High Performance FPGA Implementation of Double Precision Floating Point Adder/Subtractor

Manish Kumar Jaiswal[1] and Ray C.C. Cheung[2]
Department of Electronic Engineering, City University of Hong Kong
[1]*mkjaiswal2@student.cityu.edu.hk,* [2]*r.cheung@cityu.edu.hk*

Abstract

Floating Point (FP) arithmetic is widely used in large set of scientific and signal processing computation. Adder/subtractor is one of the common arithmetic operation in these computation. The design of FP adder/subtractor is relatively complex than other FP arithmetic operations. This paper has shown an efficient implementation of adder/subtractor module on a reconfigurable platform, which is both area as well as performance optimal. The proposed design has optimized the individual complex components of adder module (like dynamic shifter, leading one detector (LOD), priority encoder), to achieve the better overall implementation. Comparison with the best reported work has been shown in the paper, which proves the merits of proposed design.

Keywords: *Floating Point, Adder/Subtractor, FPGA, Dynamic Shifter, LOD, Rounding, Normalization.*

1: Introduction

Numbers represented in floating point can represent a much larger range than possible using the same number of bits in fixed point format. This makes floating point a natural choice for scientific and signal processing computations.

Hardware implementation of floating point arithmetic is more complex than for fixed point numbers, and this puts a performance limit on several of these applications. Over the past two decades lots of work has been dedicated to performance improvement of floating point computations, both at algorithmic level and implementation level. Several works also focused on their implementation on FPGA platforms. In spite of tremendous effort, they are still often the bottleneck in many computations. Keeping this in mind this work is dedicated to improved implementation of one of these arithmetic units, adder/subtractor, on FPGA. Several works are available in the literature, for the high performance implementation of floating point adder/subtractor unit. [1], [2] has shown the better implementation of dynamic shifter and leading-one-detector respectively to have a better overall adder architecture. [3], [4], [5] and [6] have also presented their adder architecture and claimed them to be better than previous versions of the adder implementation on FPGA platform.

Recent advances in FPGA technology made it a good choice for speeding up many computationally intensive scientific and engineering applications. Recent FPGAs have large amounts of programmable logic, memory and often come with a large set of high speed intellectual property (IP) cores to implement specific functions. The continuous increase in logic density, working speed, memory bandwidth, and wide range of IP cores (DSP cores, RAM/FIFO, high performance I/O, etc.) are increasing their suitability for numerical processing. Some recently developed reconfigurable supercomputers are using FPGA in their architecture, used for off-loading certain tasks

on them. Some examples of these are - SRC MAPstation [7] (uses Stratix II EP2S180 FPGA from Altera Corp.), SGI RASC (Reconfigurable Application Specific Computing) [8] (using Xilinx Virtex-4 high-performance FPGAs) and Cray XD1 [9] (used Xilinx Virtex-II Pro FPGAs).

Thus, FPGAs are becoming highly competitive to general purpose processors in a large range of applications. Few of these areas are scientific computing [10, 11, 12], image processing [13, 14], communications [15, 16], cryptography [17, 18, 19], etc. In view of wide applicability of FPGA, this work has also focused on a FPGA based hardware acceleration.

Many of the scientific applications described above rely on floating point (FP) computation, often requiring the use of the double precision (D.P.) format specified by the IEEE standard 754 [20, 21]. The use of D.P. data type improves the accuracy and dynamic range of the computation, but simultaneously it increases the complexity and performance of the arithmetical computation of the module. The design of high performance floating point units (FPUs) is thus of interest in this domain.

2: Background

In general, floating point arithmetic implementations involve processing separately the sign, exponent and mantissa parts, and then combining them after rounding and normalization. The basic algorithms for the computation of these operations has been discussed in this Section. IEEE (Institute of Electrical and Electronics Engineers) standard for floating point (IEEE-754) defines a binary format used for representing floating point numbers [20, 21]. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented. Some aspects of how certain arithmetic operations are to be carried out are also defined in this standard. [22] contains a lot of useful information on floating point arithmetic and implementation.

FP Addition is one of the complex unit in the floating point arithmetic operations. Addition (and the related operation of subtraction) is the most basic arithmetic operation. The hardware implementation of this arithmetic for floating point numbers is a complicated operation due to the normalization requirements. An implementation of double precision floating point adder/subtractor has been shown here. The flowchart for FPU adder/subtractor is shown in Fig. 1. For the remainder of the discussion, the term *addition* is used to refer to both *addition* and *subtraction* as the same hardware is used in both cases.

