A Radix-4 On-Line Division Algorithm*

Paul K.-G. Tu and Miloš D. Ercegovac
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

We present an on-line algorithm for radix-4 floating point division. The divisor is first transformed in to a range such that the quotient digits are computed as a function of the scaled partial remainder only.

1 Introduction

The on-line division algorithms that have been published are of two types: Type 1 is based on the non-restoring division recurrence and it is exemplified by the original Trivedi-Ercegovac on-line division[10]. It takes a conventional non-restoring division with redundant quotient and imposes the on-line restrictions on it. Irwin[7] and Trivedi and Rusnak[11] are other examples of on-line algorithms of this type. Type 2 algorithms are based on the continued product representation of the reciprocal of the divisor. For these algorithms the critical part is multiplicative (additive) normalization. Grnarov and Ercegovac describe an on-line multiplicative normalization algorithm with an on-line delay of 1[5] and a refined version in [6]. Owens describes several on-line algorithms based on the continued product/sum representations in [9] and discusses a specific on-line division algorithm in [8].

What we describe here is another type of on-line division: it consists of (1) divisor preshifting into the range [1,2); (2) the divisor scaling transformation (3 steps), and (3) on-line (non-restoring) division recurrence. In [2] and [3] division algorithms with divisor range transformation to obtain simple quotient selection independent of the divisor were discussed. The particular scaling of the divisor discussed here is based on the transformation given in [3]. The quotient selection is performed by rounding of the remainder estimate[1,2]. The scaling transformation does not use a variable shift operator (i.e., factor r^{-j}), which is costly in the redundant (carry-save) implementation, but rather a shift register. In this respect it differs from a continued product type algorithm. In the recurrence part, it differs from Type 1 on-line algorithms in the quotient-selection function. In our case, the selection is independent of the divisor. The operands and the result are in the signed-digit form. To simplify the implementation we use 2's complement representation internally. The conversion between the redundant and 2's complement forms is done using an on-the-fly conversion algorithm[4].

It is assumed that the exponent calculation is performed in conventional arithmetic, and only mantissa computation is discussed in this paper. Also, each operand and quotient mantissa has n radix-4 digits, and for $j > n$, the input digits are assumed to be zero.

2 On-Line Division Algorithm

In this division algorithm the dividend, the divisor, and the quotient are on-line. The dividend, the divisor, and the quotient are quasi-normalized floating-point numbers in signed-digit representation with radix 4 and $|x_i| \leq 2$ [13]. The dividend and the divisor are assumed to be greater than zero, and the divisor larger in absolute value than the dividend. In a signed-digit number representation system, each digit may be either positive, zero, or negative. It is hence possible for a number with no leading zeros in signed-digit representation to have leading zeros when the number is converted to non-redundant number representation. A number is said

*This research has been supported in part by the ONR Contract N00014-85-K-0159 "On-Line Arithmetic Algorithms and Structures for VLSI".

to be *quasi-normalized* if for its mantissa X

$$r^{-2} \leq |X| < 1$$

Consequently, there can be at most one leading zero.

In order to have efficient carry-save type implementation for the recurrence formula evaluation, the internal computation uses 2's complement representation. The conversion of input operands from signed-digit form to 2's complement form is performed on-the-fly, as described in [4]. For each number X to be converted, we have

$$X_j = \sum_{i=1}^{j} x_i r^{-i}$$

and

$$X_j = X_{j-1} + x_j r^{-j}$$

where X_j is in 2's complement form, and x_j in radix-4 signed-digit form.

The algorithm consists of three major phases. In the first phase, the first 3 to 5 digits of the divisor are accumulated, and shifting is performed on the dividend and divisor until the divisor falls in the range $[1, 2)$. In the second phase the divisor is transformed into the range $[1-\alpha, 1+\alpha]$ by the range transformation algorithm, where α is given its value of $\frac{1}{32}$ as discussed later. Then, in the third phase, the quotient digits are generated in on-line fashion as a function of the partial remainder.

The shifting algorithm is defined as follows, where d_{in} and n_{in} denote the incoming divisor and dividend digits, D_0 and N_0 are the divisor and dividend after preshifting is performed.

