CHAPTER 6

Square Root by Digit Recurrence

This method of performing the square root operation is conceptually very similar to the method for division discussed in the previous chapter. Consequently, we develop the algorithm in a similar fashion, providing less detail since we assume familiarity with the developments for division. Table 6.1 gives a summary of the main definitions. Moreover, we concentrate on algorithms that use estimates for the result-digit selection and redundant addition in the recurrence.

As in division, the algorithm is presented for *fractional* operand x and result s. For floating-point representation and normalized operand, it is necessary to scale the operand to have an even exponent to allow the computation of the result exponent. Consequently,

$$s = \sqrt{x}, \qquad \frac{1}{4} \le x < 1, \qquad \frac{1}{2} \le s < 1 \tag{6.1}$$

6.1 Recurrence and Step

Each iteration of the recurrence produces one digit of the result, most-significant digit first. Let us call $S[j]$ the value of the result after j iterations, that is,

$$S[j] = \sum_{i=0}^{j} s_i r^{-i} \tag{6.2}$$

The digit s_0 should be 1 for $\rho < 1$ to represent a result value greater than ρ; it can be either 1 or 0 for $\rho = 1$.

The final result is then

$$s = S[n] = \sum_{i=0}^{n} s_i r^{-i} \tag{6.3}$$

Operand	$\frac{1}{4} \le x < 1$
Result	$\frac{1}{2} \le s < 1$
Result after j iterations	$S[j] = \sum_{i=0}^{j} s_i r^{-i}$
Result-digit set	$s_i \in \{-a, \ldots, -1, 0, 1, \ldots, a\}$
Redundancy factor	$\rho = a/(r-1)$ $\frac{1}{2} < \rho \le 1$
Selection interval for $s_{j+1} = k$	$L_k[j] \le rw[j] \le U_k[j]$
Estimate of redundant shifted residual	$\widehat{y}$ with t fractional bits
Selection constants	$m_k(i)$ with c fractional bits
Result estimate	$\widehat{S}[j]$ with δ fractional bits

TABLE 6.1 Summary of definitions.

and the result has to be correct for n-digit precision; that is,

$$|x^{1/2} - s| < r^{-n} \tag{6.4}$$

The use of absolute value allows positive and negative remainders necessary for efficient implementation. We define an error function ϵ so that its value after j steps (iterations) is

$$\epsilon[j] = x^{1/2} - S[j] \tag{6.5}$$

As in division, since the minimum (maximum) digit value is $-a$ (a), we get[1]

$$-\rho r^{-j} < \epsilon[j] < \rho r^{-j} \tag{6.6}$$

Introducing (6.5) in (6.6) and transforming to eliminate the square root operation (add $S[j]$ and obtain the square), we get

$$\rho^2 r^{-2j} - 2\rho r^{-j} S[j] + S[j]^2 < x < \rho^2 r^{-2j} + 2\rho r^{-j} S[j] + S[j]^2 \tag{6.7}$$

Subtracting $S[j]^2$ we obtain

$$\rho^2 r^{-2j} - 2\rho r^{-j} S[j] < x - S[j]^2 < \rho^2 r^{-2j} + 2\rho r^{-j} S[j] \tag{6.8}$$

That is, $S[j]$ is computed such that $x - S[j]^2$ is bounded according to (6.8).

We now define a residual (or scaled partial remainder) w so that

$$w[j] = r^j (x - S[j]^2) \tag{6.9}$$

1. As in division, can make $\le$ for $\rho < 1$.

From (6.8) the bounds on the residual are

$$-2\rho S[j] + \rho^2 r^{-j} < w[j] < 2\rho S[j] + \rho^2 r^{-j} \qquad \textbf{6.10}$$

and the initial condition is

$$w[0] = x - S[0]^2 = x - s_0 \quad \text{for } s_0 = 0 \text{ or } 1 \qquad \textbf{6.11}$$

In terms of the residual we obtain the recurrence

$$w[j+1] = r\,w[j] - 2S[j]s_{j+1} - s_{j+1}^2 r^{-(j+1)} \qquad \textbf{6.12}$$

Expression (6.12) is the basic recurrence on which the square root algorithms are based. The result digit is chosen, in a way that satisfies the bounds (6.10) for $w[j+1]$, by the function

$$s_{j+1} = SEL_{SQR}(\hat{y}[j], \hat{S}[j]) \qquad \textbf{6.13}$$

where $\hat{y}[j]$ and $\hat{S}[j]$ are estimates of $rw[j]$ and $S[j]$, respectively.

Each square root iteration consists of four subcomputations (Figure 6.1(a)):

1. An arithmetic left shift of $w[j]$ by one position to produce $rw[j]$.
2. Determination of the result digit s_{j+1} using the result-digit selection function SEL_{SQR}.
3. Formation of the adder input

$$F[j] = -\left(2S[j]s_{j+1} + s_{j+1}^2 r^{-(j+1)}\right) \qquad \textbf{6.14}$$

4. Addition of $F[j]$ to $rw[j]$ to produce $w[j+1]$. As in division, to have a fast iteration, a redundant adder is used for this addition. This adder can be of the signed-digit or of the carry-save type. Since the digits of $S[j]$ are produced in signed-digit form, if a carry-save adder is used in the recurrence, it is necessary to convert the signed-digit form to two's complement form by means of a variant of the on-the-fly conversion.

The four subcomputations are executed in sequence as indicated in the timing diagram of Figure 6.1(b). Note that no time has been allocated for the arithmetic shift since it is performed by suitable wiring. Moreover, the relative magnitudes of the delay of each of the components depend on the specific implementation.

As in division, different specific versions are possible, depending on the radix, the redundancy factor, the type of representation of the residual, and the result-digit selection function.

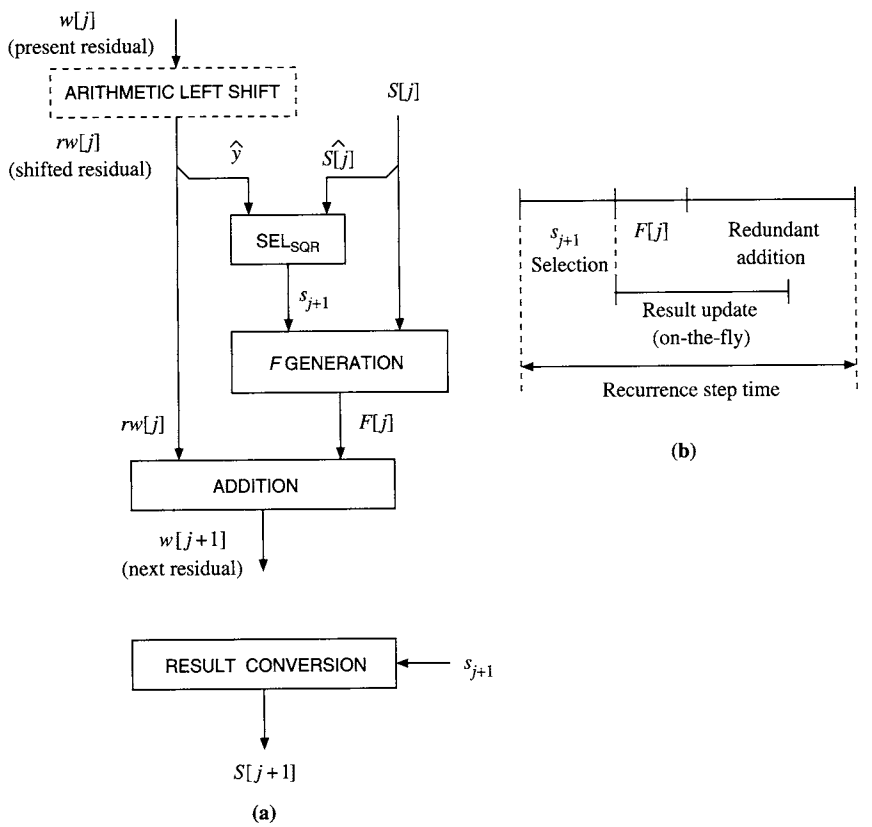

FIGURE 6.1 (a) Components of square root step. (b) Timing.

