ALGORITHM for HIGH SPEED SHARED RADIX 8 DIVISION and RADIX 8 SQUARE ROOT

Jan Fandrianto

Integrated Information Technology, Inc.
Santa Clara, California 95054

Abstract

An algorithm for performing radix 8 division and radix 8 square root in a shared hardware will be described. To achieve short iteration cycle time, it utilizes an optimized "next quotient/root prediction PLA" generally used in a radix 4 SRT division with minimal redundancy. In addition, the partial remainder, partial radicand, quotient and root are generated and saved in redundant forms, thereby eliminating the slow-carry look-ahead adder from the critical path timing of the iteration cycle. This method successfully avoids the need to generate non-trivial divisor/root multiples (3x, 5x, etc.) and also avoids the complex radix 8 next quotient prediction PLA typically used in a conventional radix 8 SRT division. It also shows that a significant amount of hardware sharing may be achieved when square root and division are performed at the same radix.

Introduction

In recent years, many VLSI floating point chips have achieved fast addition and multiplication times.[1-4] In addition to utilizing more advanced integrated circuit process technology, many have also implemented full hardware solutions; for example: fast barrel shifters, full array Booth or Wallace tree multipliers, radix 16 multiplier,[3] full array divider,[4] fast 64-bit adder and many more.

Because the frequency of add and multiply occurring in many applications is very high, designing fast adder and multiplier at the expense of large silicon area is justified. On the other hand, division operation occurs less frequently and square root occurs at even less frequency.[5] Thus, only limited silicon area can justifiably be dedicated for divide/square root hardware. Nevertheless, their speed must also catch up to avoid imbalances among add, multiply and divide. Factoring all these conditions together, the design of high-speed divide/square root needs to go in the direction of higher radix.

The algorithm for shared radix 4 division/square root previously published[6] readily extends to radix 8 or higher implementation. The advantage is that the iteration speed of radix 8 may be in about the same order as radix 4. Direct extension from radix 4 to radix 8, however, introduces two main obstacles.

- 3x divisor/root multiples must be generated. For division, 3x divisor may be generated once by adding 2x and 1x divisors at the beginning of the division. For square root, 3x root must be generated on the fly at each iteration, because the root bits are continually being formed.
- complex "next quotient/root prediction PLA" (QR-PLA) is required. If only 3x multiple is provided, i.e., minimally redundant with measure of redundancy (MoR) = 4/7, the PLA would be practically impossible to implement. Even at maximally redundant (MoR = 7/7), the PLA is still large and slow in speed,[3] and worse yet, generation of 3x, 5x and 7x multiples would be required.

Obviously, the above reasons are enough for many to abandon the choice of direct extension of SRT in favor of cascading several radix 4 arrays to form higher radix.[7] The latter means that the iteration cycle time would become the sum of several radix 4 delay times, leading to lower cycle speeds.

This paper will detail the algorithm for radix 8 division/square root that directly extends SRT radix 4 division/square root[6] into radix 8 so as to maintain its short cycle time, but modified to avoid generating the non-trivial 3x, 5x, 7x divisor/root multiples, and keep the next quotient/root PLA at the same complexity as radix 4 minimally redundant (MoR = 2/3).

Algorithm and Implementation

The algorithm and implementation described in this paper deal with performing division and square root on normalized IEEE format FP numbers and its roundings as

specified by the IEEE standards.[8] Other floating point formats can also be used with relatively little modification.

The resulting exponent can be calculated without much difficulty. For division, the divisor's exponent is subtracted from the dividend's and the exponent bias must be added back to the result. For square root, the exponent is shifted right by one bit position to reflect division by two on the exponent, and the bias must be adjusted accordingly. After the quotient/root mantissa is completely formed and rounded, the exponent result may need to be incremented by one if the mantissa requires post normalization (one bit right shift).

Forming the mantissa part of the quotient/root will be the main discussion in this paper. The paper will first provide a review of radix 4 shared division and square root algorithm; then it will explain how the algorithm is extended to radix 8 while still maintaining many of the radix 4 hardware.

Shared Radix 4 Division and Square Root Algorithm

On division, the SRT algorithm[9] is based on solving this recursive equation:

$$P_{j+1} = r * P_j - q_{j+1} * d \quad (1)$$

with the range restriction

$$| P_{j+1} | \leq \frac{n}{r-1} * d \quad (2)$$

where

- P_j = partial remainder in the j-th cycle (P_0 = dividend)
- r = radix
- q_j = quotient digit selected in the j-th cycle
- d = divisor
- n = number of divisor multiples (not including zero)

In radix 4 with minimal redundancy, i.e., n = 2, the divisor multiples (1x, 2x) can be easily formed by mere shifting. The P-D (partial remainder-divisor) plot is shown in Figure 1. Table 1 shows the next quotient prediction QR-PLA, which is used in each iteration cycle to determine the "q_{j+1}" and satisfying equation (2). The PLA can be realized in 19 product terms. Figure 2 shows the block diagram of radix 4 division hardware. On each iteration, the most significant 8 bits of the carry and sum of the partial remainder (PR) are summed in an 8-bit carry look-ahead adder (CLA) and converted into sign magnitude representation.

