Decimal Floating Point for future processors

Hossam A. H. Fahmy
Electronics and Communications Department, Cairo University, Egypt
Email: hfahmy@stanfordalumni.org
Tarek ElDeeb, Mahmoud Yousef Hassan, Yasmin Farouk, Ramy Raafat Eissa
SilMinds, LLC
Email: tarek.eldeeb@silminds.com

Abstract—Many new designs for Decimal Floating Point (DFP) hardware units have been proposed in the last few years. To date, only the IBM POWER6 and POWER7 processors include internal units for decimal floating point processing. We have designed and tested several DFP units including an adder, multiplier, divider, square root, and fused-multiply-add compliant with the IEEE 754-2008 standard. This paper presents the results of using our units as part of a vector co-processor and the anticipated gains once the units are moved on chip with the main processor.

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 his seminal paper [1] in 1959, 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.

Work partially completed under grant number C2/S1/163 from the Research, Development, and Innovation program (RDI) of the Egyptian Ministry of Higher Education and Scientific Research funded by the European Union.

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] which is widely known as the Binary Integer Decimal (BID) encoding. Hardware designs were also proposed for DFP separate operations [8]–[12] , as well as complete processors [13].

The goal of this paper is to present designs that produce the correct results according to the standard efficiently. By efficiently we mean in less time and with less energy.

A benchmarking study [14] 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 [14].

The following section explains our own designs for the DFP units and compares them to other implementations. Section 3 presents the experimental results of using our units within our DFP vector co-processor for financial applications. Finally, section 4 presents the conclusions.

2 DFP HARDWARE UNITS

Our goal is to provide functionally correct, standard compliant, high performance hardware units [15] for the major decimal arithmetic operations listed in the standard, namely: addition/subtraction, multiplication [9], fused multiply-add (FMA) [16], division, and square root. Our designs implement all these operations for the decimal encoded decimal64 and decimal128 formats of the standard. In this paper, we just briefly describe the adder and multiplier units since they are the ones used in the co-processor explained later.

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].

All units were originally designed using general ASIC coding style. Tailored versions of the cores were developed targeting Altera Stratix FPGAs and Xilinx Virtex FPGAs. The tailored versions efficiently utilize the FPGA cells.

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. Currently, the IBM team does not publish the complete test suite. Cairo University in cooperation with SilMinds developed an alternative engine. This verification work [21] included a DFP set of models compliant with the IEEE 754-2008 standard and a model-based test case generator with the same format as IBM's. We built a free tool [22] to parse the test cases generated from either Cairo University or IBM to produce output files with test vectors suitable for direct use with hardware simulators. Our tool also produces pseudo-random test vectors with a variable number of leading zeros. Each of our designs is simulated using the test vectors from IBM, Cairo University, 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.

2.1 Decimal adder

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. 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.

The DFP Adder is modified to incorporate other standard-defined functions; compare, minNumber, maxNumer, toIntegral, quantize and isQuantum.

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 [9] 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 [23] 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. 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.

3 DFP VECTOR CO-PROCESSOR EXPERIMENT

Effectiveness and efficiency of any new technology should be proved. After testing the correctness of our work its efficiency became our focus. For this purpose, a complete testing platform is developed to verify and evaluate our decimal cores.

We synthesized three parallel units of 5-stages pipelined DFP Multiplier, five parallel units of 3-stages pipelined DFP Adder (with multi-function unit) on the Xilinx FPGA. On the FPGA, our hardware connects to a main processor as a slave memory mapped component on the PCI-e bus. We report here the results of a Xilinx board connected to the host computer via a PCI-e X4 gen1 bus. The host computer runs RHEL 5.3 x64 with AMD Athlon-II X2 at 2.8 GHz.

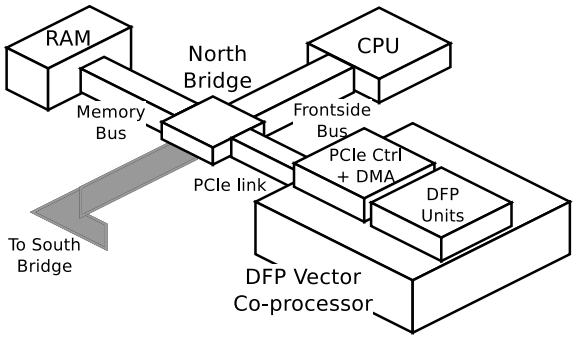

Fig. 1. PCIe linked DFP Vector Co-processor

TABLE 1
SW versus HW modes of telco benchmark

SW	1.130 s (IO overhead = 9% ≈ 100 ms)
SW no IO	1.035 s
SW no IO no toString	0.960 s
HW	0.205 s (IO overhead = 53% ≈ 100 ms)
HW no IO	0.095 s
HW no IO no toString	0.0125 s

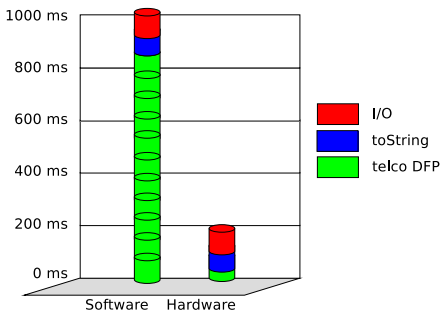

Fig. 2. SW and HW timing for telco benchmark

This system runs the simplified billing application of the C-coded telco benchmark [5] in two modes: a pure 'software' mode based on the DecNumber library [6] and a 'hardware' mode. The software mode depends on the decDouble module that is a part of the decNumber package, written in C, and implements the decimal64 format of the standard using the decimal encoding. The decimal arithmetic calculations constitute about 73.4% of the benchmark time.

