MEASURING THE COMPLEXITY OF SRT TABLES

Stuart F. Oberman and Michael J. Flynn

Technical Report: CSL-TR-95-679

November 1995

This work was supported by NSF under contract MIP93-13701.

MEASURING THE COMPLEXITY OF SRT TABLES

by

Stuart F. Oberman and Michael J. Flynn

Technical Report: CSL-TR-95-679

November 1995

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305-4055
pubs@shasta.stanford.edu

Abstract

This paper presents an analysis of the complexity of quotient-digit selection tables in SRT division implementations. SRT dividers use a fixed number of partial remainder and divisor bits to consult a table to select the next quotient-digit in each iteration. The complexity of these tables is a function of the radix, the redundancy, and the number of bits in the estimates of the divisor and partial remainder. This analysis derives the allowable divisor and partial remainder truncations for radix 2 through radix 32, and it quantifies the relationship between table parameters and the number of product terms in the logic equations defining the tables. By mapping the tables to a library of standard-cells, delay and area values were measured and are presented for table configurations through radix 32. The results show that: 1) Gray-coding of the quotient-digits allows for the automatic minimization of the quotient-digit selection logic equations. 2) Using a short carry-assimilating adder with a few more input bits than output bits can reduce table complexity. 3) Reducing the number of bits in the partial remainder estimate and increasing the length of the divisor estimate increases the size and delay of the table, offsetting any performance gain due to the shorter external adder. 4) While delay increases nearly linearly with radix, area increases quadratically, limiting practical table implementations to radix 2 and radix 4.

Key Words and Phrases: Computer arithmetic, floating point, performance tradeoffs, quotient-digit selection, SRT division, table complexity

Contents

List of Figures

List of Tables

1 Introduction

In recent years computer applications have increased in their computational complexity. High speed floating-point hardware is a requirement to meet these increasing demands. An important component of the floating point unit is the divider. There are many methods for designing division hardware, including quadratically converging algorithms, such as Newton-Raphson, and linear converging algorithms, the most common of which is SRT [14]. SRT division computes a quotient one digit at a time, with an iteration time independent of the operand length.

The theory of SRT division is discussed thoroughly in Atkins [1], Ercegovac [7], Robertson [16], and Tan [19]. Several SRT implementations have been reported, including radix 2 dividers by Knowles [10], Kuninobu [11], Vandemeulebroecke [21], and Zuras [22], radix 4 by Birman [3] and Fandrianto [8], radix 8 by Fandrianto [9] and Prabhu [15], and radix 16 by Carter [5] and Taylor [20]. SRT dividers with simplified quotient-digit selection using operand range restriction have been presented in Ercegovac [6], Montuschi [13], and Srinivas [17].

There are many performance and area tradeoffs when designing an SRT divider. One metric for comparison of different designs is the minimum required truncations of the divisor and partial remainder for quotient-digit selection. Atkins [1] and Robertson [16] provide such analyses of the divisor and partial remainder precisions required for quotient-digit selection. Burgess and Williams [4] present in more detail allowable truncations for divisors and both carry-save and borrow-save partial remainders. However, a more detailed comparison of quotient-digit selection complexity between different designs requires more information than input precision. This paper analyzes in detail the effects of algorithm radix, redundancy, divisor and partial remainder precision, and truncation error on the complexity of the resulting table. Complexity is measured by the number of product terms in the final logic equations, and the delay and area of standard-cell implementations of the tables. These metrics are obtained by an automated design flow using the specifications for the quotient-digit selection table as input, a Gray-coded PLA as an intermediate representation, and an LSI Logic 500K standard-cell implementation as the output. This paper also examines the effects of additional techniques such as table-folding and longer external carry-assimilating adders on table complexity. Using the methodology presented, it is possible to automatically generate optimized high radix quotient-digit selection tables.

The remainder of this paper is organized as follows: Section 2 presents the theory of SRT division. Section 3 discusses the implementation of SRT tables. Section 4 presents our methodology for implementing the quotient-digit selection tables. Section 5 presents the results. Section 6 is the conclusion.

2 Theory of SRT Division

SRT division belongs to the digit recurrence class of division algorithms. Digit recurrence algorithms use subtractive methods to calculate quotients one digit per iteration. Digit recurrence algorithms can be divided into *restoring* and *nonrestoring* division. Restoring

division is similar to the familiar paper and pencil division. When dividing two n-digit numbers, the division can require up to $2n$ additions or subtractions. Nonrestoring division algorithms eliminate the restoration cycles, and thus only require up to n additions. This can be accomplished by allowing negative values of the quotient as well as positive values. In this way, small errors in one iteration can be corrected in subsequent iterations.

2.1 Recurrence

In SRT division, the quotient can be computed as follows:

$$q = \frac{dividend}{divisor}$$

This expression can be rewritten as:

$$dividend = q \times divisor + remainder$$

such that

$$|remainder| < |divisor| \times ulp \quad \text{and} \quad sign(remainder) = sign(dividend)$$

where the input operands are given by *dividend* and *divisor*, and the results are q and *remainder*. The precision of the quotient is defined by the unit in the last position (ulp), where for an integer quotient ulp = 1, and for a fractional quotient using a binary representation ulp = 2^{-n}, assuming an n digit quotient. The radix r of the algorithm, typically chosen to be a power of 2, determines how many quotient bits b are retired in each iteration, such that $r = 2^b$. Accordingly, a radix r algorithm requires $\lceil n/b \rceil$ iterations to compute an n digit quotient.

