Processing Element for Reciprocal and Reciprocal Square Root

Aki Happonen, Perttu Salmela* and Adrian Burian†

Nokia Devices R&D
Elektroniikkatie 3
FIN-90570 Oulu
Finland
aki.p.happonen@nokia.com

* Department of Computer Systems
Tampere University of Technology
P.O.Box 553, FIN-33101 Tampere
Finland
perttu.salmela@tut.fi

†Nokia Devices R&D
Visiokatu 4
FIN-33720 Tampere
Finland
adrian.burian@nokia.com

Abstract-- Our topic is reciprocal and reciprocal square root and we propose processing element without iterations. Implementation is based on look-up tables and computation in logarithm domain. Simulation results and implementation complexity for 16 bits input operand with fixed point and floating point output formants are presented. Two truncation methods, LSB and half LSB, for output are presented and their impact to output error is discussed. Relative error for reciprocal is less than 5% for floating point output format.

Index Terms-- division, square root, reciprocal

I. Introduction

Reciprocal and reciprocal square root are complex functions for which the accuracy and computing speed in case of real time solutions are demanding. Traditional digital signal processor (DSP) with multiply-accumulate (MAC) architecture does not support fast implementation of reciprocal and reciprocal square root, so these functions require specific hardware (HW) acceleration. In particular, the inverse square root is required by algorithms which need vector normalization. Thus, it is a fundamental operation needed in a wide range of applications, from computer graphics or algorithms, to solving linear systems (e.g. Cholesky decomposition requires division by square root for non-diagonal elements).

Inverse square root function can be computed with methods like digit recurrence [1], piecewise linear approximation [4][10][13], Taylor series expansion [5], or polynomial approximation [12][14]. Newton's iterations can be applied on every method to obtain finer accuracy. In addition, the computation can be alleviated with logarithm and exponent functions, so to work in logarithmic domain where the computation of the inverse square root is straightforward [2][3][6] [8].

The algorithms proposed in [1] consists of a digit-by-digit part followed a linear approximation. The latter is implemented by a digit-recurrence, which uses the digits produced by the digit-by-digit part. Both parts were executed in an overlapped manner, so that the total number of cycles was half of that would be required by the digit-by-digit part alone. Because of the approximation, rounding the result cannot be obtained directly all cases; a variable-time was proposed, that produces the correctly rounded result with a small average overhead. The linear approximation has quadratic convergence and performs roughly half of the iterations as compared to conventional digit-recurrence algorithm.

The method proposed in [4] was specifically designed for QVGA graphics. Instead of providing datapath bit-exact arithmetic at a high expense regarding cost, latency, and power-consumption, the cost of the reciprocal was reduced by exploiting the limitations of the human visual system. The proposed method allows a reasonable amount of error to be introduced in the computation process, since the authors considers that this is not introducing noticeable artifacts. Their experiments on a QVGA display have suggested that a relative error of maximum 5% does not introduce visible artifacts in the generated image. A 14-bit operand was considered,the reciprocal implementation required an inexpensive operand prescaler, one 1k lookup table with 10-bit entries, and a 5-bit adder. With these, a maximum relative error of the result of only 1.5% over the entire range of the operand was achieved. Hardware synthesis was realized in a typical 0.18µm process technology, and it required 1600 standard cells to achieve a latency of 2.5ns.

An example of design and implementation of a decimal reciprocal unit is given in [5], and of a double precision floating-point number in [11]. Both these solutions are using Newton-Raphson iterations. In [5], the algorithm applied to create the look-up table is based on the first-order Taylor expansion. In this design, the method requires to read a value from table and a multiplication of this value with the modified operand to obtain the initial approximation. Three Newton-Raphson iterations are used at the second stage of the implementation. A decimal squarer, a decimal multiplier and a decimal subtracter are used to realize it. In [11], a 210 × 20 bits ROM table was needed. Two Newton-Raphson iterations were needed, and the reciprocal computation required 11 clock cycles.

