Floating-Point Division and Square Root using a Taylor-Series Expansion Algorithm

Taek-Jun Kwon, Jeff Sondeen, Jeff Draper
University of Southern California / Information Sciences Institute
Marina del Rey, U.S.A.
{tjkwon, sondeen, draper}@ISI.EDU

***Abstract*—Hardware support for floating-point (FP) arithmetic is a mandatory feature of modern microprocessor design. Although division and square root are relatively infrequent operations in traditional general-purpose applications, they are indispensable and becoming increasingly important in many modern applications. Therefore, overall performance can be greatly affected by the algorithms and the implementations used for designing FP-div and FP-sqrt units. In this paper, a fused floating-point multiply/divide/square root unit based on Taylor-series expansion algorithm is proposed. We extended an existing multiply/divide fused unit to incorporate the square root function with little area and latency overhead since Taylor's theorem enables us to compute approximations for many well-known functions with very similar forms. The proposed arithmetic unit exhibits a reasonably good area-performance balance.**

I. INTRODUCTION

Due to constant advances in VLSI technology and the prevalence of business, technical, and recreational applications that use floating-point operations, floating-point computational logic has long been an essential component of high-performance computer systems as well as embedded systems and mobile applications. The performance of many modern applications which have a high frequency of floating-point operations is often limited by the speed of the floating-point hardware. Therefore, a high-performance FPU is an essential component of these systems. Over the past years, leading architectures have incorporated several generations of FPUs. However, while addition and multiplication implementations have become increasingly efficient, support for division and other elementary functions such as square root has remained uneven [1].

Division has long been considered a minor, bothersome member of the floating-point family. Hardware designers frequently perceive divisions as infrequent, low-priority operations, and they allocate design effort and chip resources accordingly, as addition and multiplication require from two to five machine cycles, while division latencies range from nine to sixty as shown in Table I. The variation is even greater for square root.

TABLE I. MICROPROCESSOR FPU COMPARISON

Design	Cycle time (ns)	Latency/Throughput (cycles/cycles)			
		$a \pm b$	$a \times b$	$a \div b$	$\sqrt{a}$
Alpha 21164	2.0	4/1	4/1	22-60	N/A
HP PA 8000	5.0	3/1	3/1	31/31	31/31
Pentium	5.0	3/1	5/2	17/17	28/28
MIPS R10000	3.64	2/1	2/1	18/18	32/32
PowerPC 604	5.56	3/1	3/1	31/31	N/A
UltraSparc	4	3/1	3/1	22/22	22/22
Pentium 4	0.67	5/1	7/2	23/23	23/23
AMD-K7	0.83	4/1	4/1	16/13	19/16

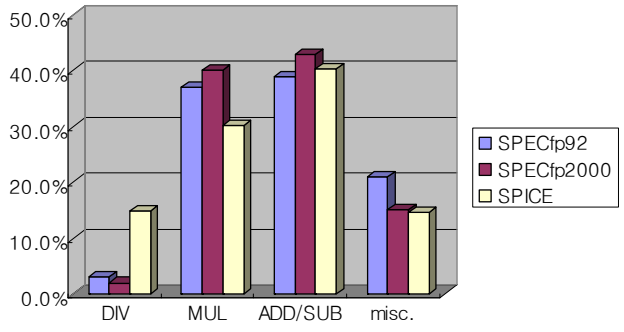

Figure 1. Average frequncy of FP-instructions

Although division and square root are relatively infrequent operations in traditional general-purpose applications, they are indispensable and becoming increasingly important, particularly in many modern applications such as CAD tools and 3D graphics rendering. Furthermore, due to the latency gap between addition/multiplication and division/square root, the latter operations increasingly become performance bottlenecks.

This research was supported by DARPA

Therefore, poor implementations of floating-point division and square root can result in severe performance degradation.