The following recurrence is used at every iteration:

$$rP_0 = dividend \tag{1}$$

$$P_{j+1} = rP_j - q_{j+1} divisor \tag{2}$$

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:

$$q_{j+1} = SEL(rP_j, divisor) \tag{3}$$

In order for the next partial remainder P_{j+1} to be bounded, the value of the quotient-digit is chosen such that

$$|P_{j+1}| < divisor \tag{4}$$

The final quotient is the weighted sum of all of the quotient-digits selected throughout the iterations, such that:

$$Q_{final} = \sum_{j=1}^{n/b} q_j \times r^{-j} \tag{5}$$

As can be noted from equations 1 and 2, each iteration of the recurrence comprises the following steps:

- Determine next quotient-digit q_{j+1} by the quotient-digit selection function
- Generate the product $q_{j+1} \times divisor$
- Subtract $q_{j+1} \times divisor$ from $r \times P_j$ to form the next partial remainder

Each of these components can contribute to the overall cost and performance of the algorithm. To reduce the time for partial remainder computation, intermediate partial remainders are often stored in a redundant representation, either carry-save or signed digit form. Then, the partial remainder computation requires only a full adder delay, rather than a full width carry-propagate addition. The rest of this paper is concerned with the quotient-digit selection component.

2.2 Choice of Radix

The fundamental method of decreasing the overall latency (in machine cycles) of the SRT algorithm is to increase the radix r of the algorithm. By choosing the radix to be a power of 2, the product of the radix and the partial remainder can be formed by shifting. Accordingly, throughout this study, only power of 2 radices are considered. Assuming the same quotient precision, the number of iterations of the algorithm required to compute the quotient is reduced by a factor k when the radix is increased from r to r^k. For example, a radix 4 algorithm retires 2 bits of quotient in every iteration. Increasing to a radix 16 algorithm will allow for retiring 4 bits in every iteration, for a 2X reduction in latency. This reduction does not come for free. As the radix increases, the quotient-digit selection becomes more complex. Since the quotient-digit selection is typically on the critical path of the algorithm, even though the number of cycles may have been reduced due to the increased radix, the time per cycle may have increased. As a result, the total time required to compute an n bit quotient may not be reduced by the factor k. Accordingly, the radix r is a fundamental parameter in determining the complexity of the quotient-digit selection table.

2.3 Choice of Quotient Digit Set

A 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, it is often desirable to utilize a *redundant digit set* which simplifies the quotient-digit selection table, thereby increasing the performance of the divider. Such a digit set can be composed of symmetric signed-digit consecutive integers, where the maximum digit is a. In particular,

$$q_j \in \mathcal{D}_a = \{-a, -a+1, \ldots, -1, 0, 1, \ldots, a-1, a\}$$

The redundancy of a digit set is determined by the value of the redundancy factor ρ, which is defined as

$$\rho = \frac{a}{r-1}, \quad \rho > \frac{1}{2} \tag{6}$$

For all partial remainders to be bounded when a redundant quotient digit set is used, the value of the quotient-digit must be chosen such that

$$|P_{j+1}| \quad < \quad \rho \times divisor \tag{7}$$

The calculation of the final quotient using a redundant quotient-digit set involves either a full carry propagate addition to subtract the negative quotient-digits from the positive quotient-digits at the completion of the iterations, or the use of on-the-fly quotient conversion techniques [7].

After the redundancy factor ρ is chosen, it is possible to derive the quotient-digit selection function. To guarantee that the shifted partial remainder remains bounded for all valid quotient-digits and divisor, expressions for the quotient-digit selection intervals must be computed. A selection interval is the region in which a particular quotient-digit can be safely chosen such that the shifted partial remainder will remain bounded. The expressions for the selection intervals are given by

$$U_k = (\rho + k)d \qquad L_k = (-\rho + k)d$$

where U_k (L_k) is the largest (smallest) value of rP_j such that it is possible for $q_{j+1} = k$ to be chosen and still keep the next shifted partial remainder bounded. The *continuity condition* requires that for all valid values of rP_j, it must be possible to select at least one quotient digit [7]. This is expressed mathematically as

$$U_{k-1} \quad \geq \quad L_k - r^{-n} \tag{8}$$

Because rP_j is represented by a maximum of n bits, the term r^{-n} is the resolution of the partial remainder.

The *P-D diagram* is a useful visual tool when designing a quotient-digit selection function. It has as axes the shifted partial remainder and the divisor. The selection interval bounds U_k and L_k are drawn as lines starting at the origin with slope $\rho + k$ and $-\rho + k$, respectively. A P-D diagram is shown in figure 1 with $r = 4$ and $a = 2$. The shaded regions are the overlap regions where more than one quotient-digit may be selected.

3 Implementing SRT Tables

3.1 Divisor and Partial Remainder Estimates

To reduce the size and complexity of the quotient-digit selection table for a given choice of r and a, it is desirable to use as input to the table estimates of the divisor and shifted partial remainder which have fewer bits than the true values. Assuming IEEE floating-point compliance, the input operands are in the range $1 \leq D < 2$. Thus, a leading integer one can be assumed for all divisors, and the table only requires fractional divisor bits to make a quotient-digit selection. The shifted partial remainder, though, requires both integer and fractional bits as inputs to the table. The shifted partial remainder rP_j and divisor d can be approximated by estimates $r\hat{P}_j$ and $\hat{d}$ using the c most significant bits of rP_j and the