Algorithm PRESHIFT
step1. [Initialization]
$D_0 \leftarrow \sum_{i=1}^{2} d_i 4^{-i}$; $f \leftarrow 1$;
$N_0 \leftarrow \sum_{i=1}^{2} n_i 4^{-i}$;

step 2. [Shift]
$D_0 \leftarrow D_0 + d_{in} \cdot 4^{-3}$;
$N_0 \leftarrow N_0 + n_{in} \cdot 4^{-3}$;

if $D_0 < \frac{1}{2}$
$D_0 \leftarrow D_0 \cdot 4$; $N_0 \leftarrow N_0 \cdot 4$;
goto step 2;

if $D_0 < 1$
$D_0 \leftarrow D_0 \cdot 2$; $f \leftarrow 2$;
$N_0 \leftarrow N_0 \cdot 2$;

end PRESHIFT.

Step 2 of the algorithm will be executed once or twice. After preshifting, D_0 has 3 to 5 input digits accumulated in it, and it has 6 or 7 bits of precision, including 1 integer bit. In each step, one more incoming digit of the divisor arrives, which may be positive or negative. Since $D_0 \in [1, 2)$, after the first phase, we have $D_j \in (1 - 2^{-5}, 2)$ for all j.

2.1 Divisor Transformation

Here we derive a radix-4 on-line range transformation algorithm for the divisor. After preshifting, the divisor is in the range

$$D_0 \in (1 - 2^{-5}, 2) \tag{1}$$

We want to find a number S such that $D \cdot S = X$, where $X \in [1-\alpha, 1+\alpha]$, and α is some small positive number to be determined later. Since D is available in on-line fashion, so is the calculation of S. More specifically, at step j of the range transformation, we want to have

$$D_j \cdot S_j = X_j \tag{2}$$

where

$$X_j \in (1 - 2 \cdot 4^{-j}, 1 + 2 \cdot 4^{-j}) \tag{3}$$

and D_j and S_j are the cumulative values of D and S at step j. We first derive the basic recurrence formula, and then discuss relevant parameters of the algorithm and the digit selection process of S_j. For the algorithm to converge, the error condition, the containment condition, and the continuity condition must be satisfied.

2.1.1 The on-line transformation recurrence

First we derive the recurrence formula for the transformation. Define

$$B_j \equiv 4^j (X_j - 1) \tag{4}$$

then from (2),

$$B_j = 4B_{j-1} + s_j D_j + d_{j+\delta} S^*_{j-1} 4^{-3} \tag{5}$$

where s_j is a digit selected in each step by a digit selection process *selects*, $\delta \in \{3, 4, 5\}$ is the number of digits accumulated in D_0, and

$$\begin{aligned} D_j &= D_{j-1} + d_{j+\delta} f 4^{-j-3} \\ & \quad d_j \in \{-2, -1, 0, 1, 2\} \\ S^*_j &\equiv S_j \cdot f = S^*_{j-1} + s_j f 4^{-j} \\ S^*_0 &= f \\ & \quad s_j \in \{-3, -2, -1, 0, 1, 2, 3\} \end{aligned} \tag{6}$$

We select s_j as a function of B_{j-1} and the incoming digit $d_{j+\delta}$. Define

$$\begin{aligned} A_{j-1} &= B_j - s_j D_j & (7)\\ &= 4B_{j-1} + d_{j+\delta} S^*_{j-1} 4^{-3} & (8) \end{aligned}$$

Then the basic recurrence for the range transformation algorithm is

$$\begin{aligned} s_j &= selects(\widehat{A}_{j-1}) & (9)\\ A_j &= 4(A_{j-1} + s_j D_j + d_{j+\delta+1} S^*_j 4^{-4}) & (10) \end{aligned}$$

where $\widehat{A}_{j-1}$ is the truncated version of A_{j-1}, and the initial condition is

$$A_0 = 4 \cdot (D_0 - 1) \qquad (11)$$

2.1.2 Error condition

From (3) we obtain the error condition

$$|X_j - 1| < 2 \cdot 4^{-j} \qquad (12)$$

Let $0 \leq \varepsilon_r < 2$ be the bound of $|B_j|$. We then require that

$$|B_j| \leq \varepsilon_r \qquad (13)$$

for all values of j for the range transformation process.

