Energy and Delay Improvement via Decimal Floating Point Units

Hossam A. H. Fahmy
Electronics and Communications Department
Cairo University
Egypt
Email: hfahmy@stanfordalumni.org

Ramy Raafat, Amira M. Abdel-Majeed, Rodina Samy,
Tarek ElDeeb, Yasmin Farouk
SilMinds
Smart Village, B115, 12577
Giza, Egypt
Email: ramy.raafat@silminds.com

Abstract

Interest in decimal arithmetic increased considerably in recent years. This paper presents new designs for decimal floating point (DFP) addition, multiplication, fused multiply-add, division, and square root. It stresses the importance of energy savings achieved by hardware implementations of the IEEE standard for decimal floating point. To the best of the authors knowledge, this is the first work to discuss energy savings in DFP and the first to present a hardware implementation of a fused multiply-add. Our Newton-Raphson based divider is over three times faster than the similar design previously reported.

1. Why decimal hardware?

Ten is the natural number base or radix for humans resulting in a decimal number system while a binary system is natural to computers. In the early days of computers, to suite the data provided by the human users many machines included circuits to perform operations on decimal numbers only. Decimal numbers were used even for the memory addressing and partitioning. In his seminal paper in 1959, Buchholz [1] presented many persuasive arguments for using binary representations instead of decimal for such machine related issues as memory addressing.

Buchholz concludes that "a combination of binary and decimal arithmetic in a single computer provides a high-performance tool for many diverse applications. It may be noted that the conclusion might not be the same for computers with a restricted range of functions or with performance goals limited in the interest of economy; the difference between binary and decimal operation might well be considered too small to justify incorporating both. The conclusion does appear valid for high-performance computers regardless of whether they are aimed primarily at scientific computing, business data processing, or real-time control."

Due to the limited capacities of the first integrated circuits in the 1960s and later years, most machines adopted the use of dedicated circuits for binary numbers and dropped decimal numbers. With the much higher capabilities of current processors and the large increase in financial and human oriented applications over the Internet, decimal is regaining its due place. The largest change in the recent revision of the IEEE standard for floating point arithmetic [2] is the introduction of the decimal floating point formats and the associated operations. Whether in software or hardware, a standard to represent the decimal data and determine the manner of handling exceptional cases in operations is important.

Simple decimal fractions such as $1/10$ which might represent a tax amount or a sales discount yield an infinitely recurring number if converted to a binary representation. Hence, a binary number system with a finite number of bits cannot accurately represent such fractions. When an approximated representation is used in a series of computations, the final result may deviate from the correct result expected by a human and required by the law [3], [4]. One study [5] shows that in a large billing application such an error may be up to $5 million per year.

Banking, billing, and other financial applications use decimal extensively. Such applications may rely on a low-level decimal software library or use dedicated hardware circuits to perform the basic decimal arithmetic operations. Two software libraries were proposed to implement the decimal formats of the new IEEE standard: one using the densely packed decimal encoding [6] and the other using the binary encoded decimal format [7]. Hardware designs were also proposed for addition [8], multiplication [9], [10], division [11], [12], square root [13], as well as complete processors [14].

A benchmarking study [15] estimates that many financial applications spend over 75% of their execution time in Decimal Floating Point (DFP) functions. For this class of applications, the speedup resulting from the use of a fast hardware implementation versus a pure software implementation ranges from a factor of 5.3 to a factor of 31.2 depending on the specific application running. This speedup is for the complete application including the non-decimal parts of the code. According to our knowledge, all the previous research focused only on the time saved. However, we see that the savings in energy are even more important. Our own preliminary estimates indicate that energy savings

for the whole application due to the use of dedicated hardware instead of a software layer are of the same order of magnitude as the time savings.

Gonzalez and Horowitz [16] argue that the process normalized Energy Delay Product (EDP) is the most suitable metric for architectural evaluations. This metric clearly indicates the architectural improvements that contribute the most to both performance and energy efficiency. A hardware implementation for DFP units is a definite winner in this case since it gives from two to three orders of magnitude improvement in EDP as a conservative estimate.

The following section explains our own designs for the DFP units and compares them to other implementations. Then, section 3 presents our preliminary energy results and its implications on real machines. Finally, section 4 presents the conclusions.