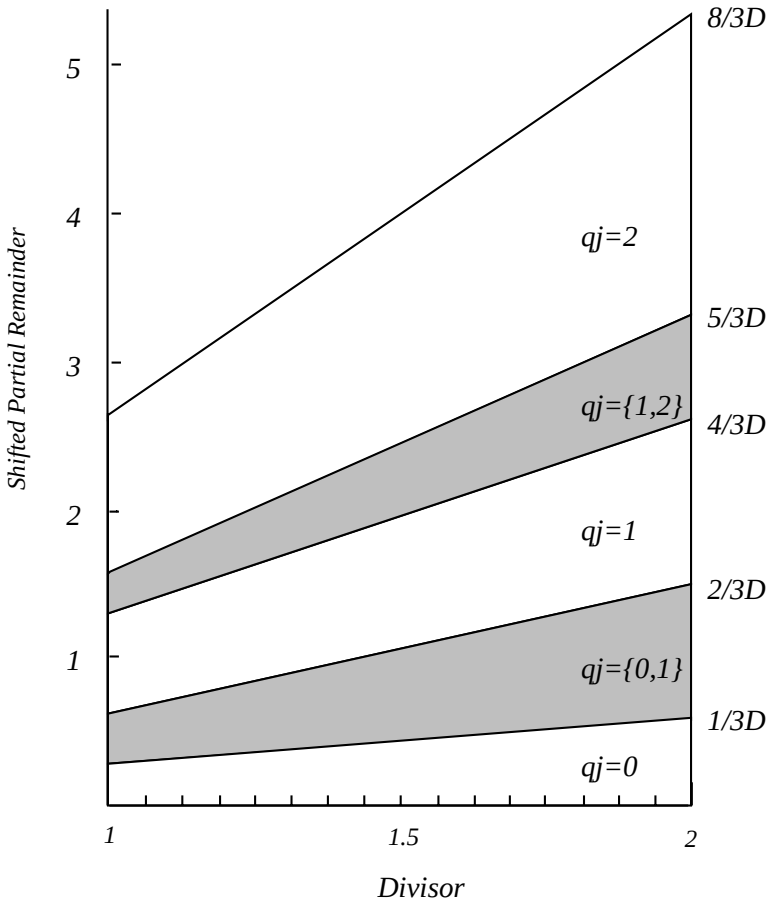

Figure 1: P-D diagram for radix 4

δ most significant bits of d. The c bits in the truncated estimate $r\hat{P}_j$ can be divided into i integer bits and f fractional bits, such that $c = i + f$. The table can take as input the partial remainder estimate directly in redundant form, or it can use the output of a short carry-assimilating adder that converts the redundant partial remainder estimate to a nonredundant representation. The use of an external short adder reduces the complexity of the table implementation, as the number of partial remainder input bits are halved. However, the delay of the quotient-digit selection function increases by the delay of the adder.

It is not possible to determine the optimal choices of δ and f analytically, as several factors are involved in making these choices. However, it is possible to determine a lower bound on δ using the continuity condition and the fact that the next partial remainder must remain bounded:

$$2^{-\delta} \leq \frac{2\rho - 1}{2(a - \rho)} \tag{9}$$

$$\delta \geq \left\lceil -\log_2 \frac{2\rho - 1}{2(a - \rho)} \right\rceil \tag{10}$$

Because the divisor is IEEE normalized with a leading one, only the leading $b = \delta - 1$ fractional bits are required as input to the table. The next quotient-digit can then be selected by using these estimates to index into a 2^{b+c} entry lookup table, implemented either as a PLA or random logic.

Assuming a nonredundant two's complement partial remainder, the estimates have nonnegative truncation errors ϵ_d and ϵ_p for the divisor and shifted partial remainder estimates respectively, where

$$\epsilon_d \leq 2^{-b} - 2^{-n} \approx 2^{-b} \tag{11}$$

$$\epsilon_p \leq 2^{-f} - 2^{-n} \approx 2^{-f} \tag{12}$$

Thus, the maximum truncation error for both the divisor and the nonredundant shifted partial remainder estimates is strictly less than 1 ulp.

For a redundant two's complement partial remainder, the truncation error depends upon the representation. For a carry-save representation, the sum and carry estimates each has nonnegative truncation error ϵ_p, assuming that both the sum and carry estimates are represented by the c most significant bits of their true values. The resulting estimate $r\hat{P}_{j(cs)}$ has truncation error

$$\epsilon_{p(cs)} \leq 2 \times (2^{-f} - 2^{-n}) \approx 2^{1-f} \tag{13}$$

Thus, the maximum truncation error for an estimate of a carry-save shifted partial remainder is strictly less than 2 ulps.

From this discussion, the number of integer bits i in $r\hat{P}_j$ can be determined analytically. Using the general recurrence for SRT division, the maximum shifted partial remainder is given by

$$rP_{j(max)} = r\rho d_{max} \tag{14}$$

For IEEE operands,

$$d_{max} = 2 - 2^{-n} \tag{15}$$

As previously stated, for a carry-save two's complement representation of the partial remainder, the truncation error is always nonnegative, and therefore the maximum estimate of the partial remainder is

$$r\hat{P}_{j(max)} = \left\lfloor r \times \rho \times (2 - 2^{-n}) \times 2^f \right\rfloor / 2^f \tag{16}$$

The minimum estimate of the partial remainder is

$$r\hat{P}_{j(min)} = \left\lceil -r \times \rho \times (2 - 2^{-n}) \times 2^f \right\rceil / 2^f - \epsilon_{p(cs)} \tag{17}$$

Accordingly, i can be determined from

$$r\hat{P}_{j(max)} - r\hat{P}_{j(min)} \leq 2^i \tag{18}$$

$$i \geq \left\lceil \log_2(r\hat{P}_{j(max)} - r\hat{P}_{j(min)}) \right\rceil \tag{19}$$

3.2 Uncertainty Regions

By using a redundant quotient-digit set, it is possible to correctly choose the next quotient-digit even when using the truncated estimates $r\hat{P}_j$ and $\hat{d}$. Due to the truncation error in the estimates, each entry in the quotient-digit selection table has an uncertainty region associated with it. For each entry, it is necessary for all combinations of all possible values represented by the estimates $r\hat{P}_j$ and $\hat{d}$ to lie in the same selection interval. For a carry-save representation of the shifted partial remainder, this involves calculating the maximum and minimum ratios of the shifted partial remainder and divisor, and ensuring that these ratios both lie in the same selection interval:

$$ratio_{max} = \begin{cases} \frac{r\hat{P}_j + \epsilon_{p(cs)}}{\hat{d}} & \text{if } P_j \geq 0 \\ \frac{r\hat{P}_j}{\hat{d}} & \text{if } P_j < 0 \end{cases} \tag{20}$$

$$ratio_{min} = \begin{cases} \frac{r\hat{P}_j}{\hat{d}+\epsilon_d} & \text{if } P_j \geq 0 \\ \frac{r\hat{P}_j + \epsilon_{p(cs)}}{\hat{d}+\epsilon_d} & \text{if } P_j < 0 \end{cases} \tag{21}$$

