2011 International Conference on Advancements in Information Technology
With workshop of ICBMG 2011
IPCSIT vol.20 (2011) © (2011) IACSIT Press, Singapore

Design and Optimized Implementation of Six-Operand Single-Precision Floating-Point Addition

N.Nalla Anandakumar [1 +] and Suganya Annadurai [2]

[1] Research Associate, Hardware Security Research Group, Society for Electronic Transactions and Security (SETS), Chennai, India.

[2] Senior Research Associate, Hardware Security Research Group, SETS, Chennai, India.

Abstract. In this paper we present an efficient design of generalized, multi-operand single-precision floating point addition. The addition operation is optimized in two ways. Firstly, the design of [4] has been modified to improve efficiency of multi-operand floating point addition in terms of speed and area. Secondly, varieties of adders were analyzed for parallel addition and identified suitable adder. Our approach is applied to six operand floating point addition to compare with the related works. The implementation is synthesized and verified on Xilinx Virtex4 FPGA. Further, the performance of proposed design was compared in terms of speed and resources with other designs, the comparison results are also presented in this paper.

Keywords: Floating point addition, FPGA.

1. Introduction

Floating point arithmetic is largely used in high speed signal processing, network security applications, search engines, data mining algorithms, etc. Floating point arithmetic is generally considered an expensive operation, in terms of area and time and it impacts overall performance of the application. In this work we present an efficient method to implement six-operand floating point addition in terms of hardware resources (area and clock speed).

1.1. Our Contribution

In the recent literature, design of networked and parallel three operands addition is presented in [4]. In this paper, better accuracy was achieved compared to three-operand addition implemented based on two IEEE floating-point adder; however, this approach requires the precision of the effective operation to increase from f to 2f+5 bits; this approach[4] requires a linear increase in number of precision bits for addition of higher number of operands. The proposed design uses the constant number of precision bits for any number of operands, without affecting the accuracy of result.

Efficient parallel adder is also analysed by employing various adders in the design. Finally, all design approaches were implemented including work of [4] and tree of two operand addition, with Six-operands targeting the same platform, so that the performance of our design can be compared with other related works effectively. The design is implemented in VHDL (Very High Speed Integrated Circuits Hardware Description Language) and targeted to Virtex4 FPGA.

The paper is organized as follows: Section 2 explains the representation of Single precision Floating point number by IEEE 754 standard. Section 3 explains the proposed design of Six-operand floating point addition and experimental results are given in Section 4. Finally, conclusion and future works are discussed.

[+] Corresponding Author : N.Nalla Anandakumar. Tel.: + 91-44-66632517; fax: + 91-44-66632501
E-mail address: nallananth@setsindia.net.

2. Floating Point Number Representation

The IEEE (Institute of Electrical and Electronics Engineers) has produced a standard for floating point arithmetic [1]. This standard specifies the standard method to represent single precision (32 bit) and double precision (64 bit) floating point numbers, as well as procedures to perform two operand arithmetic operations such as addition, subtraction, multiplication, division and etc.

The IEEE single precision floating point standard representation requires a 32 bit word, which is represented in bits position as 0 to 31, left to right. The first bit is the sign bit, 'S', the next eight bits are the exponent bits, 'E', and the final 23 bits are the fraction 'M', also called as mantissa. The bits are laid out for single-precision is shown in Table 1

0	1	...	8	9	...	31
S	E7	...	E0	M22	...	M0

Table. 1: Bits laid out for single-precision floating point number.

The value V represented by the word may be determined as follows:

- If E = 255 and M is nonzero, then V = NaN ("Not a number")
- If E = 255 and M is zero and S is 1, then V = – Infinity
- If E = 255 and M is zero and S is 0, then V = Infinity
- If $0 < E < 255$ then $V = (-1)^S \times 2^{E-127\times} 1.M$ (1)

  Where "1.M" is intended to represent the binary number created by prefixing M with an implicit leading 1 and a binary point. These are "normalized" values.
- If E = 0 and M is nonzero, then $V = (-1)^S \times 2^{-126} \times 0.M$ (2)

  These are "denormalized" values.
- If E = 0 and M is zero and S is 1, then V = – 0
- If E = 0 and M is zero and S is 0, then V = 0

The range of numbers that can be represented in single precession normalized floating point representation is $\pm 2^{-126}$ to $(2 - 2^{-23}) \times 2^{127}$ and denormalized floating point representation is $\pm 2^{-149}$ to $(1 - 2^{-23}) \times 2^{-126}$.

