. The first is that, in general, propagating a small additional number of carry bits is preferable if it simplifies the quotient selection complexity, because the time required to propagate the carry a few extra bits is likely to be less than the additional time required for a larger PLA to operate. An additional reason the second choice is better is because using divisor bits is preferable over using remainder bits since divisor bits do not change during the iterations of a division and there need be no propagation or setup time allowed for those inputs to the PLA; so, the critical path could be shortened.

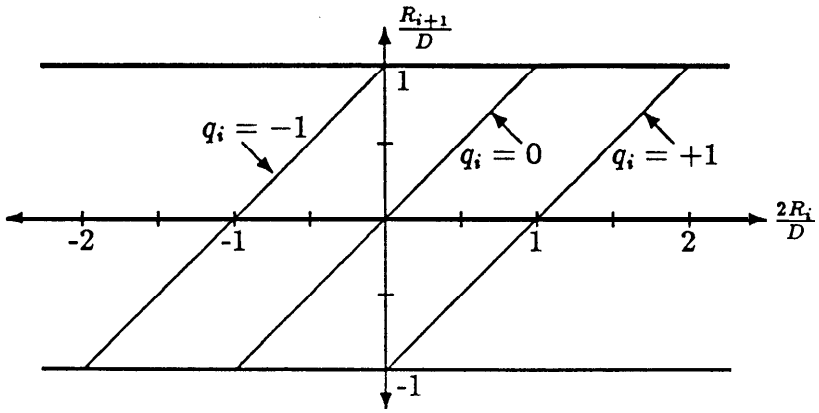


Figure **1a:** Robertson Diagram for Radix 2 with quotient digit set {-1,0,1}

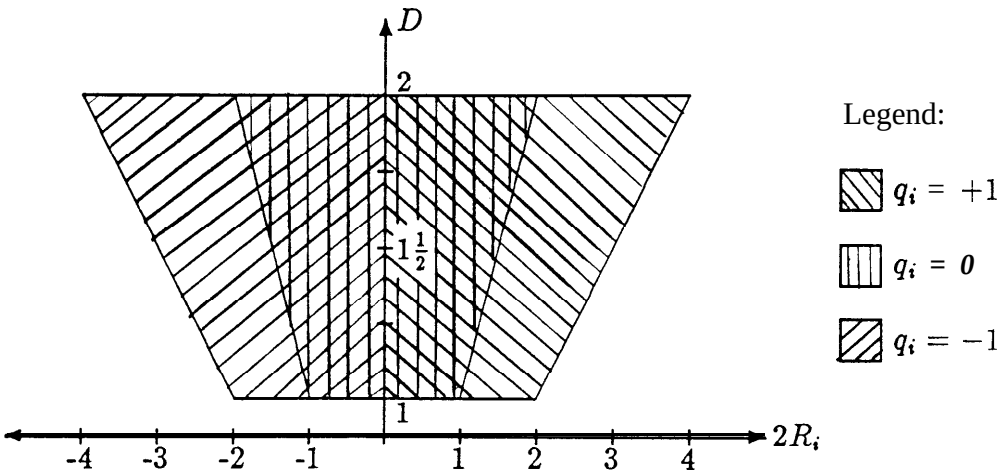


Figure **1b:** Taylor Diagram for Radix 2 with quotient digit set {-1,0,1}

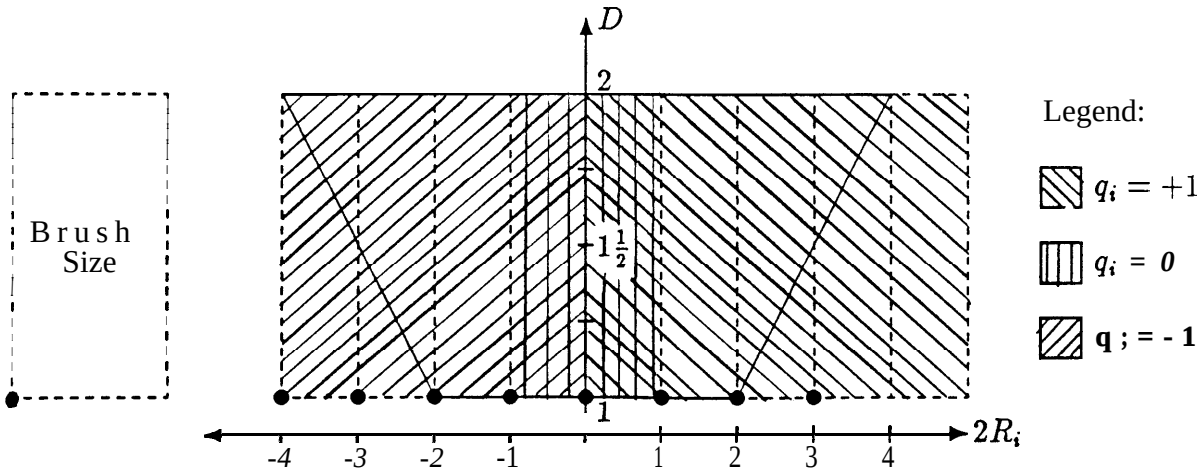


Figure **1c:** Colored Taylor Diagram for Radix 2 with quotient digit set {-1,0,1}

