2011 20th IEEE Symposium on Computer Arithmetic

Radix-8 Digit-by-Rounding: Achieving High-Performance Reciprocals, Square Roots, and Reciprocal Square Roots

J. Adam Butts, Ping Tak Peter Tang, Ron O. Dror, and David E. Shaw*
D. E. Shaw Research, New York, NY 10036, USA

***Abstract*—We describe a high-performance digit-recurrence algorithm for computing exactly rounded reciprocals, square roots, and reciprocal square roots in hardware at a rate of three result bits—one radix-8 digit—per recurrence iteration. To achieve a single-cycle recurrence at a short cycle time, we adapted the digit-by-rounding algorithm, which is normally applied at much higher radices, for efficient operation at radix 8. Using this approach avoids in the recurrence step the lookup table required by SRT—the usual algorithm used for hardware digit recurrences. The increasing access latency of this table, the size of which grows superlinearly in the radix, limits high-frequency SRT implementations to radix 4 or lower. We also developed a series of novel optimizations focused on further reducing the critical path through the recurrence. We propose, for example, decreasing datapath widths to a point where erroneous results sometimes occur and then correcting these errors off the critical path. We present a specific implementation that computes any of these functions to 31 bits of precision in 13 cycles. Our implementation achieves a cycle time only 11% longer than the best reported SRT design for the same functions, yet delivers results in five fewer cycles. Finally, we show that even at lower radices, a digit-by-rounding design is likely to have a shorter critical path than one using SRT at the same radix.**

***Keywords*-Digit recurrence, reciprocal, square root, reciprocal square root, exact rounding, prescaling, selection by rounding**

I. Introduction

A difficult problem in designing arithmetic hardware is the efficient computation of algebraic functions that produce non-terminating bit strings such as reciprocals and square roots. Realizable hardware necessarily computes a limited-precision approximation of the true result by composing many elementary, fixed-precision computations (e.g., additions and multiplications). The complexity of the overall computation leads to an enormous design space, comprising approaches that differ in achievable precision, cycle time, overall latency, and circuit area.

Digit-recurrence algorithms are commonly used for computing such functions in hardware because they offer high hardware efficiency (only a few arithmetic units are required), a straightforward rounding procedure, and the potential for a short cycle time. One well-known instance of a digit-recurrence algorithm is pencil-and-paper long division, wherein the remainder is initialized to the dividend and result digits are generated by informed trial and error. The remainder is updated by subtracting from it the product of the divisor and each new digit. After the desired number of digits has been generated, the result is rounded up if the remainder exceeds half of the divisor.

Generalizing this example, a digit-recurrence algorithm operates as follows:

1) Initialize a *remainder* based on the input operand and function to be computed.
2) Using the remainder, which is related to the error in the current partial result, determine the next digit in the partial result (*digit selection*). Result digits are generated in an implementation-defined base called the *radix*.
3) Update the remainder using the new result digit according to a *recurrence relation*.
4) Repeat steps 2–3 until rounding to the desired precision is possible.

The choice of radix is the most important decision in implementing a digit recurrence: a higher radix reduces the total number of cycles, while a lower radix enables a faster cycle time (i.e., a higher clock frequency). The highest performance is attained by targeting the lowest product of cycle count and cycle time. Real-world designs are seldom so unconstrained: cycle time is usually dictated by the co-optimization of a large number of other components. In this situation, performance is maximized by using the highest radix possible under the fixed cycle time constraint.

The favored approach to high-frequency digit recurrences—known as *SRT*, after its co-discoverers Sweeney, Robertson, and Tocher [1]—performs digit selection using a lookup table. The size (and, thus, the latency) of this table scales superlinearly in the radix, and the critical path through the recurrence computation inevitably includes the large table lookup. As a result, high-frequency implementations typically use a radix no greater than four. Some designs cascade lower-radix stages within a single cycle to achieve a higher effective radix [2], but this incurs extra hardware cost, and instances of these higher-performing implementations have been limited to computing quotients and square roots (e.g., [3], [4]).

In this paper, we develop a radix-8 digit recurrence capable of computing $1/x$, $\sqrt{x}$, and $1/\sqrt{x}$ at short cycle

*David E. Shaw is also with the Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10032. Email correspondence: David.Shaw@DEShawResearch.com.

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.28

times. To achieve this goal, we employ a *digit-by-rounding* algorithm, which formulates the remainder such that result digits are obtained from it simply by rounding [5], [6]. Compared to SRT, digit-by-rounding requires more complex computations to initialize the remainder and other recurrence variables. Essentially, the digit-by-rounding method moves complexity out of the recurrence, where its cost is multiplied by the iteration count, and places it into the initialization. Historically, this technique has been associated with multi-cycle iterations, moderate cycle times, and very high radices (radices up to 2^{14} have been suggested [7]). Applying this technique at radix 8—well outside its usual domain—we demonstrate its superiority to SRT for single-cycle iterations at short cycle times. We also leverage a new analysis technique [8] to simplify the initialization computations compared to what was deemed necessary in previous applications of this algorithm.

In developing this lower-radix version of digit-by-rounding, we devised several novel optimizations. We aggressively reduced the precisions of all recurrence variables, narrowing the width of the arithmetic components involved in their computation. One example is our "over-engineering" of internal precisions: we reduced datapath widths to a point where some inputs lead to incorrect results and then correct those results during the final rounding step. We also restructured the recurrence computations to minimize delay. By computing a biased remainder, for example, we avoid an extra addition in the critical path of digit selection.

Using these techniques collectively, we present a radix-8 design for computing correctly rounded reciprocals, reciprocal square roots, and square roots to 31 bits of precision in 13 cycles. The resulting design has a critical path of approximately 22.9 FO4, or 5.3 t_{FA}.[1] Our optimizations deliver reductions of over 10% in both cycle time and area relative to the initial radix-8 digit-by-rounding design. The best previously reported design for computing the same functions—a radix-4 SRT design—has a latency of up to 18 cycles with an estimated critical path of 4.5 t_{FA} [9]. The digit-by-rounding design achieves a clock rate within 11% of that of the lower-radix design (at equivalent pipelining overhead) while delivering results with a 30% reduction in cycle count.

II. Digit Selection by Rounding

In general terms, the digit selection by rounding method (shortened herein to digit-by-rounding) defines a remainder that can be decomposed into a product of one factor that captures the error in the current partial result and another factor that *scales* that error to effect the selection-by-rounding property. This latter factor depends only on the input value and the function to be computed, not the iteration; it is thus incorporated into the remainder prior to the first iteration and is consequently called a *prescaling factor*. The prescaling factor is not trivially computable—indeed, its ideal value is the desired result $f(x)$ in some cases—but it need only be known to limited precision. Prescaling the input operands to enable selection by rounding has long been applied to achieve high-radix digit recurrences (e.g., [7], [10]).

