2011 20th IEEE Symposium on Computer Arithmetic

Tight Certification Techniques for Digit-by-Rounding Algorithms with Application to a New $1/\sqrt{x}$ Design

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

***Abstract*—Digit-by-rounding algorithms enable efficient hardware implementations of algebraic functions such as the reciprocal, square root, or reciprocal square root, but certifying the correctness of such algorithms is a nontrivial endeavor. Traditionally, sufficient conditions for correctness are derived as closed-form formulae relating key design parameters. These sufficient conditions, however, often prove stricter than necessary, excluding correct and efficient designs. In this paper, we present a rigorous, computer-aided method for correctness certification that better approximates the necessary conditions, lowering the risk of rejecting correct designs. We also present two specific applications of this method. First, when applied to a conventional digit-by-rounding reciprocal square root design, our method enabled a fourfold reduction in lookup table size relative to the minimum dictated by a standard sufficient condition. Second, our method certified the correctness of a novel reciprocal square root design that we developed to parallelize two computational steps whose sequential execution lies on the critical path of conventional designs. The difficulty in deriving closed-form sufficient conditions to ascertain this design's correctness provided the original motivation for development of the new certification method.**

***Keywords*-Verification, error analysis, digit recurrence, selection by rounding**

I. Introduction

Digit recurrences belong to a class of computer-arithmetic algorithms suitable for implementing certain algebraic functions, such as the reciprocal or square root functions, in hardware [1]. This class of algorithms can be viewed as a generalization of the pencil-and-paper long-division algorithm. In long division, one initializes an iterative process by setting the remainder to the dividend. At every iteration, a result digit is generated based on both the remainder and the divisor, and the remainder is updated using the new digit. Provided the digits are generated correctly and computations are performed without mistakes, the partial quotients obtained at progressive iterations converge to the exact quotient.

In the broader digit-recurrence framework, the familiar decimal digit system with a base of 10 and exactly 10 possible values (0 to 9) for each digit is generalized to digit systems with other bases, also called radices. In addition, the number of possible values for each digit can exceed the value of the radix, and some digits may be negative. The remainder is defined differently for each algebraic function, but always serves to guide digit generation. A digit-recurrence algorithm using a radix-2^k digit system generates k bits of result per iteration. Consequently, for a given precision requirement, the number of iterations required decreases as the radix increases. On the other hand, the computational cost of digit generation increases with the radix.

In most digit-recurrence algorithms, digit generation requires knowing both the remainder and the input value, and involves either nontrivial operations such as multiplications, or area-intensive methods such as two-dimensional table lookups; these operations become particularly expensive at high radices. The subclass of digit-by-rounding algorithms, on the other hand, incorporates the input value in a more sophisticated initialization process; digit generation at subsequent iterations requires only knowledge of the remainder and involves only a round-to-integer computation (see Section II for a more detailed description). The area and latency increase that results from this enhanced initialization process can be kept moderate [2], [3] so that the more efficient digit-generation mechanism leads to a net performance gain. In this context, digit-by-rounding algorithms make high-radix recurrences more practical.

Correctness certification of any design, and a digit-by-rounding design in particular, is simple if exhaustive testing can be done in a reasonable time. Unfortunately, this is infeasible when the input space is large (e.g., double-precision floating-point with 53 significant bits). Without exhaustive testing, the usual correctness-certification approach for digit-by-rounding algorithms is to derive sufficient conditions in the form of closed-form constraints on various design-specific parameters [2], [3]. The reciprocal square root design described in [3], for example, has been proven correct if it satisfies the condition

$$|2M(x)\sqrt{x} - 1| < \frac{\sqrt{x}}{8r}\left(\frac{1}{4} - \frac{23}{32r}\right)$$

for all x in the domain $[1/4, 1)$, where r is the radix and $M(x)$ is a value that the algorithm produces for each value x. This approach to correctness certification is analytic and

*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.2011.29

global (i.e., constraints are applied to the entire domain).

There are two general problems with this approach. The first is that the sufficient conditions are often overly conservative. The closed-form expressions generally specify smooth and regularly shaped regions in two-dimensional space that correspond to subsets of the entire space of correct designs. In an actual digit-by-rounding design, there are many parameters and recurrence variables germane to correctness, rendering the relevant design space high dimensional. Moreover, many variables in digit-recurrence algorithms are discontinuous across the input domain. The $M(x)$ function above, for example, typically has tens or even hundreds of discontinuities over the input domain. Hence the entire space of correct designs is high dimensional and irregular. As a result, the sufficient conditions derived by a global analysis may be more stringent than necessary, and the space of designs certified correct this way may omit some efficient and legitimate options. We will show an example of this in Section IV.

The second problem is that closed-form sufficient conditions can be difficult to derive in the first place. The derivation shown in [3] or [4], for example, relies on intimate knowledge of specific details of the algorithms, as well as on their underlying regularity. Given an irregular or unconventional design, it may be difficult to obtain sufficient conditions in closed form. The new reciprocal square root design we describe in Section V is such an example, which in fact motivated the work here.

In this paper, we develop a new correctness-certification method that is local and numerical in nature. Given a specific digit-by-rounding design, we first subdivide the input domain into small subdomains, within which fluctuations of key iteration variables are small. We then apply computational interval analysis on each subdomain to track and bound the numerical ranges of those variables as the iterations proceed. As we will demonstrate, this approach significantly reduces the risk of being overly conservative. In what follows, we will first review digit recurrence in general and digit-by-rounding algorithms in particular. Our local numerical certification method is then presented as a two-step process. We will demonstrate the effectiveness of this method on a conventional reciprocal square root design. An unconventional reciprocal square root design is then presented and its correctness certified. We will discuss possible generalizations of our certification method in the conclusion.

II. Review of Digit-by-Rounding Recurrence

Let $f(x)$ be an algebraic function to be implemented as a computer-arithmetic operation. For example, $f(x) = \sqrt{x}$ for $x \in [1/4, 1)$. As a generalization of the decimal digit system suitable for representing the range of $f(x)$, we introduce an integral radix $r \geq 2$ and an integral digit set $\mathcal{D} = \{a, a+1, \ldots, b\}$, where a and b are arbitrary integers satisfying $b - a \geq r - 1$. The radix r is for all practical situations chosen to be an integral power of two: $r = 2^k$. Conceptually, a digit-recurrence algorithm that implements $f(x)$ generates iteratively, for a given input argument x, a sequence of digits $d_1, d_2, \ldots, d_j \in \mathcal{D}$, such that the partial result $d_1/r + d_2/r^2 + \cdots + d_J/r^J$ converges to $f(x)$ as $J \to \infty$. It is quite common to define the first digit differently. More precisely, we consider a radix r_1 with a digit set $\mathcal{D}_1 = \{a_1, a_1 + 1, \ldots, b_1\}$ for the first digit and define the partial result to be $P_J \stackrel{\text{def}}{=} \sum_{j=1}^{J} \frac{d_j}{r_1 r^{j-1}}$, where $d_1 \in \mathcal{D}_1$ and $d_j \in \mathcal{D}$ for $j \geq 2$. There is some freedom in how an algorithm formulates the *remainder* R_J associated with the Jth iteration. The crucial requirements are that the remainder can be used to guide digit generation and that it be computable exactly and efficiently by an update procedure at every iteration.

Convergence, as will be shown later on, is characterized more naturally not by the remainder but by a quantity that measures directly how well P_J approximates $f(x)$. We call this quantity the *residual*, $\gamma_J \stackrel{\text{def}}{=} r_1 r^{J-1}(f(x) - P_J)$. Table 1 summarizes the notation we have defined thus far.

