ISSN 2348 - 2249
Volume-2, Issue-4, October-December, 2014
International Journal for Multi Disciplinary Engineering and Business Management
(IJMDEBM)

Analysis of Double Precision Floating Point Multiplier on FPGA

Shivkan Misra
Department of ECE
Punjabi University, Patiala
India
shivkanmisra@yahoo.co.in

Harjinder Singh
Department of ECE
Punjabi University, Patiala
India
hrjindr@gmail.com

Abstract – Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation. The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point formats, viz., single precision, single extended, double precision, double extended. This paper presents the FPGA implementation of double precision floating multiplier.

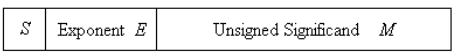
Keywords - Double Precision, Field Programmable Gate Array, Multiplier, Single Precision, Floating Point.

I. INTRODUCTION

A Floating Point Unit is the principal component in Digital Signal Processor, high performance computer systems and graphics accelerators. The floating point multiplication operations are greatly affected by how the floating point multiplier is designed. Floating point numbers are used to obtain a dynamic range for representable real numbers without having to scale the operands. Floating point numbers are approximations of real numbers and it is not possible to represent an infinite continum of real data into precisely equivalent floating point value. Number system is completely specified by specifying a suitable base β, significand (or mantissa) M, and exponent E. A floating point number F has the value

F=M βE

β is the base of exponent and it is common to all floating point numbers in a system. Commonly the significand is a signed - magnitude fraction. The floating point format in such a case consists of a sign bit S, e bits of an exponent E, and m bits of an unsigned fraction M as



The value of such a floating point number is given by:

$$F = (-1)^S M$$

The most common representation of exponent is as a biased exponent, according to which

$$E = E^{true} + bi$$

bias is a constant and Etrue is the true value of exponent. The range of Etrue using the e bits of the exponent field is

$$-2^{e-1} \le E^{true} \le 2^{e-1} -$$

The bias is normally selected as the magnitude of the most negative exponent; i.e. 2e-1, so that

$$0 \le E \le 2^e -$$

The advantage of using biased exponent is that when comparing two exponents, which is needed in the floating point addition, for example the sign bits of exponents can be ignored and they can be treated as unsigned numbers.

The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point formats, viz., single precision, single extended, double precision, double extended. The value of the floating point number represented in single precision format is

$$F = (-1)^S 1.f2^{E-1}$$

where 127 is the value of bias in single precision format (2n-1 –1) and exponent E ranges between 1 and 254, and E = 0 and E = 255 are reserved for special values.

The value of the floating point number represented in double precision data format is

$$F = (-1)^S 1.f2^{E-10}$$

Where1023 is the value of bias in double precision data format. Exponent E is in the range.

$$1 \le E \le 20$$

The extreme values of E (i.e. E = 0 and E = 2047) are reserved for special values. The extended formats have a higher precision and a higher range compared to single and double precision formats and they are used for intermediate results [2].

II. MULTIPLIER ARCHITECTURES

A) Add and Shift Multiplier

The common multiplication method is "add and shift" algorithm. The general architecture of the shift and add multiplier is shown in the figure below for a 32 bit multiplication. Depending on the value of multiplier LSB bit, a value of the multiplicand is added and accumulated. At each clock cycle the multiplier is shifted one bit to the right and its

© 2013, IJMDEBM, All Rights Reserved

ISSN 2348 - 2249
Volume-2, Issue-3, July-September, 2014
International Journal for Multi Disciplinary Engineering and Business Management
(IJMDEBM)

value is tested. If it is a 0, then only a shift operation is performed. If the value is a 1, then the multiplicand is added to the accumulator and is shifted by one bit to the right. After all the multiplier bits have been tested the product is in the accumulator. The accumulator is 2N (M+N) in size and initially the N, LSBs contains the Multiplier. The delay is N cycles maximum. This circuit has several advantages in asynchronous circuits.

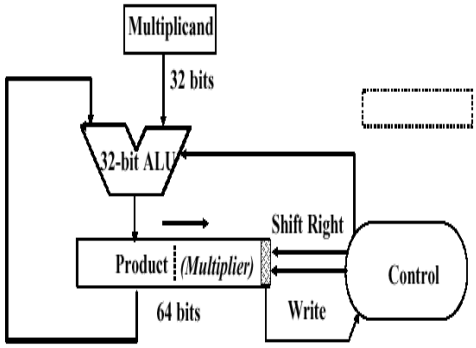


Figure 1: Add and Shift Multiplier

B) Booth Multipliers

It is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly. For the standard add-shift operation, each multiplier bit generates one multiple of the multiplicand to be added to the partial product. If the multiplier is very large, then a large number of multiplicands have to be added. In this case the delay of multiplier is determined mainly by the number of additions to be performed. If there is a way to reduce the number of the additions, the performance will get better. Booth algorithm is a method that will reduce the number of multiplicand multiples. For a given range of numbers to be represented, a higher representation radix leads to fewer digits. Since a k-bit binary number can be interpreted as K/2-digit radix-4 number, a K/3-digit radix-8 number, and so on, it can deal with more than one bit of the multiplier in each cycle by using high radix multiplication.