The remainder of this paper is organized as follows. Section 2 presents a brief description of existing algorithms used for floating-point division and square root operations followed by a detailed description of the proposed floating-point multiply/divide/square root fused unit in Section 3. Section 4 presents comparison results followed by a brief summary and conclusion in Section 5.

II. Division / Square Root Algorithms

The long latency of division and square root operations is mainly due to the algorithms used for these operations. Typically, First-order Newton-Raphson algorithms and binomial expansion algorithms (Goldschmidt's) are commonly used for division and square root implementations in high-performance systems. These algorithms exhibit better performance than subtractive algorithms, and an existing floating-point multiplier can be shared among iterative multiply operations in the algorithms to reduce area. Even so, it still results in a long latency to compute one operation, and the subsequent operation cannot be started until the previous operation finishes since the multiplier used in these algorithms is occupied by the several multiply operations of the algorithms. Liddicoat and Flynn [2] proposed a multiplicative division algorithm based on Taylor-series expansion as shown in Fig. 2. This algorithm achieves fast computation by using parallel powering units such as squaring and cubing units, which compute the higher-order terms significantly faster than traditional multipliers with a relatively small hardware overhead.

$$q = \frac{a}{b} \approx aX_0\left\{1 + (1 - bX_0) + (1 - bX_0)^2 + (1 - bX_0)^3\right\} \quad (1)$$

$$X_0 \approx \frac{1}{b} \quad (2)$$

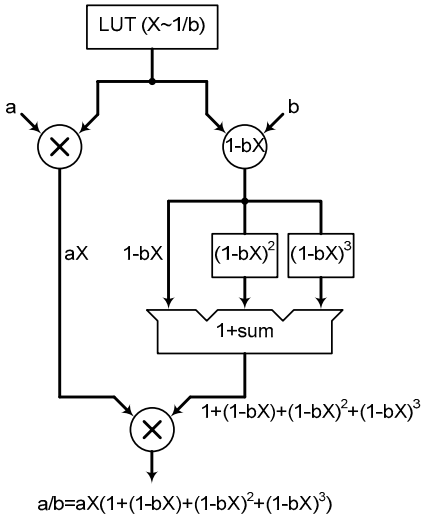

Figure 2. Division algorithm by Liddicoat and Flynn

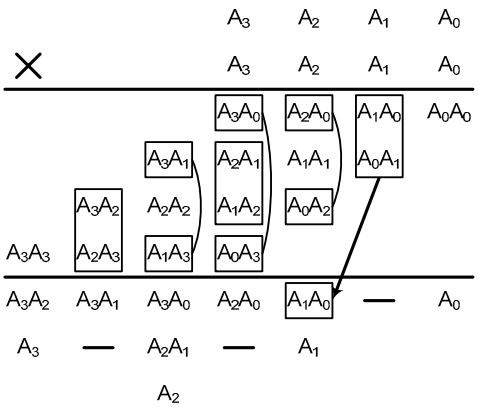

Figure 3. PPA reduction for squaring unit

An example of squaring a 4-bit number is shown in Fig. 3 [3]. Partial product terms can be combined using the equivalence $A_iA_j + A_jA_i = 2A_iA_j$ and $2A_iA_j$ is represented by placing A_iA_j one column to the left. The square of operand A can be computed with the reduced partial product array shown in the lower portion of Fig. 3. A cubing unit can be designed in a similar fashion. Further area and latency optimization can be achieved by truncating the least significant bits of these partial product terms while satisfying the required precision of floating-point operation. Specifically, 69.6% of the partial product terms from the squaring unit and 97.6% of the partial product terms from the cubing unit were truncated. As a result, the proposed squaring unit achieves an area reduction of 66.8%, and the proposed cubing unit achieves an area reduction of 89.9%, as compared with a traditional multiplier.