$f(x)$	function to be implemented
$d_1, d_2, \ldots$	digits
$r_1 = 2^{k_1}$	radix for d_1
$\mathcal{D}_1 = \{a_1, a_1 + 1, \ldots, b_1\}$	digit set for d_1
$r = 2^k$	radix for d_j, $j \geq 2$
$\mathcal{D} = \{a, a+1, \ldots, b\}$	digit set for d_j, $j \geq 2$
$P_J = \sum_{j=1}^{J} d_j/(r_1 r^{j-1})$	partial result
R_J	Jth remainder
$\gamma_J = r_1 r^{J-1}(f(x) - P_J)$	Jth residual

Table 1: Notation for digit-recurrence algorithms.

A digit-recurrence algorithm is described below.

- Generate $d_1 \in \mathcal{D}_1$, and the first remainder R_1.
- For $J = 1, 2, \ldots, J_{\text{end}} - 1$, iterate
  - $d_{J+1} = \texttt{generate-digit}(R_J, \ldots)$, $d_{J+1} \in \mathcal{D}$
  - $R_{J+1} = \texttt{update-rem}(R_J, d_{J+1}, \ldots)$
  - Round $P_{J_{\text{end}}}$ to the final result.

Methods for rounding $P_{J_{\text{end}}}$ to the destination precision are well known [5], [6] and are not the focus of our paper. We will omit this step in our exposition from this point onwards.

One class of digit-recurrence algorithms generates the digits d_{J+1} via table lookup based on some leading bits of both the remainder R_J and x. Consequently, table-size considerations usually result in a moderate radix r, typically $r \leq 4$. To avoid this limitation, an alternative

class of digit-recurrence algorithms generates the digits via arithmetic operations. To see how this may work, consider two properties of the residual that follow immediately from its definition and that of P_J:

- $f(x) - P_J = \gamma_J/(r_1 r^{J-1})$, and
- $\gamma_{J+1} = r\gamma_J - d_{J+1}$.

The first indicates that $f(x) - P_J$ converges to zero if $|\gamma_J|$ is bounded by a constant for all J. The second suggests that $d_{J+1} \leftarrow \texttt{round-to-nearest-integer}(r\gamma_J)$ is a natural strategy to keep the residuals bounded. For example, if the digit set $\mathcal{D} = \{-\frac{r}{2}, -\frac{r}{2}+1, \ldots, \frac{r}{2}\}$ is used and d_1 is somehow chosen so that $|\gamma_1| \leq 1/2$, then this digit-by-rounding recipe for the subsequent d_js will lead to $|\gamma_J| \leq 1/2$ and $|d_J| \leq r/2$ for all $J \geq 2$, implying convergence of P_J to $f(x)$. It thus seems that digit generation based on rounding-to-integer is a viable approach. There is, however, a major technical problem with this idea: the residual is not computable exactly as it requires the knowledge of $f(x)$.

Even though the residual cannot be computed exactly, it can be expressed as a product of an exactly computable factor of finite precision and another factor that may be approximated. We illustrate this with the example $f(x) = \sqrt{x}$.

$$\begin{aligned}
\gamma_J &= r_1 r^{J-1}(\sqrt{x} - P_J), \\
&= r_1 r^{J-1}(x - P_J^2)\frac{1}{\sqrt{x} + P_J}, \\
&\approx r_1 r^{J-1}(x - P_J^2)\frac{1}{2\sqrt{x}}, \quad \text{assume } P_J \approx \sqrt{x}, \\
&\approx r_1 r^{J-1}(x - P_J^2)M(x), \quad \text{where } M(x) \approx \tfrac{1}{2\sqrt{x}}, \\
&\stackrel{\text{def}}{=} R_J, \quad \text{exactly computable remainder.}
\end{aligned}$$

For this example of $f(x) = \sqrt{x}$, a digit-by-rounding algorithm may first generate $M(x)$ given an input x. Because $M(x) \approx 1/(2\sqrt{x})$, rounding $2r_1 x M(x)$ to an integer offers a viable choice for the first digit d_1. With $M(x)$ and d_1, the algorithm computes the first remainder R_1 exactly and starts the iterative process. For historical reasons, $M(x)$ is called a scale factor. For brevity, we will use M whenever we need not emphasize its dependence on x. This technique of using a scale factor to define a (computable) remainder that approximates the residual is widely applicable. Table 2 illustrates this for some functions of interest.

Provided the remainder approximates the residual well enough, digit generation can take the form

$$d_{J+1} \leftarrow \texttt{round-to-nearest-integer}(rR_J).$$

In practical implementations of digit-recurrence algorithms [1], the remainders R_J are represented in carry-save format. Hence, a round-to-integer operation at every iteration will incur too long a latency. The standard solution to this problem is to perform only an approximate round-to-integer operation, which is typically an exact round-to-integer operation on a few leading bits of both the sum and carry variables. We also note that the round-to-integer operation need not round to the nearest integer. The appropriate rounding is tied to the choice of the digit set. The round-to-nearest operation is most consistent with a symmetric digit set, $-a = b$, which is a natural choice from a performance point of view.

$f(x) =$	$\sqrt{x}$ $1/4 \leq x < 1$	$1/(2\sqrt{x})$ $1/4 \leq x < 1$	$1/(2x)$ $1/2 \leq x < 1$
$\frac{\gamma_J}{r_1 r^{J-1}} =$	$\sqrt{x} - P_J$ $\frac{x - P_J^2}{\sqrt{x} + P_J}$	$\frac{1}{2\sqrt{x}} - P_J$ $\frac{(1 - 4xP_J^2)/2}{\sqrt{x}(1 + 2\sqrt{x}P_J)}$	$\frac{1}{2x} - P_J$ $\frac{1 - 2xP_J}{2x}$
$M \approx$	$\frac{1}{2\sqrt{x}}$	$\frac{1}{2\sqrt{x}}$	$\frac{1}{2x}$
$\frac{R_J}{r_1 r^{J-1}} \stackrel{\text{def}}{=}$	$M(x - P_J^2)$	$M(1/2 - 2xP_J^2)$	$M(1 - 2xP_J)$
$d_1 \approx$	$2r_1 M x$	$r_1 M$	$r_1 M$

Table 2: Definitions of computable, residual-approximating remainders using a scale factor M. The scale factor is also used to generate the first digit.

We can now formulate digit-by-rounding algorithms as follows:

- Given x, obtain a scale factor M
- Use M and x to get d_1 and the first remainder R_1
- For $J = 1, 2, \ldots, J_{\text{end}} - 1$, iterate
  - $d_{J+1} = \texttt{approx-round-int}\,(r \times R_J)$
  - $R_{J+1} = \texttt{update-rem}(M, R_J, d_{J+1}, \ldots)$.

For a given function $f(x)$ to be implemented, a digit-by-rounding algorithm has the flexibility to choose the radices and digit sets, define the remainder, design the M factor generation procedure, and specify the `approx-round-int` operation. The correctness of a specific design is dependent on both the choices made for each of these individual parameters as well as their relationship with each other.

III. Correctness Certification

We now present our method of correctness certification for digit-by-rounding algorithms. Let us first characterize correctness precisely. We first show that $P_J \to f(x)$ if and only if the residuals γ_J lie in the interval $[a/(r-1), b/(r-1)]$ for all $J = 1, 2, \ldots$. The "if" part is straightforward. If γ_J lies in $[a/(r-1), b/(r-1)]$ for all J, then because $f(x) - P_J = \gamma_J/(r_1 r^{J-1})$, $f(x) - P_J \to 0$.

Conversely, suppose that $P_J \to f(x)$, then $f(x) = \sum_{j=1}^{\infty} d_j/(r_1 r^{j-1})$, where $d_1 \in \mathcal{D}_1$ and $d_j \in \mathcal{D}$ for $j \geq 2$. That is, for any $J \geq 1$, $\gamma_J = \sum_{j=1}^{\infty} d_{J+j}/r^j$. Since $a \leq d_{J+j} \leq b$ for all j as long as $J \geq 1$, we must have

$$a/(r-1) \leq \gamma_J \leq b/(r-1)$$

for all $J \geq 1$. This completes the proof.