If an uncertainty region is too large, the maximum and minimum ratios may span more than one selection interval, requiring one table entry to return more than one quotient-digit. This would signify that the estimate of the divisor and/or the shifted partial remainder has too much truncation error. Figure 2 shows several uncertainty regions in a radix 4 P-D plot. Each uncertainty region is represented by a rectangle whose height and width is a function of the divisor and partial remainder truncation errors. The value of $ratio_{max}$ corresponds to the upper left corner of the rectangle, while $ratio_{min}$ corresponds to the lower right corner. In this figure, the four valid uncertainty regions include a portion of

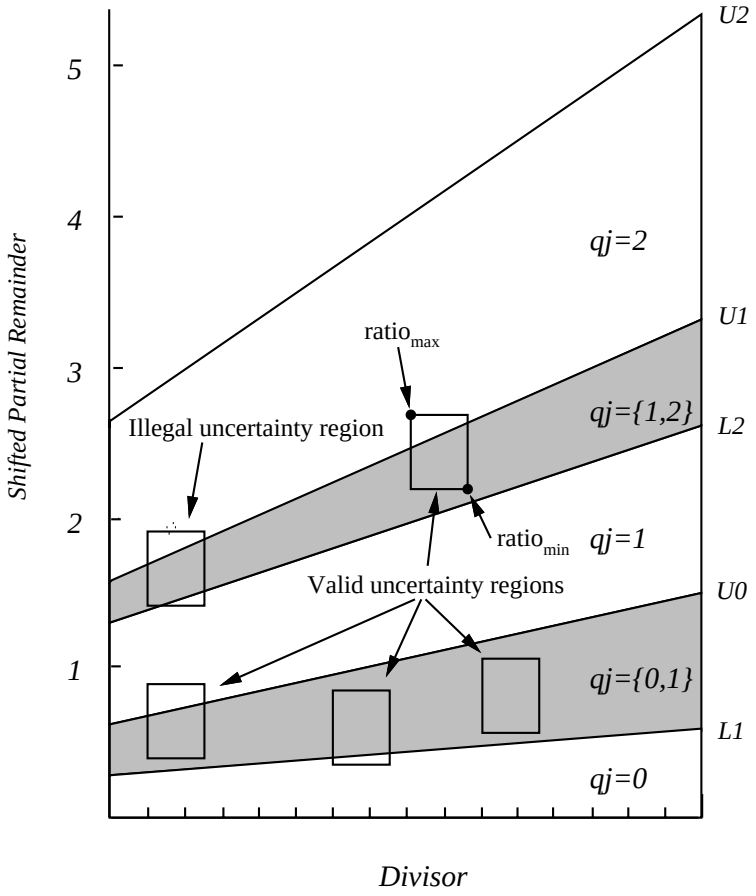

Figure 2: Uncertainty regions due to divisor and partial remainder estimates

an overlap region. Further, the lower right uncertainty region is fully contained within an overlap region, allowing the entry corresponding to that uncertainty region to take on the quotient-digits of either 0 or 1. The other three valid uncertainty regions may take on only a single quotient-digit. The upper left uncertainty region spans more than an entire overlap region, signifying that the corresponding table entry, and as a result the entire table, is not valid. To determine the valid values of b and f for a given r and a, it is necessary to calculate the uncertainty regions for all 2^{b+i+f} entries in the table. If all uncertainty regions are valid for given choices of b, i, and f, then they are valid choices.

3.3 Reducing Table Complexity

The size of the table implementation can be reduced nearly in half by folding the table entries as suggested in Fandrianto [8]. Folding involves the conversion of the two's complement representation of $r\hat{P}_j$ to signed-magnitude, allowing the same table entries to be used for both positive and negative values of $r\hat{P}_j$. This reduction does not come for free. First, it requires additional logic outside of the table, such as a row of XOR gates, to perform the representation conversion, adding external delay to the quotient digit selection process. Second, it may place further restrictions on the table design process. When a carry-save representation is used for rP_j and a truncated estimate $r\hat{P}_j$ is used to consult the table, the truncation error is always nonnegative, resulting in an asymmetrical table. To guarantee the symmetry required for folding, additional terms must be added to the table, resulting in a less than optimal implementation.

A complexity-reducing technique proposed in this study is to minimize ϵ_p. As presented previously, when using an external carry-assimilating adder for a truncated two's complement carry-save partial remainder estimate, the maximum error $\epsilon_{p(cs)}$ is approximately 2^{1-f}. This error can be further reduced by using g fractional bits of redundant partial remainder as input to the external adder, where $g > f$, but only using the most significant f fractional output bits of the adder as input to the table. The maximum error in the output of the adder is

$$\epsilon_{p(adder)} = 2^{-g} + 2^{-g} - 2^{1-n} \tag{22}$$

Then, by using f bits of the adder output, the maximum error for the input to the table is

$$\begin{aligned} \epsilon_{p(cs)} &= 2^{-f} - 2^{-g} + \epsilon_{p(adder)} \\ &= 2^{-f} + 2^{-g} - 2^{1-n} \approx 2^{-f} + 2^{-g} \end{aligned} \tag{23}$$

For the case $g = f$, the error remains approximately 2^{1-f}. However, by increasing g, the error $\epsilon_{p(cs)}$ is reduced, converging towards the error for a nonredundant partial remainder which is approximately 2^{-f}. Reducing the truncation error $\epsilon_{p(cs)}$ decreases the height of the uncertainty region in the PD diagram. This has the effect of allowing more of the entries' uncertainty regions to fully fit within overlap regions, increasing the flexibility in the logic minimization process, and ultimately reducing the complexity of the final table. A block diagram illustrating the various components of an SRT divider is shown in figure 3.