III. Proposed FP-Mul/Div/Sqrt Fused Unit

There are three major multiply operations in the Taylor-series expansion algorithm with powering units to produce a quotient with 0.5 ulp (unit in the last place) error as shown in Fig. 2. One additional multiply operation is required for exact rounding to generate IEEE-754 floating-point standard compliant results. Even though the Taylor-series expansion algorithm with powering units exhibits the highest performance among multiplicative algorithms, it consumes a larger area because the architecture consists of four multipliers, which is not suitable for area-critical applications. In earlier work, we presented a fused floating-point multiply-divide unit based on Taylor-series expansion with powering units where all multiply operations are executed by one multiplier to maximize the area efficiency, while achieving high performance by using a pipelined architecture [5][6]. By sharing the 2-stage pipelined multiplier among the multiply operations in the algorithm, the latency becomes longer (12 clock cycles) than the direct implementation of the original algorithm (8 clock cycles). However, through careful pipeline scheduling, we were able to achieve a moderately high throughput (one completion every 5 clock cycles) for consecutive divide instructions and 1.6 times smaller area. The area difference between the proposed

arithmetic unit and the direct implementation of the original algorithm is mainly because of the area occupied by multipliers.

Approximations for many well-known functions can be computed by using Taylor's theorem. Therefore, we can extend the existing fused Mul/Div unit to other elementary functions, such as square root, which is a common operation required in many modern multimedia applications such as graphics engines. The Taylor-series approximations of such elementary functions have very similar forms, and the only difference is the coefficients of each term in the polynomial as shown in equations (1) and (3). The polynomial coefficients for Taylor-series approximations are typically very low in complexity and often the coefficient multiplications can be computed using multiplexers since a simple 1-bit right shift will provide a 1/2 multiple. Therefore, extending the existing fused unit to incorporate a square root function can be achieved with little area overhead as the additional hardware components will be two lookup tables for the initial seed value of square root (odd and even operands), which is also common for other algorithms, and three multiplexers. The latency and throughput will remain the same since adding muxes to the existing datapath will incur negligible delay. A block diagram of the proposed Mul/Div/Sqrt fused arithmetic unit is shown in Fig.4 and the steps required for a division and a square root operation are described in Table 2.

$$\sqrt{b} \approx Y_0\left\{1-\frac{1}{2}\left(1-\frac{b}{Y_0^{\,2}}\right)-\frac{1}{8}\left(1-\frac{b}{Y_0^{\,2}}\right)^2-\frac{1}{16}\left(1-\frac{b}{Y_0^{\,2}}\right)^3\right\} \quad (3)$$