3. Six-Operand Floating Point Addition

Floating point operations are comparatively more complex than fixed point operations. Lots of research has been (and still is being) employed in improving the efficiency of floating point arithmetic operation. In particular, two operand additions are well studied and efficient designs are published [2, 3, 5, 6 and 7], however multi-operand addition is still researched to improve its efficiency in terms of area and speed. In multi-operand floating point addition, there are six major steps involved. Each step is discussed in detail in the following:

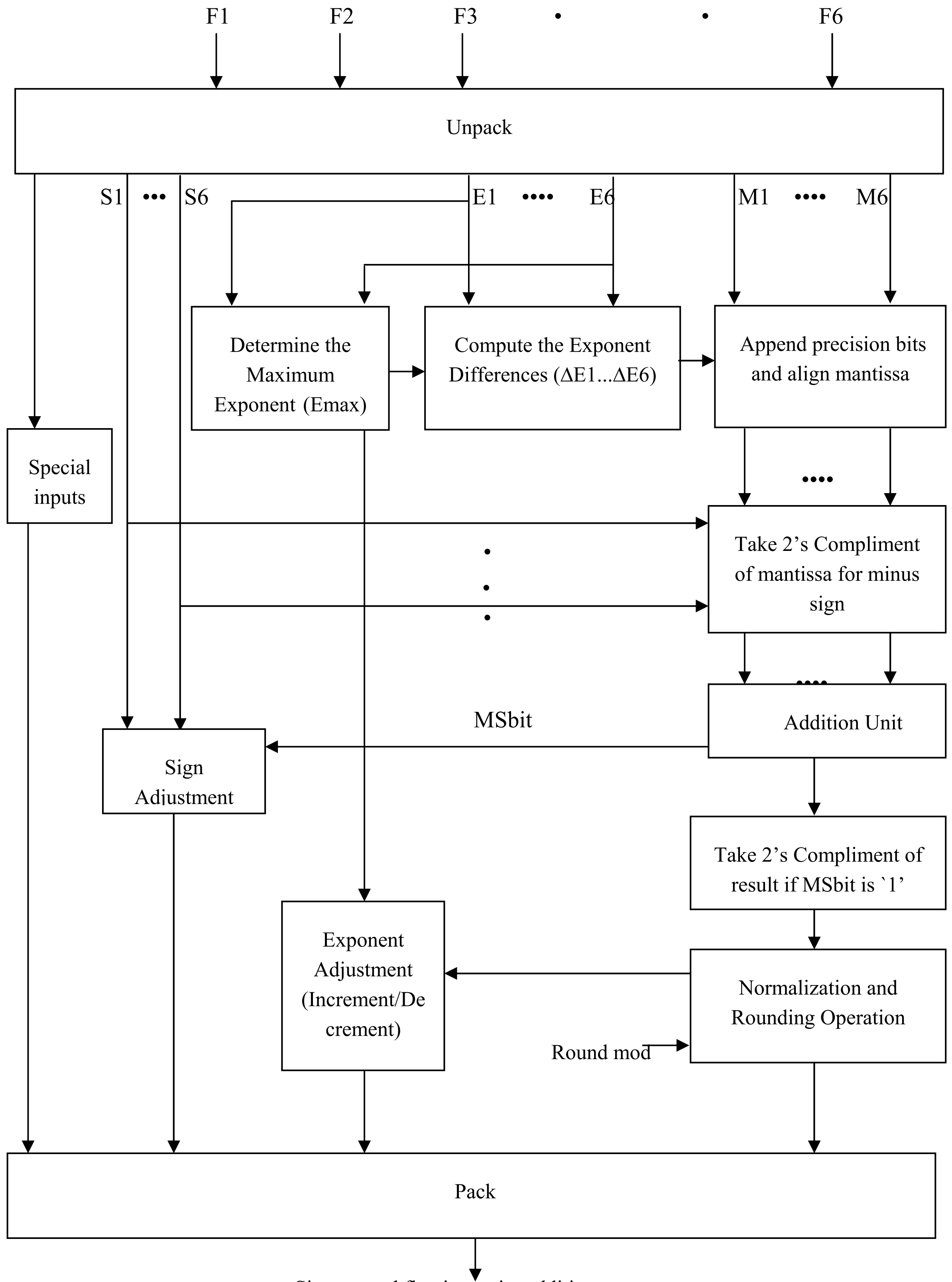
3.1. Find Maximum Exponent

Unpack each floating point operand (F_1 , F_2 , F_3 , F_4, F_5 , F_6) to obtain the sign (S_1 , S_2 , S_3, S_4, S_5, S_6), exponent(E_1 , E_2 , E_3 ,E_4 ,E_5 , E_6) and mantissa (M_1 , M_2 , M_3, M_4, M_5, M_6) bits separately. Determine the maximum exponent Emax = max (E_1, E_2, E_3, E_4, E_5, E_6) and compute the exponent differences $\Delta E_{1\text{-}6} = E_{max} - E_{1\text{-}6}$. Complexity of finding maximum exponent increases linearly when the number of operands increases. It impacts the throughput of floating point addition.

3.2. Extend and Align Mantissa Bits

Three extra bits namely guard bit, round bit, and sticky bit are appended to LSB of all mantissas to increase the accuracy of the result. These three bits are used in the rounding process. Sticky bit is the logical OR ing of any bits that are dropped during shifting of mantissa with exponent difference $\Delta E_{1\text{-}6}$ and effective parallel addition. These three bits are provided for underflow.

- One hidden bit is appended to MSB of mantissa. Hidden bit '1' is to be added if operand is normalized else hidden bit '0' is to be added if operand is de-normalised.



F1
F2
F3
F6
Unpack
S1
S6
E1
E6
M1
M6
Determine the Maximum Exponent (Emax)
Compute the Exponent Differences (ΔE1...ΔE6)
Append precision bits and align mantissa
Special inputs
Take 2's Compliment of mantissa for minus sign
Addition Unit
MSbit
Sign Adjustment
Take 2's Compliment of result if MSbit is `1'
Exponent Adjustment (Increment/Decrement)
Normalization and Rounding Operation
Round mod
Pack