To develop the algorithm, assume that we have obtained somehow the first digit of the result d_1 in radix $r_1 = 2^{b_1}$. (We will see how d_1 is related to the prescaling factor, which also must be determined at the start of the recurrence computation.) Subsequent result digits d_j, for $j > 1$, are generated by the recurrence and have a lower radix $r = 2^b < r_1$. The partial result after J iterations is

$$P_J = \sum_{j=1}^{J} \frac{d_j}{r_1 r^{j-1}}. \tag{1}$$

We seek a means to determine the d_j such that

$$\lim_{J\to\infty} P_J = f(x). \tag{2}$$

Define the *residual* γ_J to be the scaled error in the partial result after the Jth iteration:

$$\gamma_J \stackrel{\text{def}}{=} r_1 r^{J-1} \left[f(x) - P_J \right]. \tag{3}$$

Using the definitions (1) and (3),

$$\begin{aligned} \gamma_{J+1} &= r_1 r^J \left[f(x) - P_{J+1} \right] \\ &= r_1 r^J \left[f(x) - P_J \right] - d_{J+1} \\ &= r\gamma_J - d_{J+1}. \end{aligned} \tag{4}$$

Keeping the residual bounded (i.e., $|\gamma_J| < a$ for some fixed a) is equivalent to reducing the maximum error in the partial result by a factor of r each iteration. Noting that the digit must be an integer, the best possible choice to minimize the magnitude of the next residual γ_{J+1} is clearly $d_{J+1} = \text{round-to-int}(r\gamma_J)$. This choice results in $|\gamma_{J+1}| \le \frac{1}{2}$.

A constraint on digit selection is that all digits produced must belong to a predefined *digit set* $\mathcal{D}$. It is important that the digit set be *redundant*. That is, $|\mathcal{D}| > r$, and a given value will have multiple representations. Redundancy confers some slack in the digit selection, which is necessary to enable the use of an approximate residual or rounding method. The amount of redundancy allowed by the digit set ultimately determines how precise certain approximations must be for the algorithm to succeed.

Examining the form of the residual (3) reveals that it is not directly usable for computing digits of $f(x)$ because it is defined in terms of $f(x)$. Through some algebraic manipulation, however, we can write the residual as a product of a computable factor of finite precision and a non-computable factor that we can approximate. Using $f(x) = 1/(2\sqrt{x})$ as

[1] FO4 and t_{FA} are technology-independent measures of delay—the delay of a fanout-4 inverter and the propagation delay to the sum output of a single-bit full adder, respectively.

Function	M	R_J	L_J	Q_J	R_1	L_1	Q_1
$\frac{1}{2x}$	$\approx \frac{1}{2x}$	$Mr_1r^{J-1}(1-2xP_J)$	$2Mx$	0	$M(r_1-2xd_1)$	$2Mx$	0
$\frac{1}{2\sqrt{x}}$	$\approx \frac{1}{2\sqrt{x}}$	$Mr_1r^{J-1}(\frac{1}{2}-2xP_J^2)$	$4MxP_J$	$2Mx/(r_1r^J)$	$M(r_1/2-2xd_1^2/r_1)$	$4Mxd_1/r_1$	$2Mx/(r_1r)$
$\sqrt{x}$	$\approx \frac{1}{2\sqrt{x}}$	$Mr_1r^{J-1}(x-P_J^2)$	$2MP_J$	$M/(r_1r^J)$	$M(xr_1-d_1^2/r_1)$	$2Md_1/r_1$	$M/(r_1r)$

Table 1: Recurrence variable definitions: function-specific definitions of the pre-scaling factor M, the remainder R, and the coefficients L and Q used in the recurrence for the remainder. Formulae for the initialization values at $J=1$ are expanded in terms of the first digit d_1 by noting that $P_1 = d_1/r_1$.

an example,[2]

$$\begin{aligned}\gamma_J &= r_1 r^{J-1}\left(\frac{1}{2\sqrt{x}} - P_J\right)\left(\frac{1/(2\sqrt{x})+P_J}{1/(2\sqrt{x})+P_J}\right)\\ &= r_1 r^{J-1}\left(\tfrac{1}{2}-2xP_J^2\right)\left(\frac{1}{\sqrt{x}+2xP_J}\right)\\ &\approx r_1 r^{J-1}\left(\tfrac{1}{2}-2xP_J^2\right)\left(\frac{1}{2\sqrt{x}}\right),\end{aligned} \tag{5}$$

where we assumed that $P_J \approx f(x)$ (which is true if the algorithm works) or $1/(2\sqrt{x})$ in this example. Note that the first parenthesized factor is finite precision, trivially computable (i.e., using only additions and multiplications) after iteration J, and embodies the error in the current partial result. The residual need only be known to a precision high enough to perform digit selection. We can thus replace the last factor with an approximation—the prescaling factor M, which is independent of J. We revisit the accuracy requirement of this approximation at the end of the section.

We can now define the remainder R_J as a computable approximation of the residual γ_J:

$$R_J \stackrel{\text{def}}{=} Mr_1r^{J-1}\left(\tfrac{1}{2}-2xP_J^2\right) \approx \gamma_J. \tag{6}$$

Digit selection is performed by rounding the scaled remainder rR_J in lieu of the unavailable $r\gamma_J$. In practice, the remainder R_J is represented in a carry-save representation $R_J^{\text{sum}} + R_J^{\text{car}}$ to avoid a long carry-propagation delay in updating the remainder. To avoid incurring this delay in digit selection instead, we use only leading bits of the scaled remainder to compute the next digit:

$$d_{J+1} = \left\lfloor 2^{-t}\Big(\lfloor 2^t r R_J^{\text{sum}}\rfloor + \lfloor 2^t r R_J^{\text{car}}\rfloor\Big) + \tfrac{1}{2}\right\rfloor, \tag{7}$$

where the inner $\lfloor\cdot\rfloor$s represent the truncation of the sum and carry portions of the scaled remainder to t fractional bits and the outer $\lfloor\cdot+\frac{1}{2}\rfloor$ represents the rounding to an integral digit. The error introduced by the truncation may be modeled as another difference between the remainder and the residual. Choosing an appropriate t limits the error to the same order as that from using an M of limited precision.

Computing the remainder in each iteration using its definition (6) is prohibitively expensive as $2xP_J^2$ is a product of three full-precision factors. Instead, we derive a recurrence for updating the remainder by substituting $J \to J+1$ in (6) and solving for R_{J+1} in terms of R_J:

$$\begin{aligned}R_{J+1} &= Mr_1r^J\left[\tfrac{1}{2}-2x\left(P_J+\frac{d_{J+1}}{r_1r^J}\right)^2\right]\\ &= rR_J - 4MxP_Jd_{J+1} - \frac{2Mx}{r_1r^J}d_{J+1}^2.\end{aligned} \tag{8}$$

Letting L_J and Q_J be the coefficients of the linear and quadratic terms in the digit d_{J+1}, respectively,

$$R_{J+1} = rR_J - L_Jd_{J+1} - Q_Jd_{J+1}^2. \tag{9}$$

Recurrences may also be derived for the coefficients themselves:

$$L_{J+1} = L_J + 2Q_Jd_{J+1}. \tag{10}$$

$$Q_{J+1} = Q_J/r. \tag{11}$$

Similar derivations also can be performed for $f(x) = 1/(2x)$ and $f(x) = \sqrt{x}$. Table 1 lists the resulting definitions of M, R_J, L_J, and Q_J. Although the definitions depend on the particular $f(x)$, the recurrence expressions (9)–(11) are identical for all three functions considered here; the same hardware can thus compute the recurrences in each case.