Becoming input to the QR-PLA are 4 bits of the partial remainder magnitude together with the most significant 4 bits of the divisor. The outputs of the QR-PLA determine the divisor multiple to be selected (and also the next quotient digit). The carry-save adder array adds the divisor multiple to the redundant PR to form the next PR (also redundant in the form of sum and carry bits).

The next PR's are then left shifted 2 bits and latched into PR

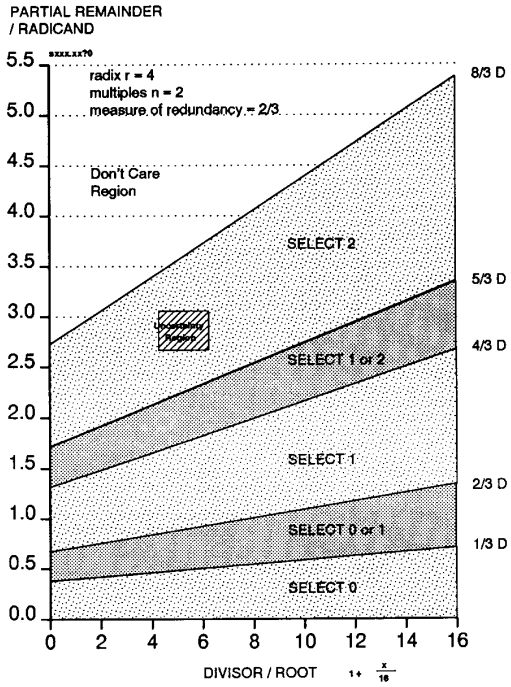

Fig. 1. SRT Division / Square -Root Graph of Redundancy

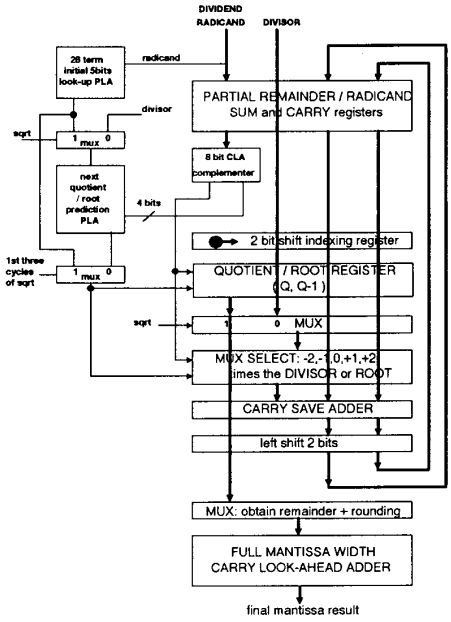

Fig. 2. Block Diagram of Shared Radix 4 Division / Square Root

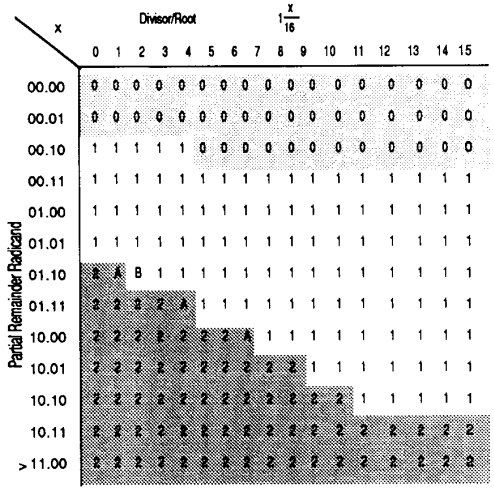

Table 1 Next Quotient/Root Selection PLA

registers for use in the next cycle. Figure 3 shows the percentages of time spent in the delay path of the iteration cycle.

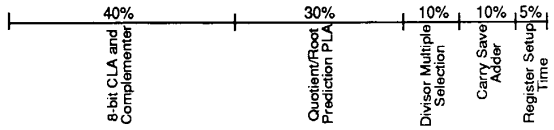

Figure 3

On square root, the root extraction is a process of completing the square,[10] where:

$$P_{j+1} = P_0 - Q_{j+1}^2, \quad (3)$$

after expanding P_0 in terms of P_j

$$P_{j+1} = r * P_j - q_{j+1} 2^{-(j+1)} \cdot (2Q_j + q_{j+1} 2^{-(j+1)}) \quad (4)$$

with range restriction

$$|P_{j+1}| \le \frac{n}{r-1} * Q_j \quad (5)$$

where

P_j = partial radicand in the j-th cycle (P_0 = radicand)
r = radix
q_j = root digit selected in j-th cycle
Q_j = root formed in j-th cycle
n = number of root multiples (not including zero)