Figure **1:** SRT diagrams for Radix 2 with quotient digit set {-1,0,1}

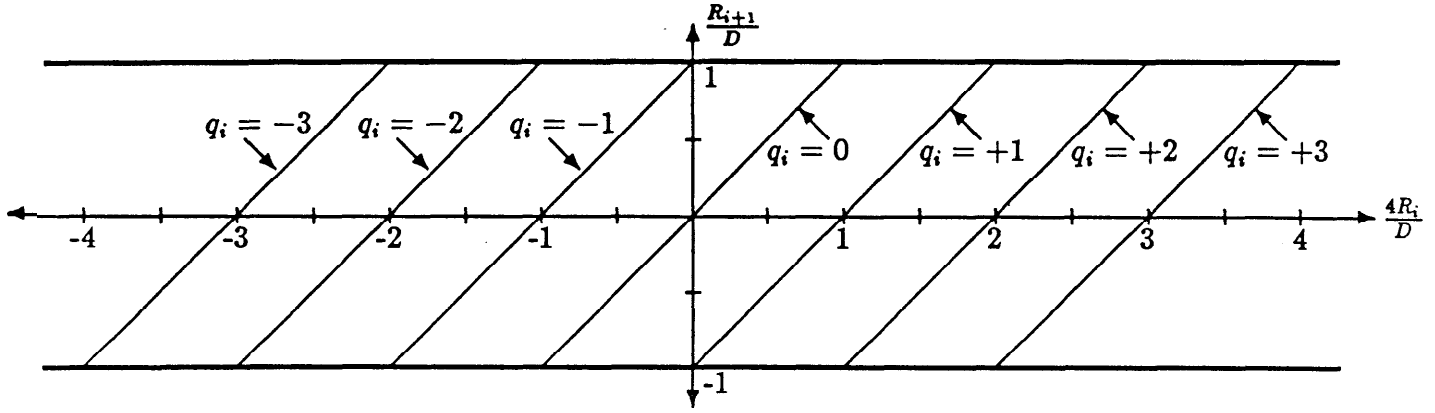


Figure 2a: Robertson Diagram for Radix 4 with quotient digit set {-3,-2,-1,0,1,2,3}

