Roll Your Own Divide/Square Root Unit

J.P. Grossman

Basic Division

We begin with a review of the basic fully restoring binary division algorithm in order to (1) establish a notation which will be used throughout, and (2) understand why it is slow and how it can be improved. We define the following symbols:

- A - numerator $\frac{1}{2} \le A < 1$
- D - denominator $\frac{1}{2} \le D < 1$
- Q_i - i^{th} partial quotient (i digits of precision after the binary decimal point)
- q_i - i^{th} quotient digit, $q_i \in \{0, 1\}$

$$Q_i = q_0.q_1q_2\cdots q_i = \sum_{k=0}^{i} q_k 2^k \qquad (1)$$

- R_i - i^{th} shifted remainder

The quotient and remainder are defined such that the invariant

$$A = Q_iD + 2^{-i}R_i \qquad (2)$$

is satisfied at all times. The reason for working with an unshifted quotient and a shifted remainder is that these are the representations that will actually be used in hardware. For division-only circuits it is often easier to compute Q in a shift register, but for unified division/square root circuits we will see that the unshifted quotient/root is required at each iteration in order to compute the square root.

The rule for updating Q is simply

$$Q_{i+1} = Q_i + 2^{-i-1}q_{i+1} \qquad (3)$$

and the rule for updating R can be derived from the invariant (2):

$$R_{i+1} = 2^{i+1}(A - Q_{i+1}D) = 2^{i+1}(Q_iD + 2^{-i}R_i - Q_iD - 2^{-i-1}q_{i+1}D) = 2R_i - q_{i+1}D \qquad (4)$$

The familiar constraint on R_i which guarantees convergence is $0 \le R_i < D$. To choose q_{i+1} we observe that R_{i+1} must satisfy this same constraint, so we must have:

$$0 \le 2R_i - q_{i+1}D < D$$

This leads to the following selection rule:

Project Aries Technical Memo ARIES-TM-01
Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA

$$q_{i+1} = \begin{cases} 0: & 0 \le R_i < \dfrac{D}{2} \\ 1: & \dfrac{D}{2} \le R_i < D \end{cases} \qquad (5)$$

This division algorithm works, but it is slow for two reasons. First, it only produces one result bit per iteration. We can reduce the number of iterations required by using radix r > 2, that is, by treating $q_0.q_1q_2\ldots q_i$ as a base r representation of Q_i rather than a base 2 representation. Typically r is a power of two. The second reason the vanilla algorithm is slow is that we need an *exact* comparison of R_i with D/2. There is no room for error; if we choose the wrong value for q_{i+1} at any iteration then we will end up with an incorrect final result. This also means that we need an exact value for R_i at each iteration, i.e. we must perform an exact subtraction to compute R_{i+1}. To get around this constraint we will use a redundant digit set.

Redundant Digit Set Radix r Division

Rather than choosing q_{i+1} from the standard radix r digit set {0, 1, …, r-1}, we will choose from the redundant digit set:

$$\{-n, -(n-1), \ldots, -1, 0, 1, \ldots, n-1, n\}$$

where $2n \ge r$. A given number may have more than one radix r representation of the form $q_0.q_1q_2\ldots$ with $q_i \in \{-n, \ldots, n\}$. This means that we can afford some sloppiness in the determination of q_{i+1}, as there may be more than one choice which will allow us to converge to the correct answer.

We can easily reformulate equations 1-4 by replacing 2 with r:

$$Q_i = q_0.q_1q_2\cdots q_i = \sum_{k=0}^{i} q_k r^k \qquad (6)$$

$$A = Q_iD + r^{-i}R_i \qquad (7)$$

$$Q_{i+1} = Q_i + r^{-i-1}q_{i+1} \qquad (8)$$

$$R_{i+1} = rR_i - q_{i+1}D \qquad (9)$$

To derive the bounds on R_i which will ensure convergence to the correct answer, we observe that given $Q_i = q_0.q_1q_2\ldots q_i$, there must exist choices for q_{i+1}, q_{i+2}, q_{i+3}, … such that