The recursive relationship of (4) differs from SRT division recursion (1) only in the formation of divisor/root multiples. Thus the same QR-PLA can be used to predict the next root bit provided that first five most significant bits of the root are known to determine the x axis's position on the QR-PLA. A look-up table can be used to obtain that 5 MSB of root. As the iteration progresses, the root (Q_j of equation 4) is coming closer and closer to the final root. Meanwhile, the root multiples are generated, selected and added to the partial radicand to form the next PR. Figure 4 shows the P-D plot for square root. The plot is based on the P-D plot for SRT division, with one difference: the boundary lines that define the regions of the next root digit selection are "fuzzy". The fuzziness of the boundary lines is proportional to the inexactness of the root bits. The root is maximally inexact at the beginning of recursion; therefore, the initial 5-root bits obtained must be carefully chosen to make sure the QR-PLA of Table 1 is still valid without having the uncertainty region touching those fuzzy boundary lines.

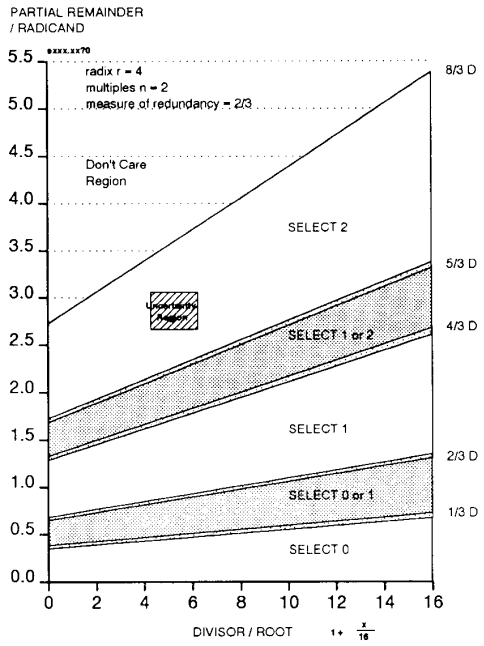

Fig. 4. Fuzzy boundary lines due to inexact root

Extension into Radix 8 Shared Division and Square Root

Based on the minimally redundant radix 4 SRT shared division and square root, the magnitude of the PR (P_{j+1}) is always within 2/3 D (range restriction requirement). For radix 8 (r = 3), after 3-bit left shift of the PR, the magnitude of the next PR ($|r * P_{j+1}|$) should be within 16/3 D. Figures 5a and 5b illustrate that relationship. Now, this PR has to be reduced again into the range of 2/3 D. This algorithm proposes a two-step overlapping reduction:

Step I. $Px_{j+1} = r * P_j - qx_{j+1} * d$ (6)
with $qx_{j+1} = -4, 0, +4$

Step II. $P_{j+1} = Px_{j+1} - qy_{j+1} * d$ (7)
with $qy_{j+1} = -2, -1, 0, +1, +2$

where Px = intermediate partial remainder
qx, qy = intermediate quotient digits selected

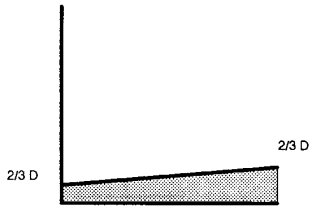

Fig. 5a. The magnitude of Pj after each iteration with MoR = 2/3

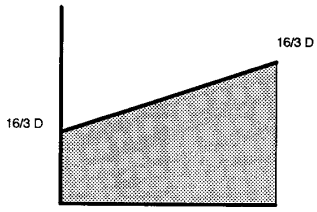

Fig. 5b. r * Pj where r = 8 (3 bit left shift of partial remainder)

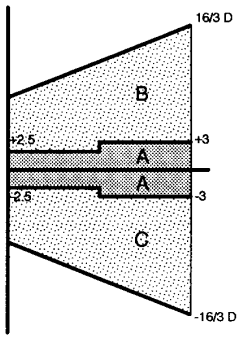

Fig. 6. The Range of the PR

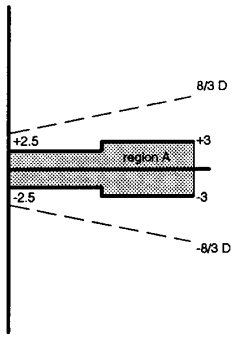

Fig. 6a. Region A stays unmodified

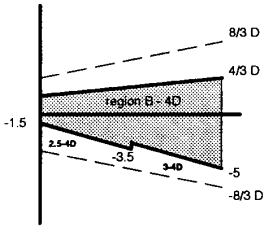

Fig. 6b. Region B after reduced by 4D

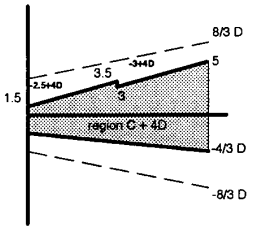

Fig. 6c. Region C after added by 4D

Step I requires a row of carry-save adder to subtract/add 4d into PR. Step I will reduce the range of PR into that of radix 4 range (< 8/3 D). Figure 6 shows the positive and negative ranges of the PR. Should the PR fall within region A, then $qx_{j+1} = 0$; and the PR goes to Step II unmodified (Figure 6a). In region B, 4d is subtracted out from PR (Figure 6b) and for region C, 4d is added (Figure 6c). Step II is exactly the same as the radix 4 version. The PR is reduced in a normal radix 4 fashion since its range is well within radix 4 SRT division/square root. The trick is now to overlap the execution of Step I with Step II.