6.2 Generation of Adder Input $F[j]$

As part of the implementation of the recurrence (6.12), it is necessary to form the adder input F with value

$$F[j] = -2S[j]s_{j+1} - s_{j+1}^2 r^{-(j+1)} \qquad \textbf{6.15}$$

so that

$$w[j+1] = rw[j] + F[j]$$

Since the digit of the result is produced in a signed-digit form, the partial result $S[j]$ is also in this form. Depending on the type of adder, $S[j]$ has to be converted to adapt to the adder. In particular, for the case of a carry-save adder the input F has to be in two's complement representation. The conversion is done on-the-fly using a variation of the scheme presented in Chapter 5. It requires that two conditional forms $A[j]$ and $B[j]$ are kept, such that

$$A[j] = S[j] \tag{6.16}$$

$$B[j] = S[j] - r^{-j} \tag{6.17}$$

These forms are updated with each result digit as follows:

$$A[j+1] = \begin{cases} A[j] + s_{j+1} r^{-(j+1)} & \text{if } s_{j+1} \geq 0 \\ B[j] + (r - |s_{j+1}|) r^{-(j+1)} & \text{otherwise} \end{cases} \tag{6.18}$$

$$B[j+1] = \begin{cases} A[j] + (s_{j+1} - 1) r^{-(j+1)} & \text{if } s_{j+1} > 0 \\ B[j] + (r - 1 - |s_{j+1}|) r^{-(j+1)} & \text{otherwise} \end{cases} \tag{6.19}$$

In a sequential implementation this conversion requires two registers for A and B, appending of one digit, and loading. For controlling this appending and loading, a shift register K is used, containing a moving 1. This implementation is shown in Figure 6.2.

In terms of these forms, the value of F is given by the following expressions. For $s_{j+1} > 0$:

$$F[j] = -2S[j] s_{j+1} - s_{j+1}^2 r^{-(j+1)} = -(2A[j] + s_{j+1} r^{-(j+1)}) s_{j+1} \tag{6.20}$$

For $s_{j+1} < 0$:

$$\begin{aligned} F[j] &= 2S[j]|s_{j+1}| - s_{j+1}^2 r^{-(j+1)} = 2(B[j] + r^{-j})|s_{j+1}| - s_{j+1}^2 r^{-(j+1)} \\ &= \left(2B[j] + (2r - |s_{j+1}|) r^{-(j+1)}\right)|s_{j+1}| \end{aligned}$$

Note that these expressions are implemented by concatenation and multiplication by one radix-r digit. The implementation is especially simple for radix 2 and radix 4 (with digit set $\{-2, \ldots, 2\}$), as shown in the examples of Section 6.3.1.

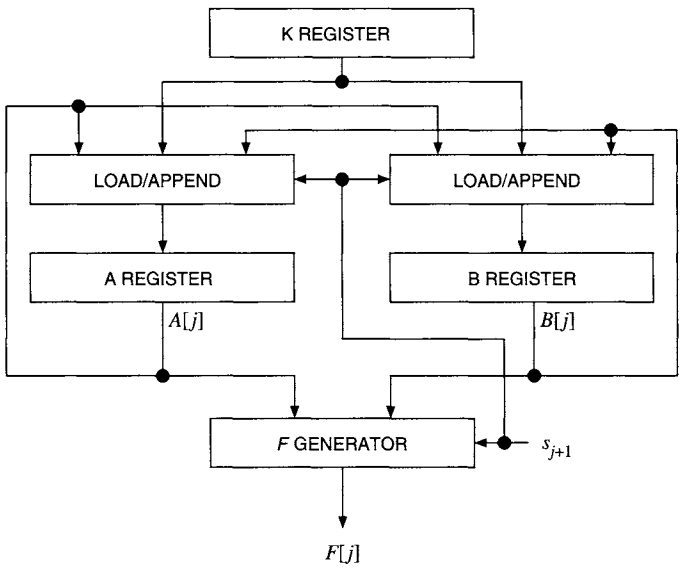

FIGURE 6.2 Network for generating F. (Adapted from (Ercegovac and Lang, 1990).)

6.3 Overall Algorithm, Implementation, and Timing

The overall algorithm is shown in Figure 6.3 and its implementation at the block-diagram level in Figure 6.4. The cycle time is

$$T_{cycle} = t_{SEL} + t_{F\text{-}GEN} + t_{ADD} + t_{load} \qquad \textbf{6.21}$$

6.3.1 Examples of Implementations

We now describe two example implementations, one radix-2 and one radix-4. The corresponding selection functions are derived in Section 6.6.

Radix-2 Square Root with Carry-Save Adder

In this case the quotient-digit set is $\{-1, 0, 1\}$ with $\rho = 1$. We choose to make $s_0 = 0$, resulting in the initial condition

$$w[0] = x$$

1. **[Initialize]** (*all assignments in parallel*)
   $w[0] \leftarrow x - s_0$; $s_0 = 0$ for $\rho = 1$ and $s_0 = 1$ for $\rho < 1$
   $A[0] \leftarrow s_0.000...000$;
   $B[0] \leftarrow (1 - s_0).000...000$; since B[0] = A[0] − 1
   $K[0] \leftarrow 0.100...000$;
2. **[Recurrence]** (*all assignments in parallel*)
   for $j = 0 \ldots n$
   $s_{j+1} = SELECT(\widehat{y}[j], \widehat{S}[j])$
   $F[j] = f(A[j], B[j], s_{j+1})$ (expressions (6.20)
   $w[j+1] \leftarrow rw[j] + F[j]$;
   $A[j+1] \leftarrow g_a(A[j], B[j], K[j], s_{j+1})$; (expressions (6.18))
   $B[j+1] \leftarrow g_b(A[j], B[j], K[j], s_{j+1})$; (expressions (6.19))
   $K[j+1] \leftarrow$ shift-right($K[j]$)
   end for
3. **[Termination]** *Correct result* (*same as for division*)

FIGURE 6.3 Square root algorithm.

Since $s_j \in \{-1, 0, 1\}$ we have $s_j^2 = |s_j|$ and hence the recurrence is

$$w[j+1] = 2w[j] - 2S[j]s_{j+1} - 2^{-(j+1)}|s_{j+1}| \qquad \textbf{6.22}$$

resulting in

$$F[j] = -(2S[j]s_{j+1} + 2^{-(j+1)})|s_{j+1}| \qquad \textbf{6.23}$$

As discussed before, we use the conditional forms A and B for the conversion of $S[j]$ to two's complement representation and for the formation of $F[j]$. However, since in this case the only nonzero values of s_{j+1} are 1 and −1, we define

$F[j]$	Value	Bit-String
$F_1[j]$	$-2S[j] - 2^{-(j+1)}$	$\{\overline{2S[j]}\}, 1, 1_{j+1}, 0, \ldots, 0$
$F_{-1}[j]$	$2S[j] - 2^{-(j+1)}$ $= 2S[j] - 2^{-(j-1)} + 3 \times 2^{-(j+1)}$	$\{2S[j] - 1\}, 1, 1_{j+1}, 0, \ldots, 0$

TABLE 6.2 F_1 and F_{-1} forms for radix 2.

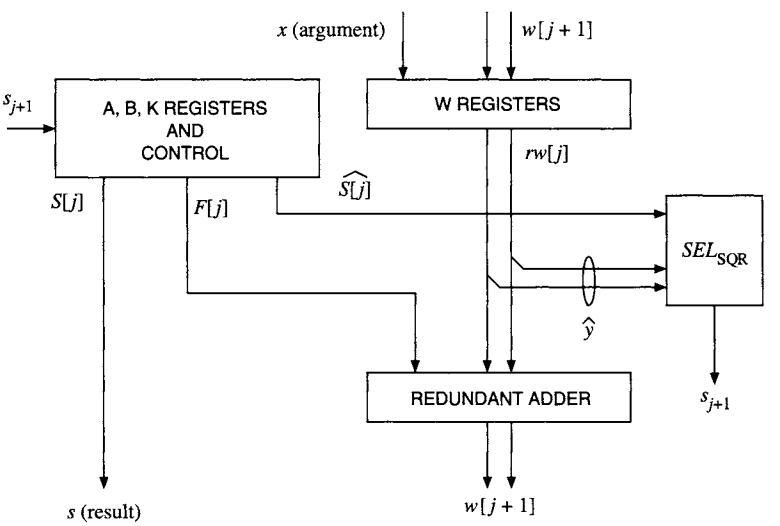

FIGURE 6.4 Block diagram of digit recurrence square root scheme. (Adapted from Ercegovac and Lang (1990).)

F_1 and F_{-1} so that[2]

$$F[j] = \begin{cases} F_1[j] & \text{if } s_{j+1} = 1 \\ 0 & \text{if } s_{j+1} = 0 \\ F_{-1}[j] & \text{if } s_{j+1} = -1 \end{cases} \qquad 6.24$$

Table 6.2 describes the values and the corresponding bit-strings of the F_1 and F_{-1} forms. In this table, $\{\overline{2S[j]}\}$ is the bit-string produced by the bit-complement of

2. In this case, A and B are used only for the conversion of $S[j]$.

s_{j+1}	$F_1[j+1]$	$F_{-1}[j+1]$
1	$(X, 0, 1, 1_{j+2}, 0, \ldots, 0)$	$(\overline{X}, 0, 1, 1_{j+2}, 0, \ldots, 0)$
0	$(X, 1, 1, 1_{j+2}, 0, \ldots, 0)$	$(Y, 1, 1, 1_{j+2}, 0, \ldots, 0)$
-1	$(\overline{Y}, 0, 1, 1_{j+2}, 0, \ldots, 0)$	$(Y, 0, 1, 1_{j+2}, 0, \ldots, 0)$

TABLE 6.3 Updating of F_1 and F_{-1} forms for radix 2.