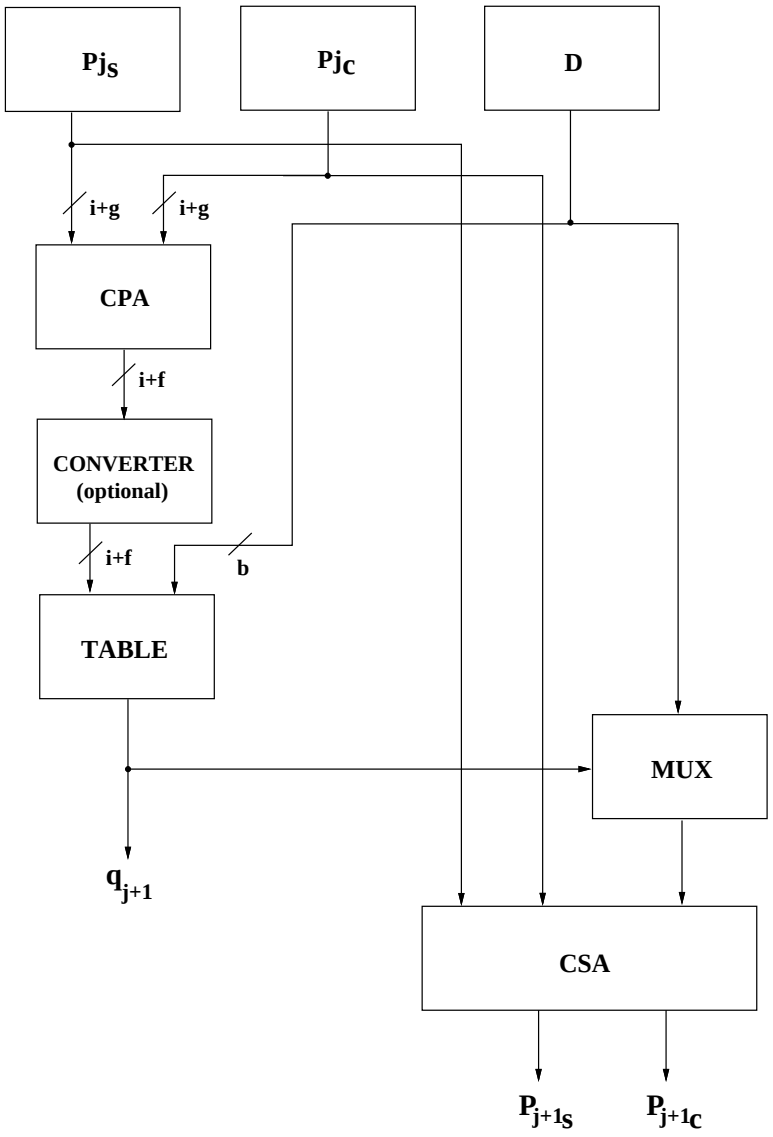

Figure 3: Components of an SRT divider

4 Experimental Methodology

The focus of this paper is to quantitatively measure the tradeoffs between the parameters r, a, b, i, f, g, and the complexity of the logic equations, measured by the number of product terms, as well as the complexity of the resulting random logic implementations of the tables, as measured by implementation delay and area. The following design flow was used to automatically generate quotient-digit selection tables in random logic:

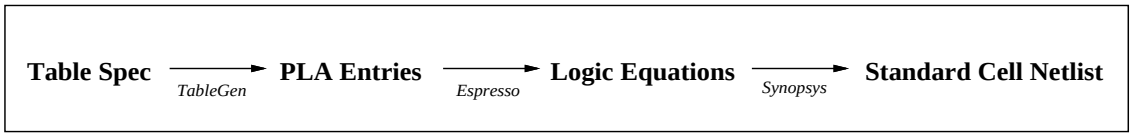

Figure 4: Design flow

In this study, a carry-save two's complement representation is used for the partial remainder in all tables.

4.1 TableGen

The program *TableGen* performs the analytical aspects of the quotient-digit selection table design. This program takes the table parameters as input, and it produces the unminimized PLA entries necessary to implement the table. First, all of the uncertainty regions for all entries in the table are computed. *TableGen* determines whether or not the choice of input parameters results in a valid table design. If the table is valid, it then computes the allowable quotient-digits for each entry in the table, based upon the size of the uncertainty region. The allowable quotient-digits are then written in PLA form for all 2^{i+b+f} possible shifted partial remainder and divisor pairs.

To allow for the greatest reduction in the complexity of the table implementations, it is proposed in this study to use a Gray code to encode an entry's allowable quotient-digits. In a Gray encoding, neighboring values only differ in one bit position [2]. This allows for the efficient representation of multiple allowable quotient-digits in the PLA output, while still only requiring $\lceil \log_2 r \rceil$ bits. The Gray-coding of the digits is recommended to ensure that given a choice between two allowable quotient-digits in an overlap region, the optimal choice can be automatically determined that will result in the least complex logic equations.

Accordingly, there are $\lceil \log_2 r \rceil$ outputs of the table which are the bits of the encoded quotient-digit. Table entries where $ratio_{min}$ and $ratio_{max}$ are both greater than $\rho \times r$ are unreachable entries. Thus, their outputs are set to *don't care*. An example of Gray-coding for $r = 4$ and $a = 3$ is shown in table 1. In this example, a value of x implies a *don't care*. Because the table stores digits in encoded form, all tables in this study require an explicit decoder to recover the true quotient-digit and select the appropriate divisor multiple. This results in the addition of a decoder delay into the critical path. However, since all tables in this study use the same encoding, this is a constant delay that only grows as the log of the radix. Alternatively, the quotient-digits could be stored in an unencoded form, removing the

Allowable Digits	Encoding
0	00
1	01
0 or 1	0x
2	11
1 or 2	x1
3	10
2 or 3	1x

Table 1: Gray encoding for maximally redundant radix 4

need for the decoder. Optimal logic minimization becomes a much more difficult problem for unencoded digits.