$$q_0.q_1q_2\ldots q_iq_{i+1}q_{i+2}\ldots = A/D$$

Now the smallest possible choice for each q_i is -n and the largest possible choice is n, so we have

$$q_0.q_1q_2\ldots q_i(-n)(-n)(-n)\ldots \le A/D \le q_0.q_1q_2\ldots q_i(n)(n)(n)\ldots$$

$$\Rightarrow \quad Q_i - \frac{nr^{-i}}{r-1} \le \frac{A}{D} \le Q_i + \frac{nr^{-i}}{r-1} \qquad (10)$$

If we agree by convention to avoid representations of A/D with trailing repeated (-n) or (n) (for example, if r = 4 and n = 3 then we will always write 0.2 instead of 0.13333333…), then the inequalities can be made strict. Multiplying (10) by D and plugging in (7) for A:

$$Q_i D - \frac{nr^{-i}}{r - 1} D < Q_i D + r^{-i} R_i < Q_i D + \frac{nr^{-i}}{r - 1} D$$

$$\Rightarrow \quad -\frac{n}{r - 1} D < R_i < \frac{n}{r - 1} D \qquad (11)$$

This, then, is the constraint that must be satisfied so that it is always possible for Q_i to converge to A/D. Conversely, if the constraint is always satisfied then (7) shows that Q_i will converge to A/D. The constraint is all that we need to worry about in choosing quotient digits.

Quotient Digit Selection

To figure out how to choose the quotient digits we will turn things around a bit. Instead of developing a set of rules of the form "If R is in this range then choose that digit" as in (5), we will answer the question "When is it acceptable to choose q_{i+1} = d?". Our only constraint is

$$-\frac{n}{r - 1} D < R_{i+1} < \frac{n}{r - 1} D \qquad (11')$$

so q_{i+1} = d is ok if

$$-\frac{n}{r - 1} D < rR_i - dD < \frac{n}{r - 1} D$$

$$\Leftrightarrow \quad \frac{1}{r}\left(d - \frac{n}{r-1}\right) D < R_i < \frac{1}{r}\left(d + \frac{n}{r-1}\right) D \qquad (12)$$

These intervals always have some overlap as there are 2n+1 of them, each of length $\frac{1}{r}\frac{2n}{r - 1} D$, all contained in the interval $\left(-\frac{n}{r-1} D, \frac{n}{r-1} D\right)$ of length $\frac{2n}{r - 1} D$, and 2n+1 > r. This means that, as expected with a redundant digit set, there is sometimes more than one possible choice for q_{i+1} which allows us to be a bit sloppy in our comparisons. Specifically, it allows us to inspect only the most significant bits of R_i when making the comparison.

Radix 4 Division

We will present the determination of the number of bits of accuracy required by way of example. Consider radix 4 division with the digit set {-3, -2, -1, 0, 1, 2, 3} (n = 3). In this case we have:

$$A = Q_i D + 4^{-i} R_i \qquad (13)$$
$$Q_{i+1} = Q_i + 4^{-i-1} q_{i+1} \qquad (14)$$

$$R_{i+1} = 4R_i - q_{i+1}D \tag{15}$$

and the constraint on R_i is

$$-D < R_i < D \tag{16}$$

Note that we can start with $Q_0 = 1$ and $R_0 = A - D$ as $\frac{1}{2} \le A < 1$ and $\frac{1}{2} \le D < 1$ imply that

$$-D \le -\tfrac{1}{2} < A - D < \tfrac{1}{2} \le D$$

so the constraint is satisfied for i = 0. From (12) we have the following seven rules:

