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

Taek-Jun Kwon, Jeff Sondeen, Jeff Draper
University of Southern California / Information Sciences Institute
Marina del Rey, CA 90292, USA
{tjkwon, sondeen, draper}@ISI.EDU

***Abstract*—Hardware support for floating-point (FP) arithmetic is an essential 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. In this paper, a fused floating-point multiply/divide/square root unit based on Taylor-series expansion algorithm is presented. The implementation results of the proposed fused unit based on standard cell methodology in IBM 90nm technology exhibits that the incorporation of square root function to an existing multiply/divide unit requires only a modest 23% area increase and the same low latency for divide and square root operation can be achieved (12 cycles). The proposed arithmetic unit also 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 a mandatory 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 thirteen to sixty as shown in Table I. The variation is even greater for square root.

TABLE I. MICROPROCESSOR FPU COMPARISON

Design	Clock Frequency	Latency/Throughput (cycles/cycles)			
		$a \pm b$	$a \times b$	$a \div b$	$\sqrt{a}$
Alpha 21164	500 MHz	4/1	4/1	22-60	N/A
HP PA 8000	200 MHz	3/1	3/1	31/31	31/31
Pentium	200 MHz	3/1	5/2	17/17	28/28
MIPS R10000	275 MHz	2/1	2/1	18/18	32/32
PowerPC 604	180 MHz	3/1	3/1	31/31	N/A
UltraSparc	250 MHz	3/1	3/1	22/22	22/22
Intel Penryn	3 GHz	3/1	4/1	13/12	13/12
AMD K10	2.3 GHz	4/1	4/1	16/13	19/16

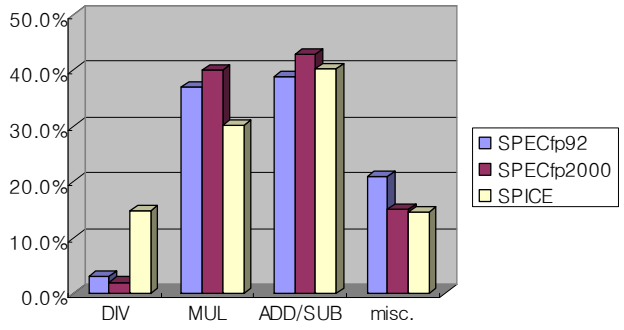

Figure 1. Average frequency of FP-instructions

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 as shown in Fig.1. Furthermore, due to the latency gap between addition/multiplication and division/square root, the latter

operations increasingly become performance bottlenecks. 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 implementation results and comparison 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, 1st-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 (3rd-order Newton method) 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.

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

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

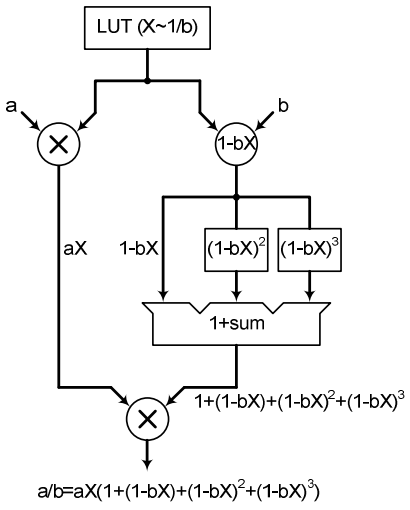

Figure 2. Division algorithm by Liddicoat and Flynn

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 a Taylor-series expansion algorithm 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.

III. Incorporating SQRT function Into The Existing FP-Mul/Div Fused Unit

Taylor's theorem enables us to compute approximations for many well-known functions such as square root, which is a common operation required in many modern multimedia applications. The Taylor-series approximations of such elementary functions have very similar forms, and the major difference is the coefficients of each term in the polynomial as shown in equations (1) and (3).

$$\sqrt{b} \approx bY_0\left\{1 + \frac{1}{2}\left(1 - bY_0^2\right) + \frac{3}{8}\left(1 - bY_0^2\right)^2 + \frac{5}{16}\left(1 - bY_0^2\right)^3\right\} \quad (3)$$

$$Y_0 \approx \frac{1}{\sqrt{b}} \quad (4)$$

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 provides a 1/2 multiple. Therefore, we can easily extend the existing Mul/Div fused unit to incorporate a square root function, and it can be achieved with a modest area and minimal latency overhead. The block diagram of the proposed Mul/Div/Sqrt fused arithmetic unit is shown in Fig.3 and the steps required for a division and a square root operation are described in Table II. The pipeline diagram of a division operation or a square root operation is also shown in Fig. 4 followed by a detailed description of the design consideration on each major datapath component.