Six-operand floating-point addition output

Fig. 1: Block diagram for the Six-operand floating point addition.

• In addition to the underflow bits and one hidden bit, additional bits are provided for overflow, which is appended to MSB of each mantissa. For example if number of operand is Six, then overflow bits (2), hidden bit (1), mantissa bits(23), underflow bits(3).

• Shift all aligned mantissas with exponent difference ΔEi to the right, so that all operands will have the same exponents.

3.3. Sign Magnitude Operation

In our approach, we have addressed floating point addition of mixed sign operand and uni-signed operand.

In mixed sign floating point addition:

• For negative operands, two's compliment is to be taken for extended mantissa and one sign bit '1' is to be appended to MSB of extended mantissa

• For positive operands, one sign bit '0' is to be appended to MSB of extended mantissa.

Uni-signed operands may have all positive sign or all negative sign.

• In case of all negative sign, mantissas will not undergo two's compliment conversion; all mantissas will be treated as positive operands. However, the resultant sign bit will be assigned as '1'.

• In case of all positive sign, the resultant sign bit will be assigned as '0'.

3.4. Effective Parallel Addition Operation

We have analysed the efficient parallel adder by implementing three varieties of adder, namely carry save adder, carry lookahead adder and ripple carry adder. Experimental result is discussed in the next section. Parallel additions of extended mantissas generate resultant mantissa M_o.

• If the MSB of M_o is '1', then the Mo requires two's compliment conversion and the resultant sign bit S_o will be assigned as '1'.

• If the MSB of M_o is '0', then the resultant sign bit will be assigned as '0'.

3.5. Perform Normalization

A normalization stage occurs after the parallel addition. Normalisation process is based on 3 bits of most significant precision bits of M_o.

• Identify the position of leading one on 3 bits of MSB bits of M_o is denoted as Q.

• Count the number of leading zeros C after Q.

• Left shift M_o by C, and decrement E_{max} by C, if Q is zero.

• Right shift M_o by Q, and increment E_{max} by Q, if Q is non-zero.