C) Optimized Wallace Tree Multiplier

Several popular and well-known schemes, with the objective of improving the speed of the parallel multiplier, have been developed in past. Wallace introduced a very important iterative realization of parallel multiplier. This advantage becomes more pronounced for multipliers of bigger than 16 bits.

In Wallace tree architecture, all the bits of all of the partial products in each column are added together by a set of counters in parallel without propagating any carries. Another set of counters then reduces this new matrix and so on, until a two-row matrix is generated. The most common counter used is the 3:2 counter which is a Full Adder.. The final results are added using usually carry propagate adder. The advantage of Wallace tree is speed because the addition of partial products is now O (logN). A block diagram of 4 bit Wallace Tree multiplier is shown in below. As seen from the block diagram partial products are added in Wallace tree block. The result of these additions is the final product bits and sum and carry bits which are added in the final fast adder (CRA).

Since Wallace Tree is a summation method, it can be used in conjunction with array multiplier of any kind including Booth array. The diagram below shows the implementation of 8 bit squarer using the Wallace tree for compressing the addition process.

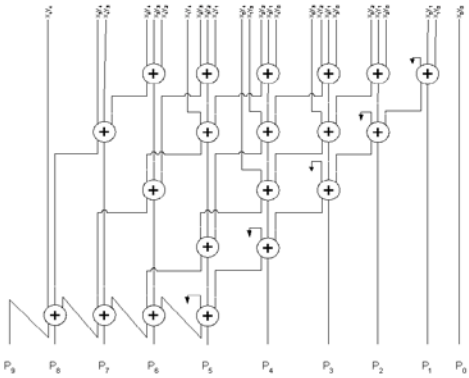


Figure 2: Wallace Tree Multiplier

III. FPGA IMPLEMENTATION OF MULTIPLIER

The FPGA-based implementations of the double precision floating point multiplier have been presented in this section. For implementation purpose, Coregen tool of Xilinx ISE 9.2i software has been used. The Core has been designed for 64 bit number with exponent width of 11 bits and fraction width of 53 bits.

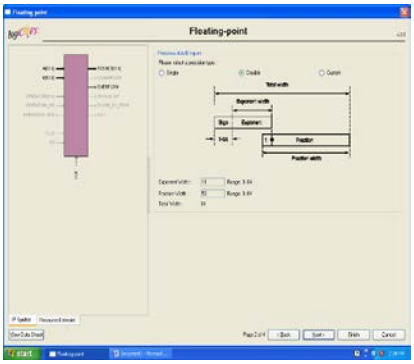


Figure 3: Screen Shot of the Core

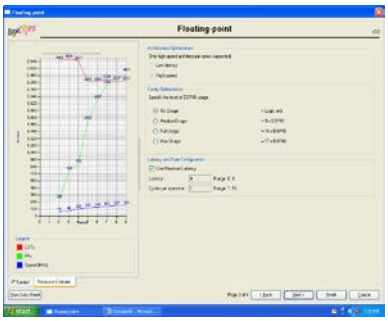


Figure 4: Screen Shot of the Resource Calculation when no DSP48 is used

© 2013, IJMDEBM, All Rights Reserved

ISSN 2348 - 2249
Volume-2, Issue-3, July-September, 2014
International Journal for Multi Disciplinary Engineering and Business Management
(IJMDEBM)

Table 1 Resource Utilization with different Latenct when no DSP48 is used

Latency	Number of LUTs	Number of FFs	Speed (MHz)
2	2661	289	74
3	2684	767	90
4	2657	892	118
5	2295	1632	143
6	2295	1987	146
7	2336	2242	163
8	2317	2317	177
9	2311	2457	185

