A STUDY ON THE FLOATING-POINT ADDER IN FPGAS

Ali Malik
Department of Electrical Engineering
University of Saskatchewan
Saskatoon, SK
email: ali.malik@usask.ca

Seok-Bum Ko
Department of Electrical Engineering
University of Saskatchewan
Saskatoon, SK
email: seokbum.ko@usask.ca

Abstract

FPAGs are increasingly being used to design high-end computationally intense microprocessors capable of handling both fixed and floating-point mathematical operations.

Addition is the most complex operation in a floating-point unit and offers major delay while taking significant area. Over the years, the VLSI community has developed many floating-point adder algorithms mainly aimed to reduce the overall latency. An efficient design of floating-point adder onto an FPGA offers major area and performance overheads. With the recent advancement in FPGA architecture and area density, latency has been the main focus of attention in order to improve performance.

Our research was oriented towards studying and implementing standard, LOP, and far and close data-path floating-point addition algorithms. Each algorithm has complex sub-operations which lead significantly to overall latency of the design. Each of the sub-operation is researched for different implementations and then synthesized onto a Xilinx Virtex2p FPGA device to be chosen for best performance.

According to our results, standard algorithm is the best implementation with respect to area but has overall large latency of 27.059 ns while occupying 541 slices. LOP algorithm improves latency by 6.5% on added expense of 38% area compared to standard algorithm. Far and close data-path implementation shows 19% improvement in latency on added expense of 88% in area compared to standard algorithm.

Keywords: *Floating-point Addition*

1. Introduction

Floating-point addition is the most frequent floating-point operation and accounts for almost half of the scientific operation in math co-processors, DSP processors, embedded arithmetic processors, and data processing units. These components demand high numerical stability and accuracy and hence are floating-point based. Floating-point adder is the most complex operation in a floating-point unit and consists of many variable latency and area dependent sub operations. In floating-point addition implementations, latency is the overall performance bottleneck. Several works have been done to improve the overall latency of floating-point adders. Various algorithms and design approaches have been developed by the VLSI community [1-4] which are analyzed for Field Programmable Gate Arrays (FPGA) in this paper.

In FPGAs, the bottleneck for designing efficient floating-point units has mostly been area. With advancement in FPGA architecture and fabrication, there is a significant increase in FPGA densities. Devices with millions of gates and frequencies reaching up to 300MHz are becoming more suitable for floating-point arithmetic reliant applications.

Floating-point unit is one of the most important custom applications needed in most hardware designs as it adds accuracy, robustness to quantization errors, and ease of use. There are many commercial products related to floating-point adders [12-14] which can be bought to be used in custom designs on FPGAs but cannot be modified for design aspects like throughput, latency, and area. Several works have been done on designing custom floating-point adder on FPGAs which is mostly concentrated on increasing the throughput by deep pipelining [6-11].

The Institute of Electrical and Electronics Engineering (IEEE) issued 754 standard for binary floating-point arithmetic in 1985 [15]. To evaluate different adder algorithms and sub operations, we are only interested in single precision format. Single-precision format uses 1-bit for sign bit, 8-bits for exponent and 23-bits to represent the fraction.

This paper is outlined as follows: Section 2 goes over the standard algorithm and brief description of implementation and analysis of different architectures of sub modules needed in floating-point adder with respect to FPGA. Sections 3 and 4 go over the advanced floating-point adder algorithms, their implementation, and design trade off analysis with respect to standard algorithm. Section 5 concludes the paper and highlights its importance in terms of floating-point design applications on FPGA and future work.

2. Standard Floating-point Adder Algorithm

This section will analyze the standard floating point algorithm architecture and the hardware modules designed as part of this algorithm including their function and use. The standard architecture is the prototype algorithm for floating-point addition in any kind of hardware and software design [16].

Micro-architecture of floating-point adder is shown in Figure 1. The operation consists of three major tasks, pre-normalization, addition and post normalization. Pre-normalization consists of exponent difference and right shift. Post-normalization consists of a Leading One Detector (LOD) to detect the leading zeros in a number after addition and a left shift operation.