$2S[j]$ in radix-2 conventional representation, and $\{2S[j] - 1\}$ is the bit-string of $2(S[j] - 2^{-j})$. The subscript of 1_{j+1} indicates that the corresponding 1 is in the $(j+1)$th position.

The bit-strings $F_1[j] = (X, 1, 1_{j+1}, 0, \ldots, 0)$ and $F_{-1}[j] = (Y, 1, 1_{j+1}, 0, \ldots, 0)$ are updated according to Table 6.3. The initial conditions are $F_1[0] = -2S[0] - 2^{-1} = -0.5 = 111.1$ and $F_{-1}[0] = 2S[0] - 2^{-1} = -0.5 = 111.1$.[3]

The updating of the registers is controlled by the control register K with contents

$$K[j] = (1, 1, 1, \ldots, 1, 1_j, 0, \ldots, 0) \tag{6.25}$$

with the initial condition $K[0] = 1111.0000...0$.

The result-digit selection developed in Section 6.6 is

$$s_{j+1} = \begin{cases} 1 & \text{if } 0 \le \widehat{y} \le 3 \\ 0 & \text{if } \widehat{y} = -1 \\ -1 & \text{if } -5 \le \widehat{y} \le -2 \end{cases} \tag{6.26}$$

where $\widehat{y}$ is an estimate of $2w[j]$ with $t = 0$ fractional bits.

EXAMPLE 6.1 We show an example of execution of the radix-2 algorithm for $x = 0.10110111$ in Figure 6.5. ■

Radix-4 with Carry-Save Adder

We develop the radix-4 case with carry-save adder and result-digit set $\{-2, -1, 0, 1, 2\}$.

3. The three integer bits are needed since $\widehat{y}$ requires four integer bits, so $w[j+1]$ requires three integer bits.

$2w[0]^+$	0001.01101110		$S[0] = 0$
	0000.00000000	$\widehat{y} = 1$ $s_1 = 1$	$S[1] = 0.1$
$F_1[0]$	111.10000000		
$w[1]$	110.11101110		
	010.00000000		
$2w[1]^+$	1101.11011100		
	0100.00000000	$\widehat{y} = 1$ $s_2 = 1$	$S[2] = 0.11$
$F_1[1]$	110.11000000		
$w[2]$	111.00011100		
	001.10000000		
$2w[2]^+$	1110.00111000		
	0011.00000000	$\widehat{y} = 1$ $s_3 = 1$	$S[3] = 0.111$
$F_1[2]$	110.01100000		
$w[3]$	011.01011000		
	100.01000000		
$2w[3]^+$	0110.10110000		
	1000.10000000	$\widehat{y} = -2$ $s_4 = -1$	$S[4] = 0.1101$
$F_{-1}[3]$	001.1011000		
$w[4]$	111.10000000		
	001.01100000		
$2w[4]^+$	1111.00000000		
	0010.11000000	$\widehat{y} = 1$ $s_5 = 1$	$S[5] = 0.11011$
$F_1[4]$	110.01011000		
$w[5]$	011.10011000		
	100.10000000		

$^+$ only three integer bits in the recurrence because of the range of $w[j]$.

FIGURE 6.5 Example of Radix-2 algorithm execution.

	$F[j]$		
s_{j+1}	Value (in terms of $S[j]$)	Value (in terms of $A[j]$ and $B[j]$)	Bit-string
0	0	0	$0\ldots00000$
1	$-2S[j]-4^{-(j+1)}$	$-2A[j]-4^{-(j+1)}$	$\overline{a}\ldots\overline{aa}111$
2	$-4S[j]-4\times4^{-(j+1)}$	$-4A[j]-4\times4^{-(j+1)}$	$\overline{a}\ldots\overline{a}1100$
−1	$2S[j]-4^{-(j+1)}$	$2B[j]+7\times4^{-(j+1)}$	$b\ldots bb111$
−2	$4S[j]-4\times4^{-(j+1)}$	$4B[j]+12\times4^{-(j+1)}$	$b\ldots b1100$

TABLE 6.4 Generation of $F[j]$ for radix 4.

The recurrence for this case is

$$w[j+1]=4w[j]-\left(2S[j]s_{j+1}+4^{-(j+1)}s_{j+1}^2\right)=4w[j]+F[j] \quad \textbf{6.27}$$

The adder input $F[j]$ is formed as discussed in Section 6.2. The resulting bit-strings are given in Table 6.4, where $a\ldots aa$ and $b\ldots bb$ are the bit-strings representing $A[j]$ and $B[j]$, respectively (shifted one position). As in the radix-2 case, the trailing location of the string is controlled by the moving 1 of register K.

The result digit is a function of $\widehat{y}$, the carry-save $rw[j]$ truncated to four fractional bits, and $\widehat{S}[j]$, the partial result also truncated to four fractional bits. As in division, the selection function is defined in terms of selection constants $m_k(i)$ such that

$$s_{j+1}=k \text{ if } m_k(i)\le\widehat{y}<m_{k+1}(i) \text{ and } \widehat{S}[j]=2^{-1}+i\times2^{-4} \quad \textbf{6.28}$$

A selection function is given in Table 6.5. Since the selection constants are all multiples of 2^{-3}, only three fractional bits of $\widehat{y}$ are used for selection, as shown in Figure 6.6. To use the same selection function for all values of j, the following transformation is performed:

$$(\widehat{S}_1,\widehat{S}_2,\widehat{S}_3,\widehat{S}_4)=\begin{cases}(1,1,0,-) & \text{if } (j=0)\\ (1,1,1,1) & \text{if } (A_0=1) \text{ and } (j\neq0)\\ (1,A_2,A_3,A_4) & \text{if } (A_0=0) \text{ and } (j\neq0)\end{cases} \quad \textbf{6.29}$$

where $(A_0, A_1, A_2, A_3, A_4)$ are the most-significant bits of A, the conventional representation of $S[j]$. Since for $j=0$, $A_0=1$ and $A_2=A_3=A_4=0$, we obtain the implementation of Figure 6.6.

i	0	1	2	3	4	5	6	7
$\widehat{S}j$	$\frac{8}{16}$	$\frac{9}{16}$	$\frac{10}{16}$	$\frac{11}{16}$	$\frac{12}{16}$	$\frac{13}{16}$	$\frac{14}{16}$	$\frac{15}{16}$
$m_2(i)^+$	12	14	16	16	18	20	20	22
$m_1(i)^+$	4	4	4	4	6	6	8	8
$m_0(i)^+$	−4	−5	−6	−6	−6	−8	−8	−8
$m_{-1}(i)^+$	−13	−14	−16	−17	−18	−20	−22	−23

+: real value is given value divided by eight.

TABLE 6.5 Selection function for radix-4 square root.

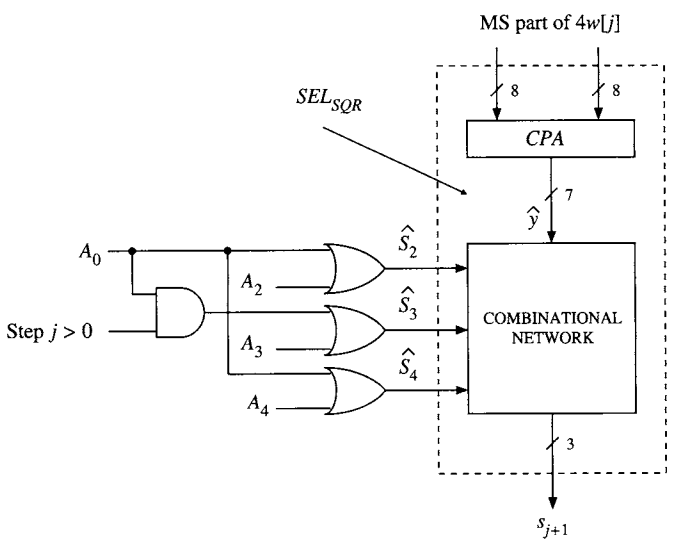

FIGURE 6.6 Selection function for $r = 4$. (Adapted from Ercegovac and Lang (1990).)

The overall algorithm follows directly the algorithm given earlier in this chapter with $r = 4$, and it is not repeated here. The cycle time is

$$\begin{aligned} T_{cycle} = {} & t_{SEL} + && \text{(8-bit CPA + 10-input q-sel)} \\ & t_{F\text{-}GEN} + && \text{(4-to-1 multiplexer)} \\ & t_{HA} + && \text{(HA part of [3:2] carry-save adder)} \\ & t_{load} && \text{(register loading)} \end{aligned}$$

This is comparable to the cycle time of a radix-4 division with carry-save adder.