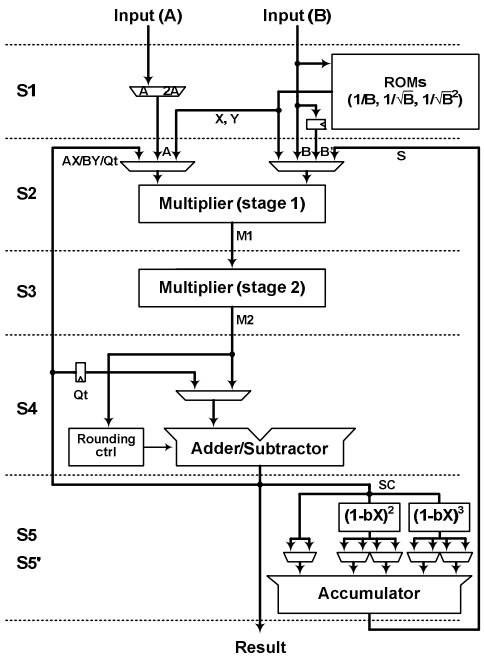

Figure 3. Block diagram of MUL/DIV/SQRT unit

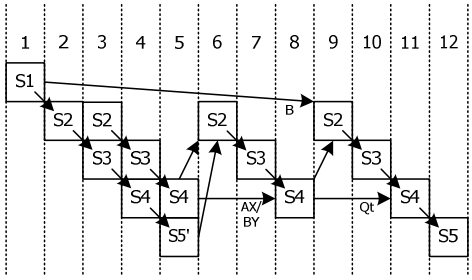

Figure 4. Pipeline diagram of fp-div and fp-sqrt

A. Look-up Tables

When a multiplicative algorithm is used to implement the square root function, generally two look-up tables for the initial approximation value Y_0 are used based on the exponent value of the input operand: odd and even. To generate IEEE-754 standard compliant results using Equation (3), an 8-bit seed is used when the exponent is even and a 9-bit seed is used when the exponent is odd, which is exhaustively verified with our simulator. In addition, look-up tables for Y_0^2 are also used to maintain and re-use the existing pipeline control logic for division operations in the Mul/Div unit as much as possible to equalize the latency of division and square root operations. As a result, five look-up tables are used in the proposed Mul/Div/Sqrt unit: X_0 for division and Y_{0_even}, Y_{0_odd}, $(Y_{0\_even})^2$, $(Y_{0\_odd})^2$ for square root. However, since all entries in several tables start with the same constants (e.g. X_0 = 0.1xxxxxxx), actual table sizes can be reduced by not storing those constants. Therefore, the sizes of each table are as follows: X_0 = 128x7b, Y_{0_even} = 128x7b, Y_{0_odd} = 256x7b, $(Y_{0\_even})^2$ =128x16b, $(Y_{0\_odd})^2$ = 256x16b.

TABLE II. STEPS FOR DIVISION AND SQUARE ROOT OPERATIONS

Cycle	Division	Square Root	Stage
1	$X = ROM_X(b)$	$Y = ROM_Y(b)$, $X = ROM_{Y^2}(b)$	S1
2	M1 = b×X (stage1)	M1 = b×X (stage1)	S2
3	M2 = b×X (stage2), M1 = a×X (stage1)	M2 = b×X (stage2) M1 = b×Y (stage1)	S3 / S2
4	SC = 1– M2, M2 = a×X (stage2)	SC = 1– M2 M2 = b×Y (stage2)	S4 / S3
5	$S = 1+SC+SC^2+SC^3$, AX = M2	$S=1+(1/2)SC+(3/8)(SC)^2+$ $(5/16)(SC)^3$, BY = M2	S5' / S4
6	M1 = AX×S (stage1)	M1 = BY×S (stage1)	S2
7	M2 = AX×S (stage2)	M2 = BY×S (stage2)	S3
8	Qt = truncate(M2)+1	Qt = truncate(M2)+1	S4
9	M1 = b×Qt (stage1)	M1 = Qt×Qt (stage1)	S2
10	M2 = b×Qt (stage2)	M2 = Qt×Qt (stage2)	S3
11	R = round(Qt)	R = round(Qt)	S4
12	Quotient = format(R)	SQRT = format(R)	S5