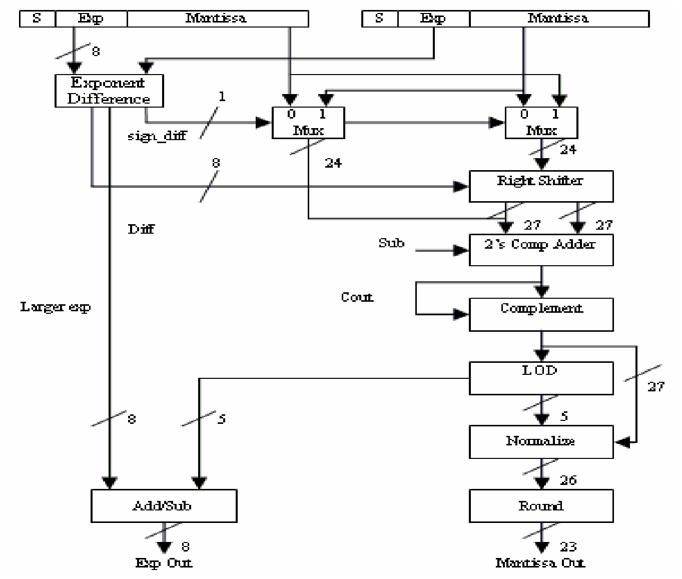

Figure 1. Micro-architecture of standard floating-point adder

All these modules add a significant delay to the overall latency of the floating-point adder. Our research was directed towards designing different available implementations of all these components separately onto a Virtex 2p FPGA device with a speed grade of 7, giving the best area latency ratio for a floating-point adder on a FPGA.

2.1. Functional Blocks

In order to compute the exponent difference, 2's complement addition, rounding, and exponent result variable width integer adders are needed. 16 bit carry look-ahead adder, carry save adder, and ripple carry adder were designed and synthesized for Virtex2p FPGA. Combinational delay and slices information were compared with the VHDL inbuilt adder function, "+." Table 1 shows the synthesis results obtained using Xilinx ISE.

Table 1. Adder implementation analysis

Adder Type	Combinational Delay (ns)	Slices
Ripple-Carry	15.91	18
Carry-Save	11.951	41
Carry-Look Ahead	9.720	39
VHDL	6.018	8

In order to pre-normalize the mantissa of the number with the smaller exponent, a right shift shifter is used to right shift the mantissa by the absolute exponent difference. Three custom shifters were designed for this purpose, for a single precision floating-point adder 24 bit mantissa acts as input, and the result is 27 bits. A typical barrel shifter was implemented using 2:1 multiplexer as its fundamental module. Another design for the right shifter named the aligned shifter was implemented using the concatenation function in VHDL and each level is implemented behaviorally. A behavioral right shift shifter was also designed using the case statements in VHDL. Table 2 shows the synthesis results obtained by using Xilinx ISE.

Table 2. Right shift shifter implementation analysis

Shifter Type	Combinational Delay (ns)	Slices
Align	10.482	71
Barrel	9.857	71
Behavioral	9.357	201

In order to post-normalize, the leading number of zeros has to be detected in the result of the adder. This amount is then used to left shift the result and normalize to obtain the mantissa out before rounding. There are a number of ways of designing a complex and complicated circuit such as LOD (Leading One Detector). LOD can be designed behaviorally and by identifying common modules which impose hierarchy on the design. Hierarchy design has low fan-in and fan-out which leads to area and delay efficient design [17], first presented by Oklobazija. Behavioral and Oklobazija type LOD were implemented and Table 3 shows the synthesis results obtained by using Xilinx ISE.

Table 3. LOD implementation analysis

LOD Type	Combinational Delay (ns)	Slices
Behavioral	9.05	20
Oklobazija	8.32	18

Using the results from the LOD, the result from the adder is left shifted to normalize the result. Table 4 gives the synthesis results obtained from Xilinx ISE implemented for Virtex2p device.

Table 4. Left shift shifter implementation analysis

LOD Type	Combinational Delay (ns)	Slices
Behavioral	8.467	80
VHDL	8.565	90

2.2. Time and Area Analysis

Using the above modules, standard floating point adder is synthesized for Virtex 2p FPGA. As the design was implemented for only one pipeline stage, the minimum clock period reported by the synthesis tool after placing and routing was 27.059ns. The levels of logic reported were 46. That means the maximum clock speed that can be achieved for this implementation is 36.95 MHz. The number of slices reported by the synthesis tool was 541.

3. LOP Algorithm

Over years, overall floating-point algorithms are researched to obtain better overall latency. One of the improvements is the Leading One Predictor (LOP) algorithm. Figure 2 shows the micro-architecture of the LOP algorithm.