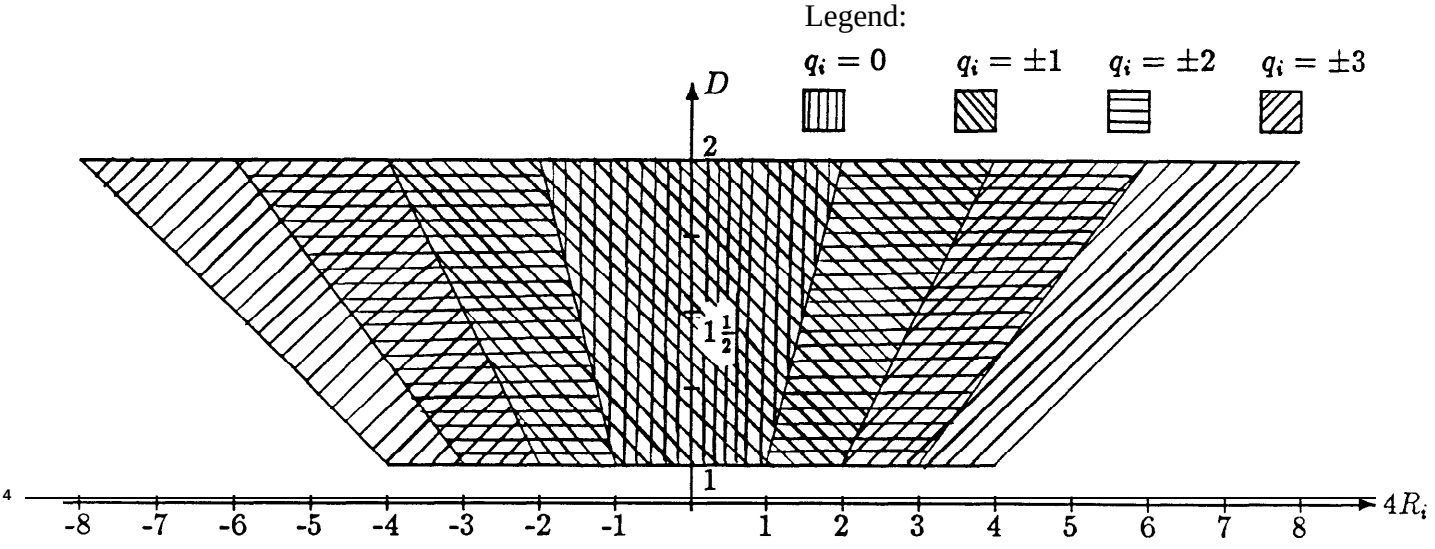


Figure 2b: Taylor Diagram for Radix 4 with quotient digit set {-3,-2,-1,0,1,2,3}

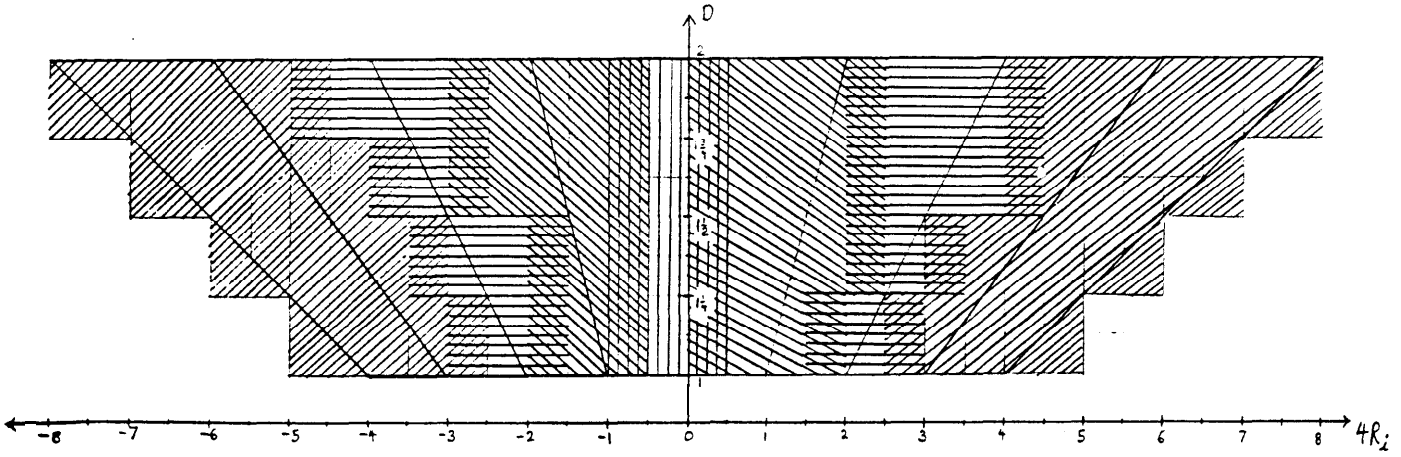


Figure 2c: Colored Taylor Diagram for Radix 4 with quotient digit set {-3,-2,-1,0,1,2,3}
Grid Size = $\left(\frac{1}{2} \times \frac{1}{4}\right)$, Brush Size = $1\left(\times \frac{1}{4}\right)$