$w[0] = x - 1 = 1.10110111$

			$S[0] = 1$
$4w[0]^+$	1110.11011100	$\widehat{S} = .1101$	
	0000.00000000	$\widehat{y} = 1110.110 = -10/8$ $s_1 = -1$	$S[1] = 0.11$
$F_{-1}[0]^+$	001.11000000		
$w[1]$	11.00011100		
	01.10000000		
$4w[1]^+$	1100.01110000	$\widehat{S} = .1100$	
	0110.00000000	$\widehat{y} = 0010.011 = 19/8$ $s_2 = 2$	$S[2] = 0.1110$
$F_2[1]^+$	100.11000000		
$w[2]$	10.10110000		
	00.10000000		
$4w[2]^+$	1010.11000000	$\widehat{S} = .1110$	
	0010.00000000	$\widehat{y} = 1100.110 = -26/8$ $s_3 = -2$	$S[3] = 0.110110$
$F_{-2}[2]^+$	011.01110000		
$w[3]$	11.10110000		
	00.10000000		

+ only two integer bits used in recurrence, because the range of $|w[j]| < 2$.

FIGURE 6.7 Example of radix-4 algorithm execution.

EXAMPLE 6.2 We give an example of execution of the radix-4 algorithm for $x = 0.10110111$ in Figure 6.7. ■

6.4 Combination of Division and Square Root

Since the recurrences of the square root and division operations have many similarities, it is possible to implement a combined unit that performs both operations. We now describe such a unit for radix 2; a generalization to higher radices is possible. Table 6.6 shows the operand and result ranges.

Operation	Operand 1	Operand 2	Result
Division	Dividend $\frac{1}{4} \le x < \frac{1}{2}$*	Divisor $\frac{1}{2} \le d < 1$	Quotient $\frac{1}{4} < q < 1$
Square root	$\frac{1}{4} \le x < 1$†		$\frac{1}{2} \le s < 1$

* Because of initial condition.
† To accommodate odd exponents.

TABLE 6.6 Operands and results ranges for combined unit.

	Division	Square Root
Recurrence $w[j+1] =$	$2w[j] - dq_{j+1}$ $\lvert w[j] \rvert < 1$ $w[0] = x/2$	$2w[j] - S[j]s_{j+1} - s_{j+1}^2 2^{-(j+2)}$ $\lvert w[j] \rvert < 1$ $w[0] = x/2$
Estimate fraction bits t	1	1
Selection		
m_1	0	
m_0	$-\frac{1}{2}$	
m_{-1}	$-\frac{5}{2}$	

TABLE 6.7 Algorithms.

Since the bound of the residual for square root is about twice that of division, to combine both recurrences it is convenient to modify the residual for square root so that

$$w[j](new) = 2^{-1}w[j](old) \qquad \textbf{6.30}$$

After making this modification (and calling $w[j](new)$ just $w[j]$), we get the algorithms described in Table 6.7. From this table we see that we can implement a generic recurrence of the form

$$w[j+1] = 2w[j] + F[j] \qquad \textbf{6.31}$$

	Division	Square Root
u_{j+1}	q_{j+1}	s_{j+1}
$F_1[j]$	$-d$	$-S[j] - 2^{-(j+2)}$
$F_{-1}[j]$	d	$S[j] - 2^{-(j+2)}$

TABLE 6.8 Correspondence.

where

$$F[j] = \begin{cases} F_1[j] & \text{if } u_{j+1} = 1 \\ 0 & \text{if } u_{j+1} = 0 \\ F_{-1}[j] & \text{if } u_{j+1} = -1 \end{cases} \qquad \textbf{6.32}$$

and $F_1[j]$, $F_{-1}[j]$, and u_{j+1} are related to the operations by the correspondence of Table 6.8. Moreover, the result-digit selection function is $u_{j+1} = Sel(\widehat{y})$, where $\widehat{y}$ is an estimate of $2w[j]$ obtained by assimilating the carry-save representation up to one fractional bit. From Table 6.7 we see that the selection function can be made to be the same for both operations (and corresponds to that described for division in Chapter 5).

The generation of the inputs to the adder is performed as described in this chapter for square root,[4] whereas for division the registers have to be loaded as indicated by the correspondence of Table 6.8. Conversion of the result is performed as described in Chapter 5. Figure 6.8 shows a block diagram and the cycle time of the combined implementation.

6.5 Integer Square Root

Integer square root (for unsigned operands) has an integer operand $0 \leq x \leq 2^n - 1$ and produces an integer result s such that

$$s = \lfloor x^{1/2} \rfloor \qquad \textbf{6.33}$$

To use the staircase selection functions discussed in this chapter, it is necessary that the result be in the range $[\frac{1}{2}, 1)$. This is achieved by shifting the operand m

4. Because of the modification of the residual for square root, now the values of $F[j]$ are one half of those reported in Table 6.2.

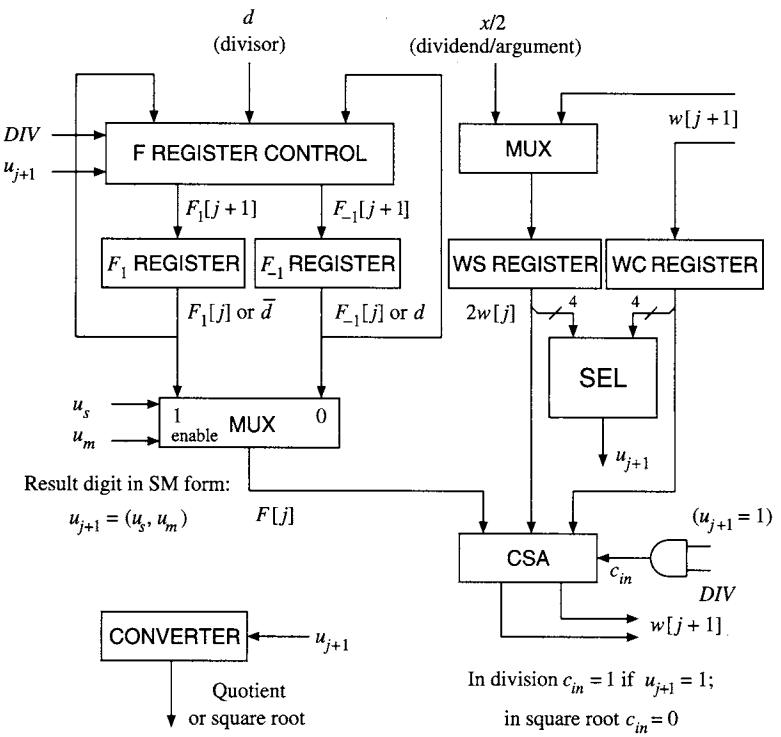

FIGURE 6.8 Overall implementation of the radix-2 combined unit. (Adapted from Ercegovac and Lang (1991).)

bits (and placing the binary point on the left) so that

$$x^* = 2^m x(2^{-n}) = 2^{-(n-m)}x \tag{6.34}$$

producing

$$s^* = 2^{-(n-m)/2}s \tag{6.35}$$

with $\frac{1}{2} \leq s^* < 1$. Then

$$s = 2^{(n-m)/2}s^* \tag{6.36}$$

To obtain s from s^* by shifting, it is necessary that $n - m$ be even. Consequently, $\frac{1}{4} \leq x^* < 1$.

The number of bits of the integer square root is $(n - m)/2$. Consequently, the number of iterations required to obtain these bits is

$$NI = \lceil (n - m)/2k \rceil \qquad \textbf{6.37}$$

where $r = 2^k$ is the radix of the result digit.

The result has to be aligned to the integer position. This can be done by placing the digits in the correct final position or placing the digits aligned to the left (to combine with fractional square root) and then performing a right shift of $(n - m)/2$ bits.

EXAMPLE 6.3 We now show an example of integer square root, radix 4 with $\rho = \frac{2}{3}$ and 8-bit operand. Consider the case $x = 27$ with binary representation $x = 00011011$.

Since $n = 8$ is even, we shift $m = 2$ bits and produce $x^* = .01101100$. The number of bits of the integer result is $(8 - 2)/2 = 3$. Consequently, two radix-4 iterations are necessary.

The radix-4 square root algorithm uses $S[0] = 1$ and $w[0] = x^* - 1 = 1.01101100$ (two's complement). As selection function we use Table 6.5 for carry-save representation of the residual.