$q_{i+1} = 3$	ok if	$\frac{1}{2}D < R_i < D$	(12a)
$q_{i+1} = 2$	ok if	$\frac{1}{4}D < R_i < \frac{3}{4}D$	(12b)
$q_{i+1} = 1$	ok if	$0 < R_i < \frac{1}{2}D$	(12c)
$q_{i+1} = 0$	ok if	$-\frac{1}{4}D < R_i < \frac{1}{4}D$	(12d)
$q_{i+1} = -1$	ok if	$-\frac{1}{2}D < R_i < 0$	(12e)
$q_{i+1} = -2$	ok if	$-\frac{3}{4}D < R_i < -\frac{1}{4}D$	(12f)
$q_{i+1} = -3$	ok if	$-D < R_i < -\frac{1}{2}D$	(12g)

The following strategy can be used to determine q_{i+1}:

(1) Compare R_i to 0
(2) If $R_i > 0$, compare R_i to $\frac{1}{2}D$
If $R_i = 0$, choose $q_{i+1} = 0$
If $R_i < 0$, compare R_i to $-\frac{1}{2}D$

Suppose we use k bits of precision after the binary decimal point. Using $[X]_k$ to denote X truncated to k bits of precision, then we are really comparing $[R_i]_k$ to $[X]_k$ where X = 0 or $\pm\frac{1}{2}D$. Now for any number X we have

$$[X]_k \le X < [X]_k + 2^{-k} \tag{17}$$

and in particular

$$[R_i]_k \le R_i < [R_i]_k + 2^{-k} \tag{18}$$

We use (17) and (18) to analyze the three possible outcomes of a comparison of $[R_i]_k$ and $[X]_k$:

$$[R_i]_k < [X]_k \Rightarrow [R_i]_k + 2^{-k} \le [X]_k \Rightarrow R_i < [X]_k \le X \tag{19}$$

sim.
$$[R_i]_k > [X]_k \Rightarrow R_i > X \tag{20}$$

$$[R_i]_k = [X]_k \Rightarrow [X]_k \le R_i < [X]_k + 2^{-k} \Rightarrow X - 2^{-k} < R_i < X + 2^{-k} \tag{21}$$

The inaccuracy lies in the third case; if $[R_i]_k = [X]_k$ then instead of being able to pinpoint R_i at X, all we can say is that R_i lies within 2^{-k} of X. However, the choice of $q_{i+1} = 0$ is valid as long as $-\frac{1}{4}D < R_i < \frac{1}{4}D$, so if $2^{-k} \le \frac{1}{4}D$ then we can take $q_{i+1} = 0$ when $[R_i]_k = 0$. Similarly, the choice of $q_{i+1} = \pm 2$ is valid as long as R_i is within $\frac{1}{4}D$ of $\pm\frac{1}{2}D$, so again if $2^{-k} \le \frac{1}{4}D$ then we can take $q_{i+1} = \pm 2$ when $[R_i]_k = [\pm\frac{1}{2}D]_k$. Since $D \ge \frac{1}{2}$, we can choose k = 3, so only three bits of precision are needed. Furthermore, since $D < 1 \Rightarrow \frac{1}{2}D < \frac{1}{2}$, so only a 2.5 bit comparison is required.

Using Carry Save Addition

Once it is realized that we only need the top few bits of R_i to select q_{i+1}, there is no longer any point in computing R_i exactly. Instead, we can use fast carry save addition at each iteration and then use carry propagate addition to compute the upper bits as needed. We will write:

$$R_i = R_i^+ - R_i^- \tag{22}$$

Abusing our own notation slightly, define

$$[R_i]_k = [R_i^+]_k - [R_i^-]_k \tag{23}$$

thus ignoring a possible borrow into the high k bits of R_i. From (17) we have

$$[R_i^+]_k \le R_i^+ < [R_i^+]_k + 2^{-k}$$

and

$$[R_i^-]_k \le R_i^- < [R_i^-]_k + 2^{-k}$$

thus

$$[R_i^+]_k - [R_i^-]_k - 2^{-k} < R_i^+ - R_i^- < [R_i^+]_k - [R_i^-]_k + 2^{-k}$$

$$\Rightarrow \quad [R_i]_k - 2^{-k} < R_i < [R_i]_k + 2^{-k} \tag{24}$$

As an aside, the reason for defining R_i as in (22) rather than $R_i = R_i^a + R_i^b$ is that with a sum formulation, (24) would read

$$[R_i]_k \le R_i < [R_i]_k + 2 \cdot 2^{-k}$$

with a lower bound that is not as clean as that in (24). The carry save adder is almost identical to the standard one; instead of solving the equation $x + y + z = 2u + v$ for u, v it solves the equation $x - y + z = 2u - v$.