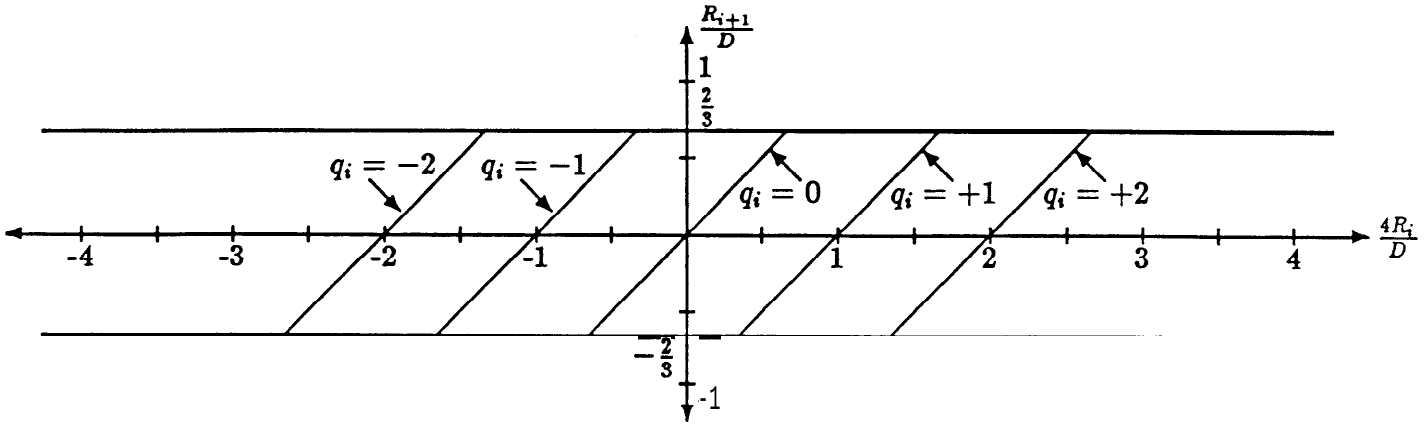


Figure 3a: Robertson Diagram for Radix 4 with quotient digit set {-2,-1,0,1,2}

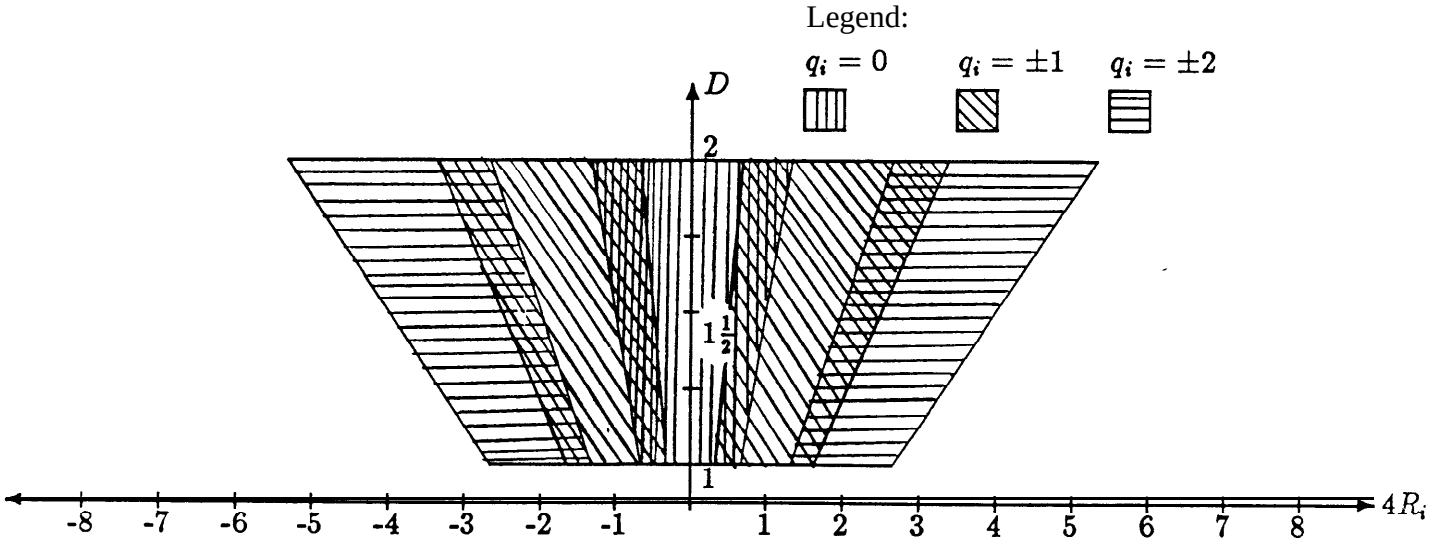


Figure 3b: Taylor Diagram for Radix 4 with quotient digit set {-2,-1,0,1,2}

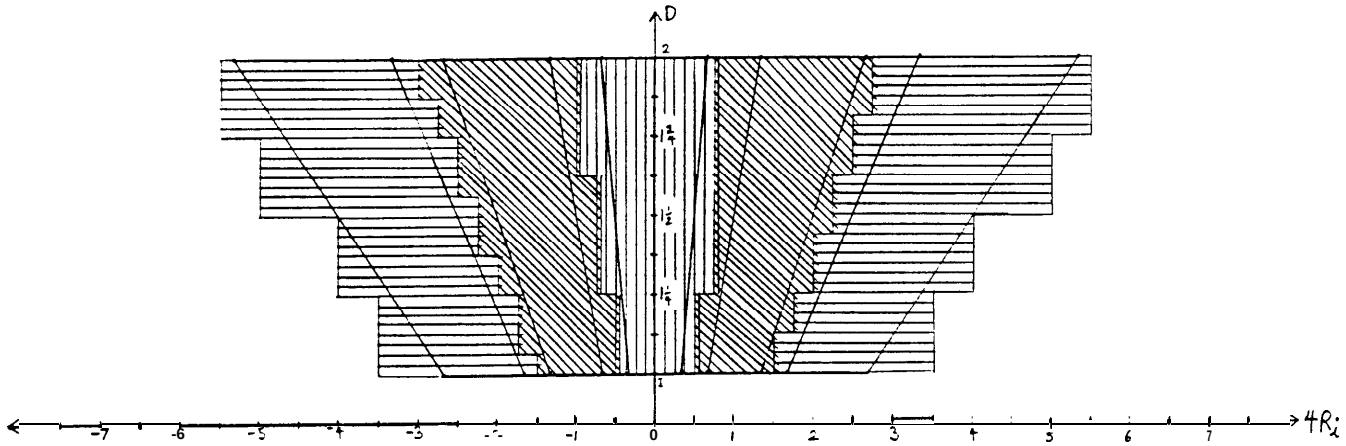


Figure 3c: Colored Taylor Diagram for Radix 4 with quotient digit set {-2,-1,0,1,2}
Grid Size = $\left(\frac{1}{4} \times \frac{1}{16}\right)$, Brush Size = $\left(\frac{5}{16} \times \frac{1}{16}\right)$

Since at radix 8 and radix 16 there is even more choice in coloring the Taylor diagrams, it was useful to write a computer program to output tables of appropriate colorings. The program verifies a coloring by checking that for its grid and brush size, there is at least one valid color for the brush when it is aligned at every grid point. The program determines the horizontal brush size required for a range of vertical brush sizes, and attempts to compose each horizontal brush size by using as large a grid as possible which is strictly smaller than brush. The grid is not allowed to be equal to the brush size, because this would require complete carry propagation in resolving the partial remainder approximation.