The iterations are as follows (for simplicity we show the residual in conventional representation):

$w[0] =$	11.01101100		$S[0] = 1$
$4w[0] =$	1101.10110000	$\hat{y} = 1101.1011$ $s_1 = -2$	$S[1] = 0.10(0.5)$
$+F =$	011.00000000		
$w[1] =$	00.10110000		
$4w[1] =$	0010.11000000	$\hat{y} = 0010.1100$ $s_2 = 2$	$S[2] = 0.1010$
($w[2]$ not needed)			

So, $s = 2^3(0.101) = 101 = (5)_{10}$. ■

6.6 Result-Digit Selection

We now develop the details of the selection function design. We give examples for radix 2 and radix 4.

6.6.1 Selection Intervals

As in division, two fundamental conditions must be satisfied by a result-digit selection: *containment* and *continuity*. These conditions determine the selection intervals and the selection constants. We now develop expressions for these selection intervals. The bounds of the residual $w[j]$ (called $\min(w[j])$ and $\max(w[j])$ below) are defined by (6.10). Note that these bounds depend on j, whereas in division they are constants. From recurrence (6.12), the interval of $rw[j]$ where $s_{j+1} = k$ can be selected is

$$\begin{aligned} U_k[j] &= \max(w[j+1]) + 2S[j]k + k^2 r^{-(j+1)} \\ &= 2\rho S[j+1] + \rho^2 r^{-(j+1)} + 2S[j]k + k^2 r^{-(j+1)} \\ L_k[j] &= \min(w[j+1]) + 2S[j]k + k^2 r^{-(j+1)} \\ &= -2\rho S[j+1] + \rho^2 r^{-(j+1)} + 2S[j]k + k^2 r^{-(j+1)} \end{aligned} \tag{6.38}$$

Since $S[j+1] = S[j] + k r^{-(j+1)}$, we get

$$\begin{aligned} U_k[j] &= 2S[j](k+\rho) + (k+\rho)^2 r^{-(j+1)} \\ L_k[j] &= 2S[j](k-\rho) + (k-\rho)^2 r^{-(j+1)} \end{aligned} \tag{6.39}$$

As in division, variations of Robertson's diagram and P-D plot can be used to represent the selection interval bounds.

The continuity condition

$$U_{k-1} \geq L_k \tag{6.40}$$

results in

$$(2\rho - 1)(2S[j] + (2k-1)r^{-(j+1)}) \geq 0 \tag{6.41}$$

The overlap between consecutive selection intervals is given by

$$U_{k-1} - L_k = (2\rho - 1)(2S[j] + (2k-1)r^{-(j+1)}) \tag{6.42}$$

As in division, this overlap is used to simplify the selection function.

Note that the bounds, selection intervals, and the overlap depend on j. This is in contrast to division, in which they are independent of j. Since now the selection intervals depend on three parameters, namely, $S[j]$, j, and k, the notation becomes more complicated. In general, we use parentheses for $S[j]$, square brackets for j, and subscripts for k; however, we skip any one of these if it is unnecessary

in a particular context. The dependence on j makes the implementation more complicated than in division, as discussed later.

6.6.2 Staircase Selection Using Redundant Adder

The basic relations that allow the use of estimates of the residual in the result-digit selection are identical to those we developed for division in the previous chapter. Since in square root the result-digit selection depends on the partial result $S[j]$ instead of on the divisor, the corresponding expressions are obtained by replacing d by $S[j]$. Instead of repeating the development of these relations, we ask the reader to refer to Chapter 5. On the other hand, some relations are different since they depend on the specific form of the recurrence; we develop these relations here.

Since the use of a redundant adder increases the speed of the implementation with a small increase in complexity, we concentrate on this type of implementation.

We now determine a staircase result-digit selection function using an estimate of the partial result and an estimate of the shifted residual obtained by truncating the redundant form.

Estimate of $S[j]$

The estimate of the result used in the result-digit selection divides the range of the result $S[j]$ into intervals, as illustrated in Figure 6.9. Specifically, if δ is the number of fractional bits of estimate $\widehat{S}[j]$, then the value

$$\widehat{S}[j] = 2^{-1} + i \times 2^{-\delta} \quad 0 \leq i \leq 2^{\delta-1} - 1 \tag{6.43}$$

defines the interval I_i. Note that the value of $\widehat{S}[j]$ for $i = 0$ is 2^{-1}, since the result is normalized.

Since the result is being produced one digit per iteration in signed-digit form, several alternatives can be used to form the estimate. This is in contrast to

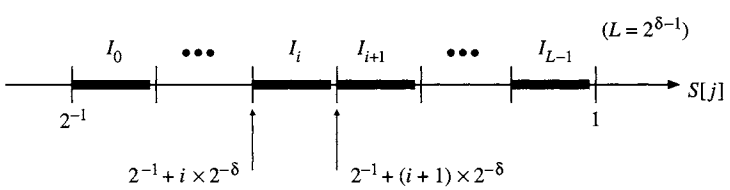

FIGURE 6.9 Generic intervals.

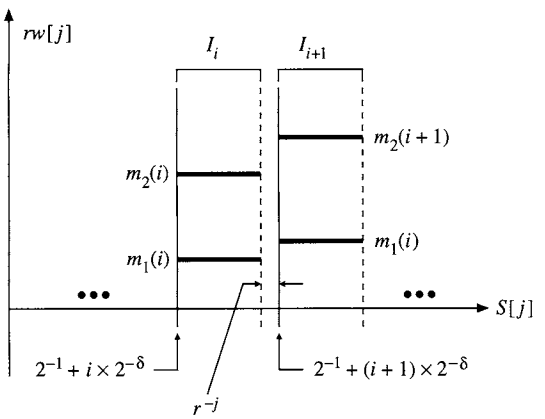

FIGURE 6.10 Selection intervals obtained by truncating conventional form.

division, where the estimate is always obtained by truncating the conventional representation of the divisor. The alternative considered here uses the truncated conventional representation of $S[j]$, which is obtained by the on-the-fly conversion. This is similar to the case of division, in which the divisor is in conventional representation. Therefore, if $\widehat{S}[j]$ is obtained by truncating $S[j]$ to δ fractional bits, then as shown in Figure 6.10, the ith interval I_i is defined by

$$\widehat{S}[j] = 2^{-1} + i \times 2^{-\delta} \leq S[j] < 2^{-1} + (i+1) \times 2^{-\delta} \qquad \textbf{6.44}$$

However, since $S[j]$ has j fractional radix-r digits, the upper bound of the interval is restricted so that I_i corresponds to

$$\widehat{S}[j] = 2^{-1} + i \times 2^{-\delta} \leq S[j] \leq 2^{-1} + (i+1) \times 2^{-\delta} - r^{-j} \qquad \textbf{6.45}$$

This restriction of the upper limit of the interval is significant for small j.

Determination of the Selection Constants $m_k(i)$

We now summarize the condition required for the result-digit selection, which is the same as for division. We then apply it to the square root case.

Using the estimate of the result, the result-digit selection is described by the set of selection constants

$$\{m_k(i) \mid 0 \leq i \leq 2^{\delta-1} - 1, \quad k \in \{-a, \ldots, -1, 0, 1, \ldots, a\}\} \qquad \textbf{6.46}$$

That is, there is one selection constant per interval and per value of the result digit. Using these selection constants, the result-digit selection is defined by

$$s_{j+1} = k \quad \text{if } m_k(i) \leq \widehat{y} < m_{k+1}(i) \text{ and } \widehat{S}[j] = 2^{-1} + i \times 2^{-\delta} \qquad \textbf{6.47}$$

where $\widehat{y}$ is an estimate of the shifted residual $rw[j]$ obtained by truncating the redundant form to t fractional bits.

As discussed in Chapter 5, the use of the estimate by truncation of the redundant form of $y = rw[j]$ results in the following condition for the selection constants:

$$\max(\widehat{L}_k(I_i)) \leq m_k(i) \leq \min(\widehat{U}_{k-1}(I_i)) \qquad \textbf{6.48}$$

Moreover, for the carry-save representation case[5]

$$\begin{aligned} \widehat{U}_{k-1} &= \lfloor U_{k-1} - 2^{-t} \rfloor_t \\ \widehat{L}_k &= = \lceil L_k \rceil_t \end{aligned} \qquad \textbf{6.49}$$

From these expressions, a feasible result-digit selection requires that

$$\min(\lfloor U_{k-1}(I_i) \rfloor_t) - \max(\lceil L_k(I_i) \rceil_t) \geq 2^{-t} \qquad \textbf{6.50}$$

Since L_k and U_{k-1} depend on j, the iteration index, the selection constants can, in general, be different for different j.

