DESIGN OF A CYCLE-EFFICIENT 64B/32B INTEGER DIVIDER USING A TABLE-SHARING METHOD*

Chua-Chin Wang†, *Po-Ming Lee, Jun-Jie Wang*

Department of Electrical Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan 80424
Tel : 886-7-525-2000 ext. 4144
Fax : 886-7-5254199
E-Mail: ccwang@ee.nsysu.edu.tw

Chenn-Jung Huang

Department of Computer Science
and Information Education
National Taitung Teachers College
Taitung, Taiwan 95004
E-Mail: cjh@cc.ntttc.edu.tw

ABSTRACT

In new generations of microprocessors, the superscalar architecture is widely adopted to increase the number of instructions executed in one cycle [2]. The division instruction among all of the instructions needs more cycles than the rest, e.g., addition and multiplication. It, then, makes division instruction an important CPI (Cycles Per Instruction) figure for modern microprocessors. In this paper, a radix 16/8/4/2 divider is proposed which uses a variety of techniques, including operand scaling, table partitioning, and table folding, to increase performance without the cost of increasing complexity.

1. INTRODUCTION

Digit recurrence is one of the oldest classes of high-speed division algorithms. A significant amount of literature has been proposed regarding algorithms, implementations, and techniques. Although prior algorithms increased division performance, many side effects and drawbacks were not considered, e.g., adjustment of remainder when used in integer division. In this paper, we employ a high-radix, 16/8/4/2, division method to reduce the required cycles for the division, while keep the hardware complexity in control. The major difference is the utilization of the table partitioning and folding.

2. DIGIT-RECURRENCE THEORY

The generalized division is defined as: $x = q \cdot d + rem$, where x is called dividend, d is divisor, q is quotient, and rem is remainder. In the digit recurrence division algorithm, 1 to b bits of quotient digit can be obtained every iteration in a radix-2^b digit recurrence division. In other words, b bits of quotient can be obtained every iteration. In [1], the digit recurrence algorithm is defined as shown:

$$w[j+1] = r \cdot w[j] - d \cdot q_{j+1}, \qquad (1)$$

where $w[j+1]$ is the residual of the $(j+1)$th iteration, r is the radix, and q_{j+1} is the quotient digit generated in the $(j+1)$th iteration. In a radix-r, r=2^b, division, the quotient digit set is defined as $q_j \in D_a = \{-a, ..., -1, 0, 1, ..., a\}$. Since $||D_a|| > r$, it uses more than r numbers to present the quotient digits, which make this quotient representation form to be a redundant form. Besides, the restriction of a is: $a \geq \lceil \frac{r}{2} \rceil$. In Eqn. (1), the quotient digits are generated in every iteration. Hence we can define the quotient-digit selection function as $q_{j+1} = \mathrm{SEL}(w[j], d)$, where the SEL() function can be simplified as a table lookup function.

*This research was partially supported by Nation Science Council under grant NSC 89-2218-E-110-014 and 89-2218-E-110-015

† Contact author

3. MIXED RADIX-16/8/4/2 64B/32B INTEGER DIVIDER

In [3], a mixed radix-8/4/2 integer divider was proposed of which performance is better than that of a normal radix-4/2 integer divider [4]. However, it paid the price of increasing the complexity of hardware, and the nearly doubling total area of the divider. In this paper, the radix will be raised up to 16 in the division algorithm to retire more bits of quotient per cycle. Nevertheless, the complexity of the hardware will be retained to a similar degree by applying several methods, including operand prescaling [6], table partitioning [7], and table folding [8].

3.1. operand scaling

In high radix dividers, the cycle time is generally determined by the quotient-digit selection operation which is basically a table lookup operation. The complexity of quotient selection function increases exponentially if the radix increases linearly. Consequently, it results in a long table lookup time. Operand scaling is a better alternative to avoid the long table lookup time.