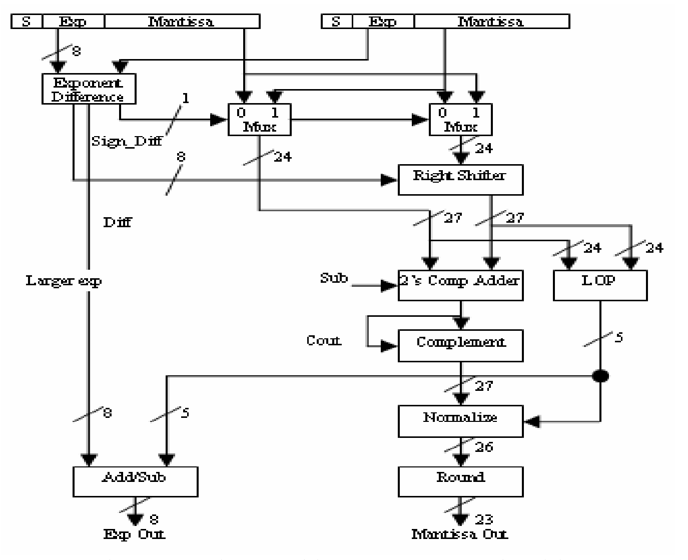

Figure 2. Micro-architecture of LOP algorithm

In this implementation, a LOP is used instead of a LOD. The main function of the module is to predict the leading number of zeros in the addition result working in parallel with the 2's complement adder. This concept was first introduced by Flynn [21] in 1991. Over years there have been number of improvements in its design and application [3, 4, 20, 22]. The most feasible design was given by J.D. Bruguera and T. Lang [3] which detects the error concurrently with the leading one detection. The design was implemented in Virtex2p FPGA and results obtained are given in Table 5.

Table 5. LOP vs. LOD implementation analysis

Module	Combinational Delay (ns)	Slices
LOP	13.6	240
LOD	8.32	18
Adder	6.893	14

In standard algorithm, LOD and adder working in parallel will have a combinational delay of 15.213 ns while the LOP offers a delay of 13.6 ns while the addition is done in parallel but as seen in the table on a added cost of large area. Using all the same modules described in Section 1, the LOP algorithm is implemented for one cycle latency and the results are summarized and compared with standard implementation in Table 6.

Table 6. Standard and LOP algorithm analysis

	Clock Period (ns)	Clock Speed (MHz)	Area (Slices)	Levels of Logic
Standard	27.059	36.95	541	46
LOP	25.325	39.48	748	35
% of imp.	+6.5%	+6.5%	-38%	+23%

The main improvement seen in LOP design is the levels of logic reduced by 23% with an added expense of increasing the area by 38%. The minimum clock period shows small improvement. Having addition in parallel with leading one detector has helped to reduce the levels of logic but this has added significant routing delay in between the LUTs as most of the added logic is in form of logic gates. This is the main reason there is not a significant improvement in latency. In VLSI design, the levels of logic effects the latency and thus LOP algorithm is a good option but for FPGA design with added area cost of 38% and just 6.5% improvement in latency in our view is not a feasible design option.

4. Far and Close Data-path Algorithm

According to the studies, 43% of floating-point instructions [20] have an exponent difference of either 0 or 1. A leading one detector or predictor is needed to count the leading number of zeros only when the effective operation is subtraction and the exponent difference is 0 or 1, for all the other cases no leading zero count is needed. Another improvement is based on compound adder which with help of additional logic does addition and rounding in one step. The main contribution towards this algorithm is presented by Oberman, Flynn, and Lang [4, 19, 20, 22].

The algorithm is potentially larger than the previously implemented algorithms but has shown significant improvement in latency and thus used in almost all the present commercial microprocessors. The micro-architecture of far and close data-path algorithm is shown in Figure 3.

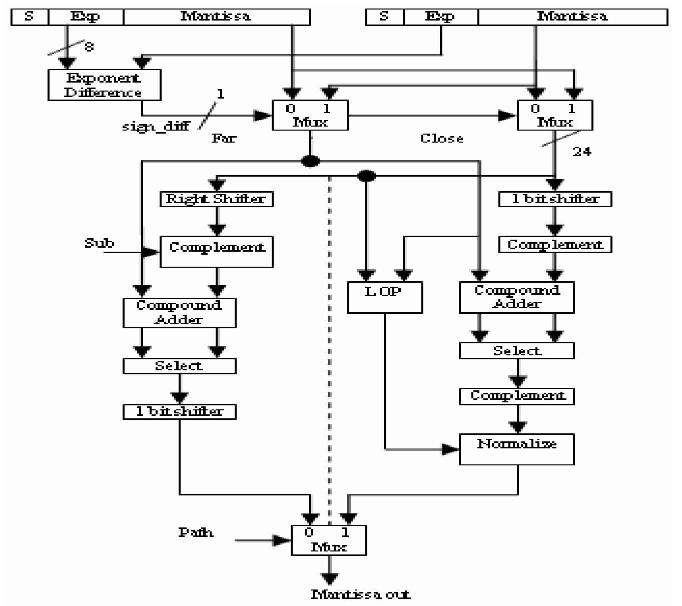

Figure 3. Micro-architecture of far and close data-path algorithm

Compared to the standard algorithm, one shifter delay and rounding delay has been removed for critical paths of data-paths with help of almost double the hardware and compound adder implementation. All the components were almost selected in Section 2 and were used to implement the far and close data-path algorithm and synthesized using Xilinx ISE. Table 7 shows a comparison between the standard algorithm and far and close data-path algorithm implementation on a Virtex 2p FPGA device.