We now apply expressions (6.48) to the square root case. For this, we use the expressions (6.39) for the selection interval and expression (6.45) for the definition of the interval I_i. Consequently, for $k > 0$, the minimum U is produced at the lower end of the interval I_i and the maximum L at the upper end of the interval. Therefore,

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - 1 + \rho) \\ &\quad + (k - 1 + \rho)^2 r^{-(j+1)} \\ \max(L_k(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-\delta})(k - \rho) \\ &\quad + (k - \rho)(k - \rho - 2r) r^{-(j+1)} \end{aligned} \qquad \textbf{6.51}$$

5. For the expressions that hold for signed-digit representation, see Chapter 5 and Exercise 6.12.

For $k \leq 0$, the minimum U is produced at the upper end of the interval I_i and the maximum L at the lower end of the interval. Substituting we get

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-\delta})(k - 1 + \rho) \\ &\quad + (1 - \rho - k)(2r + 1 - \rho - k)r^{-(j+1)} \\ \max(L_k(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - \rho) + (k - \rho)^2 r^{-(j+1)} \end{aligned} \tag{6.52}$$

These expressions are used in expression (6.48) to determine the result-digit selection. However, since they depend on j, a different selection function might result for different j. To have a single selection function for all j we develop bounds that are independent of j.

For $k > 0$, in the expression for $\min(U_{k-1}(I_i))$ the term depending on j is always positive and approaching zero for large j. Therefore, in a bound this term can be neglected. Similarly, for $\max(L_k(I_i))$ the term depending on j is negative ($k - \rho - 2r < 0$) so it can also be neglected.

For $k \leq 0$, the same situation occurs for $\min(U_{k-1}(I_i))$, but now for $\max(L_k(I_i))j$ is positive, so it cannot be neglected, and for a bound we have to use its maximum value (for $j = 0$).

The corresponding expressions independent of j are as follows. For $k > 0$:

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - 1 + \rho) \\ \max(L_k(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-\delta})(k - \rho) \end{aligned} \tag{6.53}$$

For $k \leq 0$:

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-\delta})(k - 1 + \rho) \\ \max(L_k(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - \rho) + (k - \rho)^2 r^{-1} \end{aligned} \tag{6.54}$$

To determine if a single selection function is possible, we use (6.50).[6] The worst case is $i = 0$ and $k = -a + 1$, resulting in

$$2(2^{-1} + 2^{-\delta})(-a + \rho) - 2(2^{-1})(-a + 1 - \rho) - (-a + 1 - \rho)^2 r^{-1} \geq 2^{-t} \tag{6.55}$$

6. To simplify the analysis we consider the best case, which occurs when both $\min(U_{k-1})$ and $\max(L_k)$ are multiples of 2^{-t}.

This simplifies to

$$(2\rho - 1) - (\rho r - 1)^2 r^{-1} \geq 2^{-t} + 2 \times 2^{-\delta}(a - \rho) \qquad \textbf{6.56}$$

For $r = 2$ ($\rho = 1$), this results in $2^{-1} \geq 2^{-t}$, so that a single selection function is possible. On the other hand, there is no solution for $r \geq 4$ (because the term in the left-hand side becomes negative), so that no single selection function for all j exists. A possible alternative is to find a value J so that a single selection can be used for $j \geq J$ and then consider separately the cases for $j < J$.

For the case $j \geq J$, the same considerations given before produce the following. For $k > 0$ (same as (6.53)):

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - 1 + \rho) \\ \max(L_k(I_i)) &= 2(2^{-1} + (i + 1) \times 2^{-\delta})(k - \rho) \end{aligned} \qquad \textbf{6.57}$$

For $k \leq 0$ (expression (6.54) replacing -1 by $-(J + 1)$ in the last term):

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + (i + 1) \times 2^{-\delta})(k - 1 + \rho) \\ \max(L_k(I_i)) &= 2(2^{-1} + i \times 2^{-\delta})(k - \rho) + (k - \rho)^2 r^{-(J+1)} \end{aligned} \qquad \textbf{6.58}$$

Introducing these expressions in (6.50) (for the worst case $i = 0, k = -a + 1$), we get (similar to (6.56))

$$(2\rho - 1) - (\rho r - 1)^2 r^{-(J+1)} \geq 2^{-t} + 2 \times 2^{-\delta}(a - \rho) \qquad \textbf{6.59}$$

For specific values of r and ρ we can determine the values of J, t, and δ. These are then used to determine a selection function for $j \geq J$. For example, for $r = 4$ possible solutions are ($a = 2, \delta = 4, t = 3, J = 3$) and ($a = 3, \delta = 4, t = 3$, and $J = 1$).

The case $j < J$ has to be treated separately. A possible solution is to have a table lookup to produce $S[J]$ directly from x; that is, obtain the first J digits of the result from the table.[7] As we see later in the example for $r = 4$, a detailed analysis of the cases for $j < J$ eliminates the need for this initial table lookup.

Finally, the range of $\widehat{y}$ for carry-save residual is given by

$$\lfloor r \min(w[j]) - 2^{-t} \rfloor_t \leq \widehat{y} \leq \lfloor r \max(w[j]) \rfloor_t \qquad \textbf{6.60}$$

where $\min(w[j])$ and $\max(w[j])$ are the bounds of $w[j]$.

---

7. In this case, it is possible to obtain an algorithm in which J is not an integer.

6.6.3 Selection Function for Radix 2 with Carry-Save Adder

To obtain the result-digit selection, we begin by obtaining the values of L_k and U_k from expressions (6.39). That is,

$$\begin{aligned} U_1[j] &= 4S[j] + 2^{-j+1} & L_1[j] &= 0 \\ U_0[j] &= 2S[j] + 2^{-j-1} & L_0[j] &= -2S[j] + 2^{-j-1} \\ U_{-1}[j] &= 0 & L_{-1}[j] &= -4S[j] + 2^{-j+1} \end{aligned} \tag{6.61}$$

We first determine the intervals of $S[j]$ for the staircase function and the value of t. As shown in (6.56), a single interval over the whole range $\frac{1}{2} \le S[j] < 1$ exists so that the result-digit selection is independent of the result. Since $\rho = 1$, it is possible to make $s_0 = 0$, resulting in $S[0] = 0$. To obtain the selection constants according to (6.48), the extreme values are

$$\begin{aligned} \min\left(U_0\left(\frac{1}{2}\right), U_0(1)\right) &\ge \begin{cases} 2^{-1} & \text{for } j = 0 \quad (\text{since } S[0] = 0) \\ 1 & \text{for } j > 0 \quad \left(\text{since } S[j] \ge \frac{1}{2}\right) \end{cases} \\ \max\left(L_1\left(\frac{1}{2}\right), L_1(1)\right) &= 0 \\ \min\left(U_{-1}\left(\frac{1}{2}\right), U_{-1}(1)\right) &= 0 \\ \max\left(L_0\left(\frac{1}{2}\right), L_0(1)\right) &= -1 \end{aligned} \tag{6.62}$$

This last value is obtained as follows. We have

$$\max(L_0) = -\min(2S[j]) + 2^{-j-1} \tag{6.63}$$

This value is only important when we select $s_{j+1} = -1$. Since $s \ge \frac{1}{2}$, the selection $s_{j+1} = -1$ implies that $S[j] \ge \frac{1}{2} + 2^{-j}$ (so that $S[j+1] = S[j] - 2^{-j-1} \ge \frac{1}{2}$). Consequently,

$$\max(L_0) = -2\left(\frac{1}{2} + 2^{-j}\right) + 2^{-j-1} \le -1 \tag{6.64}$$

The selection constants and the value of t are now obtained so that they satisfy (6.48) and (6.49). A possible choice is

$$t = 0 \quad m_1 = 0 \quad m_0 = -1 \tag{6.65}$$

The only case in which (6.48) is not satisfied for this choice is for $j = 0$ because for this value of j, $\min(\widehat{U}_0) < \max(\widehat{L}_1)$. However, for $j = 0$ the residual is not in carry-save form, but in conventional form, so that it is sufficient to have $U_0 \geq L_0$. Moreover, since in the result $s \geq \frac{1}{2}$, the first bit of s is always 1 (which is obtained by the selection function, since $x > 0$).

The range of the shifted residual is given by (6.60). Since $|2w[j]| < 4S < 4$, the range of the estimate is $-5 \leq \widehat{y} \leq 3$. That is, 4 bits of the carry-save residual are needed for selection.

Consequently, the result-digit selection is

$$s_{j+1} = \begin{cases} 1 & \text{if } 0 \leq \widehat{y} \leq 3 \\ 0 & \text{if } \widehat{y} = -1 \\ -1 & \text{if } -5 \leq \widehat{y} \leq -2 \end{cases} \tag{6.66}$$

6.6.4 Selection Function for Radix 4 with Carry-Save Adder

We develop the radix-4 case with carry-save adder and result-digit set $\{-2, -1, 0, 1, 2\}$.