Expressing R_i in this manner we still only need three bits for the comparison against 0 as

$[R_i]_3 > 0 \quad \Rightarrow [R_i]_3 \ge 1/8 \Rightarrow R_i > 0$
$[R_i]_3 < 0 \quad \Rightarrow [R_i]_3 \le -1/8 \quad \Rightarrow R_i < 0$
$[R_i]_3 = 0 \quad \Rightarrow -1/8 < R_i < 1/8$

However for the comparisons against ±½D we are going to need an extra bit. This is because (17) and (24) together yield

$$[R_i]_k = [X]_k \quad \Rightarrow [X]_k - 2^{-k} < R_i < [X]_k + 2^{-k} \quad \Rightarrow X - 2 \cdot 2^{-k} < R_i < X + 2^{-k} \tag{25}$$

so the lower error margin has doubled. One extra bit suffices to compensate for this, and we have the following four cases (using equation 25 to derive the bounds):

(1) $[R_i]_4 = [½D]_4 \quad \Rightarrow ½D - 1/8 < R_i < ½D + 1/16 \quad \Rightarrow ¼D < R_i < ¾D$

(2) $[R_i]_4 = [\frac{1}{2}D]_4 + 1/16 \Rightarrow \frac{1}{2}D - 1/16 < R_i < \frac{1}{2}D + 1/8 \Rightarrow \frac{1}{4}D < R_i < \frac{3}{4}D$
(3) $[R_i]_4 > [\frac{1}{2}D]_4 + 1/16 \Rightarrow [R_i]_4 \geq [\frac{1}{2}D]_4 + 1/8 \Rightarrow R_i > \frac{1}{2}D$
(4) $[R_i]_4 < [\frac{1}{2}D]_4 \Rightarrow [R_i]_4 \leq [\frac{1}{2}D]_4 - 1/16 \Rightarrow R_i < \frac{1}{2}D$

In cases (1) and (2) we can choose $q_{i+1} = 2$, in case 3 we can choose $q_{i+1} = 3$, and in case (4) we can choose $q_{i+1} = 1$. The cases when $R_i < 0$ are handled similarly.

One final and important note is that an extra bit is needed to the left of the decimal point to represent R_i. If R_i is represented exactly then since $|R_i| < D < 1$ we can express R_i using 2's complement with one sign bit to the left of the decimal point, i.e.

$$R_i = s.r_1r_2\ldots \quad (26)$$

However, it is *not* necessarily true that $|R_i^+| < 1$ and $|R_i^-| < 1$, for example we could have $R_i^+ = 1$ and $R_i^- = 1/32$. If R_i were expressed as in (26) then we would incorrectly compute

$$[R_i]_4 = 1.0000 = -1 \quad (27)$$

However, from (24) it is certainly true that $|[R_i]_4| < |R_i| + 1/16 < 17/16$, so one extra bit is more than enough to apologize for cases such as 27. Thus, we can express R_i^+ and R_i^- as:

$$R_i^+ = s^+r_0^+.r_1^+r_2^+\ldots$$

$$R_i^- = s^-r_0^-.r_1^-r_2^-\ldots$$

There are still a few details that would need to be worked out in order to implement this radix 4 division algorithm as a fully functional divide unit, namely:

- Exactly how the comparisons are handled down at the bit level when $R_i < 0$
- If the final remainder is negative it may be desirable to do a bit of cleanup so that the unit produces the same result that would come from a conventional fully restoring divider
- Handling integer division, specifically:
  - How many iterations to perform
  - How to back up by 1 bit if an odd number of quotient bits are required

These details are left as an exercise to the reader.