Table 7. Standard and F&C algorithm analysis

	Clock Period (ns)	Clock Speed (MHz)	Area (Slices)	Levels of Logic
Standard	27.059	36.95	541	46
F&C	21.821	45.82	1018	30
% of imp.	+19%	+19%	-88%	+34%

The minimum clock period reported by the synthesis tool after placing and routing was 21.821ns, 19% better than that of standard floating-point adder implementation. The levels of logic reported were 30 showing 34% improvement. Both these improvements were evident because in both the critical paths one shifter and one adder has been removed. The number of slices reported by the synthesis tool was 1018 which is a significant increase compared to standard algorithm.

5. Conclusion

In this paper, we have shown a detail design tradeoff analysis of floating-point adder with respect to basic sub components of standard adder algorithm along with the overall architectural improvements in Virtex2p FPGA. Standard algorithm implementation is analyzed and compared with LOP and far and close data-path algorithm implementation. Standard algorithm is area efficient but has larger delays in levels of logic and overall latency. LOP algorithm adds parallelism to the design thus reduces levels of logic significantly but due to added hardware and significant routing delays doesn't effect overall latency in FPGAs. Far and close data-path algorithm reduces overall latency significantly by distributing the operations into two separate paths thus reducing critical paths and effecting latency significantly but on the cost of added hardware. The main objective of this research was to develop a design resource for designers to implement floating-point adder onto FPGA. This kind of work has not been done before and we believe it will be a great help in custom implementation and design of floating-point adders on FPGAs.

Acknowledgements

The authors would like to acknowledge the Natural Science and Engineering Research Council of Canada (NSERC) for its support to this work.

References

[1] M. Farmland, "On the Design of High Performance Digital Arithmetic Units," PhD thesis, Stanford University, Department of Electrical Engineering, August 1981.

[2] P. M Seidel, G. Even," Delay-Optimization Implementation of IEEE Floating-Point Addition," IEEE transactions on computers, vol. 53, no. 2, pp. 97-113, February 2004.

[3] J. D. Bruguera and T. Lang, "Leading-One Prediction with Concurrent Position Correction," IEEE Transactions on Computers, 48(10), pp 1083–1097, 1999.

[4] S.F. Oberman, H. Al-Twaijry and M.J. Flynn, "The SNAP Project: Design of Floating-Point Arithmetic Units'" Proc. 13th IEEE Symp. on Computer Arithmetic, pp. 156-165, 1997.

[5] Xilinx, http://www.xlinix/com.

[6] L. Louca, T. A. Cook, W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," FPGAs for Custom Computing, 1996.

[7] W. B. Ligon, S. McMillan, G. Monn, F. Stivers, and K. D. Underwood, "A Re-evaluation of the Practicality of Floating-point Operations on FPGAs," IEEE Symp. On Field-Programmable Custom Computing Machines, pp 206–215, April 1998.

[8] E. Roesler, B. E. Nelson, "Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture," Field-Programmable Logic and Applications, pp 637-646, September 2002.

[9] J. Liang, R. Tessier and O. Mencer, "Floating Point Unit Generation and Evaluation for FPGAs," IEEE Symp. on Field-Programmable Custom Computing Machines, pp 185-194, April 2003.

[10] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of High-Performance Floating-Point Arithmetic on FPGAs," International Parallel and Distributed Processing Symp., pp 149b, April 2004.

[11] A. Malik and S. Ko, "Efficient Implementation of Floating Point Adder using pipelined LOP in FPGAs," IEEE Canadian Conference on Electrical and Computer Engineering, pp 688-691, May 2005.

[12] Digital Core Design, http://www.dcd.pl/

[13] Quixilica, http://www.quixilica.com/

[14] Nallatech, http://www.nallatech.com/

[15] IEEE Standard Board and ANSI, "IEEE Standard for Binary Floating-Point Arithmetic," 1985, IEEE Std 754-1985.

[16] J. Hennessy and D. A. Peterson, Computer Architecture a Quantitative Approach, Morgan Kauffman Publishers, Second edition, 1996.

[17] V.G. Oklobdzija, "An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 2, No. 1, pp. 124-128, 1994.

[18] Israel Koren, Computer Arithmetic Algorithms, A K Peters, Natick MA, 2002.

[19] J.D. Bruguera and T. Lang, "Rounding in Floating-Point Addition using a Compound Adder," University of Santiago de Compostela, Spain Internal Report, July 2000.

[20] M. J. Flynn, S. F. Oberman, Advanced Computer Arithmetic Design. John Wiley & Sons, Inc. 2001.

[21] M. Flynn, "Leading One Prediction -- Implementation, generalization, and application," Technical Report: CSL-TR-91-463, March 1991.

[22] S. F. Oberman, "Design Issues in High performance floating-point arithmetic units," Technical Report: CSL-TR-96-711, December 1996.