The steps for computing addition of two floating point numbers proceeds as follows,

1. Compare exponents and mantissa of both numbers. Decide large exponent & mantissa and small exponent & mantissa.
2. Right shift the mantissa associated with the smaller exponent, by the difference of exponents.
3. Add both mantissa if signs are same else subtract smaller mantissa from large one.
4. Do the rounding of the result after mantissa addition.
5. If the subtraction results in loss of most significant bit (MSB), then the result must be normalized. To do this, the most significant non-zero entry in the result mantissa must be shifted until it reaches the front. This is accomplished by a "Leading one detector (LOD)" followed by a shift.
6. Do normalization and adjust large exponent accordingly.
7. Final result includes sign of larger number, normalized exponent and mantissa.

For subtraction, first negate the sign of second operand and do the addition operation.

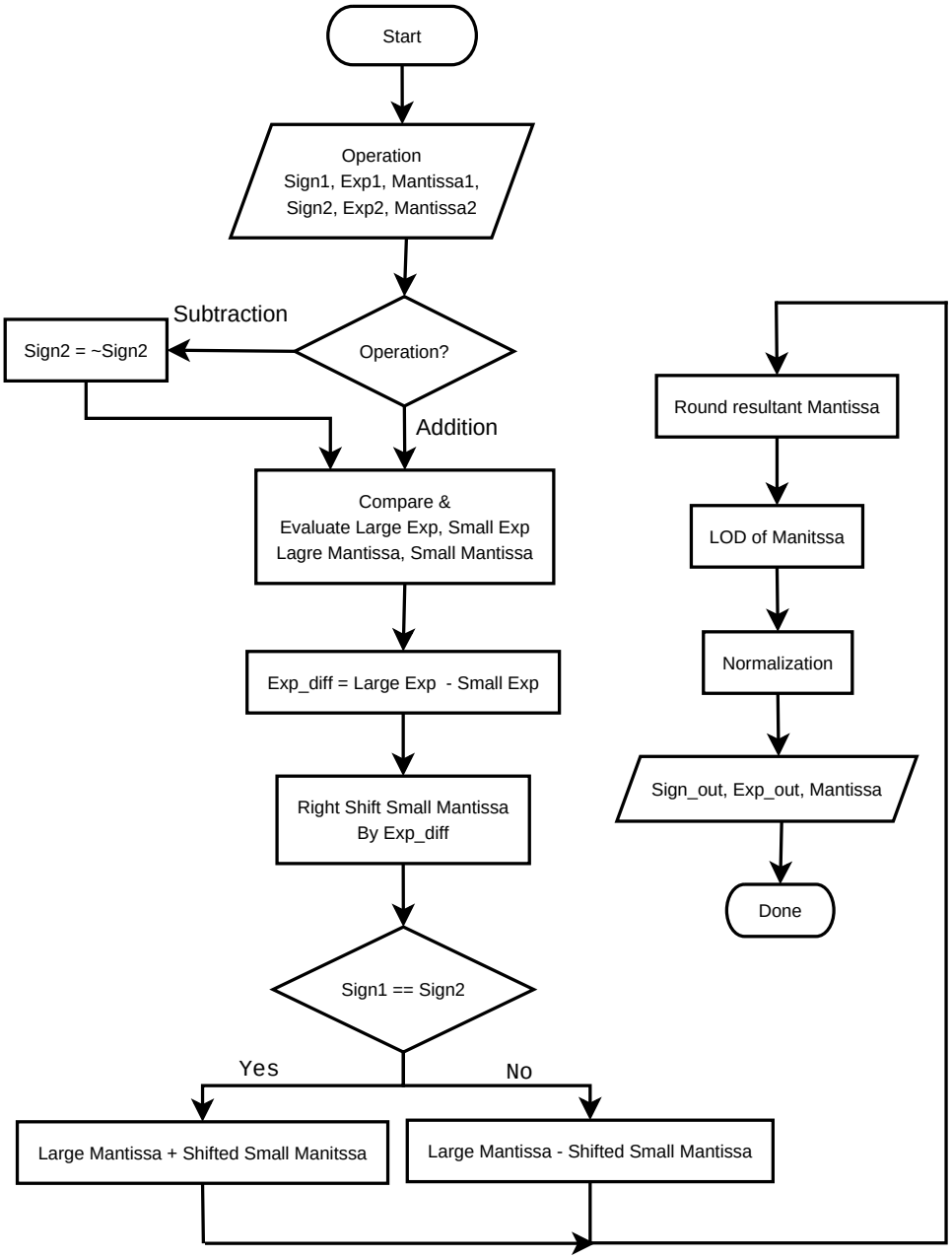

Figure 1. Flowchart for FPU Adder/Subtractor

3: Adder/Subtractor Design

This design is mainly focused on the optimization of the performance (speed) and uses the basic algorithm described earlier. Implementation has been done for both normalized and denormalized numbers. All exceptional cases have been detected, both in the inputs and output result. The design includes only "round to nearest" mode for rounding. The implementation has been made for a range of latency with a speed-area trade-off.