Figure 7 shows the block diagram of radix 8 divide/square root hardware. From the beginning of the iteration, Step I and Step II reductions are performed simultaneously. Step I assumes that the PR is either in region B or C; thus the operation required must be either to subtract 4d or to add 4d into the PR. This decision can be easily determined by performing fast-carry look-ahead on the three most significant carry and sum bits of the PR. This carry-out result does not yield the actual sign of the PR. It takes a full carry look-ahead adder to determine the sign. However, the carry-out may be different from the actual sign only if the PR is in region A of Figure 6. Therefore, it is sufficient for this carry-out logic to determine whether the PR is in region B or C.

Step II reduction is performed with the assumption that the PR is in region A and this reduction is the same as radix 4.

The model division of both steps goes through the 9-bit carry look-ahead adder (CLA), complementer (to obtain the sign magnitude representation) and QR-PLA. Figure 8 shows the 9-bit CLA of Step I and Step II. The result of 9-bit CLA of Step II are compared with 2.5 (for divisor/root < 1.5) or 3.0 (for divisor/root ≥ 1.5). This would determine whether the PR is inside region A or outside region A. The CLA is 9 bits wide with the 5 most significant bits sitting to the left of the binary point. Since the range of $|r*P_j|$ has the maximum of 16/3 D (=10-2/3 as D approaches 2), this 5-bit magnitude to the left of the binary point is sufficient to represent the maximum range of the PR (including its sign bit).

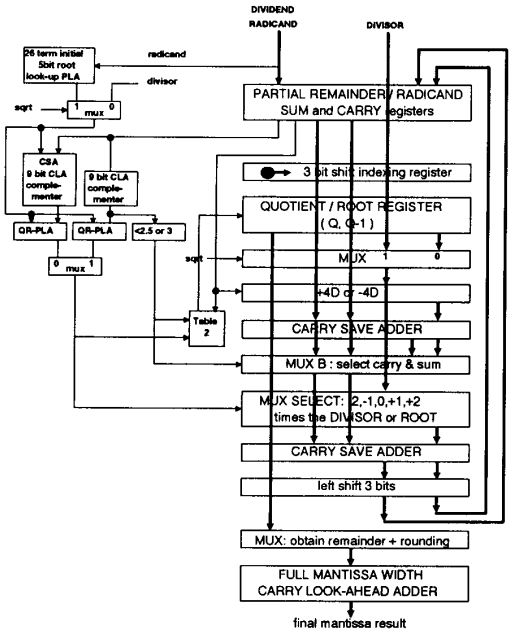

Fig. 7. Block Diagram of Shared Radix 8 Division / Square Root

If the PR is inside region A, Step I reduction is not selected and qx_{j+1} is zero and the divisor/root multiple is selected based on Step II's QR-PLA. If the PR is outside region A, then the result of the first CSA row is selected and the divisor/root multiple selection is based on Step I's QR-PLA. Combining the results of QR-PLA on both steps, Table 2 shows the quotient/root bits produced in that iteration. The quotient/root bits are also stored redundantly in "Q" and "Q-1" forms. Figure 9 shows the scheme for selecting and storing the quotient/root in the "Q" and "Q-1" forms.

For square root, the most significant 5 bits of the root including the hidden bit must be initially obtained through a look-up table. The look-up table would take the LSB of the exponent (which indicates odd/even exponent) and the 6 mantissa bits as inputs. The look-up table with 7 inputs and 5 outputs can be realized in PLA with 26 terms (Table 4). These initial root bits must be carefully chosen to make sure that after the initial two reductions of the PR, equation (5) is still satisfied (PR≤2/3 Q). Table 3 shows the choices for root multiple generation.