The method presented in [12] used a second-degree minimax polynomial approximation to obtain an initial estimate of the reciprocal and the inverse square root values. In the second stage, a modified Goldschmidt iteration was performed. The high accuracy of the initial

1-4244-2493-1/08/$20.00 ©2008 IEEE

approximation allows obtaining double-precision results by computing a single Goldschmidt iteration, significantly reducing the latency of the algorithm. For comparison purposes, other efficient methods were considered by the authors, and their execution times and area costs were estimated. Their conclusion was that the proposed method achieves an important speed-up over previous methods, with similar hardware requirements. The method from [14] used a degree-3 polynomial approximation that only required one complete multiplication and a small number of additions or subtractions. The polynomial approximation was determined using a specific method for the choice of the coefficients and the splitting of the input interval. Using some specific non-standard decomposition of the domain and quantification of some polynomial coefficients, a small and fast hardware implementation was achieved.

An interesting analysis for approximation methods that allows reasonably effective isolation of the 'worst cases' for the reciprocal function was presented in [7]. The described method provides the possibility to improve the difficulty bounds on the reciprocal square root functions for larger precisions. It should be added that the proposed method has feasibility problems for the extreme case of quad-precision reciprocal square roots. Lower bounds on the number of bits for approximation are investigated in [9]. These investigations were accurate to a unit in the last place sufficient to guarantee that correct round and sticky bits can be determined. Known lower bounds for quotients and square roots were considered, and new lower bounds for root reciprocals were proved. It was shown that algorithms can be designed for obtaining the round and sticky bits based on the bit pattern of an approximation computed to the required accuracy. Some improvement of the known lower bound for reciprocals and division is achievable at the cost of somewhat more complex hardware for rounding.

In this paper a processing element for reciprocal and reciprocal square root are presented. Reference for relative error is taken from 3D graphics application and the same reference was used in [4]. The processing elements use log space to compute reciprocal and reciprocal square root. We translate operands to log space by using look-up-tables (LUTs) and so only needed computations are subtraction and shifting operations. The outputs are translated from log space to decimal space by using LUTs respectively.

The paper is organized as following. In the Section II we introduce used architectures and complexity of processing element, simulation results are illustrated in Section III and summary of the work is given in Section IV.

II. Architectures and Complexity

In this section architecture and its implementation complexity in logic gate equivalent (GE) is presented. Both fixed point and floating point architectures are illustrated and main differences are discussed and shown in complexity analysis.

A. Architecture

The input word is translated to log space so that reciprocal and reciprocal square root computing is realized through subtracting and shifting operations. The translations from decimal scale to logarithm scale and from logarithm scale to decimal scale are done using look-up-tables (LUT) and combinational logic. The LUTs contain two ROMs, one for coarse value and the other for fine value of output. LUTs are generated to get the best piecewise approximation for log function. The output of Dec2Log is generated by adding scaled and corrected values of ROM outputs to minimize error. In Figure 1 the architecture of a Dec2Log block is illustrated. Log2Dec block has similar structure. Complexity in terms of size and data width of ROMs for 16 bits implementation used in our simulations are given in Table 1.

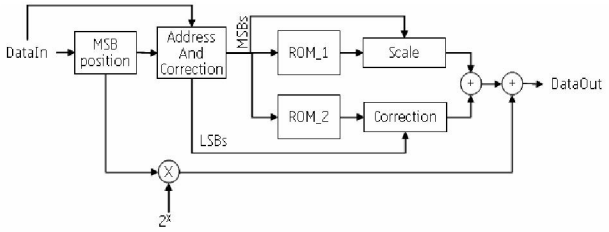

Figure 1. Dec2Log Block LUT Architecture.

Table 1. Complexity of ROMs.