The precisions[3] of M and d_1 are constrained by the need to produce a radix-r digit by rounding the remainder. By equation (6), the relative error in R_J can be no less than that in M, so M's precision m clearly must be somewhat more than b bits. Separately, d_1's radix r_1 must be high enough to support the selection of d_2 using R_1. These requirements naturally lead to $m > b$ and $b_1 > b$ (this latter constraint elucidates the higher radix $r_1 > r$ of the first digit).

For the reciprocal and reciprocal square root functions, $M \approx f(x) \approx d_1/r_1$, which means that we need obtain only one of M or d_1 during initialization since they are related by a simple scaling (neglecting for now differences in precision). Although M and d_1 are not related so simply for the square root, M is the same as that for the reciprocal square root at the same value of x, and thus we only need a method for obtaining d_1 in this case. Given d_1 for each function then, we have all necessary information to initialize the recurrence variables.

[2]The factor of $\frac{1}{2}$ is for normalization; see Section IV.

[3]Precision here refers to the number of significant bits without regard to the location of the radix point (e.g., the precisions of the digits d_j are $b_1 = \log_2 r_1$ for $j=1$ and $b = \log_2 r$ for $j > 1$).

III. Optimizing Digit-by-Rounding

In this section, we develop optimizations to the digit-by-rounding algorithm of Section II. While we describe all of the optimizations presented here in the context of the digit-by-rounding algorithm, we note that the quadratic term computation of Section III-B and the precision minimization technique of Section III-C are applicable to digit recurrences in general. At this point, we fix the radix at $r = 8$, which is assumed by some of the optimizations (notably the computation of the quadratic term). This radix also serves to demonstrate the utility of the digit-by-rounding approach over SRT in achieving high-frequency designs at a radix lower than usual for the former, yet difficult for the latter.

A. Parameter selection for convergence

The design parameters of a digit-by-rounding implementation are constrained by the requirement that the partial result converge to $f(x)$, which requires that the residual remain bounded. Another requirement is that the digits d_j fall within some fixed digit set $\mathcal{D}$ for all $j > 1$.

We described at the end of Section II some general requirements for convergence. For example, we discussed why $m > b$ is required, but did not determine by how much. Previous work derives sufficient conditions for convergence in the form of equations relating the various design parameters [7], [10]. These analyses are formulated in terms of the maximum errors introduced by the various approximations inherent in the algorithm (e.g., the finite precision of M or the truncation of the remainder to t fractional bits prior to digit selection) over the entire input domain. The global analysis results in conservatism, which manifests as parameter values (i.e., precisions) larger than strictly necessary for a correct design.

We developed a new analysis method to give us very tight bounds on the ranges of the digits possible at each iteration given a set of design parameters, allowing us to certify more efficient combinations of parameters than can be shown to be correct using the existing techniques. We describe the details of this method elsewhere [8]. In particular, we were able to prove convergence with $b_1 = m = 5$ versus the previous lower bounds of $b_1 = b + 3 = 6$ and $m = b + 6 = 9$ [7].

This refinement makes a tangible difference in the complexity of the recurrence initialization. The lower precisions for M and d_1 mean that we can obtain them directly from a lookup table; previous designs needed to perform additional computation (e.g., interpolation) after a table lookup in order to obtain M and d_1 at the required precisions from a table of reasonable size. The fact that $m = b_1$ means that no computation is required to determine M from d_1, and thus one simple table lookup of d_1 (or two lookups in the case of square root) delivers d_1 and M simultaneously.

In addition to m and r_1, our analysis also verifies values for t and l (the number of leading bits of x required to retrieve d_1). The analysis directly indicates the digit set $\mathcal{D}$ required to support a given set of precision parameters $\{r, r_1, m, t, l\}$. In our case, we use a maximally redundant digit set ($\mathcal{D} = \{d \in \mathbb{Z} : |d| < r\}$), which provides the lowest values of the precision parameters without significantly affecting the complexity of the recurrence computations.

d_{J+1}	d_{J+1}^2	Q^0	Q^1	Q^2
0	0	0	0	0
± 1	1	Q_J	0	0
± 2	4	0	$4Q_J$	0
± 3	9	Q_J	0	$8Q_J$
± 4	16	0	$16Q_J$	0
± 5	25	Q_J	$16Q_J$	$8Q_J$
± 6	36	0	$4Q_J$	$32Q_J$
± 7	49	Q_J	$16Q_J$	$32Q_J$

Table 2: Computing the quadratic term: d_{J+1} is decoded to select $Q^{0,1,2}$ such that $\sum Q^i = Q_J d_{J+1}^2$.

B. Fast radix-8 quadratic term

Computing the quadratic term of the recurrence requires multiplying the coefficient by the square of the digit. Naïvely considering the digit squared as a six-bit multiplier of the coefficient leads to five partial products (as the 2^1 bit of a square is always 0), or the need to perform a recoding after computing the square (e.g., radix-4 Booth recoding, which still yields four partial products).

There are only eight unique squares for all possible values of the digit; furthermore, none of these squares has more than three ones in its binary coding. We can thus select three appropriate partial products that compose the quadratic term based on a simple decoding of the digit according to the scheme in Table 2.

The critical path through this computation is reduced to the decoding of a mux select followed by a 2:1 mux[4] to select the shifted coefficient $2^i Q_J$. Three such partial products $Q^{0,1,2}$ compose the quadratic term for iteration J.

C. Precision minimization

At the conclusion of the final iteration N, the partial result P_N must be rounded. Exact rounding requires that the delivered result y equal the infinitely precise result $f(x)$ rounded to the target precision p. For a finite-precision result to round to the exactly rounded result, it must be accurate to some function-specific precision larger than p.

Bounds on the excess precision required to return an exactly rounded result have been examined extensively [11], [12]. Of the functions being considered here, the reciprocal square root places the highest demands on the pre-rounding precision: loose analytical bounds suggest $2p-1$ excess bits are required, although closer to p bits usually suffice.

[4]The muxes generating $Q^{1,2}$ are technically 3:1 muxes in that, in addition to two shifted coefficients, zero is a possible output; a reasonable 2:1 mux implementation, however, can also generate a zero output with no timing penalty.

As a practical matter, however, the maximum precision will be required only for a small number of cases where the exact result is very near a midpoint at the target precision. Significant hardware savings may thus be obtained by truncating the recurrence variables to a lower precision if exact rounding is not required. For example, Antelo et al. reduce the width of their datapaths by over 40% [7] by allowing the delivered result to differ from the exactly rounded result by up to one ulp (unit in the last place).