$$Y_0 \approx \sqrt{b} \quad (4)$$

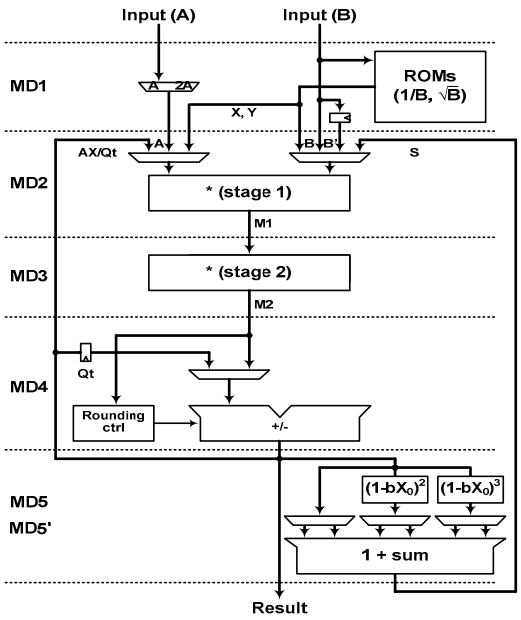

Figure 4. Block diagram of MUL/DIV/SQRT unit

TABLE II. STEPS FOR DIVISION AND SQUARE ROOT OPERATIONS

Cycle	Division	Square Root	Stage
1	$X = ROM_{DIV}(b)$	$X = ROM_{DIV}(b)$, $Y = ROM_{SQRT}(b)$	MD1
2	M1 = b × X (stage1)	M1 = b × X (stage1)	MD2
3	M2 = b × X (stage2), M1 = a × X (stage1)	M2 = b × X (stage2)	MD3 / MD2
4	SC = 1– M2, M2 = a × X (stage2)	SC = 1– M2	MD4 / MD3
5	$S = 1+SC+SC^2+SC^3$, AX = M2	$S = 1 – (1/2)SC – (1/8)(SC)^2 – (1/16)(SC)^3$	MD5' / MD4
6	M1 = AX × S (stage1)	M1 = Y × S (stage1)	MD2
7	M2 = AX × S (stage2)	M2 = AX × S (stage2)	MD3
8	Qt = truncate(M2)+1	Qt = truncate(M2)+1	MD4
9	M1 = b × Qt (stage1)	M1 = Qt × Qt (stage1)	MD2
10	M2 = b × Qt (stage2)	M2 = Qt × Qt (stage2)	MD3
11	R = round(Qt)	R = round(Qt)	MD4
12	Quotient = format(R)	SQRT = format(R)	MD5

The proposed fused arithmetic unit has a 5-stage pipelined architecture and the latency of fp-multiply operation is 5 cycles, which can be fully pipelined. The pipeline diagram of one division operation and an example of mixed fp-mul, fp-div and fp-sqrt operations are shown in Fig. 5 and Fig.6 respectively.

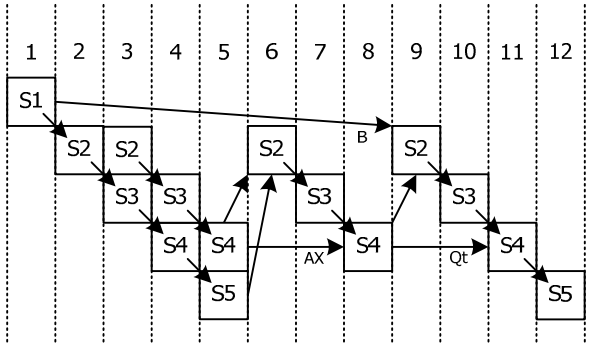

Figure 5. Pipeline diagram of one fp-div

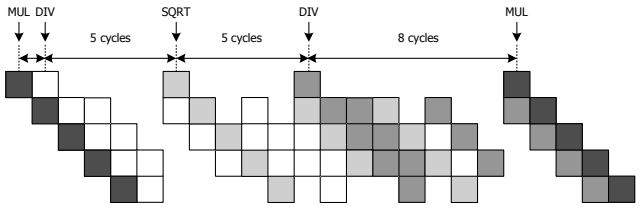

Figure 6. Pipeline diagram of mixed fp-mul, fp-div and fp-sqrt

IV. COMPARISON RESULTS

The comparison of other commonly used multiplicative algorithms and the proposed arithmetic unit for single-precision floating-point division is summarized in the following table, where an 8-bit initial seed value and a 2-stage pipelined multiplier are used for a realistic implementation. Division algorithms are commonly used for square root implementations. However, the latency of square root operation is usually longer than that of division operations.

TABLE III. FP-DIV COMPARISON OF MULTIPLICATIVE ALGORITHMS

Algorithm	Latency / Throughput	Approximate Area
1st-order Newton Raphson	18 / 14	Table + 1 FM + 1 AddSub
Goldschmidt's	18 / 14	Table + 1 FM + 1 AddSub
Taylor-series expansion with Powering units	8 / 1	≈ Table + 3 FM
Proposed Arithmetic Unit	12 / 5	≈ Table + 1.5 FM + 1 AddSub

TABLE IV. FP-DIV AND FP-SQRT COMPARISON OF LEADING ARCHITECTURES (SINGLE-PRECISION)