High speed operation was achieved through balanced pipe-lining of the algorithm for various

latencies and FPGA specific design of sub-units used in the adder computation. For example, for the purpose of mantissa addition, the carry select adder has been used. This negatively impacts area at lower latency, but gives better performance. At higher latency the hardware impact will also reduce. Other important sub-units are dynamic shifter and leading one detector (LOD), which usually put performance limits on entire circuit. Rounding to nearest value has been implemented and the final result is normalized.

3.1: Dynamic Shifter Design

This unit plays an important role in adder design. It is used at two places. First, after detecting the larger input operand, the other mantissa needs to be shifted right by the difference of exponents, to make both exponents equal for further addition of both mantissas. Second instance of its use occurs after LOD step, to left shift resultant mantissa by the place of leading one, to make the result normalized. As the amount of shifting is a variable parameter it is a complex hardware unit.

In present implementation we have used 60-bit extended format for mantissa addition. (The reason behind the use of this 60-bit extended format is to reduce the amount the rounding errors due to any possible shifting (left/right) of the mantissa.) Thus it require maximum of 60-bit left or right shifting of mantissa at both places. The design of this unit proceed as follow. A 6-bit positive binary number can represent up to 63, and hence is suitable for representing amount of shifting. The design uses a logarithmic shifting, where each bit of a binary number represents a specific amount of shift, and by taking the bits one by one complete shifting can be achieved. The architecture for dynamic shifter is shown in Fig. 2. The circuit is very simple and can be easily pipe-lined at several depth for better performance, and the pipe-lined version has been used for higher latencies.

A comparison with [1] is shown in Table 1 for a 24-bit dynamic shifter. [1] has implemented this dynamic shifter for 24-bit input, in a fully combinational fashion on a Virtex-IIpro FPGA, using three methods, and presented a comparative study. A comparison with Xilinx barrel shifter [23] is also shown in Table 2. In [23] a 32-bit barrel shifter has been shown using multiplier IP cores and logics.

Table 1. Dynamic Shifter Comparison with [1]

Shifter	Delay (ns)	Slices
Align [1]	10.482	71
Barrel [1]	9.857	71
Behavioral [1]	9.357	201
Proposed	3.229	66

Table 2. Dynamic Shifter Comparison with Xilinx Barrel Shifter [23]

