A Single/Double Precision Floating-Point Reciprocal Unit Design for Multimedia Applications

Metin Mete Özbilen[1] and Mustafa Gök[2]

[1]Mersin University, Engineering Faculty, Department of Computer Science,
33342, Mersin, Turkey
mmozbilen@mersin.edu.tr
[2]Çukurova University, Engineering Faculty, Department of Electric and Electronic,
01330, Adana, Turkey
musgok@cu.edu.tr

Abstract

Modern graphic processors, multimedia processors, and general-purpose processors with multimedia extensions provide SIMD floating-point instructions. SIMD floating-point reciprocal operation is commonly used in 2D and 3D applications, which mostly use single precision floating-point operands. Consequently, efficient single precision units are crucial for high performance systems. This paper introduces a packed floating point reciprocal unit that can perform reciprocal of either a double precision or two parallel single-precision floating-point numbers using fast multipliers.

1. Introduction

Modern general purpose processors provide special instructions for multimedia applications [1]. New generations of the general purpose processors use larger sets of multimedia instructions than the ones offered by the previous generations, since the variety of the operations and the performance requirement for multimedia applications increase [1]. Consequently, providing efficient multimedia hardware has become an important design task. In the past, modifying only integer data path were enough to implement most multimedia instructions; however today, the floating-point data path is modified as well, since many applications use floating-point operations [2-3]. Floating point division operation takes place in most of the 2D and 3D graphics applications. These applications perform vast amount of image transformation operations which require many multiplication and division operation. In general, multimedia computations do not need high accuracy i.e. single precision floating-point accuracy is adequate. On the other hand speeding up the computation is very important. The SIMD operations offer an alternative way for increasing the performance of the applications. Many general-purpose processor manufacturers implement multimedia extensions that execute SIMD type floating point operations. Processors from AMD have modified multiplier structure that can process floating point reciprocal and division operations [4]. In [5] 32/64 bit floating point division, reciprocal, square root and inverse square root unit is designed using cascade connected small multipliers to compute Newton and Raphson iteration. A High-Speed Double-Precision Computation of Reciprocal, Division, Square Root, and Inverse Square Root is design using Goldschmidt algorithm is given in [6]. A basic implementation of Newthon-Raphson reciprocal for double precision is presented in [7].

If an arithmetic unit has a fast multiplier, it can be configured to use in multiplicative iteration algorithms to speed up reciprocal and division operations. The division operation can be expressed as AB^{-1}, where A is dividend, B is divisor. The reciprocal of divisor is realized using Newton and Raphson iteration [8]. The multiplier can be also organized to operate on packed data type. In this study, we present a packed floating point reciprocal unit that can process single or double precision floating-point numbers based on the operation mode. The rest of this paper is organized as follows. In Section 2, floating point reciprocal operation is described. In Section 3, multiplicative reciprocal design and packed multiplier is described. In Section 4 the proposed design is described and in Section 5 the synthesis results are discussed.

2. Floating Point Reciprocal

The sign, exponent, and mantissa of an IEEE-754 floating number [9] x are represented as S_x, E_x, and M_x respectively. The reciprocal $(R = 1/x)$ of this number can be computed using

$$S_R = S_x \quad (1)$$
$$E_R = -E_x \quad (2)$$
$$M_R = 1/M_x \quad (3)$$

The computation of these equations can be conducted in parallel. For biased representation of exponent, the intermediate result of exponent is

$$E_{Rb} = 2B - E_{xb} + 1 \quad (4)$$

where B is bias value and small b denotes biased version of exponents.

The reciprocal of the mantissa is the most crucial part of the computation since it is the slowest step and it may introduce rounding errors. This computation is realized by following methods:

- *Digit recurrence*: simple and easy to implement, but the latency is long;
- *Functional iteration*: Algorithms such as Newton-Raphson[8] or Goldschmidt[10] iteration algorithms can be used. These algorithms are fast, scalable and have high precision.

- *Very high radix arithmetic*: This method is fast, but very complicated.

The mantissa needs to be normalized when the result of the reciprocal operation is less than 1. The normalization can be performed by a left shift and the exponent is decremented by one. Fig. 1 shows basic implementation of floating-point reciprocal unit.

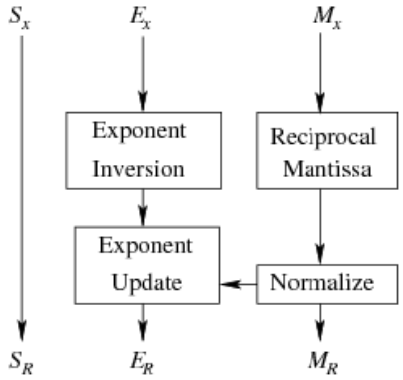

Fig. 1. Basic implementation of a floating point reciprocal unit.