Espresso is used to perform logic minimization on the output PLA. This produces a minimized PLA and the logic equations representing the PLA in sum of products form. The number of product terms in these expressions is one metric for the complexity of the tables. To verify the correctness of the tables, an SRT divider was simulated using a DEC Alpha 3000/500 with the minimized logic equations as the quotient-digit selection function. After each table was generated, the equations were incorporated into the simulator. Several thousand random and directed IEEE double precision vectors were used as input to the simulator, and the computed quotients for each table were compared with the computed results from the Alpha's internal FPU.

4.2 Table Synthesis

To quantify the performance and area of random logic implementations of the tables, each table was synthesized using a standard-cell library. The Synopsys Design Compiler [18] was used to map the logic equations describing each table to an LSI Logic 500K 0.5μm standard-cell library [12]. In the mapping stage, low flattening and medium mapping effort options were used. However, area was always sacrificed to reduce the latency. In order to minimize the true critical path of the tables, the input constraints included the late arrival time of all partial remainder inputs due to an external carry-assimilating adder.

Area and delay estimates were obtained from a Design Compiler pre-layout report. Each delay includes intrinsic gate delay, estimated interconnect wire delay, and the input load of subsequent gates. All delays were measured using nominal conditions. Each area measurement includes both cell and estimated routing area. The delay and area are reported relative to those for a base radix 4 table, which is Gray-coded with $i = 3$, $f = 3$, $g = 3$, and $d = 3$. The base table requires 43 cells and has a delay of 1.47ns.

The results are presented in tables 2 through 6. The complexity of the tables is measured by the number of product terms, the relative delay, and the relative area for each configuration. The terms result contains both the number of product terms for each output of the table, as well as the total number of terms in the table. For a given radix r, there are

exactly $n_{out} = \log_2 r$ outputs of the table. Accordingly, this column first lists the number of terms in each of the n_{out} outputs. Because there is usually some internal sharing of product terms, the last number, which is the total number of unique terms required to implement the table, is typically less than the sum of the terms for the individual outputs. The reported delay is the worst-case delay of any of the n_{out} outputs, typically corresponding to the output which has the highest number of product terms.

5 Results

5.1 Same Radix Tradeoffs

Description	a	b	i	f	g	Terms	Relative Delay	Relative Area
Baseline	2	3	4	3	3	19,8,25	1.00	1.00
	2	3	4	3	4	17,8,23	0.88	0.93
	2	3	4	3	5	17,8,23	0.84	0.91
	2	3	4	3	52	14,6,18	0.77	0.77
Folded	2	3	4	3	4	13,5,17	0.79	0.65
Folded + t-bit conv	2	3	4	3	4	12,4,16	0.71	0.57
Line Encode	2	3	4	3	4	17,7,23	0.84	0.86
Choose Highest	2	3	4	3	4	22,13,33	1.05	1.23
Max Red	3	2	5	1	1	6,9,14	0.80	0.54
	3	2	5	1	52	3,6,9	0.63	0.30

Table 2: Radix 4 Tradeoffs

Table 2 shows the results for various radix 4 configurations. The parameters varied in this table are 1) the number of g bits vs f bits, 2) folding, 3) method of choosing values in the overlap regions, and 4) the amount of redundancy. The first entries examine the effects of using g bits of the redundant partial remainder into the short CPA outside of the table, while only using f bits of the adder output as input to the table. Simply extending the CPA by one bit reduces the delay by 12% and area by 7%. Extending the CPA by two bits reduces the delay by 16% and area by 9%. In the limit, where $g = n$ and a full-mantissa carry propagate addition is required, the delay and area are both reduced by 23%. This demonstrates that increasing the width of the CPA by as little as one or two bits can reduce the complexity of the table.

The next two entries demonstrate the effects of using folded tables. For both tables, it is assumed that $f = 3$ and $g = 4$, which matches the format of the table suggested in Fandrianto [8]. The use of only a two's complement to sign-magnitude converter yields the first folded table, which achieves an additional delay reduction of 10% and an additional area reduction of 30% over the $f = 3$ $g = 4$ table without folding. This introduces a serial delay of an XOR gate external to the delay of the table. When the sign-magnitude converter

a	b	i	f	g	Terms	Relative Delay	Relative Area
1	0	3	1	1	3	0.35	0.07

Table 3: Radix 2

is combined with a "t-bit converter", which further constrains the range of input values to the table, the delay is reduced relative to the simple folded table by an additional 10%, and the area is reduced by an additional 12%. This converter introduces the serial delay of an OR gate external to the table. These results show that folding can reduce the size and delay of the table. However, the delay of the required external gates must be considered for the overall design. If the sum of the XOR and OR gate delays is less than 29% of the base table delay, table folding can result in a net decrease in delay.

Different encodings of the quotient-digits can change the complexity of the tables. The lower bound for delay and area of the table is achieved when each boundary between two consecutive quotient-digits is individually encoded. The recovery of the unencoded quotient-digit may require a large external decoder. When using such a "line" encoding scheme, again with $f = 3$ and $g = 4$, the delay and area are reduced by 5% and 8% respectively relative to the base Gray-coded table, also with $f = 3$ and $g = 4$. However, the external decoder delay grows linearly with increasing a for line encoding, while only growing as the log of a for Gray-coding. Another common encoding scheme always uses the highest digit whenever a choice is available between two consecutive quotient-digits. This is represented in the table as "choose highest" encoding. While simplifying the table generating process, this method increases the resulting table delay by 19% and area by 32% over the base $f = 3$ $g = 4$ table. Thus, this study shows that Gray-coding of the quotient-digits achieves delays and areas approaching the lower bound of line encoding, while requiring less complex external decoders.

The redundancy of the digit set has an impact on table complexity. The final entries in the table are for maximally redundant radix 4 tables, with $a = 3$. For an implementation with $f = g = 1$, the delay and area are reduced by 20% and 46% respectively. When g increases to $n = 52$, requiring a full mantissa-width CPA, the delay is further reduced by 21% and the area by 44%. These results show that if the hardware is available to generate the 3x divisor multiple, the iteration time can be reduced by over 20%, due to the reduction in table complexity and length of the external short CPA.