Shifter	Delay (ns)	Slices	MULT18x18
Xilinx [23]	3.984	39	4
Proposed	3.125	77	0

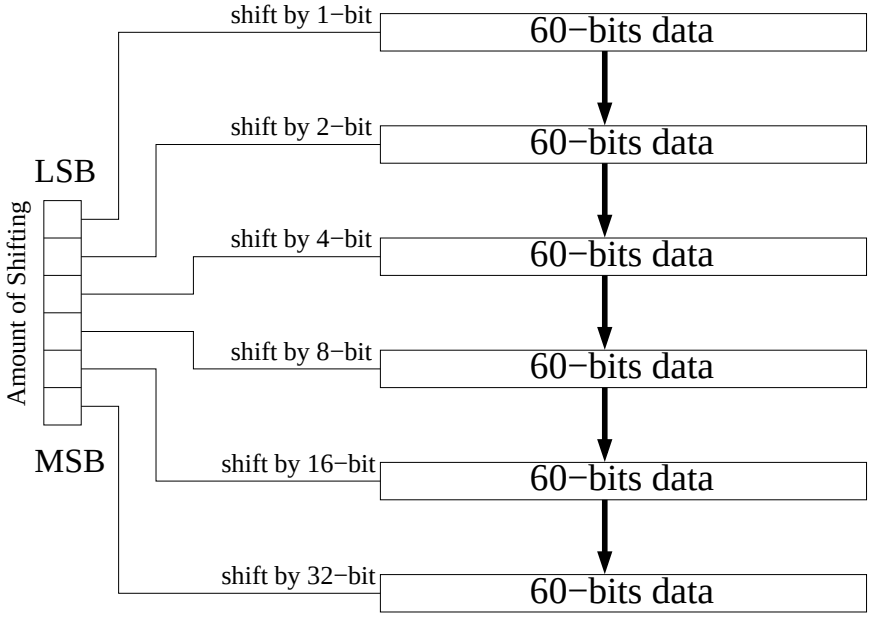

Figure 2. Dynamic Shifter for FPU Adder/Subtractor

3.2: LOD Design

Leading One Detector (LOD) is basically a priority encoder. It is required when the result of mantissa subtraction has one or more zeros at the most significant bit locations. This step is necessary to bring back the final result in normalized format. The result mantissa is priority encoded (to detect the position of the leading one) and left shifted so that the leading one becomes the MSB. This requires a 52-bit priority encoder. The direct design of a 52-bit priority encoder will have a negative impact on both area and speed. In present design it is implemented in a multi-level priority encoder fashion, and makes use of FPGA specific LUTs. For a Virtex-II pro FPGA, the 52 bits are broken up in to 13 sets of 4-bit numbers. These 4-bit numbers are encoded by a 4:3 priority encoder using 3 LUTs, and these are then finally encoded by a 13:1 priority encoder. The final result is given by the addition of an offset to encoded value. The value of offset is one of (0,4,8,12,16,20,24,28,32,36,40,44,48) and is selected by 13:1 priority encoder outputs.

A comparison with fully combinational implementation of 24-bits LOD of [1] and [2] on a Virtex-IIpro platform has been shown in Table 3. For this comparison purpose we have also designed our module for 24-bits LOD. Proposed design has better performance but at the expense of more area.

Table 3. LOD Comparison

Shifter	Delay (ns)	Slices
[1]	9.05	20
[2]	8.32	18
Proposed	4.321	29

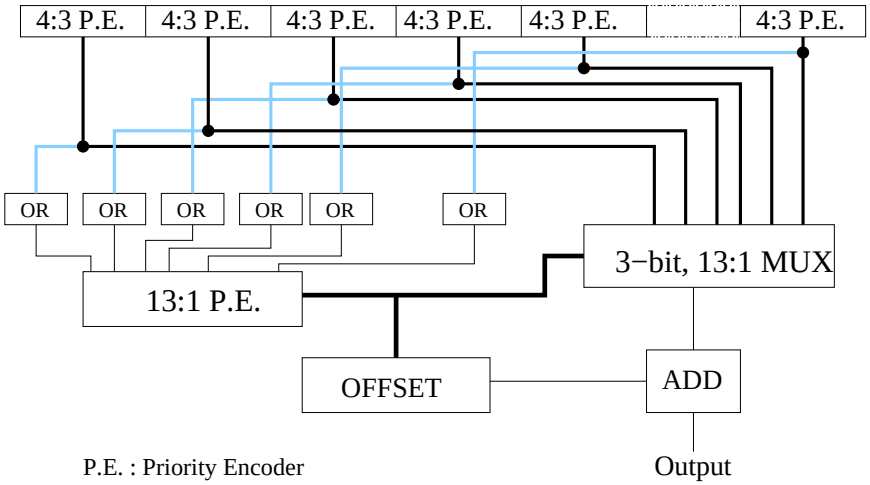

Figure 3. LOD for FPU Adder/Subtractor

3.3: Sign and Exponent Computation

The sign bit of resultant floating point number will be given by sign of the larger input operand and similarly exponent will also given by larger operand exponent, which further needs to be normalized.

3.4: Normalization and Rounding

Normalization is the step used to bring denormalized results in IEEE normalized format. When, after addition of mantissas an extra carry is generated, the exponent needs to be increased by one, along with right shifting of the addition result by one-bit. Similarly, when after subtraction of mantissa, the difference loses the MSB, it needs to be checked by the LOD, and the mantissa shifted by that amount, along with a decrease in exponent. Only rounding to nearest has been implemented in this design.

3.5: Exceptional Case Handling

The exceptional case handling in the proposed design is not fully compatible to the IEEE standard. The execution of all the exceptional cases are handled similarly to the Xilinx Core [24]. For example, if any/both of the operands is infinite we produce a `NaN` as output (with usual sign-bit). If output exponent goes `ZERO` or below `ZERO`, `UNDERFLOW` will activate, and if it goes beyond `11'h7fe` (2046 in decimal), `OVERFLOW` will activate. Entire execution is show in algorithm 1.