2.1.3 Continuity condition

For each possible value of s_j, we define a region $[\underline{K_c}, \overline{K_c}]$ such that when the value of A_j falls into this region, $s_j = c$ may be selected. $\underline{K_c}$ and $\overline{K_c}$ denote the lower and upper bound of the region, respectively. We have

$$\begin{aligned} \underline{K_c} &= -\varepsilon_r - cD_j \\ \overline{K_c} &= \varepsilon_r - cD_j & (14) \end{aligned}$$

Let Δ be the overlap between $[\underline{K_c}, \overline{K_c}]$ and $[\underline{K_{c+1}}, \overline{K_{c+1}}]$, then

$$\Delta \equiv \overline{K_{c+1}} - \underline{K_c} = 2\varepsilon_r - D_j \qquad (15)$$

For the algorithm to converge we must have $\Delta \geq 0$, and therefore

$$\varepsilon_r \geq \frac{D_j}{2} \qquad (16)$$

When Δ is greater than zero, in the range $[\underline{K_c}, \overline{K_{c+1}}]$ either c or $c+1$ is a valid choice for s_j. This feature is utilized to simplify the digit selection function.

2.1.4 Containment condition

From (14) we have

$$\begin{aligned} \overline{K_{-3}} &= \varepsilon_r + 3D_j \\ \underline{K_3} &= -\varepsilon_r - 3D_j \end{aligned}$$

Substituting (13) in (8), and noting that $S^*_j < 2$, $d_j \in$ {-2,-1,0,1,2}, we have

$$|A_j| < 4\varepsilon_r + 4^{-2} \qquad (17)$$

The containment condition requires that $A_j \in [\underline{K_3}, \overline{K_{-3}}]$, hence

$$4\varepsilon_r + 4^{-2} \leq \varepsilon_r + 3D_j \qquad (18)$$

or

$$\varepsilon_r \leq D_j - \frac{1}{3} \cdot 4^{-2} \qquad (19)$$

Combining (16) and (19) we have

$$\frac{D_j}{2} \leq \varepsilon_r \leq D_j - \frac{1}{3} \cdot 4^{-2} \qquad (20)$$

2.1.5 The digit selection process

We now discuss how to determine the comparison points for the digit selection. From (14) we have,

$$\begin{aligned} \underline{K_i} &= -(i + \frac{1}{2})D_j - (\varepsilon_r - \frac{D_j}{2}) \\ \overline{K_{i+1}} &= -(i + \frac{1}{2})D_j + (\varepsilon_r - \frac{D_j}{2}) \end{aligned}$$

For each $i \in \{-3, -2, -1, 0, 1, 2\}$, a comparison point C_i is chosen from a region

$$\begin{aligned} [\underline{K_i}, \overline{K_{i+1}}] = [&-(i + \frac{1}{2})D_j - (\varepsilon_r - \frac{D_j}{2}), \\ &-(i + \frac{1}{2})D_j + (\varepsilon_r - \frac{D_j}{2})] \end{aligned}$$

The center point of each region for C_i and i have opposite signs, and the 6 regions are symmetric to zero. To simplify the digit selection process, we first take the absolute value of the argument A_{j-1}, so that we need to compare $|A_{j-1}|$ with only 3 comparison points to get the absolute value of s_j, then the sign of s_j can be obtained directly from the sign of A_{j-1}. The 3 comparison points are in the regions

$$\begin{aligned} C_i \in [&(i + \frac{1}{2})D_j - (\varepsilon_r - \frac{D_j}{2}), \\ &(i + \frac{1}{2})D_j + (\varepsilon_r - \frac{D_j}{2})] \quad i \in \{0, 1, 2\} \end{aligned} \qquad (21)$$

Let

$$\varepsilon_r \equiv D_j - g \tag{22}$$

where

$$\frac{1}{3}4^{-2} \leq g \leq \frac{D_j}{2} \tag{23}$$

we then have

$$\Delta = 2\varepsilon_r - D_j = D_j - 2g$$

Substitute (22) into (21) we get

$$[\underline{K_i}, \overline{K_{i+1}}] = [iD_j + g, iD_j + D_j - g] \tag{24}$$

Subtract g from $|A_{j-1}|$, and the corresponding regions become

$$[\underline{K_i}, \overline{K_{i+1}}] = [iD_j, iD_j + D_j - 2g] \tag{25}$$

In the recurrence calculation, A_{j-1} is represented in carry-save form. If truncation is used to obtain an approximation A^* of $|A_{j-1}| - g$, the following holds

$$|A_{j-1}| - g \in [A^*, A^* + \gamma] \tag{26}$$

where γ is a positive quantity whose value depends on the truncation error. To obtain maximum redundancy, we use the lower bond of the region as the comparison point,