We extend this optimization to designs that demand exact rounding as follows. Assuming that the error in the pre-rounded result is always less than 1 ulp (true given any amount of excess precision), the errors in the result will manifest as an incorrect rounding decision. The correct result thus may be recovered by simply inverting the rounding decision in these cases: rounding up when no rounding was indicated and vice versa. This correction is easily performed during the final rounding step, leaving nearly the entire duration of the computation to detect whether the input requires such a correction. As the pre-rounding precision is reduced, the number of exception cases increases rapidly. The minimum practical precision will be limited ultimately by the increasing complexity of the detection logic.

The exception cases must be explicitly enumerated *a priori* and detected by the implementation. For low-enough input precision p, exhaustive testing may be used to identify the exception cases for particular choices of precision parameters. We illustrate this approach in Section IV-D. When the input domain is too large for exhaustive testing, a more focused approach is required. For certain functions, so-called *difficult-to-round cases*—those inputs leading to results lying close to a midpoint—can be computed by solving Diophantine equations [13]. Among the functions we consider here, only reciprocal and square root can be analyzed with this technique. If high-precision reciprocal square roots are required, then the suboptimal, analytically derived precisions must be used or more advanced techniques must be employed (e.g., [14]).

D. Remainder biasing

The generation of the new digit d_{J+2} is on the critical path of the recurrence.[5] According to the digit-selection equation (7), this operation consists of 1) truncating the redundant form of the scaled remainder to t fractional bits, 2) adding $\frac{1}{2}$, 3) performing a $1+b+t$-bit carry-propagate addition (CPA), and 4) truncating bits past the radix point.

Reducing the delay of digit selection involves optimizing the two addition steps (since truncation is free). As the delay of the CPA is minimized by selecting the smallest t that guarantees convergence (see Section III-A), we consider optimizing the addition of $\frac{1}{2}$ required for rounding by truncation. The extra delay to add this correction might be eliminated by hoisting it into the recurrence computation: by adding $1/(2r)$ during the generation of R_{J+1} prior to scaling, digit selection can be performed simply by truncating the scaled remainder after the CPA.

This biased-remainder strategy is beneficial only if 1) the bias can be added to R_{J+1} without requiring additional delay (i.e., the maximum number of bits of a given significance is unchanged), and 2) it can be *removed*—off the critical path—prior to using the remainder in the following iteration.

Considering each term of the remainder computation (9) singles out the quadratic term $Q_J d_{J+1}^2$ as the only candidate for the addition of the rounding correction. Both the shifted remainder rR_{J+1} and the linear term $L_{J+1}d_{J+1}$ have significant bits at the bit position corresponding to $1/(2r)$. The quadratic term *also* has a significant bit at that precision; the quadratic term, however, is itself a sum of partial products, the least significant of which (Q^0) will be either Q_J or 0 (Section III-B). The recurrence equation for Q_{J+1} (11) indicates that it decreases monotonically, so $Q_J \leq Q_1 \lesssim 1/(r_1 r) \ll 1/(2r)$, and adding $1/(2r)$ simply requires setting the (otherwise zero) $-r-1$ bit of Q^0.

Removing the rounding correction from the biased remainder $R_{J+1}+1/(2r)$ involves a subtraction that completes in parallel with the digit-generation CPA with plenty of timing margin. The net effect of biasing the remainder is thus to move the addition of the rounding correction from before the CPA, parallelizing two formerly sequential steps. Note that the bias $1/(2r)$ must also be added to R_1 during initialization.

E. Range reduction of L_J

Examining carefully the definitions in Table 1, one finds that L_J is always close to +1. For $f(x) = 1/(2\sqrt{x})$, for example, $L_J = 4MxP_J \approx 4x\left(1/(2\sqrt{x})\right)^2 = 1$. The error in this approximation is worst when $J=1$. Since M and d_1 are piecewise-constant functions of x, it is clear that L_1 is a piecewise monotonic function. Thus, the extrema of L_1 over the entire domain may be found by computing L_1 at the endpoints of each monotonic interval. We find, for our specific parameters (see Table 3), that $-2^{-m+1} \leq L_1 - 1 < 2^{-m+1}$.

We thus define a new coefficient $L'_J = L_J - 1$, which may be represented with $m-1$ fewer bits than L_J. While this change reduces the overall area slightly, the primary benefit is the narrower adder required for the update of the linear coefficient, which is not kept in redundant form. During initialization, the subtraction is accomplished simply by not computing bits above the 2^{-m+1} bit.

Because it differs from L_J by a constant, L'_J is updated by the identical recurrence (10) with $L'_J \rightarrow L_J$. A change is required to the recurrence for the residual, however:

$$R_{J+1} = rR_J - L'_J d_{J+1} - Q_J d_{J+1}^2 - d_{J+1}. \qquad (12)$$

[5] We follow Lang and Antelo [9] in ordering the recurrence computation to begin with the update of the remainder ($R_J \rightarrow R_{J+1}$) and end with the computation of the next result digit ($R_{J+1} \rightarrow d_{J+2}$). This organization minimizes the logic cut by "unrolling" the recurrence.

The subtraction of d_{J+1} is accomplished at no cost by replacing known-zero bits of the least-significant partial product of $Q_J d_{J+1}^2$ (a similar trick was used to bias the remainder: see Section III-D).

IV. Implementation

In this section, we develop a high-performance radix-8 digit-by-rounding implementation employing the optimizations described in the previous section.

The functions we will compute are

$$\begin{aligned} f_1(x) &= \tfrac{1}{2x}, && x \in \left[\tfrac{1}{2}, 1\right) \cup \left[-1, -\tfrac{1}{2}\right); \\ f_2(x) &= \tfrac{1}{2\sqrt{x}}, && x \in \left[\tfrac{1}{4}, 1\right); \\ f_3(x) &= \sqrt{x}, && x \in \left[\tfrac{1}{4}, 1\right) \cup \{0\}. \end{aligned} \tag{13}$$

The input argument x is a 1.31s number,[6] normalized to fall within $\pm[\frac{1}{2}, 1)$. Normalization is handled externally (i.e., inputs are assumed to be normalized) by left shifting to eliminate all excess sign bits (for f_1) or the largest even number less than or equal to the number of excess sign bits (for $f_{2,3}$). The factor of $\frac{1}{2}$ appearing in f_1 and f_2 ensures that the result y (i.e., $f(x)$ rounded to the target precision of 31 bits) of all three functions is also representable as a 1.31s number. This scaling factor has no hardware cost—it simply affects the assumption of the radix point location. Note that $f_1(\frac{1}{2}) = f_2(\frac{1}{4}) = +1$, which is unrepresentable in a 1.31s format. This overflow case is easily caught during normalization, and the recurrence hardware need not handle it.

The parameters of our design appear in Table 3. p and r were input parameters, and the analysis described in Section III-A supplied r_1, $\mathcal{D}_1$, $\mathcal{D}$, l, m, and t. The size of the input domain ($\approx 1.25 \times 2^{32}$ possible inputs, including the choice of function) made exhaustive testing tractable. We leveraged this capability to both verify and tune the implementation. This exercise revealed that $t = 2$ was possible versus the initial value from our analysis ($t = 3$). Exhaustive testing was also important in precision minimization (see Section III-C); the determination of the specific values of q and n is discussed in Section IV-D.