In addition, when one operand is infinite and other is denormalized, an `INVALID` operation is indicated (algorithm 2).

All other computation steps are as per Fig. 1 and the whole design is pipe-lined separately for each latency. The different latencies have been achieved mainly by pipe-lining the mantissa addition, dynamic shifting and LOD units.

Algorithm 1 Exceptional Case Handling

```
if any operands infinite then
    EXP_OUT = 11'h7ff;
    MANT_OUT = 52'h8000000000000;
else if output exponent ≤ ZERO then
    EXP_OUT = 0;
    MANT_OUT = 0;
    UNDERFLOW = 1;
else if output exponent > 11'h7fe then
    EXP_OUT = 11'h7ff;
    MANT_OUT = 0;
    OVERFLOW = 1;
else
    EXP_OUT = estimated  exponent;
    MANT_OUT = estimated  mantissa;
    UNDERFLOW = 0;
    OVERFLOW = 0;
end if
```

Algorithm 2 Invalid Operation

```
if one operand is infinite and other is denormalized then
    INVALID_OPERATION = 1;
else
    INVALID_OPERATION = 0;
end if
```

4: Implementation Details

The hardware and performance result for the adder is shown in Table 4 on Virtex-5 FPGA. Implementation results are shown for the proposed module and Xilinx IP core [24]. The proposed design has been implemented for various latencies. The data for Xilinx Core adder has also been shown for various available latencies, to have better idea of proposed design. The proposed design is taking approximately same hardware (in terms of number of LUT's and FF's count) as of Xilinx module, but have better performance speed with similar latencies. The proposed design is achieving a speed of 353 MHz than 284 MHz for Xilinx core for a latency of 12, which shows a significant performance improvement in the proposed design.

Table 4. FPU Adder/Subtractor on Virtex-5 (Speed Grade -3)

Parameters	Xilinx-Core			Proposed-Module			
Latency	4	8	12	4	8	12	16
LUT's	1262	1297	1266	1214	1260	1397	1384
FF's	367	672	1040	390	733	1013	993
Freq(MHz)	150	184	284	205	287	353	407

5: Result Comparison & Discussion

In this section, the comparison with some of the best previously reported work [4, 3, 5, 24] has been shown in Table 5. All of these reported work supports only normalized operands. Since the majority of earlier reported works had shown their implementation on Virtex-IIpro FPGA platform, for better comparison, we have also used the same FPGA platform for comparison. The area occupancy of proposed design is slightly larger than Xilinx [24] and Sandia [3] designs, but it has achieved a higher maximum frequency of operation (276 MHz on Virtex-IIpro). Main reason for this area difference is that these design does not support denormalized operands, and other reason appears to be, because these designs are hand optimized for the given platform, whereas the proposed design is a generic one & support denormalized number as well and can be ported on any FPGA platform. Whereas, if we consider the Virtex-5 FPGA platform the area occupied by proposed design is also less than the hand optimized Xilinx [24] design, and is working at relatively very high speed of operation (407 MHz). Thus, the proposed design showing a better relative area occupancy and speed of operation.

Table 5. FPU Adder Comparison on Virtex-II Pro FPGA (Speed Grade -7)

Method	Latency	Slices	Freq (MHz)
USC [4]	19 (Opti.)	933	200
USC [4]	21 (Max.)	1133	220
USC [4]	6 (Min.)	633	50
Sandia [3]	10	571	228
NEU [4, 5, 6]	8	770	54
Xilinx	12 (Max.)	753	224
Proposed	12	939	250
Proposed	16 (Max.)	936	276

6: Conclusion

This paper has shown an efficient high performance implementation of adder/subtractor unit on FPGA, a reconfigurable platform, for double precision floating point numbers. The adder module has been designed by optimizing the building blocks of the algorithmic flow. The design has achieved a high performance by balance pipeline of the architecture for different latencies. The proposed module gives excellent performance with efficient uses of resources. Design is fully compatible with the IEEE standard precision, and has shown better performance in comparisons with the reported work in the literature.

References

[1] A. Malik and S.-B. Ko, "A Study on the Floating-Point Adder in FPGAs," in *Canadian Conference on Electrical and Computer Engineering (CCECE-06)*, May 2006, pp. 86–89.

[2] V. 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, Mar 1994.

[3] K. Hemmert and K. Underwood, "Open Source High Performance Floating-Point Modules," in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-06)*, April 2006, pp. 349–350.