2.1. Newton-Raphson Iteration

The Newton-Raphson algorithm is widely used in solving non-linear equations. The Newton-Raphson technique needs an initial value x_0, which is referred as *initial guess* for the root. The derivation is carried out by Taylor series. The function $f(x)$ can be written using Taylor series expansion in period $x - x_0$ as

$$f(x) = f(x) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 = 0 \quad (5)$$

where $f'(x_0)$ is the first derivative, $f''(x_0)$ is the second derivative of $f(x)$ with respect to x. Newton-Raphson method can quickly converge when the initial guess is close enough to the desired root. This means $x - x_0$ is small, and only the first few terms is enough to get accurate estimate of the root, x_0. The series can be shorten by throwing the second term and obtain the Newton-Raphson iteration formula as [3]:

$$x_1 = x_0 - f(x)/f'(x_0) \quad (6)$$

A more general form of equation can be written as:

$$x_{i+1} = x_i - f(x_i)/f'(x_i) \quad (7)$$

An initial look-up table is used to obtain an approximate value of the root. Each iteration doubles the accuracy of the result.

The derivation of algorithm using Newton-Raphson method for computing reciprocal for mantissa M as follows:

$$x = 1/M \quad (8)$$
$$f(x) = 1/M - x \quad (9)$$
$$f'(x) = 1/x^2 \quad (10)$$

When Equations (8), (9) and (10) are put into Equation (7), the iteration equation:

$$x_{i+1} = x_i(2 - M_{x_i}) \quad (11)$$

is obtained, which can be implemented in hardware. Two multiplication and one subtraction operations are required for computing Equation (11).

2.2. Derivation of Initial Values

The *n*-bit mantissa M is represented as

$$1.m_1m_2m_3 \ldots m_{n-1}\ (m_i \in \{0,1\}, i = 1 \ldots n) \quad (12)$$

When M is divided into two parts M_1 and M_2 as

$$M_1 = 1.m_1m_2m_3 \ldots m_m \quad \text{and} \quad M_2 = 0.m_{m+1}m_{m+2}m_{m+3} \ldots m_{n-1} \quad (13)$$

The first-order Taylor expansion of M^p of number M is between M_1 and $M_1 + 2^{-m}$ and is expressed as [4]:

$$(M_1 - 2^{-m-1})^{p-1} \cdot \left(M_1 + 2^{-m-1} + p \cdot (M_2 - 2^{-m-1})\right) \quad (14)$$

The equation can be expressed as

$$C \cdot M \quad (15)$$
where $C = (M_1 - 2^{-m-1})^{p-1}$ and
$M = M_1 + 2^{-m-1} + p \cdot (M_2 - 2^{-m-1})$

C can be read from a lookup-table which is addressed by M_1, without leading one. The look-up table contains the 2^m of C values of M for special values of p, where it is -2^0 for reciprocal of M. The size of the required ROM for the look-up table is about $2^m \cdot 2m$ bits [11].

The initial approximation of floating point number M^{-1} is computed by multiplication of term C with modified operand M. The modified form of M is obtained by only complementing M_2 bitwise. The last term can be ignorable.

3. Multiplicative Reciprocal Implementation

Basic multiplicative reciprocal unit is show in Fig. 2 [3]. The mantissa modify unit process the most significant part of the M and generates M' according to Equation (15). Also, the initial approximation, C of Equation (15) is obtained from the look-up table.

In the first cycle, the first multiplexer selects modified M value, the second multiplexer selects the output of the first multiplexer. The third multiplexer selects the output of the look-up table and the forth selects also the output of third multiplexer. In the second cycle, the multiplier generates a result in carry-save format. In the third cycle the carry-save vectors are summed by a fast carry-propagate adder. At the end of the third cycle the initial value, x_i is obtained. In the fourth cycle, the first and second multiplexers select the initial value generated in the previous cycle, the third and fourth multiplexer select M. In the fifth cycle, these values are multiplied and in the sixth cycle, the vectors generated by the multiplication are added. In the seventh cycle, the two's complement of the result is selected and the stored initial value in first iteration of the Newton-Raphson is selected. In the seventh and eighth cycle, these values are multiplied and vectors are summed for final result of iteration calculation. In the ninth cycle, the final result routed to normalization to suit IEEE mantissa format.