A. Initialization

The cost of the simplified digit generation in the digit-by-rounding approach is the more complicated computations required to initialize the recurrence variables. Supporting multiple functions increases this complexity. Referring to Table 1, one finds that, in the worst case, the initialization of R_1 requires the computation of a term like Mxd_1^2 (within a power-of-two factor).

[6] We denote signed fixed-point numbers as $u.v$s, where u is the number of integer bits and v the number of fractional bits. These are represented as $u+v$-bit, twos-complement values. Constituent bits b_i are numbered with the log of their binary weights. The range of a $u.v$s is $[-2^{u-1}, 2^{u-1})$ and its least-significant bit (LSB) is b_{-v}. Unsigned fixed-point numbers are similarly denoted as $u.v$u with a range of $[0, 2^u)$.

Param.	Description	Value
p	Input x, result y precision	31
r_1	Radix of first digit d_1	32
$\mathcal{D}_1$	First digit set $d_1 \in \mathcal{D}_1$	$\pm[16, 32]$
r	Radix of digits $d_J, J > 1$	8
$\mathcal{D}$	Digit set $d_J \in \mathcal{D}, J > 1$	$[-7, 7]$
l	Leading bits of x for table lookup	7
m	Precision of M from lookup table	5
t	Fractional bits of $rR_J^{\mathrm{sum,car}}$ used in digit selection	2
q	Precision of recurrence variable Q (R and L have precision of $q-1$)	46
n	Precision of truncated $R^{\mathrm{sum,car}}$ used for rounding	29

Table 3: Iterative unit parameters.

$f(x)$	$\frac{1}{2x}$	$\frac{1}{2\sqrt{x}}$	$\sqrt{x}$	Format
S	x	$2Mx$	$2Mx$	1.35s
T	S/r_1	S/r_1	M/r_1	−4.40s
U	$-2d_1^2$	$-d_1^2$	$-d_1^2$	12.00s
V	d_1	$d_1/2$	$r_1 S/2$	1.31s
Q_1	0	T/r	T/r	−7.43u
L_1'	$2Td_1$	$2Td_1$	$2Td_1$	−3.39s
R_1	$V + TU$	$V + TU$	$V + TU$	1.46s

Table 4: Initialization of recurrence variables: two-step computation of the initial values Q_1, L_1', and R_1. The required precision and range of each variable is indicated by the associated fixed-point format.

By carefully factoring the expressions for Q_1, L_1', and R_1, it is possible to compute these values in two cycles using only two multipliers, one fused multiply-adder (FMA), one squarer, and a few muxes. Table 4 defines four function-dependent variables S, T, U, and V, all of which are computed in the first initialization cycle. In performing this factoring, we leveraged the fact that $M = P_1 = d_1/r_1$ for $f_{1,2}$. The initialization is completed in the following cycle.

The FMA used to compute R_1 during the second initialization cycle is a key critical path in the design. By Booth encoding U in the first initialization cycle, we halve the number of partial products that must be reduced during the second cycle. Also, the FMA does not require carry propagation since the recurrence formulation assumes a redundant representation for R_J. The terms of R_1 are reduced using the same 7:2 carry-save adder (CSA) used for remainder updates in the recurrence, significantly reducing the total area.

B. Lookup table

The lookup table takes $l = 7$ bits ($x_{-1} \ldots x_{-7}$) of the input as an index and returns d_1. For reciprocal only, the table values depend on x's sign bit x_0; the normalization, however, ensures that $x_{-1} = \overline{x_0}$, so x_0 is redundant in the index. Table 3 indicates that d_1 is in $\pm[16, 32]$—

representable as a 7.0s. d_1's sign always matches that of x, however, so the table stores instead a 6.0u from which d_1 is obtained by prepending the input's sign bit x_0. For $d_1 < 0$, then, the lookup table actually stores $d_1 + 2r_1$.

Letting $x_{\text{hi}} = 0.x_{-1} \ldots x_{-l} + 2^{-l-1}$ (a 0.8u),

$$\text{LUT}(i) = \begin{cases} \lfloor r_1 f\left(x_{\text{hi}}\right) + \frac{1}{2} \rfloor & f_1, i \geq 2^{l-1}, \\ \lfloor r_1 \left(f\left(x_{\text{hi}} - 1\right) + 2\right) + \frac{1}{2} \rfloor & f_1, i < 2^{l-1}, \\ \lfloor r_1 f\left(x_{\text{hi}}\right) + \frac{1}{2} \rfloor & f_{2,3}, i \geq 2^{l-2}. \end{cases} \tag{14}$$

In the case of reciprocal and reciprocal square root, M is simply d_1 reinterpreted as a 2.5u. For square root, M is obtained from d_1 of the reciprocal square root.

The lookup tables synthesized into logic gates account for less than 2% of the total design area, and they have a sub-cycle latency.

C. Remainder computation

The biased remainder is computed in carry-save form as the sum of nine terms: two terms $R^{0,1}$ corresponding to the scaled carry-save form of the remainder rR_J, two partial products $L^{0,1}$ from the linear term $L'_J d_{J+1}$, three partial products $Q^{0,1,2}$ from the quadratic term $Q_J d_{J+1}^2$, the new digit d_{J+1}, and the bias $1/(2r)$. Since the linear and quadratic terms and the new digit are subtracted, six of these terms must be negated. In addition, the two partial products comprising the linear term are conditionally negated based on the sign of the Booth digit used to generate them; this negation requires adding the signs of the corresponding Booth digits ($L^{0,1}_{\text{sign}}$) to the LSBs of the partial products.

The overall computation is shown in Figure 1. To minimize the critical path delay, the reduction is structured such that no column requires summing more than 7 bits (the addition of $1/(2r)$ and d_{J+1} are accommodated in the Q^0 term). The reduction requires a 7:2 CSA, which is also used to sum V with the six partial products representing TU in computing R_1 during initialization (see Section IV-A).

D. Rounding

Once $J = 10$, enough bits have been determined to round to the target precision of 31 bits. By the definition of the residual (3), the infinitely precise result may be written $f(x) = P_J + \gamma_J/(r_1 r^{J-1})$. Let z_i be the 2^i bit of P_{10}, and consider the exact value of $f(x)$ scaled by 2^{31} such that rounding occurs at the radix point:

$$-2^{31} z_0 + z_{-1} \ldots z_{-31} + \tfrac{1}{2}\left(z_{-32} + \gamma_{10}\right). \tag{15}$$

Let the twos-complement integer part $Z = -2^{31} z_0 + z_{-1} \ldots z_{-31}$ and the fractional part $w = \frac{1}{2}\left(z_{-32} + \gamma_{10}\right)$.