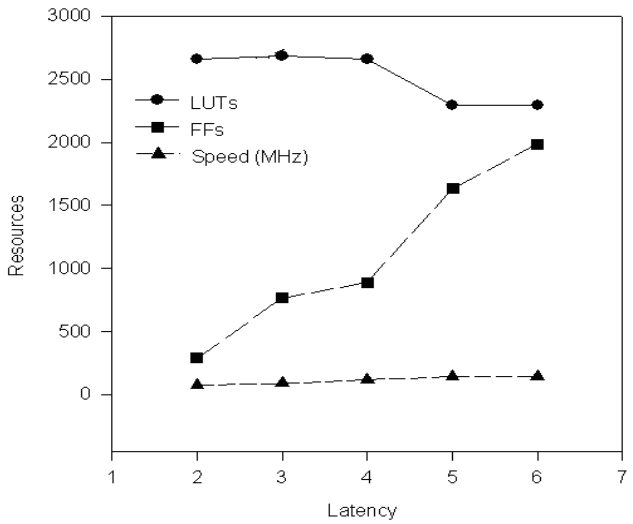


Figure 5: Variation of LUTs, FFs and Speed with Latency when no DSP48 is used

From Table 1 and Figure 5 it has been concluded that when no DSP48s has been used, as the latency increases the usage of LUTs, FFs and speed also increases. For latency of 2, 3, 4, 5, 6, 7, 8, 9 the usage of LUTs varies as 2661, 2684, 2657, 2295, 2295, 2336, 2317 and 2311 respectively. Also for latency of 4 latency of 2, 3, 4, 5, 6, 7, 8, 9 the usage of FFs varies as 289, 767, 892, 1632, 1987, 2242, 2317 and 2457 respectively. For latency of latency of 2, 3, 4, 5, 6, 7, 8, 9 the usage of speed varies as 74 MHz, 90 MHz, 118 MHz, 143 MHz, 146 MHz, 163 MHz, 177 MHz, and185 MHz respectively.

IV. CONCLUSION

In this paper FPGA implementation of double precision floating point multiplier has been presented using Xilinx coregen tool. The use of such tool reduces the design cycle by a large amount. The results has been presented in terms of the various FPGA resources used by the designed multiplies.

REFRENCES

[1] R. V. K. Pillai, "On low power floating point data path architectures", Ph. D thesis, Concordia University, Oct. 1999.
[2] David Goldberg, "Computer Arithmetic", in Computer Architecture: A quantitative approach, D. A. Patterson and J. L. Hennessy, Morgan Kaufman, San Mateo, CA, Appendix A, 1990.
[3] David Goldberg, "What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys, Vol. 23, No. 1, pp. 5-48, March 1991.
[4] Israel Koren, "Compuer Arithmetic Algorithms", Prentice Hall, Englewood Cliffs, 1993.
[5] S.Y. Shah, (M.A.Sc. 2000) Thesis Title: " Low Power Floating Point Architectures for DSP."
[6] Andrew D. Booth, "A signed binary Multiplication Technique", Quarterly J. Mechan. Appl. Math., 4: 236-240,1951.
[7] S. Shah, A. J. Al-Khalili, D. Al-Khalili, "Comparison of 32-bit Multipliers for Various Performance Measures," in proceedings of ICM 2000, Nov.2000.
[8] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashim-oto, H. Hayashida, and K. Maeguchi, "A 10-ns 54*54-b parallel Structured Full array Multiplier with 0.5 micron CMOS Technology", IEEE Journal of Solid State Circuits, vol.26, No.4, pp.600-606, April 1991.
[9] Paul J. Song Giovanni De Micheli, " Circuit and Architecture Trade-offs for High-Speed Multiplication," IEEE Journal of Solid State Circuits, vol 26, No. 9, pp.1184-1198, Sep. 1991.
[10] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili," Low Power Architecture for
[11] Floating Point MAC Fusion," in proceeding of IEE, Computers and Digital Techniques.
[12] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili," An IEEE Compliant Floating Point MAF," in proc. Of VLSI'99, Portugal, Dec.99, pp.165-168, System on Chip,
[13] C.S. Wallace, "A suggestion for a Fast Multiplier", IEEE Trans. Electronic Computers, vol.13, pp. 14-17, Feb 1964.
[14] Riya Saini and R.D.Daruwala, "Efficient Implementation of Pipelined Double Precision Floating Point Multiplier" , International Journal of Engineering Research and Applications, Vol. 3, Issue 1, pp.1676-1679, January - February 2013.
[15] Addanki Purna Ramesh and Rajesh Pattimi," High Speed Double Precision Floating Point Multiplier", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 9, pp. 647 – 650, November 2012.
[16] B.Sreenivasa Ganesh, J.E.N.Abhilash and G. Rajesh Kumar, "Design and Implementation of Floating Point Multiplier for Better Timing Performance", International Journal of Advanced Research in Computer Engineering & Technology, Vol. 1, Issue 7, September 2012.

© 2013, IJMDEBM, All Rights Reserved