The maximal overlap between quotient digits appears when the divisor is the maximum. That is, the maximal amount of overlap occurs when a divisor d is normalized and it approaches to 1. This observation leads to the concept of divisor prescaling [6]. In the first step of the scaling method, the divisor is prescaled by a factor M so that the scaled divisor z is: $1 - \alpha \leq z = M \cdot d \leq 1 + \beta$, where α and β are chosen such that the scaling factor, $S_i, i \in \{0..6\}$, is identical in all divisor intervals and the quotient-digit selection is independent of the divisor. Besides, the value of M should be chosen to minimize $(\alpha + \beta)$ which produces the smallest achievable range of z. In order to preserve the value of the quotient, three alternative ways of perform-

0-7803-7057-0/01/$10.00 ©2001 IEEE.

ing the scaling were proposed [5]. The scaling factor and scaled divisor range is tabulated in Table 1. Operand scaling process produce a scaled estimated residual: $\hat{y}$, which is generated as shown in Figure 1. In Figure 1, the "estimated residual" is chosen from the first 7 bits form $w[j]$ in Eqn. (1).

$128 \cdot d$	$64 \cdot M$	$[8192 \cdot (1\text{-}\alpha), 8192 \cdot (1+\beta))$	$128 \cdot d$	$64 \cdot M$	$[8192 \cdot (1\text{-}\alpha), 8192 \cdot (1+\beta))$
[127, 128)	64	[8128, 8192)	[126, 127)	65	[8190, 8255)
[125, 126)	65	[8125, 8190)	[124, 125)	66	[8184, 8250)
[123, 124)	66	[8118, 8184)	[122, 123)	67	[8174, 8241)
[121, 122)	67	[8107, 8174)	[120, 121)	68	[8160, 8228)
[119, 120)	69	[8211, 8280)	[118, 119)	69	[8142, 8211)
[117, 118)	70	[8190, 8260)	[116, 117)	70	[8120, 8190)
[115, 116)	71	[8165, 8236)	[114, 115)	72	[8208, 8280)
[113, 114)	72	[8136, 8208)	[112, 113)	73	[8176, 8249)
[111, 112)	74	[8214, 8288)	[110, 111)	74	[8140, 8214)
[109, 110)	75	[8175, 8250)	[108, 109)	76	[8208, 8284)
[107, 108)	76	[8132, 8208)	[106, 107)	77	[8162, 8239)
[105, 106)	78	[8190, 8268)	[104, 105)	78	[8112, 8190)
[103, 104)	79	[8137, 8216)	[102, 103)	80	[8160, 8240)
[101, 102)	81	[8181, 8262)	[100, 101)	82	[8200, 8282)
[99, 100)	82	[8118, 8200)	[98, 99)	83	[8134, 8217)
[97, 98)	84	[8148, 8232)	[96, 97)	85	[8160, 8245)
[95, 96)	86	[8170, 8256)	[94, 95)	87	[8178, 8265)
[93, 94)	88	[8184, 8272)	[92, 93)	89	[8188, 8277)
[91, 92)	90	[8190, 8280)	[90, 91)	91	[8190, 8281)
[89, 90)	92	[8188, 8280)	[88, 89)	93	[8184, 8277)
[87, 88)	94	[8178, 8272)	[86, 87)	95	[8170, 8265)
[85, 86)	96	[8160, 8256)	[84, 85)	97	[8148, 8245)
[83, 84)	98	[8134, 8232)	[82, 83)	99	[8118, 8217)
[81, 82)	101	[8181, 8282)	[80, 81)	102	[8160, 8262)
[79, 80)	103	[8137, 8240)	[78, 79)	104	[8112, 8216)
[77, 78)	106	[8162, 8268)	[76, 77)	107	[8132, 8239)
[75, 76)	109	[8175, 8284)	[74, 75)	110	[8140, 8250)
[73, 74)	112	[8176, 8288)	[72, 73)	113	[8136, 8249)
[71, 72)	115	[8165, 8280)	[70, 71)	116	[8120, 8236)
[69, 70)	118	[8142, 8260)	[68, 69)	120	[8160, 8280)
[67, 68)	121	[8107, 8228)	[66, 67)	123	[8118, 8241)
[65, 66)	125	[8125, 8250)	[64, 65)	127	[8128, 8255)

Table 1: Scaling factor and scaled divisor range.