Block		Size	Word length
Dec2Log	ROM_1	8x8	17 bits
	ROM_2	8x8	11 bits
Log2Dec	ROM_1	4x4	14 bits
	ROM_2	4x4	9 bit

The first architecture for fixed point implementation with 16-bit input and output ports is presented in Figure 2. For reciprocal computing the input data is converted to logarithm scale and that value is subtracted from constant value. The constant value represents value of one. The result of the subtraction is the converted to decimal scale. Reciprocal square root computing happens in similar manner except the shifted value of logarithm is subtracted from constant one.

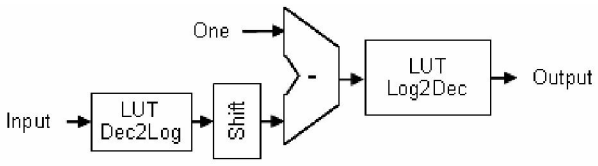

Figure 2.Fixed point architecture.

Floating point architecture - illustrated in Figure 3 - is similar to fixed point architecture except that there are two outputs from the Log2Dec block. The Output1 is 16-bit decimal number and Output2 is 4-bit fractional number.

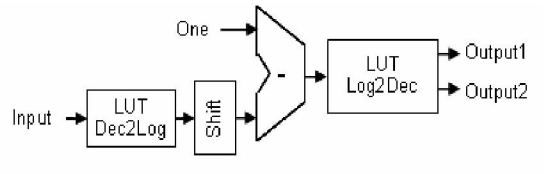

Figure 3.Floating point architecture.

B. Complexity

The processing element is synthesized using 65nm CMOS process and targeted for 160MHz clock frequency. The pipeline of the proposed architecture has three stages: read input data, convert decimal scale to logarithm scale and subtract, and convert from logarithm scale to decimal scale.

Table 2 illustrates complexity of both implementations. In both implementations the portion of non-combinational logic is roughly 500 logic GEs. Higher gate count for floating point implementation comes from higher number of outputs. The fractional output calculation requires extra logic to select bits that are truncated from final output in fixed point implementation. Differences in Dec2Log complexity numbers between fixed point and floating point implementation comes from normal variations between synthesis rounds. In both cases synthesis parameters have been the same

Table 2. Complexity of processing element in terms of GEs.

Block	Fixed point	Floating point
Dec2Log	2170	2230
Log2Dec	1640	1960
Total	4790	5160

III. SIMULATION RESULTS

Simulations are done in Matlab using bit exact simulation model of processing element and results are compared to floating point results. Processing element implementation is verified against Matlab model. In both case two rounding options are presented: truncation from least significant bit (LSB) and truncation from half LSB. In half LSB truncation the result is rounded upwards if the first cut bit is logical 1. In LSB truncation the first cut bit does not affect to the result.

A. Non-linearity of architecture

The used size of ROMs is trade-off between required accuracy and implementation complexity. The size of ROM tables defines the system performance and the processing element capabilities. Nonlinear distortion of the processing element can be illustrated by function $y = 2^{\log_2 x}$, where x is the input and y is the output of the system. If the system is linear, y is equal to x but in our system $y = x + \varepsilon$, where ε is the error. Non-linearity error depends on size of ROMs and it is one design parameter to meet required accuracy and maximum logic area.

In Figure 4, the relative error for the used LUTs is given for a 16 bits fixed point implementation. Dashed line shows LSB truncation result and solid line shows half LSB truncation results. Figure 4 is scaled to present half LSB truncation error and LSB truncation errors for small inputs are not visible. Floating point implementations have similar non-linearity due fixed point input.

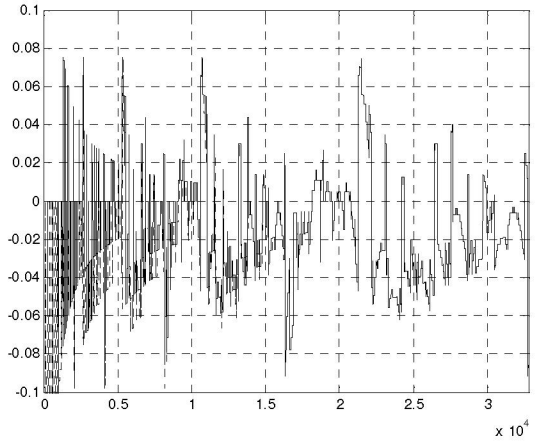

Figure 4. Relative error of the processing element.