2. DFP hardware units

Our goal is to provide functionally correct, standard compliant, high performance hardware units for the major decimal arithmetic operations listed in the standard, namely: addition/subtraction, multiplication, fused multiply-add (FMA), division, and square root. In this paper, we report our designs implementing all these operations for the decimal encoded decimal64 and decimal128 formats of the standard. (Currently, the FMA supports decimal64 only but is easily extendible to decimal128.)

All the units support seven rounding directions. Four of these are directed roundings: toward zero (RZ), away from zero (RA), toward plus infinity (RP), and toward minus infinity (RM). While the other three round to nearest but handle the tie case differently: ties to even (RNE), ties away from zero (RNA), and ties to zero (RNZ). The standard mandates the provision of RZ, RP, RM, RNE, RNA in any compliant decimal implementation. The other two (RA and RNZ) are used in some applications and defined in the BigDecimal library [17].

The proof of correct functionality and full standard compliance of floating point units is a very complicated task [18]. To get around this and still provide robust designs, companies accumulated over the years large bodies of test cases to check the critical conditions. Recently, designers used formal verification methods [19] as an alternative approach to ensure the quality and correctness of their units. For the case of DFP, the verification team of IBM developed a software engine [20] that generates test cases based on a description of the constraints in the standard. We built a free tool [21] to parse those test cases and produce output files with test vectors suitable for direct use with hardware simulators. Each of our designs is simulated using those test vectors as well as a large number of random cases to check its correct functionality in all the seven rounding directions and the correct generation of the required flags for exceptional cases according to the standard.

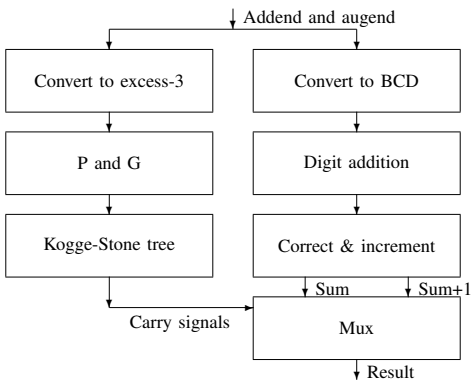

Figure 1. Fast multi-digit decimal adder

2.1. Decimal adder

For each of the decimal64 and decimal128, we designed two different adder implementations, one for high speed and the other for low area. After the correct alignments of the significands based on the exponents and leading zeros, the core of all our DFP adders uses a new fast decimal adder based on a Kogge-Stone prefix tree shown in Fig. 1. In this adder, both the addend and augend are converted to regular Binary Coded Decimal (BCD) and excess-3 encodings simultaneously. The sum of two BCD digits requires a correction only if it exceeds nine. This comparison with nine delays the generation of the corresponding carry signal. In excess-3, the sum digit requires a correction (to subtract 3) if it is nine or less and a different correction (to add 3) if it is greater than nine. However, the correct carry signal is generated quickly. Our adder combines the advantages of both encodings. It uses the excess-3 encoding to get the propagate and generate signals that are fed into the Kogge-Stone tree to quickly get the carry signal corresponding to each digit position in the significand. In parallel, it produces the sum and incremented sum of the BCD digits in each position. The carry signals then select the correct result for each position.

Furthermore and in contrast to the previous designs [8], our DFP adders generate the sticky bit in parallel with the alignment shifter then use that bit in an injection based rounding [8].

2.2. Decimal multiplier

Our multiplier [10] contains two main paths: significand path to generate the product's significand, and the exponent path to generate the product's exponent and the corresponding flags. The significand path relies on a fully parallel

decimal multiplier [22] to generate the partial products in parallel and reduce them to two vectors (sum and carry) using a carry save addition tree.

These two vectors are added using our new fast decimal carry propagation adder of Fig. 1. The exponents of both operands and the count of leading zeros determine the required amount to shift the result's significand into its correct place then it is rounded.

2.3. Decimal fused multiply add

Our decimal FMA implements the operation $\pm(a \times b) \pm c$ with a single final rounding. The FMA was introduced as a required operation in the revised standard [2]. The single rounding is not the only difference between FMA and a multiplication followed by addition. The exceptional cases are also different since the standard mandates that no underflow, overflow, or inexact exception arise due to the multiplication, but only due to the addition.

The core of the FMA uses a significand path similar to that of our multiplier explained above and introduces the operand c after any required alignment as an additional partial product in the reduction tree. For p digits per operand, the product $a \times b$ has $2p$ digits and operand c may be shifted either to its right or left for alignment. Digits shifted to the right of the product affect the sticky bit generation. Otherwise, if operand c coincides with part of the product or is shifted to its left, we use a $3p$ digits wide final adder to get the result which is later rounded.