$$C_i = iD_j \qquad i \in \{0, 1, 2\} \tag{27}$$

In obtaining A^*, assume that A_{j-1} is assimilated to the kth fractional bit. This causes an error of less than 2^{-k}. Then, assume that after calculating $|A_{j-1}| - g$, the result is truncated after the mth bit to obtain A^*, causing an error of less than 2^{-m}. Hence we must have

$$\Delta \geq 2^{-m} + 2^{-k} \tag{28}$$

In order to maximize Δ, we must minimize g. We assume $m = 1, k = 3, g = 2^{-3}$, and noting the lower bound of D_j is $1 - 2^{-5}$, we have

$$\Delta = 1 - 2^{-5} - 2^{-2} > 2^{-1} + 2^{-3} \tag{29}$$

so (28) and (23) are satisfied. The input for the digit selection function will have 5 bits from A^* and 3 bits from D_j.

To make the calculation of $A^* = |A_{j-1}| - 2^{-3}$ efficient, we calculate $A_{j-1} + 2^{-3}$ and $A_{j-1} - 2^{-3}$ concurrently with the evaluation of the recurrence expression, and then chose one of the two as the input for the digit selection function depending on the sign of A_j. Since $|d_{j+\delta+1}| \leq 2$, $S_j^* \leq 2$, for the last term in (10) we have

$$4 \cdot d_{j+\delta+1} \cdot S_j^* \cdot 4^{-4} \leq 4^{-2}$$

and $A_{j-1} + 2^{-3}$ and $A_{j-1} - 2^{-3}$ can be calculated concurrently with A_j. The selection function can be implemented by combinational logic, using A^* and D_j as input, and it has about 11 product terms of at most 5 literals.

The following is the algorithm for the divisor range transformation.

Algorithm TRANSFORM
[Divisor range transformation]
$A_0 \leftarrow 4(D_0 - 1)$

for j = 1, ···, 3 **do**

1. $s_j \leftarrow selects(A_{j-1})$
   $D_j \leftarrow D_{j-1} + d_{j+\delta} f 4^{-j-3}$
   $N_j \leftarrow N_{j-1} + n_{j+\delta} f 4^{-j-3}$

2. $A_j \leftarrow 4A_{j-1} + d_{j+\delta} S_{j-1}^* 4^{-2} + 4D_j s_j$
   $S_j^* \leftarrow S_{j-1}^* + s_j f 4^{-j}$

end TRANSFORM

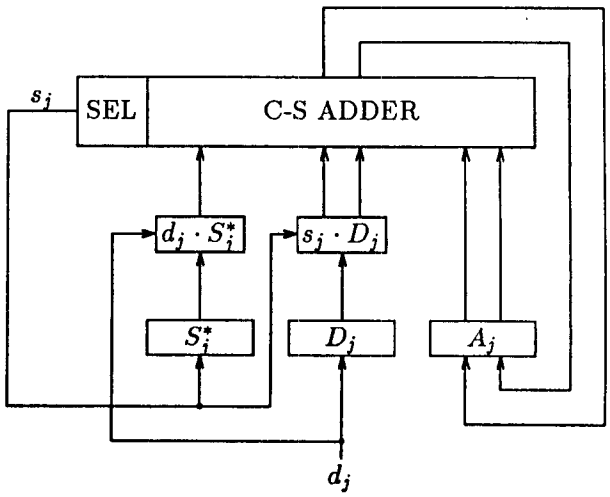

Figure 1: Divisor Range Transformation

Figure 1 is a sketch of the implementation of the divisor range transformation algorithm. The carry-save adder reduces five operands to two, while the digit selection part forms A^* with 7 assimilated bits and produces s_j. The boxes labeled S_j^* and D_j convert the inputs into 2's complement numbers and store the cumulative values of the variables.

The $x \cdot Y$ type of operation, where x is a single radix-4 digit and Y a 2's complement number, can be implemented by a multiplexer which, depending on the value of x, selects Y and/or $2 \cdot Y$ as input to the adder, or if x is negative, uses $-Y$ instead of Y. Note that since $d_j \in \{-2, -1, 0, 1, 2\}$, only one input to the adder is generated by the term $d_{j+\delta} \cdot S_j^*$, while $s_j \cdot D_j$ generates two inputs to the adder since s_j can be ± 3.