	Latency / Throughput		Algorithm	Speed	Fabrication Technology
	Div	Sqrt			
Intel Pentium 4	23/23	23/23	Radix-2 SRT	1.5 GHz	0.13 μm
AMD K7 core	16/13	19/16	Goldschmit's	1.2 GHz	0.13 μm
Proposed Arithmetic Unit	12/5	12/5	Taylor-series	333 MHz	0.18 μm

A brief survey of several floating-point dividers in leading microprocessors is summarized in Table 4 [7][8]. As shown in the table, the division algorithm used in each of these architectures is one of the basic algorithms, which requires many cycles to compute the result. These characteristics may be acceptable in some applications like typical desktop applications. However, high performance is crucial for many applications such as scientific computation, CAD tools and 3D graphics rendering which have a higher frequency of floating-point division and square root operations. It is very difficult to compare the designs where different design methodologies were used, such as full custom and standard cell approach. Generally, full custom requires 10x the design cost for 2x more speed and 1/2 the area as compared with standard cell methodology. As described earlier, the latency and the throughput of division and square root operations are worse than the direct implementation by sharing a multiplier through a non-linear pipeline scheme. However, considering the logic style and the fabrication technology, the performance of the proposed divider is better than the floating-point dividers in leading architectures, especially in terms of throughput. In terms of area, the implementations based on SRT algorithm may be smaller than multiplication-based implementations when a small radix is used. However, these algorithms require more cycles to compute the quotient with the required precision. As shown in Table 3, the area of the binomial expansion algorithm (Goldschmit's) is almost the same as the direct implementation of Liddicoat's algorithm. Therefore, the proposed arithmetic unit also exhibits an area-efficiency as compared to an implementation of Goldschmit's algorithm.

V. CONCLUSION

This paper presents a design plan for a fused floating-point multiply/divide/square root unit based on the Taylor-series expansion algorithm. The square root function can be easily incorporated into an existing multiply/divide fused unit with very minimal area and latency overhead due to the similarity of Taylor-series approximations. The resulting arithmetic unit exhibits an area efficiency as compared to implementations of other commonly used division/square root algorithms as well as high throughput and moderate latency as compared with other FPU implementations of leading architectures. We are currently in the process of implementing the fused floating-point multiply/divide/square root unit, with implementation details to follow in a future paper. However, we project that adding square root to the existing unit will increase the area by a mere 4.9%.

REFERENCES

[1] P. Soderquist, M. Leeser, "Division and Square Root: choosing the right implementation", IEEE Micro, Vol. 17, No. 4, pp.56-66, July-Aug. 1997

[2] A. Liddicoat and M.J. Flynn, "High-Performance Floating-Point Divide", *Euromicro Symposium on Digital System Design*, Sep. 2001

[3] T. C. Chen, "A Binary Multiplication Scheme Based on Squaring", IEEE Transactions on Computers, Vol. C-20, No. 6, pp. 678-680, June 1971

[4] H. M. Darley et al., "Floating Point / Integer Processor with Divide and Square Root Functions", U.S. Patent No. 4,878,190, 1989

[5] Taek-Jun Kwon, et al, "0.18μm Implementation of a Floating-Point Unit for a Processing-In-Memory System", in Proc. of the IEEE ISCAS, vol. 2, p. II-453~6, 2004.

[6] Taek-Jun Kwon, et al, "Design Trade-offs in Floating-Point Unit Implementation for Embedded and Processing-In-Memory Systems", in Proc. of the IEEE ISCAS, vol. 4, p. 3331~3334, 2005.

[7] G. Hinton, et al, "A 0.18-μm CMOS IA-32 Processor With a 4-GHz Integer Execution Unit", IEEE Journal of Solid-State Circuits, Vol. 36, No. 11, pp. 1617-1627, Nov. 2001

[8] S. F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7™ Microprocessor", in the Proceedings of IEEE Symposium on Computer Arithmetic, pp. 106-115, Apr. 1999