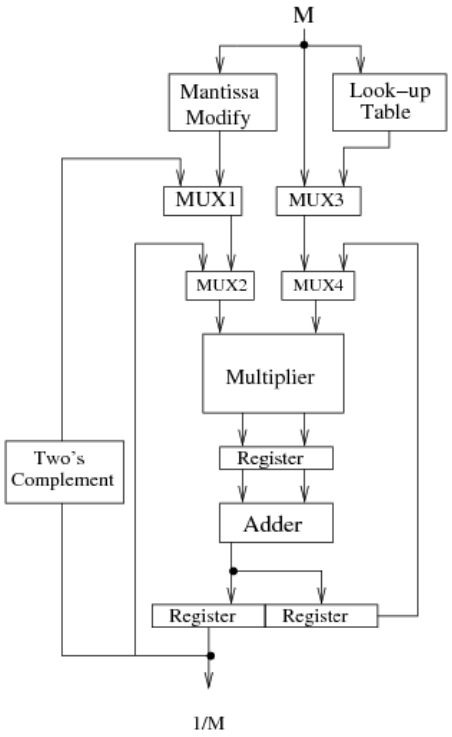

Fig. 2. A simple reciprocal unit that uses Newton-Raphson method.

Rounding is not studied here because this circuit can be coupled with a floating point multiplier for realizing floating point division operation. Rounding can be handled after multiplication by multiplication circuitry. This also minimizes the rounding error.

3. 1. Packed Multiplier

In this section, a packed multiplier design which performs the mantissa multiplications for Newton-Raphson method is described. Fig. 3a shows the alignment of one double precision floating-point mantissa and Fig. 3b shows the alignments of two single precision mantissas. Detailed description of the packed multipliers used in this work can be found in [12] and [13].

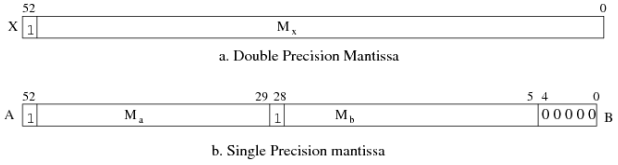

Fig. 3. The alignments of double precision and single precision mantissas.

Fig. 4 presents the adaption of the technique in [13] to implement the proposed design. In this figure, the matrices generated for two single precision mantissas multiplications are placed in the matrix generated for a double precision mantissa multiplication. All the bits are generated in double precision multiplication; the shaded areas labeled with Z1, Z2 and Z3 are not generated in single precision multiplication. The un-shaded areas are generated for single precision multiplication. The partial products within the regions Z1, Z2, Z3 are generated using equations:

$$\widehat{b_j} = s \cdot b_j \text{ and } p_{ij} = a_i \cdot \widehat{b_j} \tag{21}$$

The rest of the partial products are produced with

$$p_{ij} = a_i \cdot b_j \tag{22}$$

The signal s is used as control. When $s = '1'$ only bits with un-shaded regions are generated. When $s = '0'$, all bits are generated. The i and j are indices for appropriate partial product in the multiplication matrix [14].

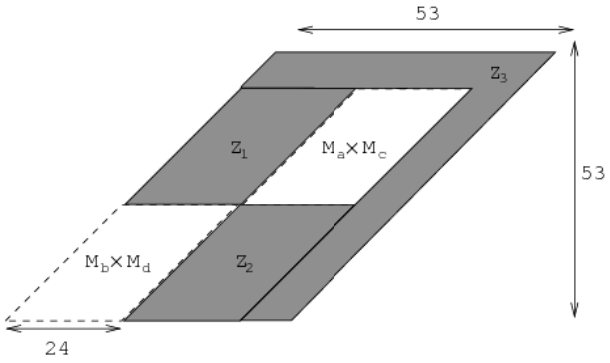

Fig. 4. Multiplication matrix for single and double precision mantissas.

4. Single/Double Precision Floating-point Reciprocal Unit Design for Packed Data

This section presents the proposed multi-precision floating-point reciprocal design. This unit uses the previous reciprocal computation methods and generates reciprocals in different precisions as follows: 1) In double precision mode the unit generates a double-precision reciprocal. 2) In first single-precision mode, the reciprocal unit generates a single-precision reciprocal and a copy of generated. 3) In the second single-precision mode, the reciprocal unit generates two different reciprocals in parallel.

The input format of modified design is shown in Fig. 5. Fig. 5a shows the input and output format in double precision mode. Fig. 5b shows the same input and output in single precision mode. An input, s signal selects operating mode.