Step II \ Step I	Select −4	Select 0	Select +4
Select 0	−4	0	+4
Select 1	−3	1	+5
Select −1	−5	−1	−3
Select +2	−2	+2	+6
Select −2	−6	−2	−2

Table 2 Combined Next Quotient/Root Digit from Step I and Step II

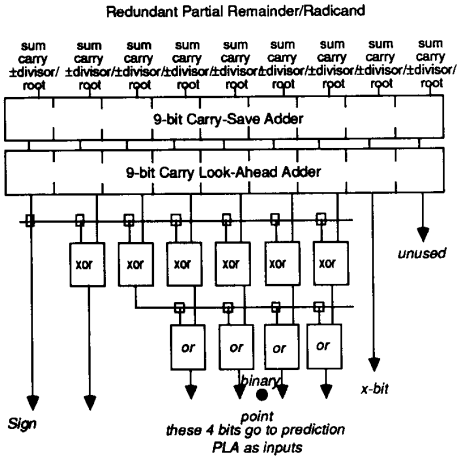

Figure 8a 9-bit CLA Complementer for Step I 's Model Division

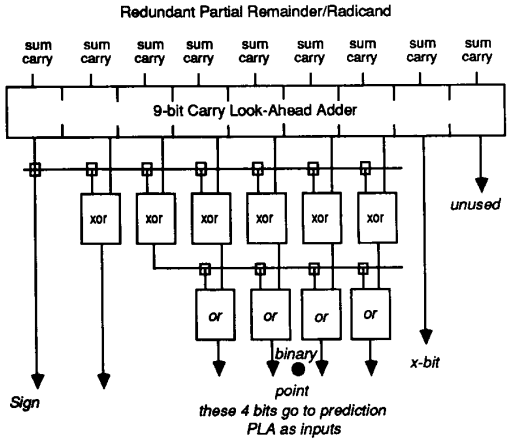

Figure 8b 9-bit CLA and Complementer for Step II's Model Division

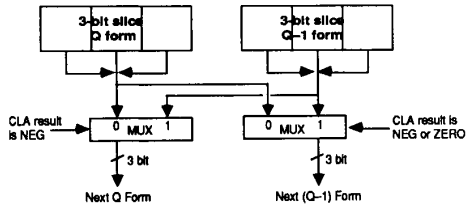

Figure 9 Quotient/Root Bits in Q and Q−1 Form

The combination of the quotient digits selected by both Step I (-4, +4) and Step II (-2, ..., +2) in fact indicates that the radix 8 digit set is from -6 to +6. The bound of the PR at the end of each iteration could then be 6/7 D, instead of 2/3 D as in pure radix 4. However, this 6/7 D bound cannot be used since the way the PR is partitioned into inside or outside, region A is not using 4 D as the dividing line. Instead, two horizontal lines of magnitude 2.5 and 3.0 are used for bounding region A. Magnitude comparison to these fixed quantities is much easier to realize than comparing PR to 4 D. Thus, determining inside/outside region A can be done in a short time.

Overall, the iteration cycle time is increased from radix 4 by a single carry-save adder and a multiplexer delay. The impact of this delay is relatively insignificant since it represents only about 15 percent additional delay on top of radix 4.

Examples of iterative square-rooting process using this algorithm are shown in the appendix.

Previous Root = 0.Q or 0.(Q-1)

Region	Select	Root Multiples (magnitude)										
A	0	0.	0	0	0	0	0	0	0	0	0	0
A	+1	0.	0	0	0	0	0	Q	0	0	0	1
B		0.	0	0	0	0	0	Q	1	0	0	1
C		0.	0	0	0	0	0	Q-1	1	0	0	1
A	-1	0.	0	0	0	0	0	Q-1	1	1	1	1
B		0.	0	0	0	0	0	Q	0	1	1	1
C		0.	0	0	0	0	0	Q-1	0	1	1	1
A	+2	0.	0	0	0	0	Q	0	0	1	0	0
B		0.	0	0	0	0	Q	1	0	1	0	0
C		0.	0	0	0	0	Q-1	1	0	1	0	0
A	-2	0.	0	0	0	0	Q-1	1	1	1	0	0
B		0.	0	0	0	0	Q	0	1	1	0	0
C		0.	0	0	0	0	Q-1	0	1	1	0	0
-	+4	0.	0	0	0	Q	0	1	0	0	0	0
-	-4	0.	0	0	0	Q-1	1	1	0	0	0	0

Table 3 Choice of Root Multiples

Rounding

The iteration continues until the guard and round bits of the quotient/root have been produced. At this time, the PR is converted into non-redundant form by a full mantissa width CLA. The sign of the PR determines which "Q" and "Q-1" forms are the intermediate quotient/root. A non-zero PR means that the sticky bit is set; otherwise, the sticky bit is zero. Depending on the rounding mode, the LSB, guard, round and sticky bits determine the round-up condition on the LSB of the intermediate quotient/root. Finally, another CLA cycle is required to increment the

Comparison to Other Schemes

This algorithm is an extension of the previously published radix 4 shared division and square root.[6]

The extension to radix 8 is achieved by overlapping the two steps of quotient/root prediction logic (Step I and Step II).

Taylor's paper on radix 16 SRT Dividers with overlapped quotient selection stages[7] mentioned the possibility of overlapping the two steps but chose not to use it in favor of overlapping the quotient selection with divisor multiple and PR formations. Indeed, Taylor's method for overlapping the 2 steps gave rise to costly duplication of quotient selection logic hardware.