Rounding requires the determination of whether $|w| > \frac{1}{2}$, indicating that the correct result is closer to $Z \pm 1$ than Z.[7]

[7]Binary results of f_1 and f_2 in our domain are always non-terminating, so $\gamma_{10} \neq 0$. The result of f_3 can sometimes be represented exactly, but in that case, $z_{-32} = 0$. Consequently, $|w| \neq \frac{1}{2}$ and there is never the need to consider rounding from an exact midpoint.

q	n				
	28	29	30	31	32
45	165/19	3/24	3/25	0/25	0/25
46	165/4	3/8	3/9	0/9	0/9
47	165/1	3/1	3/2	0/2	0/3
48	165/3	3/2	3/0	0/0	0/0

Table 5: Errors in f_1, f_2 versus precision parameters: each entry a/b, where a corresponds to f_1 and b to f_2, indicates the number of inputs in each function's domain leading to an incorrect result for the indicated precisions q and n. For the values of q and n here, f_3 is always computed correctly.

Although γ_{10} can be negative, convergence implies $|\gamma_{10}| < 1$. Thus, $w > -\frac{1}{2}$, and Z never needs to be decremented. If and only if $z_{-32} = 1$ and $\gamma_{10} > 0$ (the rounding condition), $w > \frac{1}{2}$ and Z must be incremented. In this case, the exactly rounded result $y = 2^{-31}\left(Z + 1\right)$.

γ_{10} has infinite precision, so we instead use the finite-precision approximation R_{10}. The precision of the recurrence variables, including the remainder, is minimized as described in Section III-C, leading to exception cases where R_{10}'s sign differs from that of γ_{10} when $z_{-32} = 1$. We used exhaustive testing to identify these cases for different recurrence variable precisions q. We make a further optimization by using only the first n bits after the radix point of R_{10}^{sum} and R_{10}^{car} to determine the sign of R_{10}.

Table 5 lists the number of corrections required (by function) for some values of q and n. Each such correction corresponds to a particular input that results in a rounding error. The number of errors in f_1 (reciprocal) is independent of q in the range of interest. This is to be expected as it is f_2 (reciprocal square root) that requires the highest precision of q. Reducing q nearly always results in overestimating γ_{10}: the errors in f_2 result from rounding up incorrectly. Reducing n has the opposite effect: the errors in f_1 always result from mistakenly *not* rounding. Interestingly, this means that reducing n partially compensates for the errors in f_2 resulting from a low value of q. Empirically, we find that truncation of the remainder to $q = 46$ bits and the use of $n = 29$ fractional bits for sign determination of R_{10} yields a manageable number of corrections.

E. Final design

Figure 2 shows the design of the optimized iterative unit. The initialization portion of the design (Figure 2a) occupies the first three cycles: 1) table lookup of d_1 and M, 2) operation-dependent computation of S, T, U, and V, and 3) the computation of d_2 and the initial values of the recurrence variables L'_1, Q_1, and R_1. Note that table lookup is quite fast, leaving room in the first cycle for additional operations (e.g., normalization, wiring delay).

The recurrence logic (Figure 2b) behaves identically for all three functions. Muxes here select the initial values of the recurrence variables from the initialization logic when

$-L'_J d_{J+1}$	$\overline{L^0_{-3}}$	.	$\overline{L^0_{-3}}$	$\overline{L^0_{-3}}$	$\overline{L^0_{-3}}$	$\overline{L^0_{-4}}$	$\overline{L^0_{-5}}$	...	$\overline{L^0_{3-q}}$	$\overline{L^0_{2-q}}$	$\overline{L^0_{1-q}}$	$\overline{L^0_{-q}}$
	$\overline{L^1_{-3}}$	.	$\overline{L^1_{-3}}$	$\overline{L^1_{-4}}$	$\overline{L^1_{-5}}$	$\overline{L^0_{-6}}$	$\overline{L^0_{-7}}$	...	$\overline{L^1_{1-q}}$	$\overline{L^1_{-q}}$	0	$\overline{L^0_{\text{sign}}}$
$+rR_J$	R^0_{-3}	.	R^0_{-4}	R^0_{-5}	R^0_{-6}	R^0_{-7}	R^0_{-8}	...	R^0_{-q}	$\overline{L^1_{\text{sign}}}$	0	0
	R^1_{-3}	.	R^1_{-4}	R^1_{-5}	R^1_{-6}	R^1_{-7}	R^1_{-8}	...	R^1_{-q}	0	1	1
$-Q_J d^2_{J+1} - d_{J+1} + \frac{1}{2r}$	d_0	.	0	0	0	0	1	...	$\overline{Q^0_{3-q}}$	$\overline{Q^0_{2-q}}$	$\overline{Q^0_{1-q}}$	$\overline{Q^0_{-q}}$
	1	.	1	1	1	$\overline{Q^1_{-4}}$	$\overline{Q^1_{-5}}$	...	$\overline{Q^1_{3-q}}$	$\overline{Q^1_{2-q}}$	1	1
	1	.	1	1	$\overline{Q^2_{-3}}$	$\overline{Q^2_{-4}}$	$\overline{Q^2_{-5}}$	...	$\overline{Q^2_{3-q}}$	1	1	1
$= R_{J+1} + \frac{1}{2r}$	R^{sum}_0	.	R^{sum}_{-1}	R^{sum}_{-2}	R^{sum}_{-3}	R^{sum}_{-4}	R^{sum}_{-5}	...	R^{sum}_{3-q}	R^{sum}_{2-q}	R^{sum}_{1-q}	R^{sum}_{-q}
	R^{car}_0	.	R^{car}_{-1}	R^{car}_{-2}	R^{car}_{-3}	R^{car}_{-4}	R^{car}_{-5}	...	R^{car}_{3-q}	R^{car}_{2-q}	R^{car}_{1-q}	R^{car}_{-q}

Figure 1: Next remainder computation: the seven terms contributing to the computation of R_{J+1}. $R^{0,1}$ represent $R^{\text{sum,car}}_J$, $L^{0,1}$ the partial products of the linear term, $L^{0,1}_{\text{sign}}$ the signs of the respective Booth digits, and $Q^{0,1,2}$ the partial products of the quadratic term. $1/(2r) - d_{J+1}$ has been added to the $-Q^0$ term. d_0 here represents the LSB of d_{J+1}, while $1/(2r)$ is added to speed digit selection (see Section III-D).

$J = 0$. The addition of $1/(2r) - d_{J+1}$ to Q_0 is not explicitly indicated; the subsequent correction of $R^{\text{sum,car}}_{J+1}$ by subtracting $1/(2r)$ occurs using a 5-bit half adder (labeled HA*). Rounding logic is straightforward and is not shown.

V. Results

We performed a detailed synthesis study on a spectrum of designs to demonstrate the benefit of the optimizations described in this paper. Each design implements all three functions listed in Table 1, with the choice of function to compute being an additional input to the hardware module. Designs were implemented in Verilog, synthesized, and placed (i.e., timing results account for estimated wiring delay). Results are reported in technology-independent units (FO4 and gates for cycle time and area, respectively), although complete foundry technology data for a real process and cell library were used (at the worst-case process speed, voltage, and temperature). Reported cycle times include sequencing overhead (i.e., flip-flop setup time and propagation delay and clock skew and jitter); computation-only delay is approximately 8.6 FO4 lower. Area figures are reported in unit-sized nand2 gates and include actual logic only (i.e., no whitespace), although placement was performed to a target utilization of 70%. Multiple runs were performed with a range of cycle times and unconstrained area; only runs with less than 0.5 FO4 of negative timing slack were assumed to make timing. Thus, the cycle times presented indicate the highest operating frequency for the associated designs.