A table output by the program is included as table 1 which shows the matrix of possible colorings for the different radix and quotient digit set choices. The results of the table are seen to agree with the examples already discussed. For $r = 4$ and $p = 2$, one can see the two valid choices which were illustrated in figures 2 and 3. For example, figure 3 corresponds to the case where DivisorBits= 4, RBits= 6, CBits= 8, LeftBits= 4, grid height= $\frac{1}{16}$, brush height= $\frac{1}{16}$, grid width= $\frac{1}{4}$, and brush width= $\frac{1}{4} + \frac{1}{16} = \frac{5}{16}$.

5 Modified SRT algorithms

Having developed the tools to analyze SRT division, it is easy to evaluate the feasibility and possible advantages of several modifications to the basic SRT division scheme. One idea is to try quotient digit sets which affect more than m bits of the final quotient, when the partial remainder is shifted between stages by the radix r = 2^m. This relaxes the upper restriction in equation (3) and also allows non-integral quotient digits. Because the individual quotient digits are resolved into the final quotient in a path independent of the critical path of the iteration, such a modification requiring the addition of bits from different quotient digits to form the final quotient would not slow the speed of the main iterations. Examples of this idea would be to use the quotient digit set (-1, $-\frac{1}{2}$, 0, $\frac{1}{2}$, 1} with radix 2, or the sets (-4, $-2, -1, 0, 1, 2, 4\}$ or{-4,-3,-2,-1,0,1,2,3,4} with radix 4. However, it turns out that the half-integers for radix 2 do not lower the precision required in examining the partial remainder because it is already so low, and the extra digits when $p \geq r$ in higher radices have the disadvantage that they increase the number of bits required to store the partial remainder since its allowable magnitude is increased. So although such sets are feasible, and can generate correct resulting colorings, it turns out they have no performance advantage in quotient digit selection complexity over the more standard quotient digit sets consisting of successive integers, and have the disadvantage of increased complexity in final quotient bit resolution.

Another idea we explored is to use borrow-save subtractors rather than carry-save adders as the basic arithmetic element in the columns. Graphically, this corresponds to coloring the Taylor diagram with a brush aligned to the grid by a point other than its lower left corner. The brush would extend to the right from its alignment point an amount determined by the unexamined bits of the sum component of the partial remainder, and to the left from its alignment point by an amount given by the maximum value of the unexamined or unpropagated borrow bits of the partial remainder. Such a change in coloring does affect the size of the brush required to color the Taylor diagrams, and the results of a few cases are presented in table 2. In most cases, there is no advantage obtained over just using the standard two's complement carry-save adders, but there are some attractive possibilities where borrow-save does perform better. For example, for radix 8, with $p = 6$ and using 3 divisor bits, using borrow-save allows the partial remainder to be approximated

Radix	Max digit	DivisorBits	
		0	
r	*p*	RBits	CBits
2	1	3	3

Radix	Max digit	DivisorBits							
		1		2		3		4	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits
4	2					7	7	6	8
	3			5	5	5	5	5	-
	4	7	7	5	7	5	6	5	6

Radix	Max digit	DivisorBits									
		3		4		5		6		7	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits
8	4					10	12	9	10	8	14
	5			8	9	7	9	7	8	7	8
	6	9	11	6	10	6	8	6	8	6	8
	7	7	7	6	7	6	7	6	6	6	6
	8	7	9	6	9	6	9	6	9	6	8

Radix	Max digit	DivisorBits					
		4		5		6	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits
16	8						
	9					10	12
	10			11	11	9	10
	11			9	11	8	10
	12			8	11	8	9
	13			8	10	7	10
	14	11	12	7	12	7	9
	15	9	9	7	9	7	8
	16	9	11	8	9	7	14

Table 1: Grid and Brush sizes determined by Taylor diagrams

Radix	Max digit	DivisorBits	
		0	
r	*p*	RBits	CBits
2	1	3	3

Radix	Max digit	DivisorBits							
		1		2		3		4	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits
4	2					7	7	7	7
	3			5	5	5	5	4	7
	4	7	7	5	7	5	6	5	6

Radix	Max digit	DivisorBits									
		3		4		5		6		7	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits
8	4					10	11	10	9	10	9
	5			7	10	7	9	7	8	7	8
	6	8	10	7	8	7	7	7	7	7	7
	7	7	7	6	7	6	7	5	9	5	9
	8	7	10	7	7	7	7	7	7	7	7

Table 2: Grid and Brush sizes determined **by Taylor diagrams for borrow-save operations**

with one fewer bit, and one less bit of carry to be propagated.