As indicated by expression (6.56) it is not possible to have a single selection function for all values of j. Consequently, we obtain a value of J so that a single selection function is possible for $j \geq J$ (and uses small t and δ). From (6.59) we get

$$\frac{1}{3} - \frac{25}{36}4^{-J} \geq 2^{-t} + \frac{8}{3}2^{-\delta} \tag{6.67}$$

A possible solution is $J = 3$, $t = 3$, $\delta = 4$. However, the corresponding selection function is not feasible (Exercise 6.13); this can occur because expression (6.59) has been obtained for the best case in which the limits of the interval are multiples of 2^{-t} (Exercise 6.13). Therefore, we now develop a result-digit selection with $t = 4$. It has to satisfy expression (6.48), that is,

$$\max(\lceil L_k(I_i) \rceil_4) \leq m_k(i) \leq \min(\lfloor U_{k-1}(I_i) - 2^{-4} \rfloor_4) \tag{6.68}$$

Moreover, the maximum and minimum in the interval are described by expressions (6.57) and (6.58) with $J = 3$ and $\delta = 4$. That is, for $k > 0$:

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + i \times 2^{-4})\left(k - \frac{1}{3}\right) \\ \max(L_k(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-4})\left(k - \frac{2}{3}\right) \end{aligned} \tag{6.69}$$

For $k \leq 0$:

$$\begin{aligned} \min(U_{k-1}(I_i)) &= 2(2^{-1} + (i+1) \times 2^{-4})\left(k - \frac{1}{3}\right) \\ \max(L_k(I_i)) &= 2(2^{-1} + i \times 2^{-4})\left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 4^{-4} \end{aligned} \tag{6.70}$$

Table 6.9 shows the limits of the intervals and possible selection constants. The following notation is used:

$$\begin{aligned} m\widehat{U}_{k-1}(i) &= \min(\lfloor U_{k-1}(I_i) - 2^{-4} \rfloor_4) \\ ML_k(i) &= \max(\lceil L_k(I_i) \rceil_4) \end{aligned} \tag{6.71}$$

Now we have to consider the case $j < 3$. One possible approach is to have a module to determine from a truncated x directly the value of $S[3]$ and to use it to perform the first iterations to determine $w[3]$. Another possibility is to analyze the case $j < 3$ and try to match the corresponding selection functions with that for $j \geq 3$. The particular choice of selection constants in Table 6.9 was made so that the same constants hold for all j.[8] The resulting selection function and its implementation are shown in Section 6.3.1.

The range of $\widehat{y}$ is obtained from (6.60). Since in this case we initialize to $S[0] = 1$, from (6.10) we obtain $\max(w[j]) = \max(w[1]) < \frac{4}{3} + \frac{1}{9} = \frac{13}{9}$ and $\min(w[j] > -\frac{4}{3}$. Conequently,

$$-\frac{88}{16} \leq \widehat{y} \leq \frac{92}{16} \tag{6.72}$$

Consequently, $\widehat{y}$ has four integer bits.

8. This is developed in Ercegovac and Lang (1990).

i	0	1	2	3
$\widehat{S}[j]^*$	8	9	10	11
$ML_2(i)\ m\widehat{U}_1(i)^*$	24 25	27 29	30 32	32 35
$m_2(i)^*$	**24**	**28**	**32**	**32**
$ML_1(i)\ m\widehat{U}_0(i)$	6 9	7 11	8 12	8 13
$m_1(i)$	**8**	**8**	**8**	**8**
$ML_0(i)\ m\widehat{U}_{-1}(i)$	−10 −7	−12 −8	−13 −9	−14 −9
$m_0(i)$	**−8**	**−10**	**−12**	**−12**
$ML_{-1}(i)\ m\widehat{U}_{-2}(i)$	−26 −25	−30 −28	−33 −31	−36 −33
$m_{-1}(i)$	**−26**	**−28**	**−32**	**−34**
i	4	5	6	7
$\widehat{S}[j]$	12	13	14	15, 16
$ML_2(i)\ m\widehat{U}_1(i)$	35 39	38 42	40 45	43 49
$m_2(i)$	**36**	**40**	**40**	**44**
$ML_1(i)\ m\widehat{U}_0(i)$	9 15	10 16	10 17	11 19
$m_1(i)$	**12**	**12**	**16**	**16**
$ML_0(i)\ m\widehat{U}_{-1}(i)$	−16 −10	−17 −11	−18 −11	−20 −12
$m_0(i)$	**−12**	**−16**	**−16**	**−16**
$ML_{-1}(i)\ m\widehat{U}_{-2}(i)$	−40 −36	−43 −39	−46 −41	−50 −44
$m_{-1}(i)$	**−36**	**−40**	**−44**	**−46**

* Real value is indicated value divided by 16.

TABLE 6.9 Selection function for $r = 4$ with carry-save adder ($j \geq 3$).

6.7 Exercises

Examples of Execution

6.1 Obtain the 8-bit square root of 144×2^{-8} using the following algorithms:

(a) Radix 2, $s_j \in \{-1, 0, 1\}$, conventional (nonredundant) residual
(b) Radix 2, $s_j \in \{-1, 0, 1\}$, carry-save residual
(c) Radix 4, $s_j \in \{-2, -1, 0, 1, 2\}$, carry-save residual

Characteristics of the Implementation

6.2 Design a 12-bit radix-2 square root unit at the gate level. You may use full-adders, multiplexers, registers, and gates. Provide the necessary design details to establish delays of critical paths. Use a carry-save adder to form residuals in redundant form. Give an estimate of the overall delay in gate delay units (t_g) and cost. Assume that a full-adder has a delay of $4t_g$, a 3-1 multiplexer $2t_g$, and a register load $3t_g$.

6.3 Develop the following two ways of generating the adder input F when a signed-digit adder is used:

(a) Use $S[j]$ in its original signed-digit form.
(b) Convert $S[j]$ to two's complement representation.

Discuss the design trade-offs in these alternatives.

6.4 **[Retiming of the recurrence].** As Exercise 5.7, but for square root.

6.5 **[Overlapped radix-2 stages].** Consider a radix-2 square root algorithm with the result-digit set $\{-1, 0, 1\}$ and redundant residuals in carry-save form. The result-digit selection is performed using the selection constants as described in the text. The cycle time is

$$t_{cycle} = t_{SEL_{SQRT}} + t_{buff} + t_{mux} + t_{HA} + t_{reg} = 4 + 1 + 1 + 1 + 2 = 10t_g$$

To reduce the total time, we propose to obtain all three possible values of s_{j+2}, corresponding to $s_{j+1} = -1, 0, 1$, and select the correct one once s_{j+1} is known, all of this in the same cycle. In other words, two result bits are generated per cycle.

(a) Design the network for the selection of s_{j+1} and s_{j+2}. Assume that the selection function is already implemented. Show all details; in particular, show the details of the conditional selection.
(b) Design the network to produce the next residual (assume 8 bits in the fractional part). Show all details.
(c) Determine the cycle time of the new scheme and compare the total time to obtain a result of 8 bits with the scheme described in the text. Discuss your findings.

Combination of Division and Square Root

6.6 Using the combined radix-2 algorithm described in the chapter, perform the following operations:

(a) Division of $x = 0.10110011$ by $d = 0.10001111$
(b) Square root of $x = 0.01110111$

6.7 Compare the cycle time and the cost of the combined radix-2 implementation with an implementation only for square root.

Integer Square Root

6.8 Perform the integer division algorithm for radix 4 with residual in carry-save representation for $x = 53$ and $d = 9$. Consider that operands and result are represented by 8-bit vectors.

6.9 Give an algorithm that combines fractional square root with integer square root. Show the combined implementation, highlighting the modules required to include integer square root.

Result-Digit Selection

6.10 Determine a result-digit selection function for a radix-2 algorithm in which a signed-digit adder is used in the recurrence.

6.11 Develop a bit-level implementation for the radix-2 selection function for a digit s_j represented in sign-and-magnitude by (s_s, s_m) and the residual is in carry-save representation.

6.12 For the selection function, consider using the truncated signed-digit representation or (equivalently) the fixed value of $S[\lceil \delta / \log(r) \rceil]$ (that is, the value of $S[j]$ immediately after at least δ fractional bits are produced). Show that in this case (for simplicity we consider the case in which exactly δ fractional bits are produced) the ith interval corresponds to the following range of $S[j]$:

$$2^{-1} + i \times 2^{-\delta} - \rho \times 2^{-\delta} < S[j] < 2^{-1} + i \times 2^{-\delta} + \rho \times 2^{-\delta}$$

and give a figure illustrating this method.

6.13 Following the derivation in Section 6.6.4, develop a radix-4 selection function for $J = 3, t = 3$, and $\delta = 4$. Determine the selection constants. How many fractional bits are required?