Square Root Algorithms

We can use the exact same approach as for the division algorithms to devise square root algorithms. Once again the basic working principle is that we wish to produce one root digit per iteration such that the partial root converges to the correct answer. Consider a radix r algorithm with redundant digit set {-n, …, n}. Define Q_i and R_i as before, where Q_i now denotes a partial square root rather than a partial quotient. Let A be the radicand. To avoid messing things up we only want to normalize A by shifting two binary digits at a time, so assume that $\frac{1}{4} \leq A < 1$. Our invariant is now:

$$A = Q_i^2 + r^{-i}R_i \tag{28}$$

As before, the update rule for Q is

$$Q_{i+1} = Q_i + r^{-i-1}q_{i+1} \tag{29}$$

We can use (28) to determine the update rule for R:

$$R_{i+1} = r^{i+1}(A - Q_{i+1}^2) = r^{i+1}(Q_i^2 + r^{-i}R_i - Q_i^2 - 2Q_iq_{i+1}r^{-i-1} - q_{i+1}^2r^{-2i-2})$$

$$\Rightarrow \quad R_{i+1} = rR_i - 2Q_iq_{i+1} - q_{i+1}^2r^{-i-1} \tag{30}$$

To find the constraint on R_i we again have the following convergence requirement, almost identical to the strict version of (10):

$$Q_i - \frac{nr^{-i}}{r - 1} < \sqrt{A} < Q_i + \frac{nr^{-i}}{r - 1} \tag{31}$$

For $i \geq 1$ the lower bound is positive (exercise left to the reader), so we can square all three terms and plug in (28) for A:

$$Q_i^2 - 2\frac{nr^{-i}}{r-1}Q_i + \left(\frac{nr^{-i}}{r-1}\right)^2 < Q_i^2 + r^{-i}R_i < Q_i^2 + 2\frac{nr^{-i}}{r-1}Q_i + \left(\frac{nr^{-i}}{r-1}\right)^2$$

$$\Rightarrow \quad -2\frac{n}{r-1}Q_i + \left(\frac{n}{r-1}\right)^2 r^{-i} < R_i < 2\frac{n}{r-1}Q_i + \left(\frac{n}{r-1}\right)^2 r^{-i} \tag{32}$$

Once again, this constraint is all that matters; as long as it is satisfied for each R_i the Q_i will converge to the correct square root value. We can now use (32) to determine the constraints for selecting q_{i+1}. Proceeding as before, we must have

$$-2\frac{n}{r-1}Q_{i+1} + \left(\frac{n}{r-1}\right)^2 r^{-i-1} < R_{i+1} < 2\frac{n}{r-1}Q_{i+1} + \left(\frac{n}{r-1}\right)^2 r^{-i-1} \tag{32'}$$

so, making use of (29) and (30), $q_{i+1} = d$ is ok if

$$-2\frac{n}{r-1}\left(Q_i + dr^{-i-1}\right) + \left(\frac{n}{r-1}\right)^2 r^{-i-1} < rR_i - 2Q_id - d^2r^{-i-1} < 2\frac{n}{r-1}\left(Q_i + dr^{-i-1}\right) + \left(\frac{n}{r-1}\right)^2 r^{-i-1}$$

$$\Leftrightarrow \quad \frac{2}{r}Q_i\left(d - \frac{n}{r-1}\right) + r^{-i-2}\left(d - \frac{n}{r-1}\right)^2 < R_i < \frac{2}{r}Q_i\left(d + \frac{n}{r-1}\right) + r^{-i-2}\left(d + \frac{n}{r-1}\right)^2 \tag{33}$$