The baseline design is a state-of-the-art digit-by-rounding implementation except for the use of the initialization method described in Section IV-A and the precision parameters r_1, m, t, and l obtained from our analysis. As discussed in Section III-A, the use of our tuned parameters simplifies initialization significantly relative to extant designs, yielding a substantial area benefit. To first order, performance of the baseline is unchanged by our initialization method. Minimum achievable cycle time is unaffected as the recurrence is the same and additional cycles may be added freely to handle initialization complexity. While our initialization may complete more quickly, we assume that this is balanced by the larger number of bits obtained in the first digit by a design using untuned precisions. Thus, except for the area benefit, the baseline design accurately represents the current achievable performance in a digit-by-rounding design. Note that the baseline also benefits from on-the-fly conversion [15] and rounding [16].

Figure 3 shows the results of our optimizations (note the offset of the origin). We present two versions of the baseline design. The first version uses Verilog library components to implement the multipliers within the recurrence; in the second version, the linear coefficient update and linear term computation multipliers were hand-coded using a shared, radix-4 Booth-encoded digit. Although using Booth-encoded multipliers is a well-known technique, including this optimization separately illustrates an important point: even modern synthesis tools still perform poorly when optmizing arithmetic computations. Maximizing performance often requires gate-level design of the datapath components.

Optimizations were made to the baseline design beginning with the most extensive changes. First, the computation of the quadratic term was implemented using our square-multiple technique (Section III-B). Second, we used the range-reduced L'_J instead of L_J. Third, we minimized recurrence variable precision, adding the necessary corrections to maintain exact rounding (Section III-C). Finally, we introduced the biased remainder (Section III-D), which reduces the rounding step of digit selection to a truncation.

Each successive optimization results in an area-delay curve closer to the origin, allowing one to make the design smaller or faster or both. The upper-left endpoint of a curve represents the shortest possible cycle time achievable for that design. The combined effect of our optimizations is a

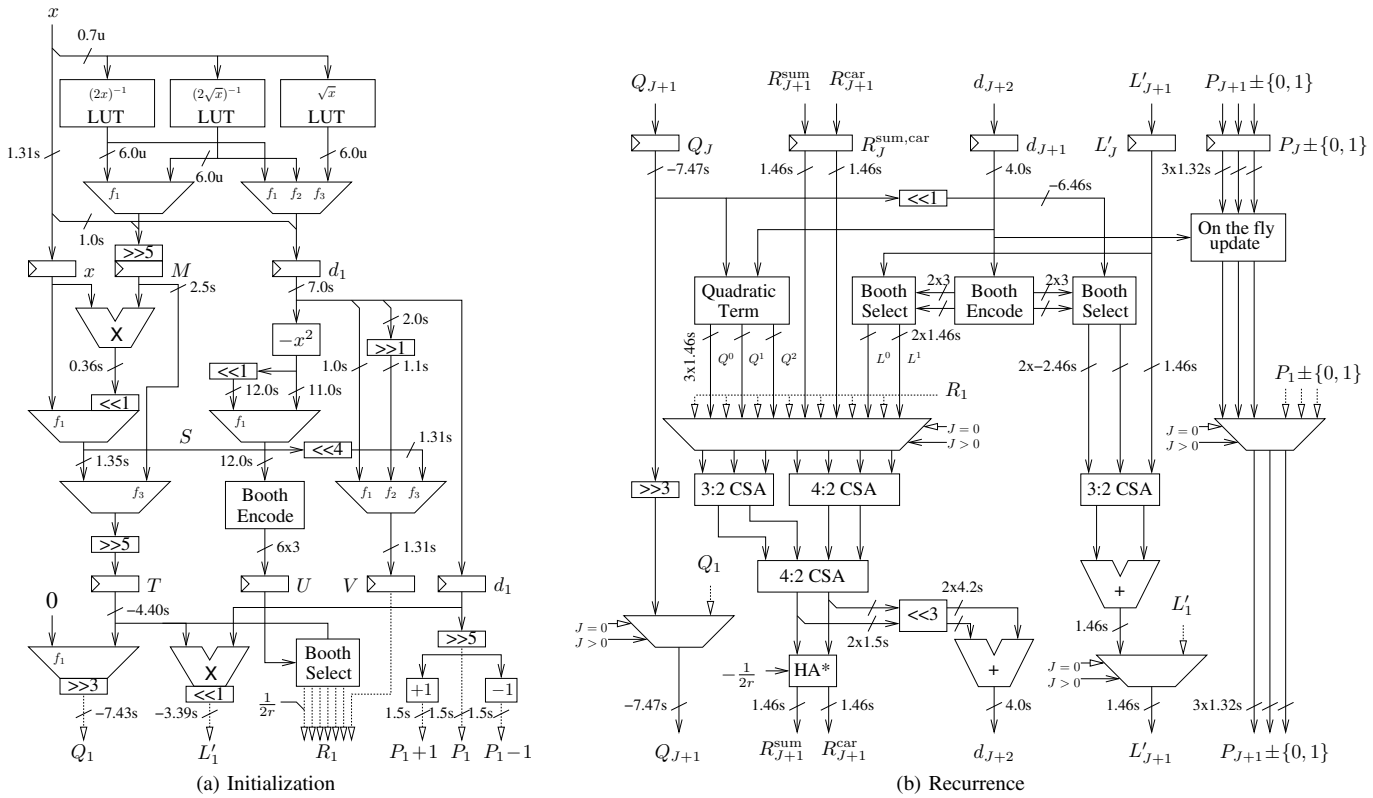

Figure 2: Detailed design: wires are annotated with their fixed-point value format. Shift operations are shown explicitly, although they simply represent a reinterpretation of the radix point location. Loss of integer or fractional bits across a non-shift operation indicates truncation. Added fractional bits indicate zero padding; signed integer bits, sign extension; and unsigned integer bits, zero extension.

9.3% decrease in minimum cycle time (10% reduction in critical path delay, accounting for the constant sequencing overhead). At the other end of the curve, where the timing pressure is much lower, the results become more chaotic (an artifact of the synthesis algorithms). Even so, the fully optimized design is over 15% smaller than the original design at the same cycle time.

The optimization to the quadratic term computation delivers the greatest benefit, one that persists even at long cycle times. The individual importance of the other optimizations is less significant although their cumulative benefit is still evident. These latter optimizations have much more impact under timing pressure: together, they are responsible for about 60% of the total decrease in cycle time.

Examining the timing paths in more detail reveals that the optimizations may be even more effective than indicated. In the unoptimized designs, logic paths in the recurrence itself limit the minimum cycle time. In the final design, the critical path is actually the delivery of the rounded result to the output pins. We reserve one-quarter cycle for output paths to account for time to deliver the result somewhere useful. The simple expedient of capturing the rounded result in flip-flops inside the iterative unit decreases the minimum cycle time to 30.6 FO4. Even in this design, the critical path is not through the recurrence, but through the initialization. Focusing additional effort on optimizing that logic could thus enable an even lower cycle time.

