International Journal of Research in Computer Science
ISSN 2249-8257 Volume 1 Issue 1 (2011) pp. 1-7

www.ijorcs.org

IEEE-754 compliant Algorithms for Fast Multiplication of Double Precision Floating Point Numbers

Geetanjali Wasson

Assistant Professor, IET Bhaddal, Ropar, Punjab
Email: geetanjlipuri@gmail.com

Abstract

In image and signal processing applications floating-point multiplication is major concern in calculations. Performing multiplication on floating point data is a long course of action and requires huge quantity of processing time. By improving the speed of multiplication task overall speed of the system can be enhanced. The Bottleneck in the floating point multiplication process is the multiplication of mantissas which needs 53*53 bit integer multiplier for double precision floating point numbers. By improving the speed of multiplication task the overall speed of the system can be improved. In this paper a comparison on the existing trends in mantissa multiplication is made. Vedic and Canonic Signed digit algorithms were compared on parameters like speed, complexity of routing, pipelining, resource required on FPGA. The comparison showed that Canonic Signed Digit Algorithm is better than Vedic algorithm in terms of speed and resources required on spartan^3 FPGA.

Keywords: CSD (Canonic Signed Digit); FPGA (Field Programmable Gated Array); Double precision floating point number.

I. Introduction

There have been several explorations for implementing floating point arithmetic units on FPGAs to perform arithmetic operations on floating point data more rapidly than software algorithms. FPGA based floating point multiplication can be categorized as IP core based and algorithm based [1-2]. In IP core based highly optimized IP cores are employed as sub units of floating point unit design and in algorithm based implementation techniques are employed to get better performance by grading system goals as speed, area and throughput optimizations [2-5].

Power consumption of multipliers is governed by its number of non zero digits. If no technique is used to reduce the number of non-zero digits then multiplication of two N bit numbers produces N partial products and N-1 adders are needed to get result of multiplication [6][10]. Different techniques are

employed to reduce the number of adder stages and to accelerate the addition of partial products generated during multiplication process. Some algorithms use compression techniques to reduce the stages of addition [13]. Some algorithms accelerated the addition process by reducing number of non-zero partial products. This is done by coding the digits of in such a way that it has minimum number of non zero digits [6-10].

II. Related Work

Image processing, Digital Signal Processing and communication systems all require competent and fast floating point multiplication. The bottleneck of floating point multiplication is mantissa multiplication which needs 53*53 bit integer multiplier for double precision floating point numbers. There are many multiplier architectures using different multiplication techniques for example, Array multiplier, redundant binary architecture and many more architectures using tree structures but they have problem of larger delay [1-5]. Different algorithms are also there that perform floating point multiplication like BOOTH algorithm, algorithms based on ancient mathematics [11], and many more. Ali Akoglu introduced an algorithm using ancient Vedic mathematics rules for fast multiplication which generate partial products concurrently.

The rest of the paper is organized as follows: In section 3, floating point multiplication technique is introduced. In section 4, BOOTH algorithm is introduced. In section 5, Vedic Algorithm is discussed, in section 6, CSD algorithm and its implementation is discussed. Results are shown in section 7 and comparison is done with Ali Akoglu's Vedic algorithm [13] and section 8 contains conclusion and future scope.

III. Floating Point Multiplication

Floating-point representation can be thought of as a computer realization of scientific notation. Unlike fixed point numbers, in floating point numbers, radix point is not fixed [3]. The advantage of floating-point representation over fixed-point representation is that it can support wider range of values. Floating point formats generally followed for computation practices, are those which confirm to IEEE-754 standard. These standards play vital role in ensuring numerical robustness and code compatibility among machines of different architectures [5]. According to IEEE-754 standards, the floating point number is represented as:

$$V= (-1)^{sign} * 2^{exponent-bias} * 1.fraction \quad [1]$$

Implicit bit is used before fraction or mantissa, which is „1 for normalized number and „0 for un-normalized number. Exponent bias is $(2^{e-1}) - 1$, which comes out to be 127 for single precision and 1023 for double precision exponent.

Floating point multiplication is not straightforward as integer multiplication. It involves three operations. Exponents of multiplier and multiplicand are added, mantissas of two numbers are multiplied and ex-or of sign bits is done to get final sign bit [1-10].

