AN EFFICIENT DIVISION ALGORITHM AND ITS ARCHITECTURE*

Sau-Gee Chen and Chieh-Chih Li

Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan, China

Abstract- *In this work, a fast division algorithm and its time/area efficient architecture is proposed. It achieves the best performance in both area and speed aspects over the existing algorithms and implementations. The proposed architecture basically consists of N simple carry-save adders (CSAs) for bit-serial/serial implementation, and N^2 CSAs for bit parallel implementation. It finishes an N-bit division in 4N carry-save addition time and the result quotient is in binary representation. In addition, there is no complicated quotient decision circuit. Also a fast algorithm is developed to convert the signed-binary number representation to binary representation.*

I. INTRODUCTION

Inherently, division operation is a sequential type of operation. Its quotient digits are produced only after the remainder signs have been detected. As such, division operation is much slower than multiplication operation. Efforts have been put in speeding up the division operation. Notably the SRT algorithm [1,2], which eliminates the restoring operations of the partial remainders. Another algorithm [2] that confines the quotient digits either to be 1 or -1, depending on the signs of remainders. The speed bottleneck of those algorithms lies in the sign detection of remainder. Fast addition algorithms such as CLA (carry-lookahead addition) shorten the operation time, at the expense of hardware area.

Recently, division algorithms based on SD (signed-digit) number representation [3] was proposed which is much faster than the previous algorithms. The algorithm shortens the time for remainder subtraction considerably by using carry-propagation-free SD addition. However, in every iteration the SD algorithm must check three MSD bits (most significant digits) of the remainders to decide quotient digits in the set of {-1,0,1}, and performs SD addition. Moreover, its finial SD result must be converted to binary representation. Also note that the signed-digit addition is more complicated than the conventional CSA.

Another type of algorithms totally avoid the slow subtract-detect-shift type of operation previously mentioned. They transform the division operation to a series of multiplication operations that converge to the original division. Among the examples are the constant convergence [4] and quadratic convergence [5,6] division algorithms which are based on Newton-Raphson algorithm. They are popular in the multiplier-based processor. It still is a sequential type of operation to certain degree, and obviously requires much more shift-and-add operations.

There is the on-line division algorithm [7] that facilitates serial/serial division operation. The advantages of the algorithm are: (a) it is pipelined at the digit level; (b) all operands and results are communicated digit serial, and (c) results digits are on-line obtained after a few initial delays. On the other hand, among some of its disadvantages are: (a) it requires more complex three-input signed-digit addition operation; (b) it needs more complicated quotient decision circuit for range detection of the remainders, and (c) output results have to be converted to binary representations.

In this work, a fast radix-2 division algorithm and its architecture is proposed. The algorithm adheres to the subtract-and-add type of division operation for it has smaller number of iteration steps than those of multiplicative approaches. The key idea behind this algorithm is to separate the sign detection operation of remainder from the remainder subtraction operation. By taking the absolute values of the remainders, we can successively subtract the remainders without knowing the signs of remain-

* This work is supported in part by National Science Council under Contract NSC82-0404-E009-154.

ders, while signs of the remainders can be decided parallely and independently. To enhance the algorithm's performance, several design techniques are incorporated into its architecture realization.

The new algorithm and its architecture try to retains the most advantages of the mentioned algorithms as possible, and simultaneously gets rid of their disadvantages. The algorithm adopts nonrestoring division operation and CSA type of operation for fast subtraction. Quotient digit set of {-1,1} is assumed for fast quotient conversion to binary representation. The algorithm is also an on-line algorithm that facilitates highly pipelined operation while it is much simpler than the existing on-line algorithm. The following sections will address the issues in the order: (a) the radix-2 division algorithm; (b) the modified CSA-like signed-digit subtraction algorithm that greatly speeds up the algorithm; (c) bit-serial/serial architecture realization of the algorithm, and (d) the conclusion.

II. THE DIVISION ALGORITHM

Given normalized n-bit sign magnitude operands $1/2<|X|<1, 1/4\le|Y|\le1/2$, quotient Q_2 of Y/X is solved as follows. Where the quotient digits $Q_2 = a_s.a_1a_2....a_n$ is in 2's complement representation and a_S is the sign bit.