Due to the PCI-e bandwidth limitation, each scalar DFP operation suffers from two PCI-e slow round trips leading to a delay of 2.3μs before the initiation of a new operation. The performance ceiling of $\frac{1}{2.3\mu s} = 435K$ Operation per second is unacceptable since it is almost 25 times slower than software. This connection, in Fig. 1, 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.

Our proposed co-processor is a vector —rather than scalar— co-processor designed with internal register banks. The internal registers are designed to support a vector datatype, where a vector is a collection of many related decimal floating point numbers. Another register file is designed to support scalar operands. This variety of supported data types allows the co-processor to handle decimal operations with operands of vector and/or scalar format. Our DFP cores are connected within the co-processor on a bus controlled by a bus arbiter to control the internal data transfers. We intend to have an instruction set to support the standard functions mentioned in IEEE Std. 754-2008 including the whole decimal operations, comparison operations, and the conversion operations. Some of the instructions developed control the internal data transfer between the DFP cores in order to execute algorithm containing several dependent decimal operations without PCI-e transfer by keeping the intermediate results stored internally. This design cuts down the PCI-e transfers enormously, resulting in much higher performance.

The intermediate results are stored in a format of sign, Exponent, a Binary Coded Decimal BCD significand and some special flags to indicate whether the number is normal or infinity or NotaNumber (NaN). Storing the received input data and the intermediate results internally in such a format eliminates the overhead of the conversion from/to the standard decimal encoding format during each decimal operation. The result of a decimal operation is considered as intermediate result by default and stored in the internal vector register banks, when the software algorithm needs a result of a certain decimal operation, it sends a vector store instruction to the co-processor to send the results back to the memory of the host computer. Since the synthesized DFP cores may have a different number of pipeline stages, we designed a simple data hazard handler to keep the data transfer synchronization between the DFP cores.

In the vector hardware mode, when a decimal operation is needed the processor of the host machine sends vector operands to SilMinds DFP hardware through a DMA transfer then reads the final result through another DMA transfer. The DMA is configured to have 2 DMA channels, a maximum payload size of 256 Bytes, a maximum read request size of 512 Bytes and a total of 16 outstanding requests. We modify the telco benchmark source code to have the decimal arithmetic calculations part be executed by our DFP hardware unit. The total time to run the benchmark includes a part for input/output manipulation of the data from files on the disk as well as changing the numerical values to printable strings. We illustrate the effect of those two factors in our results shown in table 1. Figure 2 shows the same results graphically. The use of our hardware units leads to a clear speedup for the complete program of $1.13/0.205 = 5.51$. If the computation only is considered we get even higher speedups: $1.035/0.09 = 11.5$ (no IO) and $0.96/0.0125 = 76.8$ (no IO, no toString).

4 Conclusions

This work presents the prototype of our decimal co-processor. According to the authors' knowledge, this is the first decimal co-processor in the world. We implemented our designs in FPGA and tested them using various test benches. Furthermore, we ran a typical application on a processor connected to our vector co-processor including our DFP designs.

We anticipate that support for decimal floating point may be retrofit into older architectures as add-on acceleration cards such as the one we presented in this paper. DFP might be supported directly within processors in future architectures as well.

References

[1] W. Buchholz, "Fingers or fists? (the choice of decimal or binary representation)," *Communications of the ACM*, 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*

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

[5] M. F. Cowlishaw, "The 'telco' benchmark,". Available: http://speleotrove.com/decimal/telco.html

[6] M. Cowlishaw, *The decNumber C library*, IBM Corporation, Apr. 2007, version 3.40.

[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, Montpellier, France*, Jun. 2007.

[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*, Mar. 2009.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] L.-K. Wang, C. Tsen, M. J. Schulte, and D. Jhalani, "Benchmarks and performance analysis of decimal floating-point applications," in *ICCD 25th International Conference on Computer Design*, Oct. 2007, pp. 164–170.

[15] H. A. H. Fahmy, R. Raafat, A. M. Abdel-Majeed, R. Samy, T. El-Deeb, and Y. Farouk, "Energy and delay improvement via decimal floating point units," in *Proceedings of the 19th IEEE Symposium on Computer Arithmetic, Portland, Oregon, USA*, pp. 221–224, June 2009.

[16] R. Samy, H. A. H. Fahmy, R. Raafat, A. Mohamed, T. ElDeeb, and Y. Farouk, "A decimal floating-point fused-multiply-add unit," in *Fifty-Third MidWest Symposium on Circuits and Systems, (MWSCS), Seattle, Washington, USA*, Aug. 2010.

[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] A. Sayed-Ahmed and H. A. H. Fahmy, "Three Engines to Solve Verification Constraints of Decimal Floating-Point Operations," *Submitted to the Forty-Fourth Asilomar Conference on Signals, Systems, and Computers, 2010, Asilomar, California, USA.*

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

[23] 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, June 25–27, 2007, Montpellier, France.*