Based on this understanding of convergence, we define that an algorithm is correct if and only if

1) the digits generated are within bounds, that is, $d_1 \in \mathcal{D}_1$ and $d_j \in \mathcal{D}$ for $j \geq 2$, and
2) γ_J is within the bound of $[a/(r-1), b/(r-1)]$ for $J = 1, 2, \ldots, J_{\text{end}}$.

For a given x in the input interval, the algorithm generates M and defines a *trajectory* $(d_1, \gamma_1, d_2, \gamma_2, \ldots, d_{J_{\text{end}}}, \gamma_{J_{\text{end}}})$. The digits are generated explicitly, and the residuals, though not generated explicitly, are precisely defined by the digits. For any iteration number J, $1 \leq J \leq J_{\text{end}}$, we call the vector $(x, M, d_1, \gamma_1, \ldots, d_J, \gamma_J)$ a *state*. Assessing the numerical ranges of the digits and residuals in all trajectories is the key to certifying an algorithm's correctness. Our approach makes this assessment in two steps.

- *State-Space Covering:* This can be described as a local approach. The basic idea is to perform analysis on small intervals in which the ranges of M as well as a first few digits are limited. Since in practice the range of M on the entire input domain consists only of a moderate number (typically in the hundreds) of discrete values, we focus on small intervals where there is only one possible value of M and of a first few digits as well. More precisely, for each of several early iterations $J = 1, 2, \ldots, J_{\text{launch}}$ (typically $J_{\text{launch}} = 3$ or 4), we construct a covering $\mathcal{C}_J$ consisting of a finite set of *tiles* of the form $\mathcal{T} = ([u, v], m, s_1, s_2, \ldots, s_J)$ so that for all states $(x, M, d_1, \gamma_1, \ldots, d_J, \gamma_J)$ we can find a tile in $\mathcal{C}_J$ such that

  $$x \in [u, v], \quad M = m, \quad \text{and } d_j = s_j \text{ for } j = 1, \ldots, J.$$

  Because the residual is of the form

  $$r_1 r^{J-1}(f(x) - \sum_{j=1}^{J} s_j/(r_1 r^{j-1}))$$

  (with digits fixed), its numerical range on an interval $[u, v]$ can be determined easily for monotonic $f(x)$, which is the case in practice. A covering $\mathcal{C}_J$ hence allows us to easily bound, via enumeration, the ranges of digits as well as residuals of the entire state space at iteration J.
- *Bound Propagation:* This can be described as a numerical-analysis approach. Upon completion of the state-space covering step, we have constructed in $\mathcal{C}_{J_{\text{launch}}}$ a number of tiles each containing bounds on the possible ranges of the residuals and digits. The basic idea in this step is to use interval analysis as well as other numerical methods to propagate those bounds through the iterations from J_{launch} to J_{end}. As a result, for each of the finite number of tiles in $\mathcal{C}_{J_{\text{launch}}}$, we can establish a sequence of bounds

  $$[d_j^-, d_j^+], \quad \text{and} \quad [\gamma_j^-, \gamma_j^+] \quad \text{for } J_{\text{launch}} < j \leq J_{\text{end}}.$$

  These bounds are such that, for any trajectory that emerges from this tile,

  $$d_j \in [d_j^-, d_j^+], \quad \gamma_j \in [\gamma_j^-, \gamma_j^+], \quad J_{\text{launch}} < j \leq J_{\text{end}}.$$

The state-space covering technique produces very tight bounds, but its workload grows exponentially with the number of iterations. When carried out to the last iteration J_{end}, the amount of work required will be of the same order of magnitude as that of exhaustive testing. On the other hand, bound propagation on a single tile for one step can be completed quickly. Moreover, the bounds can be tight provided the domain intervals from a tile are small. The setting of J_{launch} is a parameter that provides tradeoffs between number of tiles and the widths of resulting domain intervals. We found that a choice of $J_{\text{launch}} = 3$ or 4 makes it feasible to conduct low-miss-rate interactive exploration of the design space for certifiably correct algorithms. We now explain state-space covering and bound propagation in detail.

A. State-Space Covering

Let $\mathcal{X}$ be the input space in question. This always corresponds to some binary intervals such as $1/4 \leq x < 1$ or $1/2 \leq x < 1$ due to normalization. In addition, $\mathcal{X}$ is in fact a discrete set, consisting only of integer multiples of ϵ, the unit in the last place (ULP), thus

$$\mathcal{X} = \{x_{\min}, x_{\min} + \epsilon, x_{\min} + 2\epsilon, \ldots, x_{\max}\}.$$

Given any set $\mathcal{Y}$ bounded such that $\mathcal{Y} \subseteq [x_{\min}, x_{\max}]$, we define Expand($\mathcal{Y}$) $= [u, v]$ as the smallest interval containing $\mathcal{Y}$ (that is, $\mathcal{Y} \subseteq [u, v]$) where u and v are elements of $\mathcal{X}$.

Let $\mathcal{I}_d$ be defined as an interval such that $\mathcal{I}_d \supseteq$ `approx-round-int`$^{-1}(\{d\})$. In other words, $d =$ `approx-round-int`(rR) implies $rR \in \mathcal{I}_d$. This set is dependent on the specification of the underlying function `approx-round-int`. A common specification for an approximate round-to-nearest function is as follows. The remainder R is represented as two components in a redundant carry-save format, $R = R^{\text{sum}} + R^{\text{car}}$, implying $rR = rR^{\text{sum}} + rR^{\text{car}}$. Each of the two terms is first truncated at the 2^{-t} position, where t is an implementation choice. Their sum is then rounded to an integer. When $d =$ `approx-round-int`(rR),

$$d = \lfloor S + C + 1/2 \rfloor, \qquad \text{where}$$

$$S = 2^{-t}\lfloor 2^t r R^{\text{sum}} \rfloor, \qquad C = 2^{-t}\lfloor 2^t r R^{\text{car}} \rfloor.$$

One can show that $d =$ `approx-round-int`(rR) implies

$$d - 1/2 \leq rR < d + 1/2 + 2^{-t}.$$

The definition $\mathcal{I}_d \stackrel{\text{def}}{=} d + [-1/2, 1/2 + 2^{-t}]$ is thus appropriate. Our analysis relies only on

$$\mathcal{I}_d \supseteq \texttt{approx-round-int}^{-1}(\{d\})$$

and $\mathcal{I}_d = d + \mathcal{I}_0$, not on the exact specification of the `approx-round-int` function.

Consider next the generation of M. M is generated by mapping an algorithm-specific number of leading bits of an input x to an M value. In functional notation, we have

$$\mathcal{X} \xrightarrow{F_{\text{M}}} \{m_1, m_2, \ldots, m_L\}.$$

We construct a covering of the input space by a finite set of intervals: $\mathcal{X}_i \stackrel{\text{def}}{=} \text{Expand}(F_{\text{M}}^{-1}(\{m_i\})), \quad i = 1, 2, \ldots, L$. That is, we take successive inverse images of each possible value of M and expand them to intervals with end points in $\mathcal{X}$ itself. $\mathcal{X}_i$ is thus of the form $[u_i, v_i]$. If m_i is generated when the input is x, then we must have $x \in \mathcal{X}_i$. In other words, $\cup_i \mathcal{X}_i \supseteq \mathcal{X}$. The construction $\mathcal{C}_0 \stackrel{\text{def}}{=} \{(\mathcal{X}_i, m_i)\}$ thus covers the initial state space $\{(x, M)\}$. We note also that if the M values vary monotonically in x (which is often the case), the $\mathcal{X}_i$ values are mutually disjoint.