B. Reciprocal simulation results.

Relative error for reciprocal is defined

$$\varepsilon = \frac{\frac{1}{X_{fxd}} - \frac{1}{X_{fl}}}{\frac{1}{X_{fl}}} \qquad (1)$$

where $1/X_{fxd}$ is the reciprocal computed in proposed architecture and $1/X_{fl}$ is floating point reference. This approach is selected to measure reciprocal error because for 3D graphics application relative error should be less than 5% [4]. This sets real application benchmark requirement for proposed method.

The first simulation was run with fixed point implementation having 16-bit input and output. We noticed that fixed point implementation causes too high relative error. The results are shown in upper part of Figure 5. The lower part of Figure 5 shows relative error for floating point implementation. In both cases LSB truncations are shown as dashed line and half LSB truncation is denoted with solid line. Floating point implementation with half LSB truncation fulfils 5% error target.

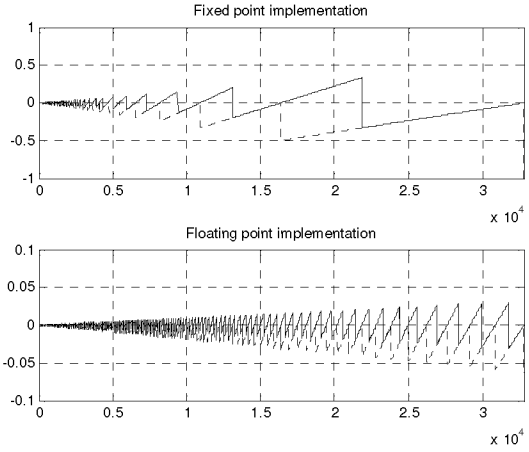

Figure 5. Relative error of reciprocal.

C. Reciprocal square root

The relative error of the reciprocal square root is calculated in a similar manner as the reciprocal relative error. Simulation results are shown in Figure 6. The upper part of Figure 6 shows error for fixed point implementation and lower part shows error for floating point implementation. In both cases LSB truncation is shown dashed line and half LSB is shown solid line.

In both calculations and implementations LSB truncation causes negative bias to error that is visible Figure 5 . Half LSB truncation cases the error signal has almost zero mean.

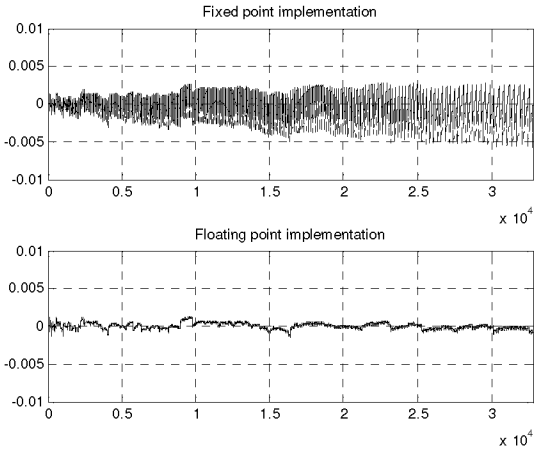

Figure 6. Relative error of reciprocal square root.

IV. CONCLUSIONS

Fixed and floating point processing elements for reciprocal and reciprocal square root operations were proposed in this paper. The architectures of the processing elements were based on alleviated computations in logarithmic domain. Transformations between decimal and logarithmic domain were carried out with the aid of dedicated units applying scaling and correction terms obtained with LUTs. The internal operation of the processing elements was described and complexity estimates of the data path and memories were presented. Both LSB and half LSB truncation impact to accuracy are presented. Finally, the inherent non-linearity of the architecture and the approximation errors as a consequence of the non-linearity were analyzed. It was shown that the processing element achieved sufficient accuracy of the reciprocal and reciprocal square root operations. The processing element could be used in applications like 3D graphic processing and Cholesky decomposition.