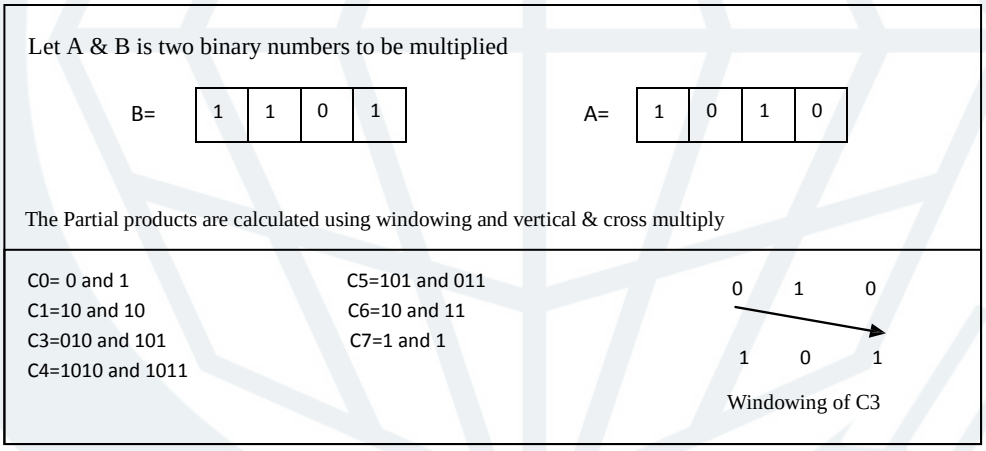
IV. Booth Multiplication Algorithm

In BOOTH multiplication algorithm to reduce the time needed for multiplication number of partial products to be added are reduced [1-3]. BOOTH recording reduces the number of adder units needed and hence reduced the delay by reducing number of nonzero bits in the multiplier [2]. In BOOTH recoding, the long sequence of 1s is replaced by two no zero bits; for example, If digits *j* through (down to) *k* are 1s, then,

$$2^j + 2^{j+1} + \ldots + 2^{k+1} + 2^k = 2^{j+1} - 2^k$$

This represents, the sequence of additions can be replaced by an addition of the multiplicand shifted by *j*+1 positions and a subtraction of the multiplicand shifted by *k* positions [5]. The drawback of BOOTH recording is the high power consumption and thus reduced efficiency [3-8].

V. Vedic Multiplication



Vedic algorithm based architectures for floating point multiplication are power efficient and are suitable for high speed applications, with reduced routing complexity [7-10],[23]. It encourages parallel computation and increases speed of multiplication by introducing inherent pipe-lined stages without any increased over head. This is achieved by using adder tree structures for addition of partial products.

This algorithm basically deals with mantissa multiplication and has three stages for multiplication.

i) Partial Product Generation
ii) Accumulation
iii) Final Addition

Partial products are generated by windowing of the multiplier and multiplicand. As shown in Figure-1. Then accumulation is done of partial products and then finally partial products are added using binary tree [5],[7],[8],[11]. Accumulation is done by using Wallace tree as shown in Figure2. The implementation of algorithm and stages of pipeline added depends on the implementation of accumulation and addition stages [8]. The speed can be increased by increasing radix of multiplication. Vedic algorithm is speed efficient and very less complex as compare to CSD but resources needed are

more [11]. And numbers of partial products are not reduced so adder units needed are more as compare to CSD. After the accumulation binary tree addition is done to get final mantissa multiplication result of 2n-1 bits [11].

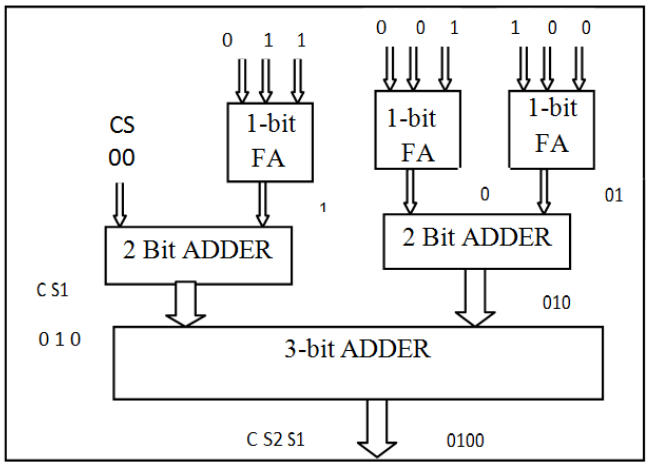


Figure 2. Accumulation stage 3:1 Compression using Wallace Tree

VI. Canonic Signed Digit

CSD is modified form of BOOTH algorithm. It utilizes the fact that the multiplication which is to be carried out by power of two can be easily obtained by simply shifting the multiplicand to right/left using some dedicated hardware and incorporating some adders. It also utilizes the fact that the multiplication by a left shift is much faster than repeated additions and more the number of 1's in multiplier [1-2], the slower the multiplication. CSD is a unique signed digit representation having smallest numbers of non-zero digits, with a property that no two consecutive bits are non zero.CSD can have n/2 non-zero digits for n which is even. Thus to multiply two n-bits integers, it takes n/2-1 number of steps [1], [7-9].
Following are the properties of CSD numbers:

i) No two consecutive bits in a CSD number are non- zero.
ii) The CSD representation of a number contains the minimum possible number of non-zero bits, thus the name canonic.
iii) The CSD representation of a number is unique.
iv) CSD numbers contains about 33% fewer non-zero bits than two's complement numbers.
v) Mantissa multiplication

CSD is minimal signed digit code; minimal representation refers to a code requiring minimum number of non-zero digits for representation of a given number.