2.4. Decimal divider and square root

Our design for the divider and square root uses a modified Newton-Raphson method [11], [13] to generate the initial approximation then iterate on the equation $x_{i+1} = x_i(2 - bx_i)$ to find the reciprocal of b or $x_{i+1} = x_i(3 - bx_i^2)/2$ to find the reciprocal square root of b. Our contribution is the use of an optimized fully parallel FMA unit with a fixed addend to generate the reciprocal and the quotient in the case of division or the reciprocal square root for the square root. Moreover, we keep all the intermediate results in a redundant form to speed up the operation. Hence, the regular multiplication circuit is modified to take as input and produce as output a redundant form. Furthermore, the proposed designs use a new and fast rounding scheme.

The rounding algorithm presented by Wang and Schulte [11] truncates the quotient to $p + 1$ digits for a significand of precision p then adds $10^{-(p+1)}$ followed by a final truncation to p digits. This scheme may produce an incorrect result in some cases. Our algorithm truncates the quotient to p digits and checks the actual remainder to decide on the correct rounding. Table 1 presents an example showing the difference when $a = +8080699100134968$, $b = +910809186219000$, $x_0 = +1.097924$, and exact $q = a/b = +8.872$.

Table 1. Example of incorrect rounding.

iteration	Our algorithm	Algorithm [11]
$x_1 = x_0(2 - bx_0)$	1.097 924 807 006 738 06	1.097 924 807 006 738 06
$x_2 = x_1(2 - bx_1)$	1.097 924 807 007 331 24	1.097 924 807 007 331 24
$q = ax_2$	8.871 999 999 999 999 87	8.871 999 999 999 999 87
	truncate keeping p digits	truncate at $p + 1$ then increment
q	8.871 999 999 999 999	8.871 999 999 999 999 9
	remainder error is 1 ulp, increment	truncate
q	8.872	8.871 999 999 999 999
	correct	incorrect

For the decimal64 format, the Digit-Recurrence divider implementation [12] has an estimated total delay of 680 Fanout of 4 (FO4) while our design gives 734 FO4, i.e. only about 8% difference. For decimal128, the digit-recurrence technique doubles the delay. However, the use of the Newton-Raphson technique in our divider leads to much better performance since only a single additional iteration is needed. Since the implementation of Wang and Schulte [11] uses a serial multiplier, its delay estimate for decimal64 is 2300 FO4, more than three times the delay of our proposal.

3. Experimental energy evaluation

To verify our designs in real hardware, we synthesized them on an Altera Cyclone II FPGA (field programmable gate array) development kit. On the FPGA, our hardware connects to a NIOS II processor as a slave memory mapped component on the Avalon bus. Due to the limitation of this configuration, the operands are transmitted to our designs on several clock cycles. This connection, however, models how an existing architecture may be retrofit with a DFP acceleration card. A much higher performance is expected from a direct implementation within a processor core.

This system runs the telco benchmark [5] in two modes: a pure 'software' mode based on the DecNumber library [6] and a 'hardware' mode. In this latter mode, when a decimal operation is needed the NIOS processor sends the operands to our hardware on the bus then reads the result. Table 2 shows the statistics for the number of clock cycles needed in the million numbers of the benchmark. Using the PowerPlay Early Power Estimator tool of Altera, the estimated average power per instruction is 109 mW when the FPGA runs at its default frequency of 50 MHz. Hence, the table also shows the estimated average energy in each case.

These preliminary results indicate that a hardware acceleration card would run 23 times faster than software on average which gives an improvement in energy-delay product of over 500. Explained differently, a user who spends just 1 minute in decimal calculations out of every

Table 2. Clock cycles and energy for HW versus SW

	HW		SW	
	Cycles	Energy (μJ)	Cycles	Energy (μJ)
max	2591	5.65	52421	114.28
min	995	2.17	12870	28.06
average	1249	2.72	29004	63.23
StdDev	226	0.49	2256	4.92
median	1444	3.15	29285	63.84

500 minutes on the computer benefits from DFP acceleration. Clearly, an architecture incorporating DFP hardware within the processor will see even greater gains.