From $\mathcal{C}_0$, we generate $\mathcal{C}_1$ as follows. The first digit, d_1, is generated based on the input x and the M value. Consider the case when $M = m_i$ and denote by g_i the first-digit generation function restricted to the condition $M = m_i$. Construct $\mathcal{X}_{i,d}$ so that $\mathcal{X}_{i,d} \supseteq \mathcal{X}_i \cap g_i^{-1}(\{d\})$ whenever the right-hand side is nonempty. In practice, $\mathcal{X}_i \cap g_i^{-1}(\{d\})$ is nonempty only for a finite number of d values.

Take for example the case of $f(x) = \sqrt{x}$ (c.f. Table 2). If the first digit is generated by the approximate rounding-to-integer of $2r_1Mx$, then $g_i(x) = \texttt{approx-round-int}\,(2r_1m_ix)$. Consequently,

$$\begin{aligned} g_i^{-1}(\{d\}) &= \texttt{approx-round-int}^{-1}(\{d\})/(2r_1m_i) \\ &\subseteq \mathcal{I}_d/(2r_1m_i) \\ &\subseteq (d + [-1/2, 1/2 + 2^{-t}])/(2r_1m_i), \end{aligned}$$

if round-to-nearest is used. Recall that $\mathcal{X}_i$ is of the form $[u_i, v_i]$. The intersection $\mathcal{X}_i \cap g_i^{-1}(\{d\})$ will be empty whenever $(d-1)/(2r_1m_i) \geq v_i$ or $(d+1)/(2r_1m_i) \leq u_i$. We define

$$\mathcal{X}_{i,d} \stackrel{\text{def}}{=} \text{Expand}(\mathcal{X}_i \cap (\mathcal{I}_d/(2r_1m_i))).$$

For any state (x, M, d, γ_1) with $M = m_i$ and $\gamma_1 = r_1 f(x) - d_1$, we must have $x \in \mathcal{X}_i$ and $x \in g_i^{-1}(\{d\})$, implying $x \in \mathcal{X}_{i,d}$. The definition

$$\mathcal{C}_1 \stackrel{\text{def}}{=} \{(\mathcal{X}_{i,d}, m_i, d) | \mathcal{X}_{i,d} \text{ nonempty}\}$$

thus provides a covering for the state space at iteration $J = 1$.

We emphasize that the construction of $\mathcal{C}_1$ considers all possible integer values of d, not just those within the set $\mathcal{D}_1$. It is possible that for some designs, one may obtain a nonempty $\mathcal{X}_i \cap g_i^{-1}(\{d\})$ for some d outside of $\mathcal{D}_1$. This will be a case where the design cannot be certified correct.

Now that $\mathcal{C}_1$ is constructed, the following procedure produces $\mathcal{C}_{J+1}$ from $\mathcal{C}_J$, $J \geq 1$. Let $\mathcal{C}_J$ be a cover for the state space at iteration J. $\mathcal{C}_J$ thus consists of a finite number of tiles $\mathcal{T} = (\mathcal{X}_{i,d_1,d_2,\ldots,d_J}, m_i, d_1, d_2, \ldots, d_J)$ where $(m_i, d_1, d_2, \ldots, d_J)$ is unique for each tile. For each tile, consider the remainder function, R_{J,m_i,γ_J}, restricted to

$$M = m_i \text{ and } P_J = d_1/r_1 + d_2/(r_1 r) + \cdots + d_J/(r_1 r^{J-1}).$$

Given any state $(x, M, d_1, \gamma_1, d_2, \ldots, d_J, \gamma_J)$,

$$d = \texttt{approx-round-int}\,(rR_{J,m_i,P_J})$$

implies $rR_{J,m_i,P_J} \in \mathcal{I}_d$, and $x \in R_{J,m_i,P_J}^{-1}(\mathcal{I}_d/r)$. We construct $\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d}$ so that

$$\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d} \supseteq \mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J} \cap R_{J,m_i,P_J}^{-1}(\mathcal{I}_d/r).$$

It is clear that then

$$\mathcal{C}_{J+1} \stackrel{\text{def}}{=} \{(\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d}, m_i, d_1, d_2, \ldots, d)\}$$

for all nonempty $\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d}$ is a cover for the $(J+1)$th iteration.

To again illustrate with the $f(x) = \sqrt{x}$ example, we have

$$R_{J,m_i,P_J} = r_1 r^{J-1} m_i (x - P_J^2),$$

leading to

$$R_{J,m_i,P_J}^{-1}(\mathcal{I}_d/r) = \frac{\mathcal{I}_d}{r_1 r^J m_i} + {P_J}^2.$$

We can thus define $\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d}$ by

$$\text{Expand}\left(\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J} \cap (\mathcal{I}_d/(r_1 r^J m_i) + {P_J}^2)\right).$$

The procedure of generating $\mathcal{C}_{J+1}$ from $\mathcal{C}_J$ allows us in theory to generate the entire sequence $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{J_{\text{end}}}$. The computational cost, however, grows exponentially with J. We therefore generate $\mathcal{C}_1, \ldots, \mathcal{C}_{J_{\text{launch}}}$, where J_{launch} is typically 3 or 4. Given the covers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{J_{\text{launch}}}$, we can bound the numerical ranges of the digits and residuals of all trajectories from iterations 1 through J_{launch}. To bound the range of all digits d_J at iteration J, we simply record the minimum and maximum of the Jth digit in all the tiles in the cover $\mathcal{C}_J$. Similarly, for each tile in a fixed cover $\mathcal{C}_J$, the associated residual is $r_1 r^{J-1}(f(x) - P_J)$ restricted to

$$x \in [u, v] = \mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J}.$$

For monotonic $f(x)$, the extrema of this residual are achieved at the end points u and v. The values

$$\max / \min \{r_1 r^{J-1}(f(u) - P_J),\ r_1 r^{J-1}(f(v) - P_J)\}$$

taken over all tiles $\mathcal{T} \in \mathcal{C}_J$ thus provide a bounding interval on the numerical ranges of all residuals at iteration J. Having obtained bounds on ranges of all the digits and residuals for iterations $J = 1, 2, \ldots, J_{\text{launch}}$, we conclude our first step.

B. Bound Propagation

We will now propagate the bounds on the ranges of the digits and residuals using interval analysis. We perform bound propagation from each of the tiles in the cover $\mathcal{C}_{J_{\text{launch}}}$ produced previously. This allows us to also exploit the small intervals $[u_i, v_i]$ and the m_i specific to each tile. In addition, we exploit whenever possible the analytic characteristics of the various functions in question such as monotonicity or accurate calculation of extrema.

Picking up from where the state-space covering step ended, we launch our propagation with $\mathcal{C}_{J_{\text{launch}}}$. For each tile $\mathcal{T}$ in $\mathcal{C}_{J_{\text{launch}}}$, we will bound the ranges of digits and residuals of all trajectories that emerge from this tile. Let $\mathcal{T}$ be a tile in $\mathcal{C}_{J_{\text{launch}}}$, $\mathcal{T} = ([u, v], m_i, d_1, d_2, \ldots, d_{J_{\text{launch}}})$, and suppose that we have at some iteration $J \geq J_{\text{launch}}$ obtained an interval $[\gamma_J^-, \gamma_J^+]$ that contains the ranges of iteration-J residuals of all the trajectories emerging from $\mathcal{T}$. We now show how one can obtain residual and digit bounds, $[\gamma_{J+1}^-, \gamma_{J+1}^+]$ and $[d_{J+1}^-, d_{J+1}^+]$, for the same set of trajectories.