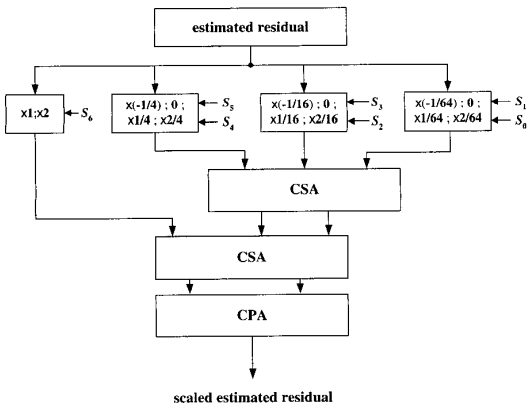

Figure 1: The generation of scaled estimated residual: $\hat{y}$

3.2. table sharing algorithm

The quotient-digit selection function in radix-8 division has been proved to be the bottleneck in each iteration [3]. Hence the radix-16/8/4/2 integer division will be infeasible unless a simplified quotient-digit selection function is developed. A modified version of table partitioning algorithm [7], called "table sharing", is presented to simplify the digit selection process. The table partition method in [7] arranged the entries in a non-mergable manner. We simply reorder the rows to place the negative entries of Lo. on top half of the table and the positive ones on the bottom. Hence, the modified table possesses a feature that is mergable for different radices. Moreover, the merge of a total of the radix-2, radix-4, radix-8, and the radix-16 quotient selection tables reduces the area.

3.2.1. quotient digit decomposition

In the proposed scheme, the maximally redundant quotient digit set is chosen for radix-16 division and decomposed into four components as follows:

$$\begin{aligned} q_{j+1} &= q_{h,h} + q_{h,l} + q_{l,h} + q_{l,l}, \\ q_{h,h} \in \{-8, 0, 8\}, &\quad q_{h,l} \in \{-4, 0, 4\}, \\ q_{l,h} \in \{-2, 0, 2\}, &\quad q_{l,l} \in \{-1, 0, 1\}, \end{aligned} \tag{2}$$

where $q_{j+1} \in \{-15, -14, ..., 0, ..., 14, 15\}$, and $q_{h,h}$, $q_{h,l}$, $q_{l,h}$, and $q_{l,l}$ are tabulated as Table 2.

		Lo.	Hi.	q_{j+1}	$q_{h,h}$	$q_{h,l}$	$q_{l,h}$	$q_{l,l}$
16		-16.5	-15	-15	-8	-4	-2	-1
		-14.5	-14	-14	-8	-4	-2	0
		-13.5	-13	-13	-8	-4	0	-1
		-12.5	-12	-12	-8	-4	0	0
		-11.5	-11	-11	-8	0	-2	-1
		-10.5	-10	-10	-8	0	-2	0
		-9.5	-9	-9	-8	0	0	-1
		-8.5	-8	-8	-8	0	0	0
	8	-7.5	-7	-7	0	-4	-2	-1
		-6.5	-6	-6	0	-4	-2	0
		-5.5	-5	-5	0	-4	0	-1
		-4.5	-4	-4	0	-4	0	0
	4	-3.5	-3	-3	0	0	-2	-1
		-2.5	-2	-2	0	0	-2	0
	2	-1.5	-1	-1	0	0	0	-1
		-0.5	-0	-0	0	0	0	0
		0.5	1	1	0	0	0	1
		1.5	2	2	0	0	2	0
		2.5	3	3	0	0	2	1
		3.5	4	4	0	4	0	0
		4.5	5	5	0	4	0	1
		5.5	6	6	0	4	2	0
		6.5	7	7	0	4	2	1
		7.5	8	8	8	0	0	0
		8.5	9	9	8	0	0	1
		9.5	10	10	8	0	2	0
		10.5	11	11	8	0	2	1
		11.5	12	12	8	4	0	0
		12.5	13	13	8	4	0	1
		13.5	14	14	8	4	2	0
		14.5	16	15	8	4	2	1

Table 2: The decomposition of q_{j+1} for 16/8/4/2 radix divider.

According to Table 2, the selection intervals for q_{j+1} in radix-16, radix-8, radix-4, and radix-2 divisions are included and tabulated in Table 2, as indicated by the 4 braces, respectively. Besides, *Lo.* and *Hi.* in Table 2 denote the lower bound and the upper bound of the the shifted estimated residual, respectively. Notably, by inspecting Table 1, the bounds of the scaled shifted residual, $\hat{y}$, are derived form $|w[j]| \leq \frac{8288}{8192}$ and the corresponding 2-bit truncation error. Namely, $\lfloor -r \cdot \frac{8288}{8192} - 2^{-2} \rfloor \leq \hat{y} \leq \lfloor r \cdot \frac{8288}{8192} \rfloor$ Nevertheless, since the quotient-digit table is shared by different radices, the highest order digit will be incorrectly enabled if the value of $\hat{y}$ is close to the bounds as illustrated in the first and last rows indicated by the braces in Tables 2. Fortunately, this can be fixed easily later at the quotient-digit assimilation stage where $q_{h,h}, q_{h,l}, q_{l,h}$, and $q_{l,l}$ re-compose the quotient digits.