4. Conclusions

To the best of the authors' knowledge, this is the first paper to indicate the energy savings resulting from hardware implementations of decimal floating point operations. Using the energy delay product as a metric, a hardware implementation is beneficial even if the user runs decimal applications only rarely.

This work also reports the first hardware implementation of the FMA and the fastest hardware DFP divider using Newton-Raphson iterations.

References

[1] W. Buchholz, "Fingers or fists? (the choice of decimal or binary representation)," *Communications of the ACM*, vol. 2, no. 12, pp. 3–11, Dec. 1959.

[2] "IEEE standard for floating-point arithmetic," New York, NY, Aug. 2008, (IEEE Std 754-2008).

[3] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," in *16th IEEE Symposium on Computer Arithmetic: ARITH-16 2003: proceedings: Santiago de Compostela, Spain, 15–18 June, 2003*.

[4] European Commission, *The Introduction of the Euro and the Rounding of Currency Amounts*, European Commission Directorate General II Economic and Financial Affairs, Brussels, Belgium, 1997.

[5] M. F. Cowlishaw, "The 'telco' benchmark," World-Wide Web document., Hursley, UK, 2002. [Online]. Available: http://speleotrove.com/decimal/telco.html

[6] M. Cowlishaw, *The decNumber C library*, IBM Corporation, Apr. 2007, version 3.40. [Online]. Available: http://speleotrove.com/decimal/

[7] M. Cornea, C. Anderson, J. Harrision, P. Tang, E. Schneider, and C. Tsen, "A software implementation of the IEEE 754r decimal floating-point arithmetic using the binary encoding," in *Proceedings of the IEEE International Symposium on Computer Arithmetic, 25–27 June, 2007, Montpellier, France*.

[8] L.-K. Wang, M. J. Schulte, J. D. Thompson, and N. Jairam, "Hardware designs for decimal floating-point addition and related operations," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 322–335, Mar. 2009.

[9] M. A. Erle, M. J. Schulte, and B. J. Hickmann, "Decimal floating-point multiplication via carry-save addition," in *Proceedings of the IEEE International Symposium on Computer Arithmetic, 25–27 June, 2007, Montpellier, France*.

[10] R. Raafat, A. M. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, M. Elkhouly, and H. A. H. Fahmy, "A decimal fully parallel and pipelined floating point multiplier," in *Forty-Second Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA*, Oct. 2008.

[11] L.-K. Wang and M. J. Schulte, "A decimal floating-point divider using Newton–Raphson iteration," *Journal of VLSI Signal Processing*, vol. 49, no. 1, pp. 3–18, Oct. 2007.

[12] H. Nikmehr, B. Phillips, and C.-C. Lim, "Fast decimal floating-point division," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 951–961, Sep. 2006.

[13] L.-K. Wang and M. J. Schulte, "Decimal floating-point square root using Newton–Raphson iteration," in *16th IEEE International Conference on Application-Specific Systems, Architectures, and Processors: ASAP 2005: 23–25 July 2005, Samos, Greece*.

[14] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, "Decimal floating-point support on the IBM system z10 processor," *IBM Journal of Research and Development*, vol. 53, no. 1, 2009.

[15] L.-K. Wang, C. Tsen, M. J. Schulte, and D. Jhalani, "Benchmarks and performance analysis of decimal floating-point applications," *IEEE*, pp. 164–170, 2007.

[16] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, Sep. 1996.

[17] Sun Microsystems, *BigDecimal (Java 2 Platform SE v1.4.0)*, Sun Microsystems, Mountain View, CA, USA, 2002. [Online]. Available: http://java.sun/com/products

[18] D. M. Russinoff, "A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7^{TM} processor," *LMS Journal of Computation and Mathematics*, vol. 1, pp. 148–200, 1998.

[19] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: a survey," *ACM Transactions on Design Automation of Electronic Systems.*, vol. 4, no. 2, pp. 123–193, Apr. 1999.

[20] "Floating point test suite." [Online]. Available: http://www.haifa.ibm.com/projects/verification/fpgen/ieeets.html

[21] "Silminds decimal parsing tool." [Online]. Available: http://www.silminds.com/index.php?option=com_content&task=view&id=10&Itemid=37

[22] A. Vazquez, E. Antelo, and P. Montuschi, "A new family of high-performance parallel decimal multipliers," in *Proceedings of the 18th IEEE Symposium on Computer Arithmetic, 25–27 June, 2007, Montpellier, France*.