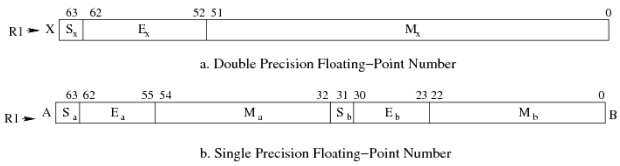

Fig. 5. The alignments of double and single precision floating point numbers.

The block diagram for the proposed design is shown in Fig 6. The explanations of the main units are as follows:

- *Exponent Unit* generates the exponents of one double precision or two single precision results. In single precision mode exponents are obtained with Equation

(23). In double precision mode Equation (23) is connected in cascade.

$$E_r = 11111111 - \widehat{E_x} \qquad (23)$$

- *Mantissa Modifier* generates modified mantissas based on the operation mode in order to prepare the inputs ready for the packed multiplier like in Fig. 3.
- *Lookup Table* contains look-up tables needed for initial approximation required for Newton-Raphson method. These are *C* values of Equation (15). They are pre-computed values generated by computer software such Maple, MatLab, etc.
- *Operand Modifier* modifies the operands required for initial value calculation. The value evaluated here is M' of Equation (15). It is evaluated by inverting the digits starting from 10^{th} digit for this design. The modification of operand(s) depends on the selected operation mode.
- *State Counter* drives the multiplexers to select correct inputs to the packed multiplier during the computation of Newton-Raphson iteration. The computation of Equation (11) requires three multiplications. Depending on selected operation mode the inputs of multiplexers are in double precision or packed single precision format as shown in Fig. 3. In the second cycle of circuit multiplexers are arranged for multiplication of look-up value(s) and modified mantissas as in the Equation (15). In the fourth cycle, multiplexers are arranged for multiplication of computed initial approximation value(s) and the input mantissa(s) in the Equation (11). And, in the sixth cycle, multiplexers are arranged for multiplication of stored initial value(s) and computed value(s) of inside parenthesis of the Equation (11) .
- *Packed Multiplier* is 53 by 53 multiplier slightly modified to handle two single and one double precision number as described. The input format of multiplier is shown in Fig 2. Multiplication output depends on selected operation mode.
- *Packed Product Generator* processes the output of *packed multiplier* and generates output used in next stages of iteration. The output of this unit is stored in a register. The format of output is truncated one 53-bit double mantissa or two 24-bit single mantissas depending on selected mode. The mantissas arranged as in Fig. 3.
- *I.A.Store* unit stores the Initial Approximation value(s) computed in the second cycle of circuit. These are x_i values in Equation (15), which are needed in fourth cycle.
- *Inverter* inverts the stored multiplication result(s) to compute the expression in the parenthesis of equation (11). The inversion is done depending of selected operation mode.
- *Single Normalizer(s)* normalize the result in single-precision mode. *Double Normalizer* normalizes the result in double precision mode. The normalization is one left shift if required.
- *Exponent Updater* updates the exponents depending on the normalization results. Two decrementers are separately used to update 8-bit exponents in single mode or in double mode these decrementers are connected cascade to update 11-bit exponent.

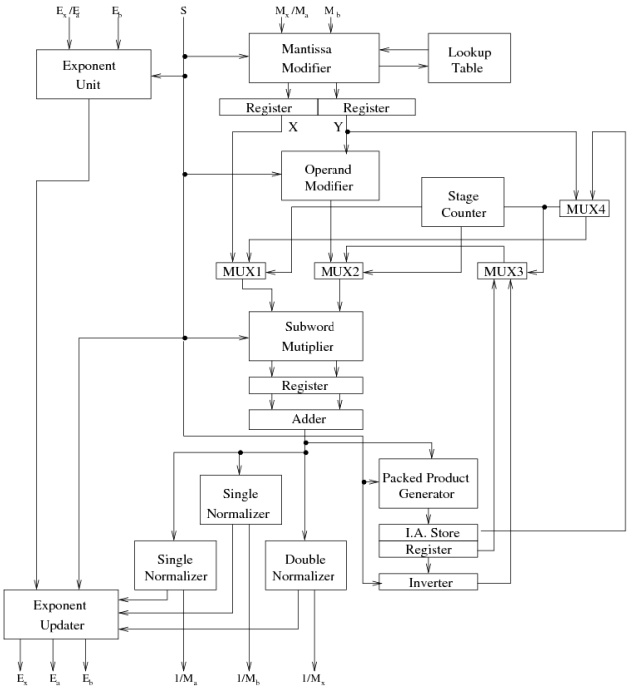

Fig. 6. The proposed single/double precision reciprocal unit

5. Synthesis Results

In this section we present the synthesis results for the proposed single/double precision floating point reciprocal unit. We use the design in [7] as reference standard double precision floating-point reciprocal unit with some estimation. The estimations include design of unsigned radix-2 multiplier, carry-propagate-adders and a controlling logic for the multiplexers. Both circuits are modeled using structural VHDL code. Synthesis of the circuit is done using TSMC 0.18 micron ASIC library and Leonardo Spectrum program. Both circuits are optimized for delay. The clock delays and area estimates (in terms of number of gates) for both designs are given in Table 1. The values in Table 1 are in nanoseconds for time and in number of gate for area.