3.3. table folding

By inspecting Table 2, the entries of the top half are identical to the opposite ones in the bottom half. It allows us to simply implement only the positive half. Accordingly, the proposed scheme needs only 6 bits, besides the common sign bit, as the input to the quotient-digit selection table, including 5 integer bits and 1 fractional bit of $\hat{y}$, in contrast to 11 bits required in the radix-8 division presented [3]. Thus, a total of 7 bits, $\hat{y} = y_5 y_4 y_3 y_2 y_1 y_0 y_{-1}$ are used to derive $q^h_{h,h}$, $q^h_{h,l}$, $q^h_{l,h}$, $q^h_{l,l}$, $q^l_{h,h}$, $q^l_{h,l}$, $q^l_{l,h}$, and $q^l_{l,l}$. Then, we get

$$
\begin{aligned}
q^h_{h,h} &= q^h_{h,l} = q^h_{l,h} = q^h_{l,l} = y_5, \\
q^l_{h,h} &= \bar{y}_4 y_3 + \bar{y}_4 y_2 y_1 y_0 y_{-1} + y_4 \bar{y}_3 \bar{y}_2 \bar{y}_1 \bar{y}_0 \bar{y}_{-1}, \\
q^l_{h,l} &= \bar{y}_4 y_3 y_2 + \bar{y}_4 y_2 \bar{y}_1 + \bar{y}_4 y_2 \bar{y}_0 + \bar{y}_4 y_2 \bar{y}_{-1} + \\
&\quad \bar{y}_4 \bar{y}_2 y_1 y_0 y_{-1} + y_4 \bar{y}_3 \bar{y}_2 \bar{y}_1 \bar{y}_0 \bar{y}_{-1}, \\
q^l_{l,h} &= \bar{y}_4 y_1 \bar{y}_0 + \bar{y}_4 y_1 \bar{y}_{-1} + \bar{y}_4 y_3 y_2 y_1 + \bar{y}_4 \bar{y}_1 y_0 y_{-1} + \\
&\quad y_4 \bar{y}_3 \bar{y}_2 \bar{y}_1 \bar{y}_0 \bar{y}_{-1}, \\
q^l_{l,l} &= \bar{y}_4 \bar{y}_0 y_1 + \bar{y}_4 y_0 \bar{y}_{-1} + \bar{y}_4 y_3 y_2 y_1 y_0 y_{-1} + \\
&\quad y_4 \bar{y}_3 \bar{y}_2 \bar{y}_1 \bar{y}_0 \bar{y}_{-1} \qquad (3)
\end{aligned}
$$

Notably, all the four radices can share the expressions given in Eqn. (3) without any changes. The scenario requiring many quotient selection tables for different radices appearing in the prior designs no longer exists.

3.4. quotient digit assimilation unit

The quotient digit assimilation unit has the task of performing the assimilation of the selections $q^l_{h,h}$, $q^l_{h,l}$, $q^l_{l,h}$, and $q^l_{l,l}$ into the single digit $q_{j+1} = q_h \cdot 4 + q_l$, where q_h is formed by assimilating $q^l_{h,h}$ and $q^l_{h,l}$, and q_l is created by assimilating $q^l_{l,h}$ and $q^l_{l,l}$. Meanwhile, the assimilation unit can be shared by different radices. The q_h and q_l are functions of $q^h_{h,h}$, $q^h_{h,l}$, $q^h_{l,h}$, $q^h_{l,l}$, $q^l_{h,h}$, $q^l_{h,l}$, $q^l_{l,h}$, and $q^l_{l,l}$ for radix-16 through radix-2 divisions as shown in Table 3.