A. CSD Algorithm:

In CSD algorithm, coding of mantissa of multiplier is done so as to minimize the number of non-zero digits. The algorithm computes the recoded number starting from the least significant digit and proceeding to the left. Let Y is the auxiliary carry, X is the mantissa of multiplier, and C is multiplier s CSD code then, equations 1a and 1b are used to produce CSD code of multiplier.

Starting from LSB of X, first auxiliary carry variable Y0 is set to 0 and subsequently the binary number X is scanned two bits at a time [18].The canonically Recoded digit C_i and the next value of the auxiliary binary variable Y_{i+1} for i = 0, 1,2….. N is generated. Y_{i+1} is carry vector so it follows the rules of binary addition i.e. Y_{i+1} is 1 if there are 2 or three ones among Y_i, X_{i+1}, X_i.

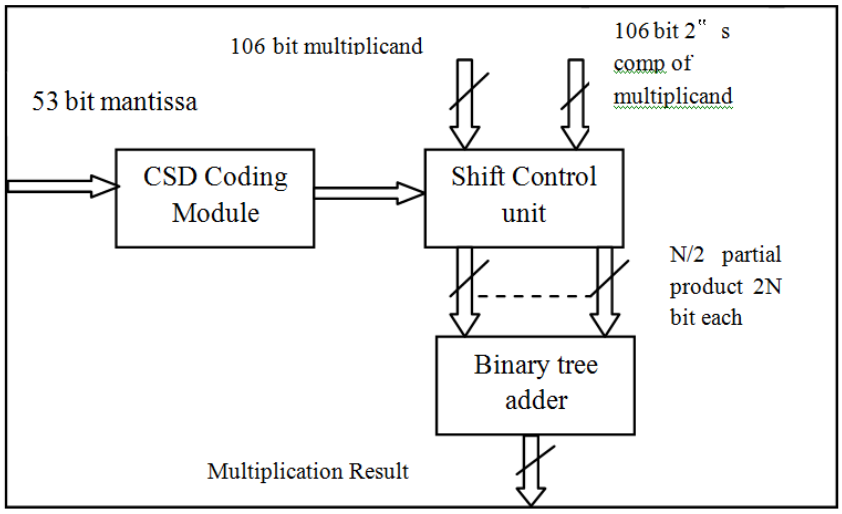


Figure 3: Design flow of CSD algorithm

$Y_{i+1}=Y_i+X_{i+1}+X$ ……. (a)

$C_i=Y_i+X_i-2X_{i+1}$ .....……. (b)

Example: Take a number (13)10 = (01101)2 then using 13 = 16 - 4 + 1. CSD representation of 23 results 10-01. Notation (-) represents (-1).

B. Methodology of CSD algorithm

Product of two N bit numbers is obtained in N/2-1 steps. Figure 3 shows the methodology of multiplication using CSD coding.CSD multiplication unit contains a CSD vector generation unit, a twos compliment generation unit, shift and control logic unit, and adder unit.

i) Multiplier and multiplicand are stored in registers.

ii) Mantissa of multiplicand is fed to twos compliment unit to get twos complement of multiplicand.

iii) CSD coding of mantissa of multiplier is done and CSD code is fed to the shift control logic to produce the required number of left shifts to multiplicand or two s complimen depending on sign and position of non-zero digit in coded vector. Barrel shifter is used for shifting the multiplicand. Utilizing the property of CSD code that no two consecutive bits can be non-zero. N/2 partial products are generated.

iv) These N/2 partial products are given to binary tree adder. Tree adder has property of providing pipeline without increasing overhead. Ten stages of pipeline are achieved and output of tree adder is result of mantissa multiplication.

v) The result is fed to normalization and rounding unit and then after exponent adjustment the results are packed to get final result.

VII. Results

Table 1 show that our algorithm outperforms any other algorithm in terms of maximum clock frequency. Our algorithm needs 10 pipeline stages for mantissa multiplication. Table 2 compares the results of CSD with other fast algorithm for mantissa multiplication and it is observed that CSD is better than Vedic algorithm when maximum clock frequency is prime concern.