This paper, on the other hand, shows how early determination of the sign of the PR and the introduction of multiplexer B in the array (Figure 7) eliminate the above mentioned duplication of quotient selection logic, while still achieving true parallelism. Importantly, this overlapping execution of Step I and Step II which naturally works for division, also works for square-rooting. This is contributed to the carefully chosen digits on QR-PLA that avoid touching the fuzzy boundaries of Figure 4.

It must also be noted that performing square root in the same radix as division enables hardware sharing in the following logic groups:

- Quotient/root prediction logic
- Carry-save adders
- Registers holding PR and quotient/root
- Sequencing logic that controls the number of iterations
- Radix controlled shifting logic for PR and quotient digits

Input = 7 bits : LSB of unbiased exponent (actual exp – IEEE bias)
6 MSB (bits) of mantissa (right after the hidden bit
Output = 5 bits : Hidden bit followed by 4 MSB of the initial root
Output polarity is positive true
Number of terms = 26

1-10111	00001	011000-	00001
-01111-	00100	00011--	00010
11011--	00010	10011--	00001
01-11--	00001	011--1-	00010
1--101-	00001	10000--	10111
011-1--	00010	00-10--	00001
00-01--	00001	0101---	00001
11-01--	00001	-011-0-	00011
--110--	00010	-010---	00010
10--1--	11000	11-10--	11101
10-1---	11000	110----	11100
1-10---	11010	-1-0---	00100
01-----	00100	0------	10000

Table 4 Look-Up PLA for Prediction of the First 5 Root Bits

Conclusion

Radix 8 SRT division and square root provides an attractive solution to faster division and square root operation, thereby closing on the speed gap that exists as faster solutions on add and multiply become available.

This paper has demonstrated the viability of a hardware solution on radix 8 shared division and square root being an extension of the radix 4 algorithm.

The main advantages are:

- avoid the need to generate non-trivial divisor/root multiples;
- maintain the complexity of the next quotient/root prediction PLA to the same level as radix 4;
- need relatively little additional hardware on top of radix 4;
- introduce little additional delay to the iteration cycle time;
- efficient hardware sharing between division and square root.

Acknowledgements

The author is indebted to Selfia Halim and the Symposium reviewers for their incisive comments on the technical accuracy and to Lea McGowan and Tobing Soebroto for editing and typesetting the paper.

References

[1] Molnar, K., et al, "A 40 MHz 64b Floating Point Coprocessor," ISSCC Digest of Technical Papers, February 1989.

[2] Ciminiera, L., "Parallel Multipliers Based on Horizontal Compressors," Proceedings of the 8th Symposium on Computer Arithmetic, May 1987, pp. 63–69.

[3] Weitek Corp., "WTL 2264/WTL2265 Floating Point Multiplier/Divider and ALU," Specifications, July 1986.

[4] McAllister, et al, "An NMOS 64b Floating Point Chip Set," ISSCC Digest of Technical Papers, February 1986, pp. 34–35.

[5] Whetstone Program: A Benchmark for Floating Point Performance.

[6] Fandrianto, J., "Algorithm for High Speed Shared Radix 4 Division and Radix 4 Square Root," Proceedings of the 8th Symposium on Computer Arithmetic, May 1987, pp. 73–79.

[7] Taylor, G. S., "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages," Proceedings of the 7th Symposium on Computer Arithmetic, June 1985, pp. 64–71.

[8] IEEE Standard for Binary Floating Point Arithmetic, 1985.

[9] Atkins, D. E., "Higher-Radix Division Using Estimates of the Divisor and Partial Remainder," IEEE Transactions on Computers, 17, No. 10, October 1968, pp. 925–934.

[10] Hwang, K., "Computer Arithmetic: Principles, Architecture and Design," New York, John Wiley and Sons, Inc., 1978.