B. Parallel Powering Units

A significant portion of the least significant columns in the partial product array of the powering units in the existing Mul/Div unit were truncated to reduce the hardware required for implementation and the latency of the powering units while satisfying adequate precision of floating-point division operations. To incorporate a square root function, the existing powering units must be verified to check whether they also provide enough precision for square root operations; otherwise additional PPA columns must be added to improve the worst-case error. However, in fact, the quotient computation requires more precision from the powering units as the coefficients of the 2nd and the 3rd order terms in Equation (1) is larger than those of Equation (3), which is also verified with our simulator. Therefore, we can re-use the existing powering units without adding PPA columns. Additional muxes and a modified accumulator which adds all terms in the Taylor-series polynomial are used to compute the coefficient multiplication in Equation (3). In the case of the 2nd order term, since 3/8=2/8+1/8, 2-bit shifted and 3-bit shifted values of $(1-bY_0^2)^2$ are accumulated, and in the case of the 3rd order term, since 5/16=4/16+1/16, 2-bit shifted and 4-bit shifted values of $(1-bY_0^2)^3$ are accumulated. Even though muxes are added in the datapath and the accumulator is widened from 3 inputs to 5 inputs, the latency and throughput of a square root operation will remain the same as a division operation since the described design changes incur negligible delay.

C. Multiplier, Adder/Subtractor, etc.

The 2-stage pipelined multiplier in the existing Mul/Div unit is shared among several multiplications in the Taylor-series expansion algorithm and floating-point multiply operations. Since the multiplication which generates the widest result is step 6 and 7 in Table II, and the width of the

results in these steps are the same for both division and square root, the existing multiplier can be re-used without any modification. In case of the adder/subtractor, the width is determined by step 4 where $(1-bX_0)$ or $(1-bY_0^2)$ is computed. To accommodate a square root function, the width of the existing adder/subtractor is increased since the width of Y_0^2 (18-bit) is larger than that of X_0 (8-bit). However, this also does not affect the overall performance of the proposed arithmetic unit since the adder/subtractor is not the critical path. The existing pipeline control logic for division operations also can be re-used for square root operations since both operations have the same latency and throughput.

IV. Implementation Results and Comparison

The proposed Mul/Div/Sqrt fused unit has been described in Verilog and was synthesized to IBM 90nm CMOS technology under the timing constraint of a 500MHz clock frequency using Synopsys Design Compiler. A brief summary of area comparisons on the major datapath components in Mul/Div unit and Mul/Div/Sqrt unit is presented in Table III.

TABLE III. Summary of Area Comparison Between Mul/Div and Mul/Div/Sqrt

Datapath Component	Mul/Div	Mul/Div/Sqrt	Increase
Look-up Tables	793.1	4989.3	529.1 %
Parallel Powering Units	6098.5	10042.0	64.7%
2-Stage Pipelined Multiplier	15063.9	15078.3	0.1%
Adder / Subtractor	3853.3	3960.8	2.8%
Control Logic, Register, etc.	11914.0	12367.3	3.8%
Total Area (μm)	37722.8	46437.7	23.1%

As shown in the table, the major area overhead is due to the additional look-up tables for the square root function in the Mul/Div/Sqrt unit. Since look-up tables are synthesized using standard cells, rather than implemented with ROM, the actual area increase is less than the value (985.7%) expected based on the number of bits in the look-up table. The additional muxes and the widened accumulator in the parallel powering unit block also contribute to the area increase. As a result, the proposed arithmetic unit can be extended from the existing Mul/Div unit to incorporate a square root function with a modest 23% area increase and minimal performance overhead. However, when considering incorporating a square root function to an entire FPU [5], the area increase is a mere 14%.

TABLE IV. FP-Div and FP-Sqrt Comparison of Leading Architectures (Single-Precision)

	Latency / Throughput		Algorithm	Speed	Fabrication Technology
	Div	Sqrt			
Intel Penryn	13/12	13/12	Radix-16 SRT	3 GHz	45 nm
AMD K10	16/13	19/16	Goldschmit's	2.3 GHz	65 nm
Proposed Arithmetic Unit	12/5	12/5	Taylor-series	500 MHz	90 nm

A brief survey of several floating-point division and square root implementations in leading microprocessors is summarized in Table IV [7][8]. It is very difficult to compare the designs where different design methodologies were used, such as full custom and standard cell approach. 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.

V. Conclusion

This paper presents the design and implementation of a fused floating-point multiply/divide/square root unit based on a Taylor-series expansion algorithm. The design considerations and impact of incorporating a square root function into an existing fused floating-point multiply/divide unit are also presented. The square root function can be easily incorporated into an existing multiply/divide fused unit with a modest area and minimal latency overhead due to the similarity of Taylor-series approximations. The resulting arithmetic unit also exhibits high throughput and moderate latency as compared with other FPU implementations of leading architectures.

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] *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, Intel Corporation, Nov.2007

[8] *Software Optimization Guide for AMD Family 10h Processors*, Advanced Micro Devices, Inc., Jan 2008