2.2 Quotient Generation

The quotient generation of the division algorithm consists of evaluating the recurrence formula and generating the quotient digits by the digit selection function *selectq*. We first derive the basic recurrence, and then discuss parameters of the algorithm.

Let N denote the dividend, D the divisor, and Q the quotient. Denote the transformed divisor as

$$X_j = D_j S^* \tag{30}$$

where $S^* = S_3^*$ is obtained in the divisor range transformation process, and the partial remainder as

$$Y_j = N_j S^* - Q_j X_{j-1} \tag{31}$$

By substituting (30) into (31) and expanding N, Q, and D, we have

$$\begin{aligned} Y_j &= N_j S^* - Q_j D_{j-1} S^* \qquad (32)\\ &= Y_{j-1} + n_{j+\delta_d} S^* 4^{-j-\delta_d} - \\ &\quad d_{j+\delta_d-1} Z_{j-1} 4^{-j-\delta_d+1} - \\ &\quad q_j X_{j-1} 4^{-j} \end{aligned}$$

where

$$\begin{aligned} X_{j-1} &= X_{j-2} + d_{j+\delta_d-1} S^* 4^{-j-\delta_d+1} \\ Z_j &\equiv S^* Q_j = Z_{j-1} + q_j S^* 4^{-j} \end{aligned}$$

and δ_d is the division on-line delay, which is the number of input digits accumulated in the operands before the quotient generation begins. Define R_j as the scaled partial remainder

$$R_j = 4^j Y_j \tag{33}$$

We then have the basic recurrence for quotient generation

$$\begin{aligned} q_j &= selectq(\widehat{4R}_{j-1}) \qquad (34)\\ R_j &= 4R_{j-1} + n_{j+\delta_d} S^* 4^{-\delta_d} - \\ &\quad d_{j+\delta_d-1} Z_{j-1} 4^{-\delta_d+1} - q_j X_{j-1} \qquad (35) \end{aligned}$$

Initially,

$$R_0 = Y_0 = N_0 S^* \tag{36}$$

and the digit selection function is

$$selectq(W) = \text{sign(W)} \cdot \lfloor |W| + \frac{1}{2} \rfloor \tag{37}$$

where $W = \widehat{4R}_{j-1}$.

Now we discuss the conditions and restrictions under which (35) and (34) will generate the correct quotient. From (32), we have

$$Q_j = \frac{N_j}{D_{j-1}} - \frac{Y_j}{D_{j-1} S^*} \tag{38}$$

and hence $Q_j \to N_j / D_{j-1}$ as $Y_j \to 0$. We define the error condition as

$$|Y_j| < 4^{-j} \tag{39}$$

we then have

$$\left| Q_n - \frac{N_{n+\delta_d}}{D_{n+\delta_d-1}} \right| < \frac{4^{-n}}{1-\alpha} \tag{40}$$

Since $\delta_d > 1$ we have $D_{n+\delta_d-1} = D$, $N_{n+\delta_d} = N$, we have

$$\left| Q - \frac{N}{D} \right| < \frac{(1+\alpha) \cdot 4^{-n}}{1-\alpha^2} \tag{41}$$

Substituting (39) into (33), we have

$$|R_j| < 1$$

Let $0 \le \varepsilon_q < 1$ be the bound of $|R_j|$. To satisfy the error condition (39), we require

$$|R_j| \le \varepsilon_q \tag{42}$$

Assume R_j is in carry-save form and let $\widehat{4R_j}$ denote $4R_j$ with k assimilation bits. Then

$$0 \le 4R_j - \widehat{4R}_j < 2^{-k} \tag{43}$$

From (37) and (43) we have

$$\begin{aligned} |\widehat{4R}_{j-1} - q_j| &\le 0.5 \\ |4R_{j-1} - q_j| &< 0.5 + 2^{-k} \end{aligned}$$