One way of making the quotient selection easier is to restrict the range of the divisor using pre-normalization. Previously published work has examined normalization extensively as a means of doing division[2], but it is also possible to just perform a very simple pre-normalization before using SRT division for the main algorithm. For example, if the divisor is less than $\frac{3}{2}$, then both the divisor and dividend can be multiplied by $\frac{5}{4}$ with a single shift and add, and then the divisor will always be in the range [$\frac{5}{4}$, 2). Further, the dividend multiplication can be performed for free by initializing both the sum and carry portions of the first partial remainder adder, using the sum input for the original dividend and the carry input for the shifted dividend. Normalizing the divisor corresponds graphically to moving up the lower horizontal bound of the Taylor diagram, and since the tightest cases for coloring always occur at the bottom, this will allow a bigger brush size to be used and simplify the quotient selection logic. The results for a divisor restricted to the range [$\frac{5}{4}$, 2) are shown in table 3. The results yield some interesting feasible possibilities for division schemes. For example, by using 4 bits of the divisor, and 6 of the partial remainder into which 8 carry bits have been propagated, radix 8 division may be performed with the quotient digit set (-6, ... 6). These are the same number of bits required for the quotient selection logic of radix 4 division with quotient set (-2, ... 2). So, with the addition of a single adder to form the extra divisor **multiples, direct radix 8 division,** achieving a $\frac{3}{2}$ factor speed improvement, may be performed for approximately the same quotient selection logic complexity as radix 4 division.

Radix	Max digit	DivisorBits							
		1		2		3		4	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	**CBits**
4	*2*			*8*	*8*	*6*	*8*	*6*	*7*
	3	*6*	*6*	*5*	*5*	*4*	*6*	**4**	*6*
	4	*5*	*7*	*5*	*5*	*5*	*5*	*4*	*8*

Radix	Max digit	DivisorBits									
		3		4		5		6		7	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits	RBits	CBits	RBits	**CBits**
8	**4**					**9**	**11**	*8*	**11**	*8*	**10**
	5			**7**	**9**	**7**	**8**	**6**	**12**	**6**	**10**
	6	**7**	**9**	**6**	**8**	*6*	*7*	*6*	*7*	*6*	*7*
	7	*6*	*7*	*5*	*8*	*5*	*8*	*5*	*8*	*5*	*7*
	8	*7*	*8*	*6*	*8*	*6*	*8*	*6*	*7*	*6*	*7*

Radix	Max digit	DivisorBits					
		4		5		6	
r	*p*	RBits	CBits	RBits	CBits	RBits	CBits
16	8						
	9					10	10
	10			9	11	8	11
	11			8	10	8	9
	12	11	11	8	9	7	10
	13	8	11	7	10	7	8
	14	8	10	7	8	7	8
	15	8	8	7	8	6	10
	16	8	11	7	11	7	9

Table 3: Grid and Brush sizes determined by Taylor diagrams for a Divisor pre-normalized to be in the range $[\frac{5}{4},2)$

6 Hardware Implementation

6.1 Implementation Options

The "best" choice of radix and quotient digit set depends strongly on the implementation method taken. There are several overall approaches to implementing SRT division in hardware. The inherent iteration can either be done in time by looping repeatedly through the same hardware elements, or in space by replicating the hardware elements to form an array of the division stages. The most common approach, and the one requiring the fewest transistors, uses only one instance of each hardware element (full carry-save adder, short carry-propagate adder, quotient selection logic) and clocks these elements iteratively to compute a new quotient digit each cycle. Much previous work using the iterative approach has been done using ECL gate arrays[8]. The time iterative approach requires that the clock have a period greater than the worst case propagation delay through the arithmetic elements in the critical path around the loop from a register's outputs back to its inputs, plus the setup time for the register and an additional allowance for clock skew. The worst case times must allow for both the worst case arithmetic operations and the worst case fabrication possibilities. The total division time will be fixed at the worst case single iteration cycle time multiplied by the number of quotient digits. Higher radix schemes reduce the number of iterations but will increase the cycle time because of increased complexity. But since the cycle time will probably not double to go from radix 2 to radix 4 or from radix **4** to radix 16, such moves do increase performance. Since the generation and distribution of high speed clocks is difficult, a higher radix scheme is preferable even for the same nominal performance, since it will have a slower clock. Another benefit of using a higher radix is that the complexity and silicon area of the quotient selection logic becomes more balanced with that of the carry-save adder in each array stage.