Table 3 shows the complexity of a basic radix 2 table. This table can be implemented by a single three-input gate, as it only has 3 PLA terms, which can be contrasted with the 25 terms in the baseline radix 4 table. The resulting delay is 65% less than the base radix 4 table, while 93% less area is required. Accordingly, the radix 2 table is 2.86 times faster than the base radix 4 table, and 2.40 times faster than a $g = 5$ radix 4 table.

5.2 Higher Radix

Tables 4, 5, and 6 show the complexity for tables that directly implement radix 8, 16,

a	b	i	f	g	Terms	Relative Delay	Relative Area
4	7	5	4	4	137,59,114,292	1.85	10.2
	7	5	4	5	111,48,94,240	1.76	8.80
	6	5	5	5	110,50,94,240	1.70	8.85
	5	5	6	6	104,50,85,221	1.67	8.21
5	5	5	3	3	42,19,69,122	1.45	5.21
	4	5	4	4	35,14,57,103	1.47	4.05
6	5	5	2	2	26,35,39,92	1.46	4.82
	4	5	3	3	24,33,38,88	1.46	4.46
	3	5	5	5	27,29,33,78	1.36	4.03
7	6	6	1	1	16,23,43,76	1.46	3.90
	3	6	2	2	15,21,35,64	1.41	3.53

Table 4: Radix 8

and 32 respectively. The allowable choices of i, b, and f determined in this study correspond with the results presented in [4] for radix 8 and 16. In our study, we extend the allowed operand truncations to radix 32. For radix 16 and radix 32, the minimally redundant configurations required 20 or more inputs to the table. Due to computational constraints, table optimization was limited to configurations containing fewer than 20 inputs. Those configurations where optimization was infeasible are denoted with a dagger in the tables.

For a given choice of radix and redundancy, there exists more than one possible table design. As discussed previously, a minimum number of divisor estimate bits is required as input for a given configuration. This corresponds to a maximum number of partial remainder bits that need be used. However, it is possible to trade an increase in divisor bits for a reduction in the number of partial remainder bits. This might initially seem desirable, as the partial remainder bits must first be assimilated in an external adder, adding to the overall iteration time. By using a fewer number of partial remainder bits in the table, the external adder can be smaller, reducing the external delay. However, for carry-save partial remainders, the maximum partial remainder truncation error $\epsilon_{p(cs)}$ is greater than the maximum divisor truncation error ϵ_d. By trading-off fewer partial remainder bits for more divisor bits, the height of the uncertainty region increases at approximately twice the rate at which the width of the region decreases. As a result, the overall uncertainty region area increases as fewer partial remainder bits are used. This result can be seen quantitatively in tables 4, 5, and 6. For any given choice of radix and redundancy, the use of the maximum number of divisor bits and minimum number of partial remainder bits results in the largest number of total product terms, and typically the largest delay and area. As the number of divisor bits is reduced and the number of partial remainder bits increased, the number of product terms, the delay, and the area are all typically reduced.

This study confirms that as the radix increases, the complexity of the tables also increases. Fitting the average area at a given radix to a curve across the various radices

a	b	i	f	g	Terms	Relative Delay	Relative Area
8	11	6	5	5	†	†	†
	8	6	6	6	†	†	†
	7	6	8	8	†	†	†
9	7	6	4	4	198,98,176,481,871	2.56	32.9
	6	6	5	5	170,81,154,410,745	2.50	29.0
10	7	6	3	3	120,58,231,280,616	2.37	24.8
	6	6	4	4	105,57,191,240,530	2.32	21.4
	5	6	5	5	96,49,183,227,497	2.24	21.1
11	6	6	3	3	80,44,159,258,484	2.21	21.1
	5	6	4	4	68,39,142,208,418	2.10	18.9
12	7	6	2	2	79,138,144,228,481	2.17	25.0
	6	6	3	3	66,112,119,172,373	2.09	19.1
13	6	6	2	2	60,105,105,207,393	2.14	20.4
	5	6	3	3	62,104,99,186,344	2.07	18.5
14	5	6	2	2	48,101,136,169,383	2.14	20.2
	4	6	6	6	54,92,110,139,310	2.06	16.5
15	9	7	1	1	47,76,135,193,383	2.16	19.2
	5	7	2	2	39,69,99,136,281	2.03	15.4
	4	7	3	3	42,61,93,125,261	1.94	14.8

Table 5: Radix 16

determines that the area increases geometrically with increasing radix:

$$\text{Area} \quad = \quad .1R^2 \tag{24}$$

for radix R, where this area is the table area relative to that of the base radix 4 divider. Similar analysis of average delay demonstrates that delay increases only linearly with increasing radix. For radix 8, the delay is on the average about 1.5 times that of the base radix 4 table. However, it can require up to 10 times as much area. While radix 16 tables have about 2 times the base delay, they can require up to 32 times the area. In the case of radix 32, it was not even possible to achieve a delay of 2.5 times the base delay, the maximum desired delay, with actual delays between 3.5 and 4.7. The area required for radix 32 ranges from 57 to 141 times the base area. These results show that radix 16 and 32 are clearly impractical design choices, even ignoring practical implementation limitations such as generating all divisor multiples. This study shows that it is possible to design radix 8 tables with reasonable delay and area; a minimally-redundant radix 8 table is demonstrated to be a practical design choice.