Step 1: $a_s = x_s \oplus y_s$

Step 2: Solve partial remainder R_{i+1}, by modifying the conventional nonrestoring algorithm [2] as follows. The signed-binary quotient Q_i has its quotient digit $q_i \in \{-1,1\}$

$$R_{i+1} = 2|R_i - q'_i X| \tag{1}$$

where $R_0 = |Y|$. $Q_0 = 1$, q'_i is the *ith* pseudo quotient digit. Since R_{i+1} is always positive, Eq. (1) can be rewritten as

$$R_{i+1} = 2|R_i - X| \tag{2}$$

$$q_i = \begin{cases} 1, \ if \ S_{i-1} = 0 \\ 1, \ if \ Z_i = 0 \\ -1, \ if \ S_{i-1} = 1 \end{cases} \tag{3}$$

where

Sr_i =The sign of remainder R_i
S_i =True sign of i-th remainder of $S_{i-1} \oplus Sr_i$
Z_i = Zero Flag, Z_0=0
$S_0 = Sr_0 = \text{Sign}\{R_0\} = 0$

Step 3: Convert the signed-binary representation q_i(-1,1) to 2's complement binary representation a_i.

$$a_i = \begin{cases} 1, \ if \ q_{i+1} = 1 \\ 0, if \ q_{i+1} = -1 \end{cases} \tag{4}$$

As can be seen the conversion rule is pretty straightforward. Fast conversion is achieved by checking the polarity of individual digit.

Step 4: Proceed to the next step if either when $i=N$ or $Z_i=0$. Otherwise go to Step 2.

Step 5: Obtain $Q_2 = a_s.a_1a_2....a_{n-1}$.

Since absolute values of the partial remainders are computed instead of their actual values, the algorithm facilitates parallel computations of partial remainder and quotient digit. To further speed up the operation of subtraction, we modified the signed-digit operation.

III. MODIFIED SIGNED-DIGIT SUBTRACTION

Since computations of Eq. (2) involves only the subtraction operation of two positive numbers, we can speed up the computation by defining the CSA-like operation as follows.

$$y_i - x_i = 2c_{i+1} + t_i \tag{5.a}$$

$$t_i + c_i = s_i \tag{5.b}$$

where

$y_i, \ s_i \in \{-1,0,1\}$
$x_i, \ t_i \in \{0,1\}$
$c_i \in \{-1,0\}$

Here, a signed-digit bit y_i subtracts a binary digit x_i, then generates carry c_{i+1} and intermediate result t_i. The finial result is obtained by adding t_i and the carry-in bit c_i. Since $s_i \in$(-1,0,1), there will be no carry generated from $t_i + c_i$. As a result, the signed-digit subtraction efficiently eliminates carry propagation. In addition, the complexity of this operation is similar to that of conventional CSA. Example 1 depicts the division algorithm with the modified addition algorithm.

IV. THE RADIX-2 BIT-SERIAL DIVIDER

Figure 1 shows the bit-serial divider for *Y/X*, $1/2<|X|<1$, $1/4\le|Y|\le1/2$. Where FA0 module performs operations of $a_s = x_s \oplus y_s$, and Eq. (5). The FA modules perform Eq. (5). The Neg modules take absolute values of Eq. (5.b). The Sign module determines the actual signs of remainders and correspondingly the quotient digits. If the sign flag of a remainder is negative then $q_i = -1$, $a_i = 0$, otherwise $q_i = 1$, $a_i = 1$, which is dictated by Eq. (4). The sign of current remainder is negative if either previous remainder's sign is positive and current sign flag is negative, or previous sign is negative and current sign flag is positive. The sign of current remainder is positive if either previous remainder's sign is positive and current sign flag is positive, or previous sign is negative and current sign flag is negative. If zero flag is set then there is exact division. Under such condition, the division is completed and terminated.

V. CONCLUSION

In summary, the division algorithm have the advantages as follows:

a) It uses a smaller quotient digit set of {-1,1} than {-1,0,1}, that simplifies the quotient decision circuits like some known algorithms do, but achieves the exact division and trivial conversion of the results from signed-binary representation to binary representation.
b) It needs no quotient estimator.
c) In each iteration, the algorithm computes partial remainders without knowing the signs of previous remainders and decides the signs of remainders independently and parallely. In addition, these two operations are done in a pipelined fashion and in digit level with maximum throughput rate.
d) Its architecture is basically consists of the simple and CSA cells.
e) It can handle either positive or negative operands, by minor modification of the architecture.