Appendix

```
ODD EXPONENT
     What is the radicand (two 32 bit integers) ?
                40023456 789abcde

predicted root bits : 11000
 Region B and sel2=0 sel1=1 negate=1
sum PR: 0010010001101000101011001111000100110101011110011011110000
cry PR: 0000000000000000000000000000000000000000000000000000000000
Qj  I :-010000000000000000000000000000000000000000000000000000000
Qj II :+000111000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=0 negate=0
sum PR: 0111101000110000010101110110010100000011101010011101110000
cry PR: 1000100100010101000100000010010010101000001001000000010000
Qj  I :-000000000000000000000000000000000000000000000000000000000
Qj II :-000000000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=1 negate=0
sum PR: 0110011011010101110001011111001010100011100100010011111000
cry PR: 1011001101010101011101100101101010111010110111011100001000
Qj  I :-000000000000000000000000000000000000000000000000000000000
Qj II :-000110000010000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=1 negate=0
sum PR: 1001001101111010011000101011111100110101100110000001111000
cry PR: 0111110101011111011111111010101110111101110111111110001000
Qj  I :-000000000000000000000000000000000000000000000000000000000
Qj II :-000110000010001000000000000000000000000000000000000000000
 Region C and sel2=1 sel1=0 negate=0
sum PR: 0100111111000111000101110101101110111101110000000001111000
cry PR: 0111010111110111111110111111101110110111111111111110001000
Qj  I :+011000001000110000000000000000000000000000000000000000000
Qj II :-001100000100010100000000000000000000000000000000000000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 1111010001000001101011110100010100100010000000000011111000
cry PR: 1011101111101110111010110111011101110111111111111100001000
Qj  I :+011000001000101110000000000000000000000000000000000000000
Qj II :-000110000010001011001000000000000000000000000000000000000
 Region C and sel2=1 sel1=0 negate=0
sum PR: 0100101101110101001010100011100100010000000000000011111000
cry PR: 0111100101010101011010111010110111011111111111111100001000
Qj  I :+011000001000101100110000000000000000000000000000000000000
Qj II :-001100000100010110010100000000000000000000000000000000000
 Region C and sel2=0 sel1=0 negate=0
sum PR: 0111110111011010011100011100100010000000000000000011111000
cry PR: 0010101010100111000111010110111011111111111111111100001000
Qj  I :+011000001000101100101110000000000000000000000000000000000
Qj II :-000000000000000000000000000000000000000000000000000000000
 Region B and sel2=0 sel1=1 negate=1
sum PR: 1100100011111110001111001000100000000000000000000011111000
cry PR: 0111111101100111101010110111011111111111111111111100001000
Qj  I :-011000001000101100101100010000000000000000000000000000000
Qj II :+000110000010001011001011000111000000000000000000000000000
 Region A and sel2=0 sel1=0 negate=1
sum PR: 0111000100010111110110111111111111111111111111111111110000
cry PR: 1000111011010100011010001100000000000000000000000000010000
Qj  I :+000000000000000000000000000000000000000000000000000000000
Qj II :+000000000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=0 negate=1
sum PR: 1111111000011101100110011111111111111111111111111100000000
cry PR: 0000000101000100100011000000000000000000000000000100000000
Qj  I :+000000000000000000000000000000000000000000000000000000000
Qj II :+000000000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=0 negate=1
sum PR: 1111101011001000101011111111111111111111111111100000000000
cry PR: 0000000001001000100000000000000000000000000000100000000000
Qj  I :+000000000000000000000000000000000000000000000000000000000
Qj II :+000000000000000000000000000000000000000000000000000000000
 Region A and sel2=1 sel1=0 negate=1
sum PR: 1101010000000000101111111111111111111111111110000000000000
cry PR: 0000010010001000000000000000000000000000010000000000000000
Qj  I :+000000000000000000000000000000000000000000000000000000000
Qj II :+001100000100010110010110001011111111111100000000000000000
 Region B and sel2=0 sel1=1 negate=1

sum PR: 0000011001100111010011101000000000000100000000000000000000
cry PR: 0100000000010001011000101111111111110100000000000000000000
Qj  I :-011000001000101100101100010111111111100100000000000000000
Qj II :+000110000010001011001011000101111111111100111000000000000
 Region A and sel2=0 sel1=1 negate=1
sum PR: 0110111001010101100011000100000000010001000111111111110000
cry PR: 1000001000101100111010101111111101100111000000000000010000
Qj  I :+000000000000000000000000000000000000000000000000000000000
Qj II :+000110000010001011001011000101111111111100101111000000000
```

```
 Region B and sel2=0 sel1=1 negate=1
sum PR: 10100010110111011001000101000011011101001000011111100000000
cry PR: 10100010010011001101010101111011001100101111000001000000000
Qj  I :-01100000100010110010110001011111111111001011101000000000000
Qj II :+00011000001000101100101100010111111111110010111011100000000
 Region A and sel2=0 sel1=1 negate=1
sum PR: 00011011111101111111001001001110111111001011100011111110000
cry PR: 11001010000110000011100101100110010100101011100000010000000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00011000001000101100101100010111111111110010111011011110000
 Region A and sel2=0 sel1=1 negate=1
sum PR: 01001110011010000000001111110101000100111000010001000000000
cry PR: 10100011001011111101101000111011111110101011101011100000000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00011000001000101100101100010111111111110010111010110111110

Root  : 00110000010001011001011000101111111111110010111010110110111

Radicand   = 40023456 789abcde
Final root = 3ff822cb 17ff2eb7



EVEN EXPONENT
     What is the radicand (two 32 bit integers) ?
                3ffabcde 98765431