The 38-bit CSA in Figure 2 is fed with the shifted residual, $-q_{hh} \cdot d$, $-q_{hl} \cdot d$, $-q_{lh} \cdot d$, $-q_{ll} \cdot d$, $r \times wc[j]$, and $r \times ws[j]$ to generate $wc[j+1]$ and $ws[j+1]$, where $q_{hh} + q_{hl} = q_h$, and $q_{lh} + q_{ll} = q_l$. Notably, q_{hh}, q_{lh}, q_{hl}, q_{ll} must be numbers of 2's power. For instance, a pre-computation of $3 \cdot d$ can be decomposed into $3 \cdot d = (2^0 + 2^1)d$ in one pass through the CSA and saved in registers right after the divisor d is scaled in the first cycle. Multipliers then will not be required in the high radix division implementations.

radix		bit 2 (MSB)	bit 1	bit 0 (LSB)
16	q_h	$q^h_{l,l}$	$q^l_{h,h}$	$q^l_{h,l}$
	q_l	$q^h_{l,l}$	$q^l_{l,h}$	$q^l_{l,l}$
8	q_h	$q^h_{l,l}$	$q^l_{h,h} + q^l_{h,l}$	$q^l_{l,h}$
	q_l	$q^h_{l,l}$	$q^l_{l,l}$	0
4	q_h	$q^h_{l,l}$	$q^l_{h,h} + q^l_{h,l} + q^l_{l,h}$	$q^l_{h,h} + q^l_{h,l} + q^l_{l,l}$
	q_l	$q^h_{l,l}$	0	0
2	q_h	$q^h_{l,l}$	$q^l_{h,h} + q^l_{h,l} + q^l_{l,h} + q^l_{l,l}$	0
	q_l	$q^h_{l,l}$	0	0

Table 3: generation of q_h and q_l for different radices

Note that the expressions of q_l and q_h for different radices are identical except for the cases that the higher order quotient digit is incorrectly set to 1 when the value of $\hat{y}$ is close to the bounds as illustrated in the boundary rows of each radix in Table 2. The complete scheme for the mixed radix 16/8/4/2 64b/32b integer divider is presented in Figure 2.

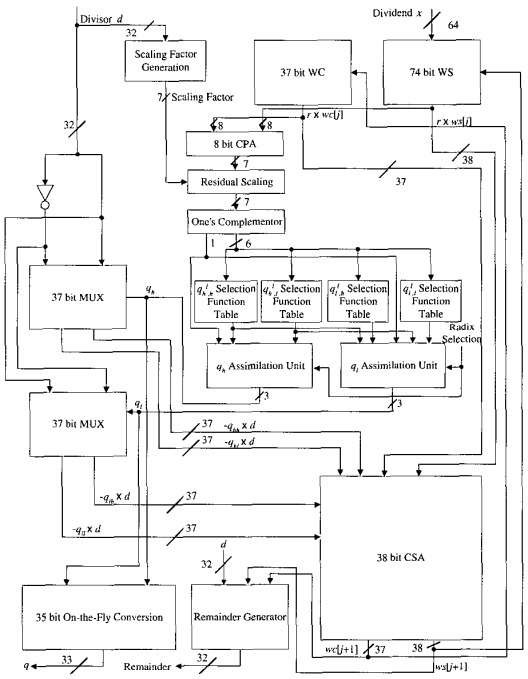

Figure 2: The architecture of the radix 16/8/4/2 64b/32b integer divider.

4. SIMULATION AND IMPLEMENTATION

The chip is designed by TSMC (Taiwan Semiconductor Manufacturing Company) 0.35 μm 1P4M technology, and simulated by Cadence SDF simulation tools. The simulation result is shown in Figure 3. The highest working clock of this radix 16/8/4/2 64/32 bit divider is 76.9 MHz. Figure 4 is the DIE photo of the proposed divider. Table 4

demonstrates the comparison of execution cycles for integer division of current X86 microprocessors and our design. Our design possesses the advantage of cycles to execute a division instruction. Table 5 is the comparison of mixed radix-4/2, mixed radix-8/4/2, and our design.