Table 1. The comparison of the standard double precision and proposed floating-point reciprocal designs.

Design	# of Gates	Latency
Reference Double Precision	31979	3.86 Ns
Single /Double Precision	33997	3.94 Ns

The single/double precision reciprocal unit has approximately 6% more area and has about 3% more critical delay. The most critical delay occurs in the multiplier. Because of the multiplier we used is slightly modified a negligible difference occurs in delay. The additional circuits cause also negligible grows in design. The floating-point reciprocal units used in modern processors are usually pipelined designs. The design performs two single-precision reciprocal with about same latency which is dissolved in pipeline stages.

6. Conclusions

This paper presented a reciprocal unit for multimedia applications. The design operates on SIMD type data input. The accuracy of the results are 20 bits for each iteration. Compared to the previous reference designs less than 1% area increase and delay increase is reported based on synthesis results. However the functionality of the reciprocal unit is improved to support three operation modes. The mode that generates two different reciprocals simultaneously is expected to double the performance of single precision division operations. The proposed unit can be expanded to support reciprocal-square-root operation with additional circuit and modifications

7. References

[1] V. Lappalainen, T. D., Hämäläinen, P., Liuha, "Overview of Research Efforts on Media ISA Extensions and Their Usage in Video Coding", *IEEE Transactions on Circuits And Systems For Video Technology*, Vol: 12, No: 8, 2002.

[2] R.B., Lee, "Multimedia extensions for general purpose processors", Signal Processing Systems, 1997", *SIPS 97 - Design and Implementation., IEEE Workshop*, 1997, pp 9-23.

[3] M. D. Ercegovac, T. Lang, "Digital Arithmetic", Morgan Kaufmann, Los Atlas, CA, 2004.

[4] S.F., Oberman, N., Juffa, F., Weber, "Method and Apparatus For Calculating Reciprocals and Reciprocal Square Roots", Advanced Micro Devices Inc., Patent Number 6.115.773.

[5] C., Shuang, W. Dong, Z. Tie, H. Chao, "Design and Implementation of a 64/32-bit Floating-point Division, Reciprocal, Square root, and Inverse Square root Unit," *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on* , pp.1976-1979, 2006.

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

[7] U. Kucukkabak, A. Akkas, "Design and implementation of reciprocal unit using table look-up and Newton-Raphson iteration", *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, 2004, pp. *249-253.*

[8] F., Domenico, "A Division Method Using a Parallel Multiplier", *Electronic Computers, IEEE Transactions on* , 1967,vol. EC-16, no.2, pp 224-226.

[9] *IEEE Standard for binary floating-point arithmetic*, ANSI-IEEE Standard 754-1985, 1985.

[10] P., Markstein, "Software division and square root using Goldschmidt's algorithms", *Journal of Universität Trier*, pp. *146*, 2004.

[11] N. Takagi, "Generating a power of an operand by a table look-up and a multiplication", in *Proceedings of 13th Sym. on Computer Arithmetic*, Asilomar, CA, 1997, pp. *126-131.*

[12] M. Gok, S. Krithivasan, M. J. Schulte, "Designs for Subword-Parallel Multiplications and Dot Product Operations", in *Workshop on Application Specific Processors*, 2004, pp. *27-31.*

[13] S. Krithivasan, M. J. Schulte, "Multiplier Architectures for Media Processing", in *Conference Records of 37th Asilomar Conference on Signals, Systems and Computers*, Asilomar, CA, 2003, vol. 2, pp. *2193-2197.*

[14] M. Gok, M. M. Ozbilen, "A Single/Double Precision Floating-Point Multiplier Design for Multimedia Applications", *Istanbul University Journal Of Electrical & Electronics Engineering*, 2009, vol. 9, pp. *827-831.*

[15] S., Oberman, G., Favor, F.,Weber, "AMD 3DNow! Technology: architecture and implementations", *IEEE Micro*, vol.19, no.2, pp.37-48, 1999.