predicted root bits : 10101
 Region B and sel2=0 sel1=1 negate=1
sum PR: 00011010101111001101111010011000011101100101010000011000100
cry PR: 00000000000000000000000000000000000000000000000000000000000000
Qj  I :-01000000000000000000000000000000000000000000000000000000000000
Qj II :+00011100000000000000000000000000000000000000000000000000000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 01100001110101001110001010111011001010000011101011000100000
cry PR: 01010100000100100001001000010001010100001000010010000000000
Qj  I :+01011000000000000000000000000000000000000000000000000000000000
Qj II :-00010100100000000000000000000000000000000000000000000000000000
 Region C and sel2=0 sel1=0 negate=0
sum PR: 00110100110010000101101011100010011010111011111011111000000
cry PR: 10010110011011110100101100111011001010011001010010000001000
Qj  I :+01010011000000000000000000000000000000000000000000000000000000
Qj II :-00010100100100000000000000000000000000000000000000000000000000
 Region B and sel2=0 sel1=0 negate=1
sum PR: 10110010110000111101001100010111011110011010111000011111000
cry PR: 10011010011110010101110111010101001011010110011111000001000
Qj  I :-01010010101000000000000000000000000000000000000000000000000000
Qj II :+00000000000000000000000000000000000000000000000000000000000000
 Region C and sel2=1 sel1=0 negate=0
sum PR: 10001010100101100111010001001101010111100111011111111110000
cry PR: 01001010010100110001001011001010100001100010000000000010000
Qj  I :+01010010101111000000000000000000000000000000000000000000000000
Qj II :-00101001010111010000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=1 negate=1
sum PR: 10001111101111011100010111010110011010111000000001111110000
cry PR: 01100010100001100111010101010011101011001111111110000001000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00010100101011101111100000000000000000000000000000000000000000
 Region B and sel2=0 sel1=1 negate=1
sum PR: 11001100101010100010000101110001110111111111111111110000000
cry PR: 01101010111011101010101001000101000100000000000000001000000
Qj  I :-01010010101110111110100000000000000000000000000000000000000000
Qj II :+00010100101011101111011100000000000000000000000000000000000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 00110011100110100001110101110001110111111111111111111110000
cry PR: 10011000110010111110010100010100010000000000000000000010000
Qj  I :+01010010101110111110101101100000000000000000000000000000000000
Qj II :-00010100101011101111010100100000000000000000000000000000000000
 Region B and sel2=0 sel1=0 negate=0
sum PR: 10111100011001000111010011010111000000000000000000111111000
cry PR: 10010111001100101010011001101100111111111111111111000001000
Qj  I :-01010010101111011110101010100000000000000000000000000000000000
Qj II :-00000000000000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=0 negate=0
sum PR: 00011001001010100101001110001000000000000000000000000001000
cry PR: 11101101101011111101101111111111111111111111111111111111111000
Qj  I :-00000000000000000000000000000000000000000000000000000000000000
Qj II :-00000000000000000000000000000000000000000000000000000000000000
 Region B and sel2=0 sel1=1 negate=1
sum PR: 01011011110100111011110001000000000000000000000000001111000
cry PR: 11011010111111011011111111111111111111111111111111110001000
Qj  I :-01010010101111011110101011000000100000000000000000000000000000
Qj II :+00010100101011101111010101100000011000000000000000000000000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 01111011011111010000001011100100011111111111111111111110000
cry PR: 01000101010100111110111100001011100000000000000000000000010000
Qj  I :+01010010101111011111010101100000010110000000000000000000000000
Qj II :-00010100101011101111010101100000010100100000000000000000000000
 Region A and sel2=1 sel1=0 negate=0
sum PR: 11101010011111010111011100111011100000000000000111111000
cry PR: 00110111000100001001000110101000110111111111111111000001000
Qj  I :-00000000000000000000000000000000000000000000000000000000000000
Qj II :-00010100101011101111010101100000010101001000000000000000000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 01011110011111111001110101100000011001000000000000001111000
cry PR: 01100011000000010101001110111010110111111111111111110001000
Qj  I :+01010010101111011110101011000000101010011100000000000000000000
Qj II :-00010100101011101111010101100000010101001100100000000000000000
 Region A and sel2=0 sel1=0 negate=1
sum PR: 00000011000111011101010000001110010000100100000000011111000
cry PR: 11111010101010000101101011110101011100110111111111100001000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00000000000000000000000000000000000000000000000000000000000000
 Region A and sel2=0 sel1=1 negate=1
sum PR: 10010010011010110000111100101001100010011111111111110000000
cry PR: 01010000100001001010000001010010000100100000000000010000000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00010100101011101111010101100000010101001100111111110000000
 Region C and sel2=0 sel1=1 negate=0
sum PR: 10111110011001011101111011011111100100110000010000000000
cry PR: 00000010101101000010100100000000001001100111111100000000000
Qj  I :+01010010101111011111010101100000010101001100111111010000000
Qj II :-00010100101011101111010101100000010101001100111111010010000
 Region A and sel2=0 sel1=0 negate=1
sum PR: 00000010100001111010001110011111011100000110110011011111000
cry PR: 11110110101100001010100011101001001101110000010000000001000
Qj  I :+00000000000000000000000000000000000000000000000000000000000000
Qj II :+00000000000000000000000000000000000000000000000000000000000000

Root  : 00101001010111011110101011000000101010011001111101001111111

Radicand   = 3ffabcde 98765431
Final root = 3ff4aef5 6054cfa8
```