3.6. Rounding Operation

The IEEE standard has four different rounding modes[1] namely round to nearest value, round towards zero, round towards positive infinity and round towards negative infinity.

We have implemented all rounding modes. The addition operation uses any one of the rounding mode based on the input, which is received along with operands. Based on the mode selected and resultant underflow bits (guard, round and sticky bits), M_o is altered. If the alteration causes overflow in Q value, then M_o is shifted by one and the E_{max} is incremented. Extract 23 bits, eliminating sign bit, overflow bits (2 bits), hidden bit, guard bit, round bit and sticky bit, from M_o to make resultant M_o. Finally S_o, E_{max} and M_o will be packed into 32 bit floating point resultant value.

4. Experimental Results

Six-operand floating point addition can be implemented in two methods. The straight-forward method is network of 2-operand floating point addition. Another method is parallel addition of Six operand floating point is proposed and the various adders are analysed to perform parallel addition to make the design more efficient. Efficiency of our approach is compared with the network of two operand floating point addition.

VHDL is used for implementation, ModelSim SE Plus 6.4f is used for simulation and Xilinx ISE9.1i is used to synthesis. The target FPGA board is Xilinx ML410 and the board uses Xilinx FPGA (Virtex 4vfx60ff1152-11).The synthesis results for the proposed design and comparisons with other related work are illustrated in Table 2.

It's clear from the table 2, the design has less allocated resources and performs better in terms of operation frequency than the network tree of two input floating point addition and Tenca [4]. From the analysis, it is found that Carry save adder is efficient for multi-operand addition when compared to ripple carry adder and carry look-a-head adder. Same approach is implemented with more number of operands and carry save adder shows significant improvement when the number of operand increseases.

Design	Number of slices	Operating frequency (Clock speed)
Networked architecture approach	4140 (16 %)	42.73Mhz
Approach of Tenca[4]	8712 (34 %)	11.50 Mhz
Proposed Design(Ripple carry adder)	3840 (15 %)	44.020Mhz
Proposed Design(Carry lookahead adder)	3764 (14 %)	44.769Mhz
Proposed Design(Carry save adder)	3682 (14 %)	45.34Mhz

Table. 2: Performance results of various design of six operand floating point adder

5. Conclusion

In this paper, the design approach of Six-operand floating point addition is thoroughly analysed and presented. It is proven that the proposed design uses reduced number of slices and increased operating frequency as compare to Networked architecture approach and other approach. Our future work is extended to design and implementation of pipelined multi-operand floating point addition to improve the throughput further. Currently, in this paper constant number of overflow bits is appended to mantissa before parallel addition. This approach works for six-operand addition, without affecting the accuracy of result. However, we are working towards generalisation of overflow bits to arrive at an equation for n number of operands.

6. Acknowledgements

I thank Hardware Security Research Group head, Mr.R.Uppili for his continuous motivation for this work and my colleague Mr.R.Vijayasarathy for his insightful discussions and valuable inputs.

7. References

[1] IEEE Std 754, "IEEE Standard for Binary Floating-Point Arithmetic", IEEE, 1985.

[2] Peter-Michel Seidel and Guy Even. "Delay-optimized implementation of IEEE floating-point addition", IEEE Transactions on Computers, volume.53, NO.2, February 2004.

[3] Karlstrom, P.; Ehliar, A.; Liu, D.; "High Performance, Low Latency FPGA based Floating Point Adder and Multiplier Units in a Virtex 4", IEEE 14th Norchip Conference, 2006.

[4] Alexandre F.Tenca, "Multi-operand Floating-point Addition", IEEE International Symposium on Computer Arithmetic, 2009.

[5] Hamid, L.S.A.; Shehata, K.; El-Ghitani, H.; ElSaid, M. "Design of Generic Floating Point Multiplier and Adder/Subtractor Units", IEEE 12th International Conference on Computer Modelling and Simulation, 2010.

[6] Ali Malik; Seok-Bum Ko; "A STUDY ON THE FLOATING-POINT ADDER IN FPGAS", IEEE CCECE/CCGEI, Ottawa, May 2006.

[7] Krueger, S.D.; Seidel, P.-M.; "Design of an on-line IEEE floating-point addition unit for FPGAs", Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004.