[4] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs," in *Proceedings of 18th International Parallel and Distributed Processing Symposium*, April 2004, pp. 149–156.

[5] P. Belanovic and M. Leeser, "A Library of Parameterized Floating-Point Modules and Their Use," in *12th International Conference on Field-Programmable Logic and Applications (FPL-02)*. London, UK: Springer-Verlag, Sep. 2002, pp. 657–666.

[6] M. Leeser, "Vfloat: The northeastern variable precision floating point library," *http://www.ece.neu.edu/groups/rpl/projects/floatingpoint/*, Feb 2008.

[7] "SRC Supercomputers," 2008. [Online]. Available: http://www.srccomp.com/

[8] "SGI Supercomputers." [Online]. Available: http://www.sgi.com/

[9] "Cray XD1 Supercomputers," 2008. [Online]. Available: http://www.cray.com/

[10] O. Storaasli, R. C. Singleterry, and S. Brown, "Scientific Computation on a NASA Reconfigurable Hypercomputer," Sept. 2002.

[11] K. Underwood and K. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-2004)*, April 2004, pp. 219–228.

[12] M. Smith, J. Vetter, and X. Liang, "Accelerating Scientific Applications with the SRC-6 Reconfigurable Computer: Methodologies and Analysis," in *Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium*, April 2005, pp. 157b–157b.

[13] G. Zhi, N. Walid, V. Frank, and V. Kees, "A quantitative analysis of the speedup factors of FPGAs over processors," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays (FPGA-04)*. New York, NY, USA: ACM, 2004, pp. 162–170.

[14] V. Aggarwal, A. D. George, and K. C. Slatton, "Reconfigurable computing with multiscale data fusion for remote sensing," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays (FPGA-06)*. New York, NY, USA: ACM, 2006, pp. 235–235.

[15] H. Parizi, A. Niktash, A. Kamalizad, and N. Bagherzadeh, "A Reconfigurable Architecture for Wireless Communication Systems," *Third International Conference on Information Technology: New Generations*, pp. 250–255, 2006.

[16] K. A., N. Bagherzadeh, and C. Pan, "A fast parallel Reed-Solomon decoder on a reconfigurable architecture," in *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2003, pp. 59–64.

[17] W. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, Feb. 2008.

[18] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA implementation of high performance elliptic curve cryptographic processor over GF(2^{163})," *J. Syst. Archit.*, vol. 54, no. 10, pp. 893–900, 2008.

[19] H. M. Choi, C. P. Hong, and C. H. Kim, "High Performance Elliptic Curve Cryptographic Processor Over GF(2^{163})," *IEEE International Workshop on Electronic Design, Test and Applications*, pp. 290–295, 2008.

[20] IEEE, "IEEE standard for binary floating-point arithmetic," *ANSI/IEEE Std 754-1985*, Aug 1985.

[21] ——, "IEEE standard Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–58, Aug 2008.

[22] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, 1991.

[23] P. Gigliotti, "Xilinx Application Notes, XAPP195-Implementing Barrel Shifters Using Multipliers ," Tech. Rep., Aug. 2004. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp195.pdf

[24] Xilinx, "Xilinx Floating-Point IP Core." [Online]. Available: http://www.xilinx.com

Authors

Manish Kumar Jaiswal received his B.Sc. and M.Sc. in Electronics in 2002 and 2004 respectively from from D.D.U. Gorakhpur University, Gorakhpur, India. He completed his M.S. (By Research) in VLSI Design in 2009 from the Indian Institute of Technology, Madras, India. He has worked as an Assistant Professor, for two years, in The ICFAI University, Dehradun, India. Currently, he is pursuing his PhD studies at City University of Hong Kong. His research interest includes Digital VLSI Design, Reconfigurable Computing, VLSI Implementation of DSP, High-Performance Algorithmic Synthesis, Hardware Synthesis of ANN.

Ray C.C. Cheung (M'07) received the B.Eng. and M.Phil. degrees in computer engineering and computer science and engineering at the Chinese University of Hong Kong (CUHK), Hong Kong, in 1999 and 2001, respectively, and the Ph.D. degree in computing at Imperial College London, London, U.K., in 2007. In 2009, he worked as a visiting research fellow in the Department of Electrical Engineering, Princeton University. He is currently an assistant professor at City University of Hong Kong (CityU). His research team, CityU Architecture Lab for Arithmetic and Security (CALAS) focuses on the following research topics: reconfigurable trusted computing, SoC VLSI designs, cryptography, and embedded biomedical VLSI designs.