Given any trajectory, and in particular one that emerges from $\mathcal{T}$,

$$\begin{aligned} \gamma_{J+1} &= r\gamma_J - d_{J+1} \\ &= r(\gamma_J - R_J) + (rR_J - d_{J+1}). \end{aligned}$$

Because $rR_J - d_{J+1} \in \mathcal{I}_0$, to bound the range of γ_{J+1}, we aim to bound the range of the function $\Delta_J = \gamma_J - R_J$. This function is known explicitly and varies with $x \in [u, v]$ and $\gamma_J \in [\gamma_J^-, \gamma_J^+]$. Interval arithmetic techniques can be applied in general to bound Δ_J's range, but exploiting specific properties of Δ_J yields tighter bounds. We illustrate this with the $f(x) = \sqrt{x}$ example:

$$\begin{aligned} \Delta_J &= r_1 r^{J-1}((\sqrt{x} - P_J) - m_i(x - P_J^2)) \\ &= r_1 r^{J-1}(\sqrt{x} - P_J)(1 - m_i(\sqrt{x} + P_J)) \\ &= \gamma_J[1 - m_i(2\sqrt{x} - (\sqrt{x} - P_J))] \\ &= \gamma_J[(1 - m_i 2\sqrt{x}) + \gamma_J/(r_1 r^{J-1})]. \end{aligned}$$

Note that Δ_J is monotonic in x and hence the extrema of Δ_J on $x \in [u, v]$ and $\gamma_J \in [\gamma_J^-, \gamma_J^+]$ are achieved on the boundaries $x = u$ and $x = v$. On each of these two boundaries, Δ_J is a quadratic in γ_J. The extrema of Δ_J in the entire domain $[u, v] \times [\gamma_J^-, \gamma_J^+]$ can thus be found among the points (u, γ_J^-), (u, γ_J^+), (v, γ_J^-), (v, γ_J^+) as well as the critical points of the two quadratic functions in γ:

$$\gamma(1 - 2m_i\sqrt{x_0} + \gamma/(r_1 r^{J-1})), \quad x_0 = u, v.$$

A critical point needs to be considered only if it falls in $[\gamma_J^-, \gamma_J^+]$. We tabulate the Δ_J functions for the examples of interest here in Table 3.

$f(x)$	$\Delta_J(x, \gamma_J)$
$\sqrt{x}$	$\gamma_J[(1 - m_i 2\sqrt{x}) + \gamma_J/(r_1 r^{J-1})]$
$1/(2\sqrt{x})$	$\gamma_J[(1 - m_i 2\sqrt{x}) + 2m_i x\gamma_J/(r_1 r^{J-1})]$
$1/(2x)$	$\gamma_J(1 - m_i 2x)$

Table 3: $\Delta_J(x, \gamma_J) = \gamma_J - R_J$ for several $f(x)$ of interest. We exploit monotonicity and location of critical points of Δ_J in small domains of (x, γ_J).

The bounds obtained for Δ_J using this technique will be the actual ranges achieved on the domain $[u, v] \times [\gamma_J^-, \gamma_J^+]$. Denoting the range of Δ_J by $[\Delta_J^-, \Delta_J^+]$, we can then define

$$[\gamma_{J+1}^-, \gamma_{J+1}^+] \stackrel{\text{def}}{=} r[\Delta_J^-, \Delta_J^+] + \mathcal{I}_0,$$

which is a bound on the residuals of the next iteration.

Turning to bounds on the digits, we note that

$$\begin{aligned} d_{J+1} &= \texttt{approx-round-int}(rR_J) \text{ implies} \\ rR_J &\in \mathcal{I}_{d_{J+1}} = d_{J+1} + \mathcal{I}_0 \quad \text{implies} \\ d_{J+1} &\in rR_J - \mathcal{I}_0 \\ &\in r\gamma_J + r(R_J - \gamma_J) - \mathcal{I}_0 \\ &\in r[\gamma_J^-, \gamma_J^+] + r[-\Delta_J^+, -\Delta_J^-] - \mathcal{I}_0. \end{aligned}$$

Bounds on the $(J+1)$th digits of all trajectories emerging from tile $\mathcal{T}$ can thus be constructed by

$$d_{J+1}^- \stackrel{\text{def}}{=} \lceil (r(\gamma_J^- - \Delta_J^+) - \max(\mathcal{I}_0)) \rceil,$$

and

$$d_{J+1}^+ \stackrel{\text{def}}{=} \lfloor (r(\gamma_J^+ - \Delta_J^-) - \min(\mathcal{I}_0)) \rfloor.$$

We have shown that for a fixed tile and a bound $[\gamma_J^-, \gamma_J^+]$, we can derive bounds $[\gamma_{J+1}^-, \gamma_{J+1}^+]$ and $[d_{J+1}^-, d_{J+1}^+]$. Starting with any tile in the cover $\mathcal{C}_{J_{\text{launch}}}$ and the associated $[\gamma_{J_{\text{launch}}}^-, \gamma_{J_{\text{launch}}}^+]$ and $[u, v]$, we obtain bounds on all residuals and digits of the entire state space

$$[\gamma_j^-, \gamma_j^+], \quad [d_j^-, d_j^+], \quad J_{\text{launch}} < j \leq J_{\text{end}}.$$

This concludes our second step as well as the correctness certification method.

When a design is certified correct, it is correct for all specific realizations such as particular recoding methods or peculiar arrangements of CSA trees. It is possible that the method fails to certify correct designs that are in fact correct. We call this a false negative. There are two main sources of false negatives. The first comes from $\mathcal{I}_d$. The interval $\mathcal{I}_d$ has to be constructed based on the worst inaccuracies that may be incurred by the `approx-round-int` function. These

correspond to very specific bit patterns of the redundant representations in R_J that are seldom achieved. The second source is from the conservative nature of the triangular inequality. From $\gamma_J = r(\gamma_J - R_J) + (rR_J - d_{J+1})$, for example, we obtain the inequality

$$|\gamma_J| \leq |r(\gamma_J - R_J)| + |rR_J - d_{J+1}|,$$

which is in general an overestimate. Despite these potentials for false negatives, we demonstrate that the correctness certification method works very well in practice.

IV. A Demonstration

The correctness certification methodology was implemented as a design-exploration tool in the numerical computation environment Matlab [7]. We demonstrate the effectiveness of our methodology by applying our tool to the reciprocal square root function $f(x) = 1/(2\sqrt{x})$. The factor $1/2$ in $1/(2\sqrt{x})$ is for normalization purposes so that $1/2 \leq f(x) < 1$ for $1/4 \leq x < 1$. The remainder is defined as in Table 2:

$$R_J = r_1 r^{J-1} M(1/2 - 2xP_J^2)$$

leading to the update recurrence formula

$$R_{J+1} = rR_J - 4MxP_J d_{J+1} - 2Mxd_{J+1}^2/(r_1 r^J).$$

Conventional reciprocal square root designs, where M is obtained by table lookup, are parametrized as follows:

- Input $x = 0.x_1x_2\ldots x_n$, $1/4 \leq x \leq 1 - 2^{-n}$
- For a given input x, M is obtained by
  - $x_{\rm hi} = 0.x_1x_2\ldots x_{\rm M\_in} + 2^{-({\rm M\_in}+1)}$
  - $M = 2^{-{\rm M\_out}} \times \text{nearest-integer}(2^{\rm M\_out}/(2\sqrt{x_{\rm hi}}))$

  where the positive integers M_in and M_out are design parameters.
- $d_1 = \texttt{approx-round-int}\,(r_1 M)$
- $R_1 = r_1 M(1/2 - 2x(d_1/r_1)^2)$
- For $J = 1, 2, \ldots, J_{\rm end} - 1$
  - $d_{J+1} = \texttt{approx-round-int}\,(rR_J)$
  - $R_{J+1} = rR_J - 4MxP_J d_{J+1} - 2Mxd_{J+1}^2/(r_1 r^J)$.