Since $X_j \in [1-\alpha, 1+\alpha]$, we have, in the worst case,

$$\begin{aligned} |4R_{j-1} - q_j X_j| &\le |4R_{j-1} - q_j| + |q_j \alpha| \\ &< 0.5 + 2^{-k} + |q_j \alpha| \qquad (44) \end{aligned}$$

substituting (44) into (35) and noting that $|S^*| \leq 2$, $|n_{j+\delta_d}| \leq 2$, $|d_{j+\delta_d}| \leq 2$, $|q_j| \leq 2$, $|Q_{j-1}| < 1$, $|Z_{j-1}| \leq 2$, we have

$$|R_j| < 0.5 + 2^{-k} + 2 \cdot \alpha + 5 \cdot 4^{-\delta_d+1} \qquad (45)$$

On the other hand, we require that $|q_j| \leq 2$, which in turn requires that

$$|4R_j| + 0.5 < 3$$

or

$$|R_j| < \frac{2.5}{4} = 0.625 \qquad (46)$$

From (45) and (46) we get

$$\begin{aligned} 0.5 + 2^{-k} + 2\alpha + 5 \cdot 4^{-\delta_d+1} &\leq 0.625 \\ \alpha + \frac{5}{8} \cdot 4^{-\delta_d+2} + 2^{-k-1} &\leq \frac{1}{16} \end{aligned} \qquad (47)$$

α, δ_d and k are chosen such that (47) is satisfied. Since the value of α affects the number of steps needed for the divisor range transformation, and the quotient generation process begins after the range transformation is finished, too small a value for α will increase the on-line delay of the quotient generation. Hence the value of α should be as large as possible. From (47), a minimum of 3 steps is needed for the range transformation, yielding

$$\alpha = \frac{1}{32} \qquad (48)$$

Then for $\delta_d \geq 5$, which is satisfied since quotient generation starts after the divisor range transformation terminates, and $k = 5$, we have

$$\alpha + \frac{5}{8} \cdot 4^{-\delta_d+2} + 2^{-k-1} = \frac{1}{32} + \frac{5}{8} \cdot \frac{1}{64} + \frac{1}{64} < \frac{1}{16}$$

and (47) is satisfied.

The implementation scheme of the quotient calculation is similar to that of the divisor range transformation, and is shown in Figure 2. The C-S adder needs 8 bits of assimilation, which includes one sign bit, 2 bits of integer part, and 5 bits of the fractional part. The R_j box simply stores the intermediate output of the adder to be used in the next step, and S^* is a constant value in the quotient generation phase, and is calculated in the divisor range transformation phase. The X and the $-Z$ boxes have the same structure. To avoid the time delay caused

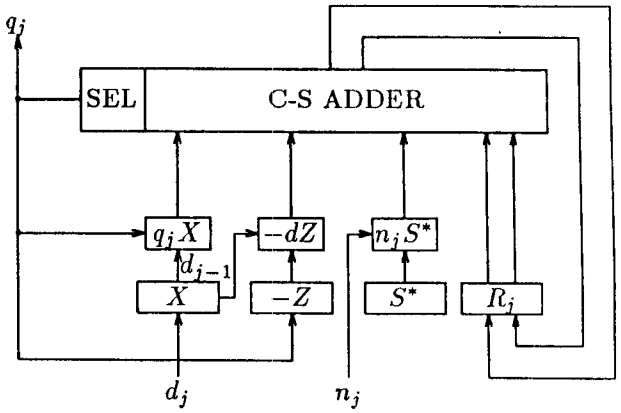

Figure 2: Quotient Generation

by multiplying two numbers, these two terms are generated incrementally,

$$\begin{aligned} X_0 &= D_0 \cdot S_0 = D_0 \\ X_j &= D_j \cdot S_j \\ &= X_{j-1} + d_{j+3} S_j^* 4^{-j-3} + s_j D_{j-1} 4^{-j} \\ &\quad j = 1, \cdots, 3 \\ X_j &= D_j \cdot S_3 = X_{j-1} + d_{j+3} S^* 4^{-j-3} \qquad (49) \\ &\quad j \geq 4 \\ Z_0 &= 0 \\ Z_{j-3} &= S^* \cdot Q_{j-3} = Z_{j-4} + q_{j-3} S^* 4^{-j+3} \qquad (50) \\ &\quad j \geq 4 \end{aligned}$$

Since S has a fixed precision of 7 bits, (49) and (50) can be implemented by a combination of a set of shift registers and carry propagate adders. 7 bits of carry propagate is needed. A detailed description of this scheme is given in [12]. The other parts of the diagram are similar to those in Figure 1.

The following is a summary of the radix-4 on-line division algorithm.