Table 1: Comparison of mantissa multiplication performance on Virtex 2 device (virtex2p xc2vp50)

Mantissa Multiplier	**OURS**	**LEE**	**AKOLGU**	**GOVINDU**
Speed(In Mhz)	341.530	67	231	200
Pipelines	10	2	6	7

Table 2: Comparison of our d.p. floating point multiplication with akoglu vedic algorithm on spartan3 device (3s1500fg676)

FP MULTIPLICATION	**VEDIC**	**OURS**
Speed (In Mhz)	312.891mhz	370.5096mhz
Area(In Slices)	6848	5036
4 Input Luts	11895	8959
Pipeliones/Latency	12	10

VIII. Conclusion And Future Work

This paper presents CSD algorithm implementation for double precision binary floating point multiplication. A signed digit coding is used by which number of partial products and hence addition levels in adder stage are reduced. Mantissa multiplication performance reaches to 385.505MHz on virtex4, 341.530MHz on virtex2 and 370.096MHz on Spartan^3.While performing multiplication on double precision binary floating point numbers, for mantissa multiplication task Canonic Signed digit algorithm is better than Vedic algorithm as: Maximum Clock Frequency is more with Canonic Signed digit algorithm than that allowed by Vedic Algorithm. Slices needed by CSD algorithm are 14% less than that required for Vedic Algorithm. LUTs needed for CSD algorithm are 11% less than that needed by Vedic Algorithm. Flip-Flops needed for CSD implementation are 9% more than required by Vedic Algorithm. Future work involves the optimization of resources and implementation of division and square root operations.

IX. References

[1] B .C. Jinaga, Saroja. V Siddamal, R.M Banakar, "Design of high-speed floating point multiplier" 4th IEEE International Symposium on Electronic Design, Test & Applications, IEEE computer society, 2008, pp. 285,289

[2] E.M.Saad, M.Taher, "High speed area efficient FPGA based floating point arithmetic modules", National conference on radio science (NRSC 2007), March-2007, pp 1-8. doi:10.1109/DELTA.2008.19

[3] Sang-min Kim, "Low error fixed width CSD multiplier with efficient sign extension", IEEE Transactions on circuits and systems-II: analog and digital signal processing,vol50,December 2003. doi:10.1109/TCSII.2003.820231

[4] "Iterative radix-8 multiplier structure based on a novel real time CSD recoding", ACSSC 2007, conference record of forty first Asilomar conference on signals, systems & computers, 2007, pp 977 to 981.

[5] Himanshu Thapliyal, "Modified Montgomery Modular Multiplication using 4:2 Compressor and CSA Adder", Proceedings of the third IEEE international workshop on electronic design, test and applications (DELTA 06), Jan 2005

[6] B.Lee, N. Burgess, "Parameterizeable Floating-point Operations on FPGA" Signals, Systems and Computers, 2002. Conference Record of the 36th Asilomar Conference on Volume 2, 3-6 Nov. 2002, pp. 1064-1068

[7] G. Govindu, L. Zhuo, S. Choi and V. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs,"l8th International Parallel and Distributed Processing Symposium, IPDPS 2004, Apr 26-30 2004, pp. 2043-2050. doi:10.1109/IPDPS.2004.1303135

[8] G.Lienhart, A.Kugel and R.Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific body simulations", IEEE Symposium on Field-Programmable Custom Computing Machines. IEEE Computer, April 2002, pp.182-191.

[9] Kwen-Siong Chong, "A Micro-power Low-Voltage Multiplier With Reduced Spurious Switching", IEEE transactions on VLSI systems, vol13, No.2, Feb 07 , pp 255-265

[10] Mark A. Erle, "Decimal Floating-Point Multiplication Via Carry-Save Addition", 18th IEEE Symposium on Computer Arithmetic (ARITH'07), 2007. doi:10.1109/ARITH.2007.14

[11] Nachiket Kapre, "Optimistic Parallelization of Floating-Point Accumulation", 18th IEEE Symposium on Computer Arithmetic(ARITH'07), 2007. doi:10.1109/ARITH.2007.25

[12] Ali Akoglu, Sandeep K Venishetti, "A Highly Parallel FPGA based IEEE-754 Compliant Double Precision Binary Floating Point Multiplication Algorithm" International conference on field programmable technology (FPT), Dec-2007, pp 145-152. doi:10.1109/FPT.2007.4439243

[13] G.Govindu, L.Zhuo, "Analysis of high-performance floating point arithmetic on FPGAs", 18th International Parallel and Distributed Processing Symposium, IPDPS 2004, Apr 2004, pp 2043-205.

How to cite

Geetanjali Wasson, "IEEE-754 compliant Algorithms for Fast Multiplication of Double Precision Floating Point Numbers". *International Journal of Research in Computer Science*, 1 (1): pp. 1-7, September 2011. doi:10.7815/ijorcs.11.2011.001