Our consideration here is certifying the correctness of particular choices of parameters. We do not need to be concerned with implementation issues such as representing the update formula for R_{J+1} in terms of carry-save adder trees (CSA). It suffices to assume that R_{J+1} will be obtained in a carry-save format after compressions of partial products and sums. We apply our tool with the following specializations:

State-Space Covering:

- The set of values $\{m_1, m_2, \ldots, m_L\}$ that M can take on is obtained by simple enumeration. L obviously never exceeds $2^{\rm M\_out}$, but in practice does not achieve this maximum because different, but nearby, $0.x_1x_2\ldots x_{\rm M\_in}$ do occasionally map to a common m_j.
- We construct $\mathcal{C}_0 = \{(\mathcal{X}_i, m_i)\}$ as follows. For each m_i, we determine by enumeration the minimum number of the form $0.x_1x_2\ldots x_{\rm M\_in}$ that is mapped to m_i by M. Denote this by μ_i. Similarly, we determine the maximum number of the same form that is mapped to m_i by M. Denote this by ν_i. We define

  $$\mathcal{X}_i \stackrel{\rm def}{=} [u_i, v_i], \text{ where } u_i = \mu_i,\ v_i = \nu_i + 2^{-{\rm M\_in}} - 2^{-n}.$$
- Consider the first digit generation. Because $M \approx 1/(2\sqrt{x}) = f(x)$, a design can choose M_out $= k_1$ (recall that $r_1 = 2^{k_1}$) and thus in effect generate the first digit by table lookup. For this case, we construct $\mathcal{C}_1 = \{(\mathcal{X}_{i,d}, m_i, d) | d = r_1 m_i\}$. If a design chooses M_out $> k_1$, then the first digit generation, given $M = m_i$, is $d = \texttt{approx-round-int}\,(r_1 m_i)$, which implies $x \in \mathcal{I}_d/(r_1 m_i)$. We therefore construct

  $$\mathcal{X}_{i,d} = \text{Expand}\,(\mathcal{X}_i \cap (\mathcal{I}_d/(r_1 m_i)))\,.$$
- Since the remainder function R_{J,m_i,P_J} is

  $$y = r_1 r^{J-1} m_i (1/2 - 2xP_J^2),$$

  the inverse R^{-1}_{J,m_i,P_J} is

  $$x = (2P_J^2)^{-1}\left(1/2 - \frac{y}{m_i r_1 r^{J-1}}\right).$$

Bound Propagation:

- Consider the function $\Delta_J = \gamma_J - R_J$ (see Table 3),

  $$\Delta_J = \left((1 - 2m_i\sqrt{x}) + 2m_i x\gamma_J/(r_1 r^{J-1})\right)\gamma_J$$

  as a function in x and γ_J. Partial differentiation yields

  $$\frac{\partial \Delta_J}{\partial x} = \left(2\gamma_J/(r_1 r^{J-1}) - 1/\sqrt{x}\right) m_i \gamma_J.$$

  Note that because $1 < 1/\sqrt{x} \leq 2$,

  $$1/\sqrt{x} > 2\gamma_J/(r_1 r^{J-1}), \quad \text{for all } \gamma_J \in [\gamma_J^-, \gamma_j^+]$$

  as long as $J \geq 2$ and r_1 and r are of realistic values. This implies that Δ_J is monotonic in x and hence its extremal values in the domain $[u_i, v_i] \times [\gamma_J^-, \gamma_J^+]$ are attained on the boundaries (u_i, γ) and (v_i, γ), $\gamma \in [\gamma_J^-, \gamma_J^+]$. Our tool first ascertains the monotonicity of Δ_J in x and then determines the range of Δ_J by exploiting its quadraticity on the boundaries (u_i, γ) and (v_i, γ).

We show the results of applying this methodology to two instances of this conventional reciprocal square root design. Both instances utilize the digit sets $r_1 = 32$, $\mathcal{D}_1 = \{16, 17, \ldots, 32\}$; $r = 8$, $\mathcal{D} = \{-7, -6, \ldots, 7\}$. The function `approx-round-int` corresponds to a round-to-nearest integer operation and hence $\mathcal{I}_0 = [-1/2, 1/2 + 2^{-t}]$. A design is thus certifiably correct as long as $\gamma_J \in [-1, 1]$ for $J \geq 1$; $d_1 \in \mathcal{D}_1$; and $d_J \in \mathcal{D}$ for $J \geq 2$. The

first instance generates the first digit d_1 by table lookup ($\mathrm{M\_out} = k_1$); the second example generates d_1 by approximate rounding. We use $n = 31$, which corresponds to 32-bit signed fixed-point arithmetic. Because of the relatively low precision, we are able to exhaustively test software versions of the designs under certification to validate our methodology. A software-implemented design reflects one specific realistic hardware implementation where, for example, the remainder is represented in carry-save format and its update is obtained by compression of partial products. The digit generation is implemented by $d_{J+1} = \lfloor rR_J + 1/2 - \beta \rfloor$ where $\beta \in [0, 2^{-t}]$ is a parameter that can be set constant for all inputs or allowed to vary randomly per input value.

Table 4 displays the bounds obtained from our method alongside the actual ranges observed by two exhaustive testings corresponding to $\beta = 0$ and $\beta = 2^{-t}$. The extrema of the observed residuals in the simulation are within ± 0.03 of the tool-generated bounds, implying that false negatives on the residuals are rare—they are only possible if the residuals of a correct design fall within 3% of the maximum allowable value of 1.

The second instance illustrates that the space of designs certifiable correct by means of the global-analytic approach can exclude efficient and correct instances. Instance 2 in Table 4 conforms to the framework of [3]. In particular, the closed-form conditions implying correctness are (1) the first radix is at least 3 bits wider than the general radix, that is, $r_1 \geq 8r$, and (2) the factor $M(x)$ which is designed to approximate $1/(2\sqrt{x})$ satisfies

$$-\frac{\sqrt{x}}{8r}\left(\frac{1}{4} - \frac{23}{32r}\right) < 2M(x)\sqrt{x} - 1 < \frac{\sqrt{x}}{8r}\left(\frac{1}{4} - \frac{23}{32r}\right).$$

The design that is certified correct here violates both conditions. The first radix is smaller: $r_1 = 4r$ (see Table 4), and the $M(x)$ factor does not approximate $1/(2\sqrt{x})$ as accurately as prescribed by the smooth analytic bounds above. This is shown in Figure 1 where the actual $2M(x)\sqrt{x} - 1$ (the discontinuous graph) is plotted in conjunction with the manually derived lower and upper bounds (the smooth envelopes). In order to satisfy these conservative bounds, one would need $\mathrm{M\_in} = 9$, increasing the table size fourfold. In other words, the design Instance 2, which offers a fourfold reduction in lookup table size, would have been rejected if we rely solely on certification by the global-analytic approach.

V. A New $1/\sqrt{x}$ Design

Let us first review in detail a conventional $1/(2\sqrt{x})$ design in which the first digit is obtained by table lookup. In our notation, $\mathrm{M\_out} = k_1$ and $d_1 = r_1 M$. Consider the remainder update recurrence

$$R_{J+1} = rR_J - 4(MxP_J)d_{J+1} - 2(Mx)d_{J+1}^2/(r_1 r^J).$$

$r_1 = 32, \mathcal{D}_1 = \{16, 17, \ldots, 32\}$; $r = 8, \mathcal{D} = \{-7, 6, \ldots, 7\}$
Certified if $d_1 \in \mathcal{D}_1$, $d_J \in \mathcal{D}$, $J \geq 2$, and $\gamma_J \in [-1, 1]$, $J \geq 1$