a	b	i	f	g	Terms	Relative Delay	Relative Area
17	9	7	5	5	†	†	†
	8	7	6	6	†	†	†
18	9	7	4	4	†	†	†
	8	7	5	5	†	†	†
	7	7	7	7	†	†	†
19	8	7	4	4	351,208,352,945,1633,3119	4.56	141
	7	7	5	5	309,191,308,860,1445,2767	4.18	79.1
20	9	7	3	3	312,164,660,891,1527,3218	4.69	144
	7	7	4	4	257,156,531,727,1215,2592	4.24	106
21	8	7	3	3	237,128,507,649,1274,2499	4.13	73.2
	7	7	4	4	206,118,424,541,1060,2099	4.06	94.8
	6	7	6	6	180,108,366,466,912,1826	3.91	80.5
22	7	7	3	3	192,119,450,733,1032,2127	4.31	101
	6	7	5	5	178,90,356,596,794,1696	4.07	80.9
23	7	7	3	3	158,85,365,607,946,1865	4.11	92.5
	6	7	4	4	140,84,316,527,814,1621	3.86	78.6
24	9	7	2	2	207,408,421,678,1073,2243	4.62	92.5
	7	7	3	3	147,327,314,491,780,1678	3.63	86.8
	6	7	4	4	142,286,277,458,699,1497	3.61	73.3
25	8	7	2	2	185,330,318,543,985,1978	4.36	99.6
	6	7	3	3	144,252,258,425,747,1534	3.70	76.3
26	8	7	2	2	154,291,299,607,838,1783	3.95	91.6
	6	7	3	3	123,249,235,505,687,1475	3.73	75.1
27	7	7	2	2	141,263,266,535,798,1620	3.73	86.4
	6	7	3	3	126,221,227,439,661,1377	3.63	73.7
28	7	7	2	2	146,233,359,494,717,1578	3.86	82.0
	6	7	3	3	134,218,322,413,599,1327	3.76	74.4
29	7	7	2	2	226,233,342,431,697,1480	3.75	82.1
	6	7	3	3	116,188,259,349,573,1241	3.73	69.8
30	6	7	2	2	145,220,318,461,656,1433	4.14	76.0
	5	7	7	7	165,184,252,351,505,1143	3.78	61.0
31	11	8	1	1	†	†	†
	6	8	2	2	89,170,241,399,555,1158	4.05	61.3
	5	8	4	4	86,152,219,333,486,1021	3.52	57.2

Table 6: Radix 32

6 Conclusion

This study has demonstrated a methodology for generating quotient-digit selection tables from a table specification through an automated design flow. Using this process, performance and area tradeoffs of quotient selection tables in SRT dividers have been presented for several table configurations. The use of Gray-coding is shown to be a simple yet effective method that allows automatically determining optimal choices of quotient-digits which reduce table complexity.

Short external carry-assimilating adders are necessary to convert redundant partial remainders to a non-redundant form. By extending the length of these adders by as little as one or two bits, it is shown that table complexity can be further reduced. The conventional wisdom for SRT table specification has been whenever possible, the length of the partial remainder estimate should be reduced at the expense of increasing the length of the divisor estimate in order to reduce the width, and thus the delay, of the external adder. However, this study quantitatively demonstrates that such a choice also increases the size and delay of the table, mitigating the performance gain provided by the narrower adder. Accordingly, the overall iteration time is not reduced through such a tradeoff.

As the radix increases, it is shown that the table delay increases linearly. However, the area increases quadratically with increasing radix. This fact, combined with the difficulty in generating all of the required divisor multiples for radix 8 and higher, limits practical table implementations to radix 2 and radix 4.

References

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

[2] A. Barna and D. Porat. *Integrated Circuits in Digital Electronics.* John Wiley and Sons, 1973.

[3] M. Birman, A. Samuels, G. Chu, T. Chuk, L. Hu, J. McLeod, and J. Barnes. Developing the WTL 3170/3171 Sparc floating-point co-processors. *IEEE Micro*, 10(1):55–63, February 1990.

[4] N. Burgess and T. Williams. Choices of operand truncation in the SRT division algorithm. *IEEE Transactions on Computers*, 44(7):933–937, July 1995.

[5] T. Carter and J. Robertson. Radix-16 signed-digit division. *IEEE Transactions on Computers*, 39(12):1243–1433, December 1990.

[6] M. D. Ercegovac and T. Lang. Simple radix-4 division with operands scaling. *IEEE Transactions on Computers*, C-39(9):1204–1207, September 1990.

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

[8] J. Fandrianto. Algorithm for high-speed shared radix 4 division and radix 4 square root. In *Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, pages 73–79, May 1987.

[9] J. Fandrianto. Algorithm for high-speed shared radix 8 division and radix 8 square root. In *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, pages 68–75, July 1989.

[10] S. Knowles. Arithmetic processor design for the T9000 transputer. *ASPAAI-2*, 1991.

[11] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Tanaguchi, and N. Takagi. Design of high speed MOS multiplier and divider using redundant binary representation. In *Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, pages 80–86, 1987.

[12] LSI Logic lcb500k standard-cell library, 1994.

[13] P. Montuschi and L. Ciminiera. Over-redundant digit sets and the design of digit-by-digit division units. *IEEE Transactions on Computers*, 43(3):269–277, March 1994.

[14] S. Oberman and M. Flynn. An analysis of division algorithms and implementations. Technical Report No. CSL-TR-95-675, Computer Systems Laboratory, Stanford University, July 1995.

[15] J. A. Prabhu and G. B. Zyner. 167 MHz Radix-8 floating point divide and square root using overlapped radix-2 stages. In *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pages 155–162, July 1995.

[16] J. E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, EC-7(3):88–92, September 1958.

[17] H. Srinivas and K. Parhi. A fast radix-4 division algorithm and its architecture. *IEEE Transactions on Computers*, 44(6):826–831, June 1995.

[18] Synopsys Design Compiler version v3.2b, 1995.

[19] K. G. Tan. The theory and implementation of high-radix division. In *Proceedings of the 4th IEEE Symposium on Computer Arithmetic*, pages 154–163, June 1978.

[20] G. S. Taylor. Radix 16 SRT dividers with overlapped quotient selection stages. In *Proceedings of the 7th IEEE Symposium on Computer Arithmetic*, pages 64–71, June 1985.

[21] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P.G.A. Jespers. A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor. *IEEE Journal of Solid State Circuits*, 25:748–756, June 1990.

[22] D. Zuras and W. McAllister. Balanced delay trees and combinatorial division. *IEEE Journal of Solid State Circuits*, SC-21(5):814–819, October 1986.