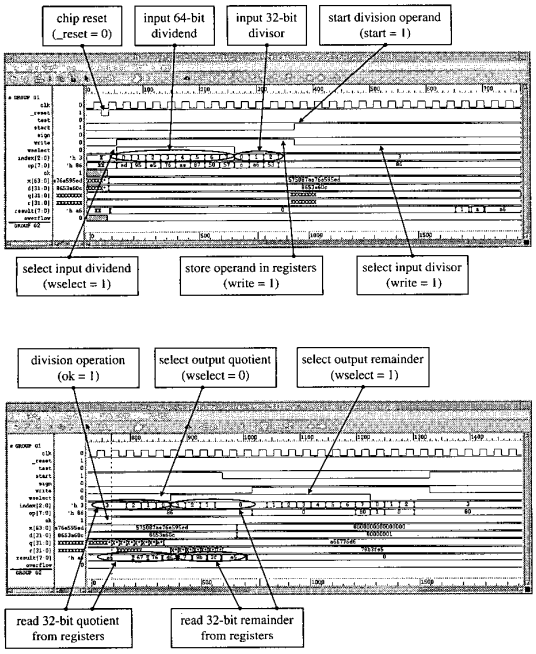

Figure 3: The simulation result of the proposed divider

	Pentium Pro	AMD Athlon	Cyrix 6x86 MII 1998	[3]	Ours
Integer division	12-6	47-22	45-13	18-3	13-3

Table 4: Comparison of modern microprocessor integer division cycles and proposed mixed radix-16/8/4/2 divider.

	Mixed Radix-4/2 [4]	Mixed Radix-8/4/2 [3]	Ours
Area by SYNOPSYS	9984.75	17468.76	19996.42
Number of cycles	23-2	18-3	13-3
Clock rate	33 MHz	66 MHz	76.9 MHz

Table 5: Comparison of mixed radix-4/2, mixed radix-8/4/2, and proposed mixed radix-16/8/4/2 divider.

5. SUMMARY

The division performance of the integer divider can be improved by either reducing the division cycles or decreasing the time in each iteration without adding significant

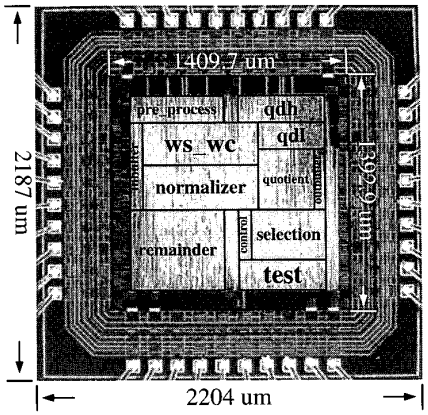

Figure 4: DIE photo of the radix 16/8/4/2 64b/32b integer divider.

complexity in hardware. In this paper, we have proposed a novel scheme to meliorate the performance of integer division. What we present employs operand scaling, table folding, and table partitioning techniques to realize the mixed radix 16/8/4/2 quotient selection tables.

6. REFERENCES

[1] K. Hwang, "Computer arithmetic: principles, architecture, and design," Reading: John Wiley & Son, 1979.

[2] S.-F. Oberman, and M.-J. Flynn, "Design issues in division and other floating-point operations" *IEEE Trans. Computers*, vol. 46, no. 2, pp. 154-161, Feb. 1997.

[3] C.-C. Wang, C.-J. Huang, and I.-Y. Chang, "Design and analysis of Radix-8/4/2 64b/32b integer divider using COMPASS Cell library," *VLSI Design, (accepted, no. 001025)* vol 46, no. 2, pp. 154-161, Feb. 1997.

[4] C.-C. Wang, C.-J. Huang, and G.-C. Lin, "A chip design of radix-4/2 64b/32b signed and unsigned integer divider using COMPASS cell library," *1999 Inter. Symp. on Circuits & Systems (ISCAS'99)* pp. 439-442, June, 1999.

[5] M. D. Ercegovac, and T. Lang, "Division and square root - digit-recurrence algorithms and implementations," Reading: Kluwer Academic Publishers, 1994.

[6] M. D. Ercegovac, and T. Lang, "Simple radix-4 division with operands scaling," *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1204-1208, Sept. 1990.

[7] P. Montuschi, and L. Ciminiera, "Design of a radix 4 division unit with simple selection table," *IEEE Trans. Computers*, vol. 41, no. 12, pp. 1606-1611, Dec. 1992.

[8] J. Fandrianto, "Algorithm for high-speed shared radix 4 division and radix 4 square root," *Proc. 8th IEEE Symp. on Computer Arithmetic*, pp. 73-79, Como, Italy, May 1987.