A second hardware implementation approach is to make a fully combinational array of the arithmetic elements. This approach has only recently become feasible due to the high density available in small geometry MOS fabrication technologies. An example of this approach is the nMOS division chip designed at Hewlett-Packard[4]. A fully combinational array of the partial remainder adders and quotient selection logic has the advantage that the internal logic will propagate at the fastest possible speed, since the propagation of results can proceed from one stage to the next without waiting for clock cycles to occur. However, only a small proportion of the transistors are active at any instant, and the area required for the whole array is certainly large. The HP implementation used a whole die of size 6.7x7.1mm to produce a 64 bit result, and it would be difficult to combine such a design with the other parts of a general floating-point chip while keeping within a reasonable die size. The fully combinational array implementation is also harder to extend to higher radix division schemes because the array must contain as many copies of the quotient selection logic as there are stages in the array. Since at a radix higher than 2, the quotient selection logic becomes large compared to the adders in each stage, the full array approach would expand greatly in area for higher radix division implementations.

An intermediate approach between the fully iterative and fully combinational approach is one which has a small array of several copies of the arithmetic elements required for each stage, with the results looped around so that the array iterates several times in order to calculate the quotient to the desired precision. If the array contains enough stages then there need not be any registers in the loop since the outputs of the last stage can be valid long enough to feed back around to the first stage in the array because of the finite propagation time through the loop. If the loop is self-timed

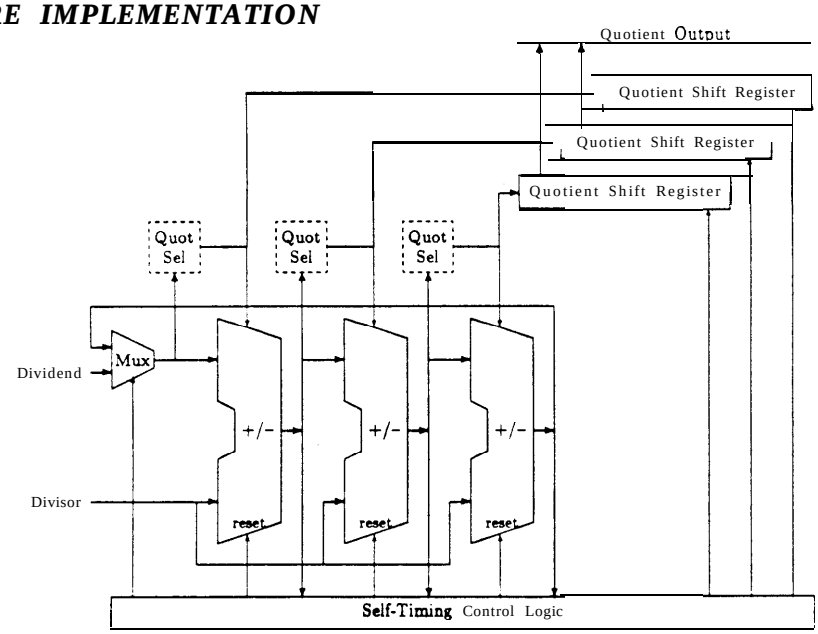


Figure 4: Block diagram of Self-Timed Radix 2 division chip

and iterates without synchronization to external clocks, a useful way to think of this approach is that the division stages form a ring oscillator which has a side effect of spinning off quotient bits as each stage evaluates. Thus, this approach has the speed advantage of the fully combinational array, since it contains no clocked registers, but the silicon area and yield economies of the iterative approach, since only a few copies of the elements for a stage are necessary.

6.2 Our Implementation

We have chosen to implement a radix 2 division chip using this approach of iterating around a small combinational array[10]. We chose the radix 2 scheme because we wanted to demonstrate the feasibility of self-timing the array when the arithmetic operations were the primary component in the critical path rather than the quotient selection logic. To control the propagation times around the array, each stage within the array uses precharged function block logic where the precharge control signals are derived from the logic of the array itself. The result is a self-timed array because the signals which control the propagation, and hence speed, of the array are derived internally.

The main array of the division circuit consists of a series of columns connected cyclically. A block diagram of the structure is shown in figure 4. Each column computes one stage of the division by performing a computation of the next partial remainder, and performing the quotient selection logic for the next stage. The columns operate sequentially as the wavefront of evaluation progresses around the columns. The evaluation will continue in the cyclic pattern until stopped by a completion signal coming from outside of the main array. This completion signal is generated by the on-chip shift registers collecting the quotient digits as they are produced, and the completion signal is also an output of the chip to signal when the final quotient is valid.

Each column of the main array contains a carry-save adder to compute the next partial remainder and quotient selection logic to select the quotient digit determined by that remainder. This quotient digit multiplexes the appropriate multiple of the divisor for the carry-save adder of

the next column. Because a carry-save adder is used to compute the full partial remainder, the carrys need not be propagated up the whole width of the array and hence the speed of the circuit is independent of the word width. The critical path of the circuit lies only in the quotient selection logic and the most significant bits of the adder.

The carry-save addition produces a partial remainder from which the top three bits are used to determine the next quotient digit. A 3 bit carry-propagate addition is performed to combine the sum and carry components of these bits produced by the carry-save adder. This option makes the quotient selection logic very simple, since then the partial remainder bits input to the quotient select logic are irredundant. Another option would have been to use the sum and carry bits of the redundant remainder in carry-save form as inputs, and to not use a carry-propagate adder at all, but this would make even the radix 2 quotient selection logic complex enough to require a PLA with 23 product terms.