Radix 4 Square Root

We will once again continue with an example. Consider radix 4 square root with the digit set {-3, -2, -1, 0, 1, 2, 3} (n = 3). In this case we have:

$$A = Q_i^2 + 4^{-i}R_i \quad (34)$$

$$Q_{i+1} = Q_i + 4^{-i-1}q_{i+1} \quad (35)$$

$$R_{i+1} = rR_i - 2Q_iq_{i+1} - q_{i+1}^2 4^{-i-1} \quad (36)$$

and the constraint on R_i is

$$-2Q_i + 4^{-i} < R_i < 2Q_i + 4^{-i} \quad (37)$$

We can start with $Q_0 = 0$ and $R_0 = A$, which satisfies equation (31) for i = 0. From (33) we have the following seven rules:

$q_{i+1} = 3$	ok if	$Q_i + 4^{-i-1} < R_i < 2Q_i + 4^{-i}$	(33a)
$q_{i+1} = 2$	ok if	$\frac{1}{2}Q_i + 4^{-i-2} < R_i < 3/2Q_i + 9\cdot4^{-i-2}$	(33b)
$q_{i+1} = 1$	ok if	$0 < R_i < Q_i + 4^{-i-1}$	(33c)
$q_{i+1} = 0$	ok if	$-\frac{1}{2}Q_i + 4^{-i-2} < R_i < \frac{1}{2}Q_i + 4^{-i-2}$	(33d)
$q_{i+1} = -1$	ok if	$-Q_i + 4^{-i-1} < R_i < 0$	(33e)
$q_{i+1} = -2$	ok if	$-3/2Q_i + 9\cdot4^{-i-2} < R_i < -\frac{1}{2}Q_i + 4^{-i-2}$	(33f)
$q_{i+1} = -3$	ok if	$-2Q_i + 4^{-i} < R_i < -Q_i + 4^{-i-1}$	(33g)

The following strategy, almost identical to the division strategy, can be used to determine q_{i+1}:

(1) Compare R_i to 0
(2) If $R_i > 0$, compare R_i to $Q_i + 4^{-i-1}$
If $R_i = 0$, choose $q_{i+1} = 0$
If $R_i < 0$, compare R_i to $-Q_i + 4^{-i-1}$

Determining the margin of error for the comparisons was straightforward for division, but will require a bit more work for square root. We make use of the following two lemmas:

<u>lemma 1:</u> $Q_i \geq \frac{1}{2}$ for all $i \geq 1$

<u>proof:</u> From (31) we have $Q_i + 4^{-i} > \sqrt{A}$. But $A \geq \frac{1}{4}$ so $Q_i > \frac{1}{2} - 4^{-i}$
$\Rightarrow \quad 4^iQ_i > 2\cdot4^{i-1} - 1$. But 4^iQ_i is an integer, so $4^iQ_i \geq 2\cdot4^{i-1}$ and the result follows.

<u>lemma 2:</u> If $R_i < 0$ then $Q_i \geq \frac{1}{2} + 4^{-i}$

<u>proof:</u> Choose $j \leq i$ such that $R_j < 0$ but $R_{j-1} \geq 0$. This means that $R_j < R_{j-1}$ which, by inspection of (36), can only be the case if $q_j \geq 1$. Thus $Q_j \geq Q_{j-1} + 4^{-j} \geq \frac{1}{2} + 4^{-j}$ (applying lemma 1). Finally, since $q_k \geq -3$ for $j < k \leq i$, we have

$$Q_i \geq \frac{1}{2} + 4^{-j} - 3\cdot(4^{-j-1} + \ldots + 4^{-i}) = \frac{1}{2} + 4^{-i} \text{ as required}$$

We can now determine the margins of error for each of the comparisons. We begin with the comparison against 0. If $R_i > 0$, then from rule (33d) the margin of error is $\frac{1}{2}Q_i + 4^{-i-2} \geq \frac{1}{4}$ (applying lemma 1). If $R_i < 0$, then the margin of error is $\frac{1}{2}Q_i - 4^{-i-2} \geq \frac{1}{4} + \frac{1}{2}\cdot 4^{-i} - 4^{-i-2} > \frac{1}{4}$ (applying lemma 2). In fact, if we apply lemma 1 to (33b) and lemma 2 to (33f), we find that all margins of error are $> \frac{1}{4}$, so we only need two bits of precision for the comparison against 0 and three bits of precision for the comparisons against $\pm Q_i + 4^{-i-1}$, fewer precision bits than are required for division! However, we end up doing the exact same amount of work as in the division case since all three precision bits of $\pm Q_i + 4^{-i-1}$ are compared against, and as we will see shortly we need one extra bit to the left of the decimal point in the representation of R_i.