At this point, we can compare the critical path of a digit-by-rounding design to that of SRT, the most commonly used recurrence method. Both designs must necessarily update the remainder each iteration; furthermore, the recurrence equations to do so are the same in each case. The difference in the designs is thus solely in the digit-selection computation, which depends on some leading bits of the new remainder.

For our radix-8 design, digit selection involves only a six-bit addition on the leading bits of the redundant remainder. In contrast, a radix-4 SRT design for computing the same functions requires an eight-bit adder on leading bits of the redundant remainder followed by an 11-bit table lookup [9]. While this computation is overlapped with the update of the remainder (by performing a faster, limited-precision remainder update to compute the table index), the same technique can be applied to the digit-by-rounding approach. At equivalent radix and digit set redundancy, we thus expect

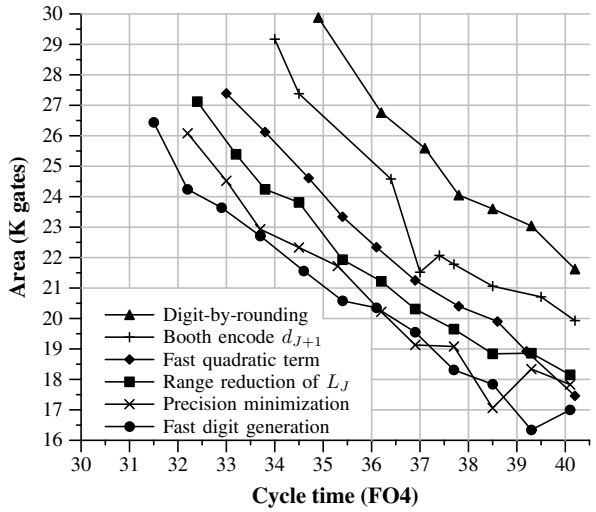

Figure 3: Synthesis results: area versus delay in technology-independent units as optimizations are added cumulatively to a baseline digit-by-rounding design.

an SRT design to require *more* bits of assimilated remainder than a digit-by-rounding design in order to perform digit selection. Even were the number required the same, however, SRT fundamentally requires a table lookup after the partial assimilation of the redundant remainder, while an optimized digit-by-rounding approach requires only the latter.

VI. Conclusion

Reciprocals, reciprocal square roots, or square roots occur in most nontrival numerical computations. When exact rounding, short cycle time, and minimum area are required, digit-recurrence algorithms are favored. To achieve low overall latency, higher radices are required in these algorithms, but radices exceeding four make table-based digit selection (e.g., SRT) both area-intensive and difficult to achieve at short cycle times.

By employing a digit-by-rounding algorithm, normally associated with higher radices, it is possible to shift some of the computational burden from the recurrence to the initialization, where extra latency is not multiplied by the iteration count. We find that for the selection of radix-8 digits, the critical recurrence path of the digit-by-rounding approach is demonstrably superior to that of SRT, the traditional algorithm for achieving short cycle times.

We have presented a compact initialization method based on our tuning of design parameters and several optimizations to the digit-by-rounding algorithm and the recurrence computation in particular. The net effect of these optimizations is to decrease both cycle time and area by over 10%, pushing the design critical path out of the recurrence logic. Even higher operating frequencies appear to be possible with further optimization of the initialization and rounding stages. Finally, we note that several of the optimization techniques we have presented apply directly to other digit-recurrence algorithms, including SRT.

Acknowledgments

The authors wish to thank David Borhani, Amy Perry, Brannon Batson, Jeffrey Kuskin, and John Salmon for their detailed feedback on earlier drafts of this work. We are grateful to Li-Siang Lee for building the environment that we used to obtain our synthesis results. Finally, we appreciate the helpful comments offered by the anonymous reviewers.

References

[1] C. Freiman, "Statistical analysis of certain binary division algorithms," *Proc. IRE*, vol. 49, pp. 91–103, Jan. 1961.

[2] S. Obermann and M. Flynn, "Division algorithms and implementations," *IEEE Trans. Comput.*, vol. 46, pp. 833–854, Aug. 1997.

[3] J. Coke *et al.*, "Improvements in the Intel® Core 2 Penryn™ processor family architecture and microarchitecture," *Intel Technology J.*, vol. 12, pp. 179–192, Oct. 2008.

[4] J. Fandrianto, "Algorithm for high speed shared radix 8 division and radix 8 square root," in *Proc. 9th Symp. Comput. Arithmetic*, Sep. 1989, pp. 68–75.

[5] M. Ercegovac, "A higher-radix division with simple selection of quotient digits," in *Proc. 6th Symp. Comput. Arithmetic*, 1983, pp. 94–98.

[6] E. Krishnamurthy, "On range-transformation techniques for division," *IEEE Trans. Comput.*, vol. C-19, pp. 157–160, Feb. 1970.

[7] E. Antelo, T. Lang, and J. Bruguera, "Computation of $\sqrt{x/d}$ in a very high radix combined division/square-root unit with scaling and selection by rounding," *IEEE Trans. Comput.*, vol. 47, pp. 152–161, Feb. 1998.

[8] P. T. P. Tang *et al.*, "Tight certification techniques for digit-by-rounding algorithms with application to a new $1/\sqrt{x}$ design," in *Proc. 20th Symp. Comput. Arithmetic*, July 2011.

[9] T. Lang and E. Antelo, "Radix-4 reciprocal square-root and its combination with division and square root," *IEEE Trans. Comput.*, vol. 52, pp. 1100–1114, Sep. 2003.

[10] T. Lang and P. Montuschi, "Very-high radix combined division and square root with prescaling and selection by rounding," in *Proc. 12th Symp. Comput. Arithmetic*, Jul. 1995, pp. 124–131.

[11] C. Iordache and D. Matula, "On infinitely precise rounding for division, square root, reciprocal and square root reciprocal," in *Proc. 14th Symp. Comput. Arithmetic*, Apr. 1999, pp. 233–240.

[12] T. Lang and J.-M. Muller, "Bounds on runs of zeros and ones for algebraic functions," in *Proc. 15th Symp. Comput. Arithmetic*, Jun. 2001, pp. 13–20.

[13] M. Parks, "Number-theoretic test generation for directed rounding," in *Proc. 14th Symp. Comput. Arithmetic*, Apr. 1999, pp. 241–248.

[14] D. Stehlé, V. Lefèvre, and P. Zimmermann, "Searching worst cases of a one-variable function using lattice reduction," *IEEE Trans. Comput.*, vol. 54, pp. 340–346, Mar. 2005.

[15] M. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Trans. Comput.*, vol. C-36, pp. 895–897, Jul. 1987.

[16] M. Ercegovac and T. Lang, "On-the-fly rounding [computing arithmetic]," *IEEE Trans. Comput.*, vol. 41, pp. 1497–1503, Dec. 1992.