From above discussion, the proposed division algorithm and its architecture is very efficient. The new algorithm's realization is composed of highly regular cellular array, which is suitable for VLSI implementation and can be easily extended to bit-parallel implementation.

The algorithm can be extended to higher radix divisions such as radix-4 division, and to square-root operations. Since the remainders are taken absolute values, the digit set contains only digits 1 and 2 is sufficient for the entire radix-4 operation. This greatly reduces the number of search regions for the quotient digits, in contrast to the bigger set of {0,1,2,3} that existing algorithms allow. Those possible algorithm extensions are currently under investigation.

REFERENCES

[1] C. V. Freiman, "Statistical Analysis of Certain Binary division algorithms," Proc. IRE, Vol. 49, Jan. 1961, pp. 91-103.

[2] K. Hwang, Computer Arithmetic: Principles, Architectures, and Design, 1979, pp. 222-223.

[3] S. Kuninobu et al., "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation," IEEE Proceeding of Symposium on Computer Arithmetic, 1987, pp. 80-86.

[4] S. Waser and M. J. Flynn, Introduction to Arithmetic for Digital Systems Designers, New York: CBS College Publishing, Chap. 5, 1982.

[5] P. Markenstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor," IBM Journal of Research and Development, Vol. 34, 1990, pp. 111-119.

[6] D. A. Patterson and J. L. Hennessy, Computer: A Quantitative Approach, San Mateo, CA, Morgan Kaufman, 1990

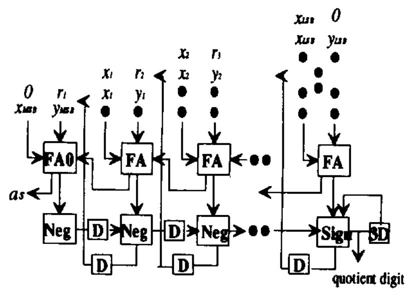
[7] K. S. Trivedi and M. D. Ercegovac, "On-Line Algorithms for Division and Multiplication," IEEE Trans. on Computers, Vol. C-26, No 7, July 1977.

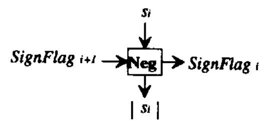
Example 1: $Y/X,\ Y = 01010001_2 = 81$

$X = 00001001_2 = 9$

$$\begin{array}{lrl}
q_0 = 1 & 1010001 & 2R_i \\
 & \underline{-1001000} & X \\
 & 0011001 & t_i \\
 & \underline{-0001000} & c_i \\
q_1 = 1, & 0001001 > 0 & s_i \\
a_1 = 1 & & \\
 & 0010010 & 2R_{i+1} \\
 & -\ \underline{1001000} & \\
 & 1011010 & \\
 & -\underline{10011000} & \\
q_2 = -1, & \bar{1}1001010 < 0 & \\
a_2 = 0 & |\bar{1}100101001| & \\
 & 1\bar{1}00\bar{1}0\bar{1}00 & \\
 & -\ \underline{1001000} & \\
 & 111011100 & \\
 & -\underline{110111} & \\
q_3 = -1, & 001\bar{1}00100 > 0 & \\
a_3 = 0 & & \\
 & 1\bar{1}001000 & \\
 & -\ \underline{1001000} & \\
 & 10000000 & \\
 & -\ \underline{10000000} & \\
q_4 = 1, & 0 & \\
a_4 = 1 & &
\end{array}$$

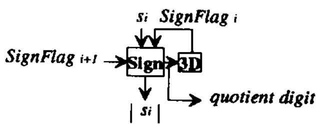
The quotient=*1001*$_2$=*9*, and remainder=*0*





$$SignFlag_i = \begin{cases} SignFlag_{i+1}, & \text{if } SignFlag_{i+1} = 1 \text{ or } -1 \\ Sign(s_i), & \text{if } SignFlag_{i+1} = 0 \end{cases}$$

$$|s_i| = \begin{cases} s_i, & \text{if } SignFlag_{i+1} = 1 \text{ or } 0 \\ -s_i, & \text{if } SignFlag_{i+1} = -1 \end{cases}$$



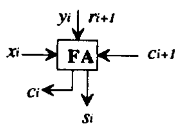


Figure 1. The radix-2 division implementation