REFERENCES

[1] E. Antelo, T. Lang, P. Montuschi, A. Nannarelli, "Low latency digit-recurrence reciprocal and square-root reciprocal algorithm and architecture", in Proc. of the 17th IEEE Symposium on Computer Arithmetic, 27-29 June 2005, pp. 147 - 154.

[2] J. N. Coleman and E. I. Chester, "A 32 bit logarithmic arithmetic unit and its performance compared to floating-point," in Proc. 14th IEEE Symp on Computer Arithmetic, Adelaide, Australia, Apr. 1999, pp. 142–151.

[3] C.-H. Chen and C.-Y. Lee, "A cost effective lighting processor for 3D graphics application," in Proc. Int. Conf. on Image Processing, vol. 2, Kobe, Japan, Oct. 1999, pp. 792–796.

[4] D. Crisu, S. Vassiliadis, S. Cotofana, P. Liuha, "Low cost and latency embedded 3D graphics reciprocation", in Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS, Volume 2, 23-26 May 2004, pp. II - 905-8.

[5] Dongdong Chen and Seok-Bum Ko, "Design and Implementation of Decimal Reciprocal Unit", Proc. of the Canadian Conference on Electrical and Computer Engineering, CCECE 2007, 22-26 April 2007, pp. 1094 - 1097.

[6] A. Happonen, E. Hemming, and M. Juntti, "A novel coarse grain reconfigurable processing element architecture," in Proc. Midwest Symp. Circuits and Systems, vol. 3, Cairo, Egypt, Dec. 2003, pp. 827–830.

[7] J. Harrison, "Isolating critical cases for reciprocals using integer factorization", in Proc. of the 16th IEEE Symposium on Computer Arithmetic, 15-18 June 2003, pp. 148 - 157 .

[8] M. Haselman, M. Beauchamp, K. Underwood, and K. S. Hemmert, "A comparison of floating point and logarithmic number systems for FPGAs," in Proc. 13th Ann. IEEE Symp. on Field-Programmable Custom Computing Machines, Napa, CA, USA, Apr. 2005, pp. 181–190.

[9] C. Iordache and D.W. Matula, "On infinitely precise rounding for division, square root, reciprocal and square root reciprocal", in Proc. of the 14th IEEE Symposium on Computer Arithmetic, 14-16 April 1999, pp. 233 - 240 .

[10] P. Kornerup and D.W. Matula, "Single precision reciprocals by multipartite table lookup", in Proc. of the 17th IEEE Symposium on Computer Arithmetic, 27-29 June 2005, pp. 240 - 248 .

[11] U. Kucukkabak and A. Akkas, "Design and implementation of reciprocal unit using table look-up and Newton-Raphson iteration", in Proc. of Euromicro Symposium on Digital System Design, 31 Aug.- 3 Sept. 2004, pp. 249 - 253.

[12] J.A. Pineiro and J.D. Bruguera, "High-speed double-precision computation of reciprocal, division, square root, and inverse square root", IEEE Transactions on Computers, Vol. 51, Issue 12, Dec. 2002, pp. 1377 - 1388.

[13] M.J. Schulte, J.E. Stine, K.E. Wires, "High-speed reciprocal approximations", in Proc. of the Thirty-First Asilomar Conference on Signals, Systems & Computers, Volume 2, 2-5 Nov. 1997, pp. 1183 - 1187.

[14] A. Tisserand, "Hardware reciprocation using degree-3 polynomials but only 1 complete multiplication", in Proc. of the 50th Midwest Symposium on Circuits and Systems, MWSCAS, 5-8 Aug. 2007, pp. 301 - 304.