	Instance 1: $\mathrm{M\_in} = 7,\ \mathrm{M\_out} = 5,\ t = 3$				Instance 2: $\mathrm{M\_in} = 7,\ \mathrm{M\_out} = 7,\ t = 2$			
	Bounds vs. *Actual*				Bounds vs. *Actual*			
J	residuals γ_J		digits d_J		residuals γ_J		digits d_J	
1	$[-0.635,$	$0.622]$	$[16,$	$32]$	$[-0.831,$	$0.512]$	$[16,$	$32]$
	[−0.635,	*0.622]*	*[16,*	*32]*	*[−0.830,*	*0.511]*	*[16,*	*32]*
2	$[-0.500,$	$0.806]$	$[-5,$	$5]$	$[-0.504,$	$0.886]$	$[-7,$	$6]$
	[−0.499,	*0.805]*	*[−5,*	*5]*	*[−0.503,*	*0.886]*	*[−7,*	*4]*
3	$[-0.621,$	$0.805]$	$[-4,$	$7]$	$[-0.548,$	$0.814]$	$[-4,$	$7]$
	[−0.605,	*0.805]*	*[−4,*	*6]*	*[−0.540,*	*0.813]*	*[−4,*	*5]*
4	$[-0.632,$	$0.791]$	$[-5,$	$6]$	$[-0.557,$	$0.806]$	$[-5,$	$6]$
	[−0.610,	*0.790]*	*[−5,*	*6]*	*[−0.537,*	*0.806]*	*[−4,*	*6]*
5	$[-0.667,$	$0.817]$	$[-5,$	$6]$	$[-0.568,$	$0.811]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*
6	$[-0.674,$	$0.832]$	$[-5,$	$6]$	$[-0.570,$	$0.812]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*
7	$[-0.680,$	$0.836]$	$[-5,$	$6]$	$[-0.570,$	$0.812]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*
8	$[-0.679,$	$0.838]$	$[-5,$	$6]$	$[-0.570,$	$0.812]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*
9	$[-0.680,$	$0.838]$	$[-5,$	$6]$	$[-0.570,$	$0.812]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*
10	$[-0.680,$	$0.838]$	$[-5,$	$6]$	$[-0.570,$	$0.812]$	$[-5,$	$6]$
	[−0.647,	*0.809]*	*[−5,*	*6]*	*[−0.538,*	*0.806]*	*[−4,*	*6]*

Table 4: Tool-generated bounds versus actually observed ranges on two instances of a conventional reciprocal square root design. At each iteration we show tool-generated bounds (top) and actual observed ranges (bottom in italics). Actual observed bounds are gathered by combining two exhaustive testings, with β set to 0 and 2^{-t}, respectively.

The latency incurred in computing this recurrence is nontrivial, and the following rather standard techniques in the literature are used to reduce this latency. Both $\tilde{x} = Mx$ and $\tilde{P}_1 = \tilde{x}P_1$ are computed during the initialization phase and at the first digit generation. Moreover, $\tilde{P}_{J+1} = \tilde{x}P_J$ is computed from d_{J+1}, $\tilde{P}_{J+1} = \tilde{P}_J + \tilde{x}d_{J+1}/(r_1 r^J)$, during the iterations right after d_{J+1} is obtained. As a result, the variables $\tilde{P}_J$ and $\tilde{x}$ in the update

$$R_{J+1} = rR_J - 4d_{J+1}\tilde{P}_J - 2\tilde{x}d_{J+1}^2/(r_1 r^J)$$

are no longer in the critical path. In essence, parallelism is used to reduce latency. The outline of this common approach is provided below.

A Standard Inverse Square Root Design:

- Input $x = 0.x_1x_2\ldots x_n$, $1/4 \leq x \leq 1 - 2^{-n}$
- For a given input x, M is obtained by
  - $x_{\mathrm{hi}} = 0.x_1x_2\ldots x_{\mathrm{M\_in}} + 2^{-(\mathrm{M\_in}+1)}$
  - $M = \frac{1}{r_1} \times \text{nearest-integer}(r_1/(2\sqrt{x_{\mathrm{hi}}}))$
- $d_1 = r_1 M$; $\tilde{x} = Mx$; $\tilde{P}_1 = \tilde{x}d_1/r_1$

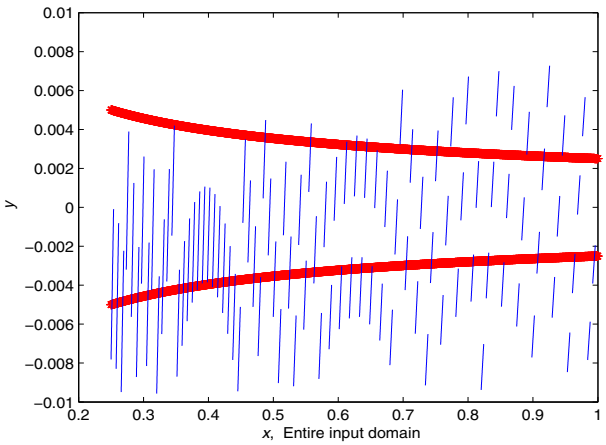

Figure 1: $y = 2M(x)\sqrt{x} - 1$ (discontinuous graph) vs. $y = \pm\frac{\sqrt{x}}{8r}\left(\frac{1}{4} - \frac{23}{32r}\right)$ with $r = 8$ (smooth envelopes). Sufficient conditions require $2M(x)\sqrt{x} - 1$ stay within the smooth envelopes. The design is certified correct despite the fact that it clearly violates the sufficient conditions.

- $R_1 = r_1 M(1/2 - 2xP_1^2)$
- For $J = 1, 2, \ldots, J_{\text{end}} - 1$
  - $d_{J+1} = \texttt{approx-round-int}\,(rR_J)$
  - In parallel, compute
    - $R_{J+1} = rR_J - 4d_{J+1}\tilde{P}_J - 2\tilde{x}d_{J+1}^2/(r_1 r^J)$
    - $\tilde{P}_{J+1} = \tilde{P}_J + \tilde{x}d_{J+1}/(r_1 r^J)$.

Even with the parallelism, however, the inner loop latency is still nontrivial mainly because of the complicated term $2\tilde{x}d_{J+1}^2/(r_1 r^J)$. The multiplication of a square d_{J+1}^2 with $\tilde{x}$ involves not only generating a number of partial products, but also the eventual compression of those products down to two variables—a step that has to be completed before the next iteration can start. That these computations cannot begin until d_{J+1} is generated further exacerbates the difficulties. Indeed, existing work such as [3] chooses to realize this update in two cycles when it is implemented in conjunction with a reciprocal function that runs at a speed of one iteration per cycle.

We will now introduce a new design of the reciprocal square root function that has a reduced latency. We observe that while the term $2\tilde{x}d_{J+1}^2/(r_1 r^J)$ is complicated, the $1/(r_1 r^J)$ factor seems to suggest that this term is inconsequential in determining the outcome of d_{J+2}. We thus propose a new approach to using an outdated version of this complicated term. More precisely, denote by $\tilde{Q}_J$ the term $\tilde{x}d_J^2/(r_1 r^{J-1})$. We define an unconventional remainder by the recurrence $\hat{R}_{J+1} = r\hat{R}_J - 4d_{J+1}\tilde{P}_J - 2r\tilde{Q}_J$. Let us tentatively sketch a new design based on this remainder.

Towards a New Inverse Square Root Design:

- Input $x = 0.x_1x_2\ldots x_n$, $1/4 \le x \le 1 - 2^{-n}$
- For a given input x, M is obtained by
  - $x_{\text{hi}} = 0.x_1x_2\ldots x_{\text{M\_in}} + 2^{-(\text{M\_in}+1)}$
  - $M = \frac{1}{r_1} \times$ nearest-integer$(r_1/(2\sqrt{x_{\text{hi}}}))$
- $d_1 = r_1 M$; $\tilde{x} = Mx$; $\tilde{P}_1 = \tilde{x}d_1/r_1$
- $\hat{R}_1 = r_1 M(1/2 - 2xP_1^2)$; $\tilde{Q}_1 = 0$
- For $J = 1, 2, \ldots, J_{\text{end}} - 1$
  - $d_{J+1} = \texttt{approx-round-int}\,(r\hat{R}_J)$
  - In parallel, compute
    - $\hat{R}_{J+1} = r\hat{R}_J - 4d_{J+1}\tilde{P}_J - 2r\tilde{Q}_J$
    - $\tilde{P}_{J+1} = \tilde{P}_J + \tilde{x}d_{J+1}/(r_1 r^J)$
    - $\tilde{Q}_{J+1} = \tilde{x}d_{J+1}^2/(r_1 r^J)$

The remainder update critical path for $\hat{R}$ has a shorter latency than that for R as the term $\tilde{Q}_J$ is ready at the beginning of the iteration, independent of d_{J+1}. One can see that $\hat{R}_1 = R_1$ and $\hat{R}_2 = R_2 + 2\tilde{Q}_2$. Using the recurrence definition of $\hat{R}_J$, we find that

$$\hat{R}_J = R_J + 2\tilde{Q}_J, \quad \text{for all } J \ge 2.$$

We can readily apply our correctness certification method to this unconventional design.