6.14 Determine a result-digit selection function for a radix-4 square root algorithm with $\rho = 1$ and carry-save residual. Give only the portion for selection of $s_{j+1} = 2$.

6.8 Further Readings

A survey of square-root algorithms is presented in Montuschi and Mezzalama (1990). Parts of this chapter are based on the monograph Ercegovac and Lang (1994) which presents an in-depth study of digit-recurrence methods for division and square root.

Early work in square rooting algorithms with redundancy in the result-digit set to maximize the number of zero digits is developed in Metze (1967).

Radix 2

Radix-2 algorithms and implementations are discussed in Taylor (1981) and Majerski (1985).

Radix 4 and Radix 8

Specific radix-4 implementations are presented in Fandrianto (1987) and Ercegovac and Lang (1989, 1991) and a radix-8 alternative in Fandrianto (1989). The radix-4 algorithm described in this chapter which uses a single selection function for all iterations, is developed in Ercegovac and Lang (1990).

Higher Radix

Higher radix algorithms and implementations are considered in Ciminiera and Montuschi (1990). A very-high-radix implementation, with prescaling and selection by rounding, is described in Lang and Montuschi (1992, 1999).

Combined Division and Square Root

Combined division/square root implementations are presented in a number of places (Taylor 1981; Zurawski and Gosling 1987; Fandrianto 1987, 1989; Ercegovac and Lang 1991; McQuillan and McCanny 1994; Prabhu and Zyner 1995). In Srinivas and Parhi (1999) the residuals are kept in signed-digit

representation and the result digit is overredundant, which simplifies the selection function. Self-timed designs are presented in Matsubara et al. (1995) and Guyot et al. (1996).

Reciprocal Square Root

Digit recurrence algorithms and implementations for reciprocal square root have recently been presented in Lang and Antelo (2001) and Takagi (2001). A very-high-radix version is presented in Antelo et al. (1998).

Low-Power Design

Radix-4 combined division/square root low-power units are described in Kuhlmann and Parhi (1998) and Nannarelli and Lang (1999). A self-timed low-power design for combined division/square-root is presented in Matsubara and Ide (1997).

Combinational Implementation

Combinational implementations of square root as linear arrays are reported in Majithia and Kitai (1971), Majithia (1972), Agrawal (1979), Cappuccino et al. (1998, 1999), and Corsonello et al. (2000). Combinational implementations of combined division/square root schemes are described in McQuillan et al. (1991, 1993).

Miscellaneous

A square root scheme for integers is presented in Hashemian (1990). A radix-2 square root implementation for field-programmable gate arrays is developed in Louie and Ercegovac (1993). Skipping of zero result digits is considered in Montuschi and Ciminiera (1993).

Area/Delay Analysis

Area/performance of square root units is discussed in Soderquist and Leeser (1996).

Verification

Verification of square root implementations is presented in Leeser and O'Leary (1995) and of combined multiplication, division, and square root implementations in Walter (1995).

6.9 Bibliography

Agrawal, D. P. (1979). High-speed arithmetic arrays. *IEEE Transactions on Computers*, C-28(3):215–24.

Antelo, E., T. Lang, and J. D. Bruguera (1998). Computation of $\sqrt{x/d}$ in a very high radix combined division/square-root unit with scaling. *IEEE Transactions on Computers*, 47(2):152–61.

Cappuccino, G., G. Cocorullo, P. Corsonello, and S. Perri (1999). High speed self-timed pipelined datapath for square rooting. *IEE Proceedings—Circuits, Devices and Systems*, 146(1):16–22.

Cappuccino, G., P. Corsonello, and G. Cocorullo (1998). High performance VLSI modules for division and square root. *Microprocessors and Microsystems*, 22(5):239–46.

Ciminiera, L., and P. Montuschi (1990). Higher radix square rooting. *IEEE Transactions on Computers*, 39(10):1220–31.

Corsonello, P., S. Perri, and G. Cocorullo (2000). Performance comparison between static and dynamic CMOS logic implementations of a pipelined square-rooting circuit. *IEE Proceedings—Circuits, Devices and Systems*, 147(6):347–55.

Ercegovac, M. D., and T. Lang (1989). Radix-4 square root without initial PLA. In *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, pages 162–68.

Ercegovac, M. D., and T. Lang (1990). Radix-4 square root without initial PLA. *IEEE Transactions on Computers*, C-39(8):1016–24.

Ercegovac, M. D., and T. Lang (1991). Module to perform multiplication, division and square root in systolic arrays for matrix computations. *Journal of Parallel and Distributed Computing*, 11(3):212–21.

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

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

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

Guyot, A., M. Renaudin, B. El Hassan, and V. Levering (1996). Self timed division and square-root extraction. In *Ninth International Conference on VLSI Design*, pages 376–81.

Hashemian, R. (1990). Square rooting algorithms for integer and floating-point numbers. *IEEE Transactions on Computers*, C-39(8):1025–29.

Kuhlmann, M., and K. Parhi (1998). Power comparison of SRT and GST dividers. In *Proceedings of SPIE—Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*, volume 3461, pages 584–94.

Lang, T., and E. Antelo (2001). Correctly rounded reciprocal square root by digit recurrence and radix-4 implementation. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 94–100.

Lang, T., and P. Montuschi (1992). Higher radix square root with prescaling. *IEEE Transactions on Computers*, 41(8):996–1009.

Lang, T., and P. Montuschi (1999). Very high radix square root with prescaling and rounding and a combined division/square root unit. *IEEE Transactions on Computers*, 48(8):827–41.

Leeser, M., and J. O'Leary (1995). Verification of a subtractive radix-2 square root algorithm and implementation. In *International Conference on Computer Design: VLSI in Computers and Processors*, pages 526–31.

Louie, M. E., and M. D. Ercegovac (1993). A digit-recurrence square root implementation for field programmable gate arrays. In *IEEE Workshop on FPGAs for Custom Computing Machines*.

Majerski, S. (1985). Square-root algorithms for high-speed digital circuits. *IEEE Transactions on Computers*, C-34(8):724–33.

Majithia, J. C. (1972). Cellular array for extraction of squares and square roots of binary numbers. *IEEE Transactions on Computers*, C-21(9):1023–24.

Majithia, J. C., and R. Kitai (1971). A cellular array for the nonrestoring extraction of square roots. *IEEE Transactions on Computers*, C-20(12):1617–18.

Matsubara, G., and N. Ide (1997). A low power zero-overhead self-timed division and square root unit combining a single-rail static circuit with a dual-rail dynamic circuit. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 198–209.

Matsubara, G., N. Ide, H. Tago, S. Suzuki, and N. Goto (1995). 30-ns 55-b shared radix-2 division and square root using a self-timed circuit. In *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pages 98–105.

McQuillan, S., and J. V. McCanny (1994). Fast algorithms for division and square root. *Journal of VLSI Signal Processing*, 8(2):151–68.

McQuillan, S. E., J. V. McCanny, and R. Hamill (1993). New algorithms and VLSI architectures for SRT division and square root. In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 80–86.

McQuillan, S. E., J. V. McCanny, and R. F. Woods (1991). High performance VLSI architecture for division and square root. *Electronics Letters*, V27(1):19–21.

Metze, G. (1967). Minimal square rooting. *IEEE Transactions on Computers*, EC-14(2):181–85.

Montuschi, P., and L. Ciminiera (1993). Reducing iteration time when result digit is zero for radix-2 SRT division and square root with redundant remainders. *IEEE Transactions on Computers*, 42(2):239–46.

Montuschi, P., and M. Mezzalama (1990). Survey of square rooting algorithms. *IEE Proceedings E: Computers and Digital Techniques*, 137(1):31–40.

Nannarelli, A., and T. Lang (1999). Low-power radix-4 combined division and square root. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'99)*, pages 236–42.

Prabhu, J. A., and G. B. Zyner (1995). 167 MHz radix-8 divide and square root using overlapped radix-2 stages. In *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pages 155–62.

Soderquist, P., and M. Leeser (1996). Area and performance tradeoffs in floating-point division and square root implementations. *ACM Computing Surveys*, 28(3):518–64.

Srinivas, H. R., and K. K. Parhi (1999). A radix 2 shared division/square root algorithm and its VLSI architecture. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 21(1):37–60.

Takagi, N. (2001). A hardware algorithm for computing reciprocal square root. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 94–100.

Taylor, G. S. (1981). Compatible hardware for division and square root. In *Proceedings of the 5th IEEE Symposium on Computer Arithmetic*, pages 127–34.

Walter, C. D. (1995). Verification of hardware combining multiplication, division and square root. *Microprocessors and Microsystems*, 19(5):243–45.

Zurawski, J. H. P., and J. B. Gosling (1987). Design of a high-speed square root, multiply, and divide unit. *IEEE Transactions on Computers*, C-36(9):13–23.