Bounding R_i

Unlike division, the bound (37) on R_i depends on i. However, it is not too difficult to change this into a bound which is independent of i. We note that $Q_i < 1 \Rightarrow 4^i Q_i < 4^i$. As in the proof of lemma 1, $4^i Q_i$ is an integer and so $4^i Q_i \leq 4^i - 1 \Rightarrow Q_i \leq 1 - 4^{-i} \Rightarrow 2Q_i + 4^{-i} \leq 2 - 4^i < 2$. Thus, from (37) we see that $|R_i| < 2$. As with division, an exact representation of R_i would only require a sign bit and a 2^0 bit to the left of the binary decimal point, but for the carry save representation we need a sign bit, a 2^0 bit and a 2^1 bit.

On-line Conversion of Q_i

Since the digits q_i are taken from a redundant digit set, at some point we must convert Q from redundant digit to standard binary form. Furthermore, since Q_i appears in the update rule (36) for R_i, it is desirable to perform this conversion on-line so as to have the binary representation of Q_i available at every step.

We can accomplish this fairly easily by storing both Q_i and $Q_i' = Q_i - 4^i$. Assuming that Q_i and Q_{i+1} are both stored in standard binary form, we can update them using the one of the following three rules, depending on the sign of q_{i+1}:

1)	$q_{i+1} > 0$	$Q_{i+1} = Q_i + q_{i+1}4^{-i-1}$	$Q_{i+1}' = Q_i' + (q_{i+1} - 1)\cdot 4^{-i-1}$
2)	$q_{i+1} = 0$	$Q_{i+1} = Q_i$	$Q_{i+1}' = Q_i' + 3\cdot 4^{-i-1}$
3)	$q_{i+1} < 0$	$Q_{i+1} = Q_i' + (4 + q_{i+1})\cdot 4^{-i-1}$	$Q_{i+1}' = Q_i' + (3 + q_{i+1})\cdot 4^{-i-1}$

In each case Q_{i+1} and Q_{i+1}' are formed by adding one of $\{0, 4^{-i-1}, 2\cdot 4^{-i-1}, 3\cdot 4^{-i-1}\}$ to one of Q_i, Q_i'; the update can therefore be accomplished by simply ORing in the new bits.

References

Stanislaw Majerski, “Square-Rooting Algorithms for High-Speed Digital Circuits”, IEEE Transactions on Computers, Vol. C-34, No. 8, August 1985, pp. 724-733

Hosahalli R. Srinivas, Keshab K. Parhi, “A Floating Point Radix 2 Shared Division/Square Root Chip”, Proc. International Conference on Computer Design, 1995, pp. 472-478

Sau-Gee Chen, Chieh-Chih Li, “Efficient Designs of Unified 2’s Complement Division and Square-root Algorithm and Architecture”, IEEE TENCON, 1994

Vitit Kantabutra, “A New Algorithm for Division in Hardware”, Proc. International Conference on Computer Design, 1996, pp. 551-556

Tzu-His Pan, Hyon-Sok Kay, Youngsun Chun, Chin-Long Wey, “High-Radix SRT Division with Speculation of Quotient Digits”, Proc. International Conference on Computer Design, 1995, pp. 479-484

Jan Fandrianto, “Algorithm for High Speed Shared Radix 8 Division and Radix 8 Square Root”, Proc. International Symposium on Computer Arithmetic, 1989, pp. 68-75