Algorithm OLDIV

step 1. [Initialization and shifting]
execute **Algorithm** PRESHIFT

step 2. [divisor range transformation]
execute **Algorithm** TRANSFORM

step 3. [quotient generation]
$S^* \leftarrow S_3^*$
$R_3 \leftarrow N_3 \cdot S^*$
$X_3 \leftarrow D_3 \cdot S_3$
$Z_0 \leftarrow 0$

for j = 4, $\cdots$, $n + \delta + 3$ **do**

3.1 $q_{j-3} \leftarrow selectq(\widehat{4R}_{j-3})$

3.2 $X_j \leftarrow X_{j-1} + d_{j+\delta} S^* 4^{-j-\delta}$

$R_j \leftarrow 4R_{j-1} + n_{j+\delta} S^* 4^{-2}$

$\quad -d_{j+\delta-1} Z_{j-4} 4^{-2} - q_{j-3} X_{j-1}$

$Z_{j-3} \leftarrow Z_{j-4} + q_{j-3} S^* 4^{-j+3}$

end OLDIV.

The critical path of the algorithm consist of $q_j \rightarrow q_j \cdot X_{j-1} \rightarrow$ 5-2 ADDER $\rightarrow SEL \rightarrow q_{j+1}$. The total number of steps needed is $n + \delta + 3$, where n is the number of digits of the quotient.

3 Summary

We have presented a radix-4 on-line division algorithm with the following properties. The operands of the algorithm are in radix-4 signed-digit form. The divisor is first transformed into a range such that the quotient digits are selected by rounding the scaled partial remainder. The recurrence formulas (10) and (35) are similar to the recurrence formula for on-line multiplication algorithm[10], and hence may be implemented using similar building blocks.

The on-line division algorithm we presented has an on-line delay of $\delta_d = \delta + 3$, which is from 6 to 8, which is quite large comparing to other on-line division algorithms. On the other hand, comparing to other published on-line division algorithms, this algorithm has a shorter digit-step time. In applications where a large number of arithmetic operations are to be performed in sequence, since in on-line arithmetic all operations are synchronized, a long digit-step time will slow down the whole operation. Since division generally appears less frequently than other operations such as addition and multiplication, an algorithm with a shorter digit-step time may be a better solution than one with a longer digit-step time, even if it has longer on-line delay. Further study is under way to reduce this delay.

References

[1] M. D. Ercegovac. *A General Method for Evaluation of Functions and Computations in a Digital Computer.* PhD thesis, University of Illinois at Urbana-Champaign, 1975.

[2] M. D. Ercegovac. A higher-radix division with simple selection of quotient digits. In *Proceedings of IEEE 1983 Sixth Symposium on Computer Arithmetic*, pages 94–98, 1983.

[3] M. D. Ercegovac and T. Lang. A division algorithm with prediction of quotient digits. In *Proceedings of the Seventh Symposium on Computer Arithmetic*, Urbana, Illinois, 1985.

[4] M. D. Ercegovac and T. Lang. *On-the-Fly Conversion of Redundant into Conventional Representations.* Technical Report CSD 850026, UCLA Computer Science Department, August 1985.

[5] A. L. Grnarov and M. D. Ercegovac. On-line multiplicative normalization. In *UCLA Computer Science Department Quarterly*, UCLA Computer Science Department, July 1979.

[6] A. L. Grnarov and M. D. Ercegovac. On-line multiplicative normalization. In *Proceedings of IEEE 1983 Sixth Symposium on Computer Arithmetic*, pages 151–155, 1983.

[7] M. Irwin. *An Arithmetic Unit for On-Line Arithmetic Computation.* PhD thesis, University of Illinoins, 1977.

[8] R. M. Owens. *Compound Algorithms for Digit Online Arithmetic.* Technical Report CS-81-1, Department of Computer Science, the Pennsylvania State University, January 1981.

[9] R. M. Owens. *Digit On-Line Algorithms for Pipeline Architectures.* PhD thesis, Penn State Universily, 1980.

[10] K. S. Trivedi and M. D. Ercegovac. On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26(7):681–687, July 1977.

[11] K. S. Trivedi and J. G. Rusnak. Higher radix online division. In *Proceedings of the Fourth Symposium on Computer Arithmetic*, pages 164–174, Santa Monica, CA, Oct 1978.

[12] P. K. Tu and M. D. Ercegovac. *A Radix-4 Online Division Algorithm.* Technical Report, UCLA Computer Science Department, Feb 1987.

[13] O. Watanuki. *Floating-point on-line arithmetic for highly concurrent digit-serial computation: application to mesh problems.* Technical Report CSD810529, Computer Science Department, UCLA, May 1981.