Since the quotient selection logic cannot evaluate until the outputs of the carry-propagate adder are valid, it was also chosen to use the results of the carry-propagate adder rather than the carry-save adder as the input for the top bits of the partial remainder input to the next stage. This decision reduced the number of wires that need to be shipped from one stage to the next for the top bits. Due to the possibility that $rR_i < -\frac{2rp}{r-1}$ can occur by an amount equal to the maximum value of the unpropagated carry, an additional bit to the left of the binary point might have otherwise been necessary, but an additional advantage of using the results of the carry-propagate adder as the remainder input to the next stage is that the maximum value of the unpropagated carry is reduced to a value where this special case can be detected. When $rR_i < -\frac{2rp}{r-1}$ is detected, the quotient selection logic of the current stage and the next stage can both be forced to select the most negative quotient digit so that the correct result is always produced.

Because the output of the carry-propagate adder is used as the output for the partial remainder computation as well as the input to the quotient selection logic in each stage, the carry-propagate adder has been implemented in the layout by interleaving the carry-save logic blocks with the carry-propagate adder. The latter comprises blocks to provide propagate, generate, and kill signals to a carry chain. The carry chain is a dual-rail Manchester precharged design, and snakes up through the top three bit slices to produce the correct sum bits of the partial remainder by gating out the appropriate rail in each bit slice.

Since the design is self-timed, the speed at which it performs a given division will be dependent on the values of the operands. The variance would even be greater with a higher radix implementation using this self-timed method. The self-timing provides a performance increase over a clocked methodology for most numbers, but for a system to truly take advantage of the self-timing, it must sense and synchronize on the completion signal output from the chip.

A test chip containing 5430 transistors and measuring 3.39x4.59mm (active area = 2.18x3.46mm) was fabricated on a MOSIS 3μ CMOS run to test the top bits, quotient select logic and the self-timing. A full version of the array containing 14,000 transistors to compute quotients for a floating-point double precision operand length of 48 bits has been fabricated in MOSIS 2μ CMOS technology. Since the active portion fits in an area of 6.0x1.6mm, it is suitable to be used as the division portion of a complete floating-point chip. Testing has verified that the chips function with a measured average speed of 13nS per quotient bit, allowing the computation of a 48 bit quotient in 625nS.

7 Conclusions

We have shown how the drawing of a series of two-dimensional diagrams illustrates the arithmetic constraints required to implement SRT division. The diagrams allow comparison of different radix and quotient digit sets, and illustrate the tradeoffs in determining the number of bits to use in approximating the divisor and partial remainder. In modern VLSI implementations, these tradeoffs directly affect the time and space required since custom designs use only the required number of bits. We examined several modifications to the typical SRT division algorithms, and found, in particular, that simple pre-normalization will allow radix 8 division to be performed with approximately the same quotient selection complexity as radix 4 division, and also that some of the higher radix cases can be implemented with fewer bits in the quotient selection logic by using a borrow-save rather than a carry-save format for the remainder arithmetic.

We have successfully designed, fabricated and tested a CMOS chip implementing a self-timed algorithm for performing SRT division on normalized floating-point mantissas. The intermediate implementation strategy between a fully iterative and fully combinational approach allows the design to have nearly the speed of a full combinational array, but with a reduced area. The self-timing of the array frees the chip of requiring carefully distributed high speed clocks, and allows it to run as fast as the operand values, technology, and temperature allow.

References

[1] D.E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders," ***IEEE Transactions on Computers,*** vol. C-17, pp. 925-934, October 1968.

[2] 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, May 1983.

[3] K. Hwang, ***Computer Arithmetic: Principles, Architecture, and Design,*** John Wiley & Sons, 1979.

[4] W.M. McAllister and D. Zuras, Hewlett-Packard, "An nMOS 64b Floating-Point Chip Set," IEEE International Solid-State Circuits Conference, February 1986.

[5] J.E. Robertson, "A New Class of Digital Division Methods," ***IRE Tmns. Electronic Computers,*** vol. EC-7, pp. 218-222, September 1958.

[6] A.L. Sangiovanni-Vincentelli, R.K. Brayton, et. al., Logic Minimization ***Algorithms for VLSI Synthesis,*** Kluwer Academic, 1984.

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

[8] G.S. Taylor, "Radix 16 SRT Dividers With Overlapped Quotient Selection Stages," ***Proceedings of the Seventh IEEE Symposium on Computer Arithmetic,*** pp. 64-71, May 1985.

[9] T.D. Tochner, "Techniques of Multiplication and Division for Automatic Binary Computers," ***Quarter J. Mech App. Math.,*** vol. 2, pt. 3, pp. 364-384, 1958.

[10] T.E. Williams, M. Horowitz, et. al., "A Self-Timed Chip for Division," ***Advanced*** *Research* in ***VLSI, Proceedings of the 1987 Stanford Conference,*** pp. 75-95, March 1987.