A. State-Space Covering

Since M is generated by table lookup, the initial cover $\mathcal{C}_0 = \{(\mathcal{X}_i, m_i)\}$ is readily obtainable. The same is true for $\mathcal{C}_1$ because the first digit is exactly $r_1 m_i$: $\mathcal{C}_1 = \{(\mathcal{X}_i, m_i, d) | d = r_1 m_i)\}$. To generate $\mathcal{C}_{J+1}$ from $\mathcal{C}_J$, we derive the inverse of the remainder function restricted to $M = m_i$ and the first J digits to the $d_1, d_2, \ldots, d_J$ corresponding to a specific tile in $\mathcal{C}_J$. The function $\hat{R}_{J,m_i,P_J}$ is thus given by

$$\begin{aligned} y &= R_J + 2\tilde{Q}_J \\ &= r_1 r^{J-1} m_i(1/2 - 2xP_J^2) + 2m_i x d_J^2/(r_1 r^{J-1}) \\ &= r_1 r^{J-1} m_i/2 - 2m_i x(r_1 r^{J-1} P_J^2 - d_J^2/(r_1 r^{J-1})). \end{aligned}$$

This is a linear function in x whose inverse is

$$x = \frac{r_1 r^{J-1} m_i/2 - y}{2m_i(r_1 r^{J-1} P_J^2 - d_J^2/(r_1 r^{J-1}))}.$$

For each $A = \mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J}$ and d, we can thus obtain $\mathcal{X}_{i,m_i,d_1,d_2,\ldots,d_J,d}$ as

$$\text{Expand}\left(A \cap \left(\frac{r_1 r^{J-1} m_i/2 - \mathcal{I}_d}{2m_i(r_1 r^{J-1} P_J^2 - d_J^2/(r_1 r^{J-1}))}\right)\right).$$

B. Bound Propagation

At the end of the state-space covering step, we have with each tile a small interval $[u, v]$, a fixed M value m_i, and the bounds $[\gamma_J^-, \gamma_J^+]$ and $[d_J^-, d_J^+]$, $J = J_{\text{launch}}$. We will propagate these bounds using interval arithmetic techniques,

making use of the previously derived relationship between the residual and the conventional remainder R:

$$\begin{aligned}\gamma_{J+1} &= r\gamma_J - d_{J+1} \\ &= r(\gamma_J - \hat{R}_J) - (r\hat{R}_J - d_{J+1}) \\ &= r(\gamma_J - R_J) + r(R_J - \hat{R}_J) + (r\hat{R}_J - d_{J+1}) \\ &= r\Delta_J - 2r\hat{Q}_J + (r\hat{R}_J - d_{J+1}),\end{aligned}$$

where Δ_J was listed in Table 3. We thus compute using standard interval arithmetic, for $J \geq J_{\text{launch}}$,

$$\begin{aligned}[\gamma_{J+1}^-, \gamma_{J+1}^+] &= r[\Delta_J^-, \Delta_J^+] + \mathcal{I}_0 \\ &\quad - \frac{2rm_i}{r_1 r^{J-1}}[u, v] \cdot [d_J^-, d_J^+]^2.\end{aligned}$$

For bounds on the digits, we follow similarly the derivation in Section III-B:

$$\begin{aligned}d_{J+1} \in\ & r([\gamma_J^-, \gamma_J^+] + [-\Delta_J^+, -\Delta_J^-]) \\ & + \mathcal{I}_0 \\ & + \frac{2rm_i[u, v] \cdot [d_J^-, d_J^+]^2}{r_1 r^{J-1}}.\end{aligned}$$

We choose $r_1 = 32$, $\mathcal{D}_1 = \{16, \ldots, 32\}$, $r = 8$, $\mathcal{D} = \{-7, \ldots, 7\}$, M_in $= 7$, M_out $= 5$, and $t = 4$. The requirements for correctness are thus $\gamma_J \in [-1, 1]$ for $J \geq 1$ and the digits belong to their corresponding sets $\mathcal{D}_1$ and $\mathcal{D}$. We obtained the following bounds from the state-space covering step with $J_{\text{launch}} = 4$:

J	residuals γ_J		digits d_J	
1	$[-0.635,$	$0.622]$	$[16,$	$32]$
2	$[-0.499,$	$0.725]$	$[-5,$	$5]$
3	$[-1.125,$	$0.598]$	$[-4,$	$6]$
4	$[-0.765,$	$0.768]$	$[-9,$	$5]$

It is obvious that the design cannot be certified correct due to the bounds for γ_3 and d_4 (testing reveals actual failures as well). Nevertheless, the reported bounds suggest an easy remedy. Because $\hat{R}_J = R_J + 2\tilde{Q}_J$, one would typically obtain a digit that is biased in the positive direction. This bias is most significant in the generation of d_3. We thus make a minor modification to the digit generation to attenuate the bias: $d_{J+1} = \texttt{approx-round-int}\,(r\hat{R}_J - \beta_J)$, where $\beta_J = 0$ always except for β_2. It is advantageous to use a β_2 with short precision. We thus restrict β_2 to integer multiples of $2^{-t} = 1/16$. After a few educated guesses, $\beta_2 = 5/16$ emerges as a solution. With this modification, the resulting design is certified correct. The bounds generated by state-space covering on the first four iterations are:

J	residuals γ_J		digits d_J	
1	$[-0.635,$	$0.622]$	$[16,$	$32]$
2	$[-0.499,$	$0.725]$	$[-5,$	$5]$
3	$[-0.873,$	$0.898]$	$[-4,$	$6]$
4	$[-0.673,$	$0.710]$	$[-7,$	$7]$

Subsequent bound propagation certifies that from iterations 5 to 10, the digits generated stay within $[-6, 6]$ and the residuals stay within $[-0.68, 0.76]$. In summary, the novel reciprocal square root design presented here eliminates the fundamental dependency of the $(J+1)$th remainder on the quadratic term $d_{J+1}^2 Mx/(r_1 r^J)$.

VI. Conclusion

We have developed a rigorous, computer-aided correctness-certification method for digit-by-rounding algorithms, which has been implemented as an interactive design exploration tool. This tool enabled a fourfold reduction of the lookup table of a conventional reciprocal square root design, relative to the requirements of a conventional correctness-certification method. We have also presented a novel reciprocal square root design that utilizes an unconventional definition of the remainder as well as a digit generation with a bias. The generality of our certification method allowed us to naturally accommodate and prove the correctness of this new design.

We have focused on digit-by-rounding digit recurrences, but we believe our method is applicable to digit-recurrence algorithms more generally. For other digit-recurrence designs, one would only need to characterize the inverse image of the digit-generation function. We plan to explore this generalization in the future.

Acknowledgments

We thank Rebecca Kastleman and Amy Perry for their editoral assistance.

References

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

[2] M. Ercegovac, T. Lang, and P. Montuschi, "Very-high radix division with prescaling and selection by rounding," *IEEE Trans. Comput.*, vol. 43, pp. 909–918, Aug. 1994.

[3] 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.

[4] 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*, 1995, pp. 124–131.

[5] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993.

[6] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Francisco, CA: Mogan Kaufmann Publishers, 2004.

[7] Matlab R2010a. The MathWorks, Inc. Natick, MA.