VOL. 10, NO. 14, AUGUST 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

IEEE 754 COMPLIANT FLOATING POINT FUSED ADD SUB UNIT

Sharmila Hemanandh[1] and Siva Subramanian[2]
[1]Department of Electronics and Communication Engineering, Sathyabama University, India
[2]School of Electronics, Vellore Institute of Technology, Tamil Nadu, India
E-Mail:sharmilaahemanandh@gmail.com

ABSTRACT

Floating point arithmetic is a key component in the development of many algorithms for DSP applications that require large dynamic range and high level of accuracy. This paper proposes a floating point fused add sub unit that computes the sum and the difference of two operands simultaneously. Algorithms like FFT and DCT require the sum and the difference of two operands. The proposed fused unit is compliant with both single precision and double precision IEEE 754 standard of floating point representation. It also supports all rounding modes. When compared with the conventional floating point adder, the results obtained using the fused unit are more accurate since the number of rounding operations is reduced in the proposed unit. The fused unit is implemented using Verilog HDL and synthesized using Xylinx 14.1.

Keywords: IEEE 754 standard, floating point arithmetic, fused add sub, Xylink 14.1.

1. INTRODUCTION

Numbers in real life are almost numbers with fractional part. These fractional numbers in general are represented using fixed point representation or floating point representation to ease storage and manipulation. A fixed point number consists of an integer and a fraction part. The term fixed point refers to the fact that the number of digits after the radix point is fixed. Fixed point number representation is characterized based on the position of the radix point and the number of bits used to represent the number [1]. The bits to the right of the radix point represent the fraction and those to the left represent the integer part of the number. Though fixed point representation is a simple way to represent fractional numbers, it represents the range of numbers limited by the significant digits used to represent the number.

1.1 Representation of floating point numbers

In floating point number representation the radix point can float. It can be placed anywhere relative to the significant digits of the number. Here the number of bits before and after the radix point is not fixed. In general a number is represented by scientific notation using two fields, namely the mantissa (also called the significant) and the exponent. Moreover, using this representation very small numbers or very large numbers can be represented with a great deal of precision. In general, a floating point number is represented by three parts, namely sign(s), mantissa (m), and exponent (e).

The sign bit is 0 for a positive number and 1 for a negative number. Mantissa is always a positive number that holds the significant digits of the floating point number [2]. The exponent is the integer power to which the radix is raised. The radix here is 2 for binary, 10 for decimal, and 16 for hexadecimal representations.

1.2 EEE 754 floating point standard

In the early years major microprocessor manufacturers designed floating point units with varying word sizes. No uniform rounding procedure or overflow/ underflow conditions were handled. In 1980 IEEE standardized the format used to represent floating point numbers to deal with the problem of non standard floating point representations [3]. Since then the IEEE 754 standard has been the commonly used format for all floating point computations.

The IEEE standard for binary floating point arithmetic defines two basic floating point formats, namely single precision (32 bit) and double precision (64 bit). The number of bits used to represent the sign, exponent and mantissa is as shown in Table-1.

Table-1. Floating point representation.

S.No.	Forma	Sign	Exponent	Mantissa
1	Single Precision	1[31]	8[30-23]	23[22-0]
2	Double precision	1[63]	11[62-52]	52[51-0]

This standard not only specifies the basic and extended floating point number formats but also supports basic arithmetic operations such as addition, subtraction, multiplication, and division of two floating point numbers.

Rounding modes, conversion of integer into floating point format and vice versa, conversion among different floating point formats, floating point exceptions and handling are also a part of the standard [4]. Nowadays, all modern engineering and technology applications prefer floating point in the field of signal processing for scientific computations due to its high dynamic range when compared with fixed point representation.

2. FLOATING POINT FUSED ADD SUB UNIT

The implementation of many DSP algorithms like FFT and DCT require the sum and the difference of two operands simultaneously to perform butterfly operation [5]. Such applications are benefited by fused units such as Fused add sub unit (FAS). The FAS unit shown in Fig 1 computes the sum and the difference of two operands simultaneously.

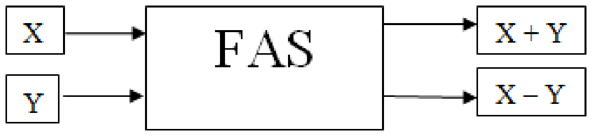


Figure-1. Fused add sub unit.

In the conventional method two design approaches are possible. In the first case two floating point adders are connected in parallel to compute the sum and the difference of two operands. Since two floating point adders use the area, power consumption is a limitation and this approach is expensive. In the second case only a single adder is used to compute the sum first and the same adder calculates the difference between the same operands in a serial manner [6]. Though this approach is efficient in terms of area the time required to complete the operation is high.

Hence an FAS unit is designed to increase the throughput which is a major parameter of concern in many signal processing applications. There is also a significant reduction in power and silicon area because FAS unit executes the shared logic when compared with the traditional approach. This paper proposes the FAS unit that increases the performance and accuracy of DSP algorithms.

2.1 Design approach

Let X and Y be the two operands. The FAS unit computes the sum(X+Y) and the difference(X-Y) simultaneously. To begin with the exponent compare logic compares the exponents of the two operands to identify the smaller number and also computes the difference between the two exponents. The exponents of the two operands must be made equal before addition or subtraction. If the exponent of X is greater than that of Y, then right shift the mantissa of Y by the difference value of the two exponents. Similarly if the exponent of Y is greater than that of X, then right shift the mantissa of X by the difference value of the two exponents. By doing so the exponents of the two operands are made equal. Next the sum of the two operands is computed. Normalization is required if the results lie outside the permitted range. If the result obtained after adding the two operands is not in the prescribed format, then it has to be rounded.

The IEEE floating point arithmetic supports four rounding modes:

Round to +∞: The result obtained is rounded up to a representable value close to positive infinity but not greater than the true result.

Round to -∞: The result obtained is rounded down to a representable value close to negative infinity but not less than the true result.

Round to zero: The number of bits required is retained and the rest of the bits are discarded (also called truncation)

Round to the nearest: It is the most commonly used rounding mode in floating point applications. Here the result is rounded up or down. When the result is exactly half way through, then the result is rounded to the nearest even.

The proposed fused add sub unit supports all the four rounding modes.

The IEEE 754 standard allows the representation of four types of special values:

Signed zero: Two types of zero exist based on the sign, namely +0 and -0. Though the magnitudes are equal in both the cases, ~~But~~ certain operations like 1/ (+0) = +∞ and 1/ (-0) = -∞ yield different results based on the sign.

Subnormal numbers: Also called de-normalized numbers where the value of the exponent is the smallest value and the leading significant bit is 0 instead of 1.

Infinites: The standard allows the representation of positive infinity and negative infinity. This representation is much useful when the results of certain operations are undefined. The sign bit is either zero (positive infinity) or one (negative infinity). The significant bits are all zeros and the exponent bits are all ones.

Nan: It is used to represent the result of an invalid operation such as dividing by zero, finding the square root of a negative number. The sign bit is either zero or one, all the bits of the exponent being ~~are~~ one and the significant is a combination of zero's and one's. Each combination represents the type of invalid operation.

The proposed fused add sub unit is designed to handle all the four special values described above.

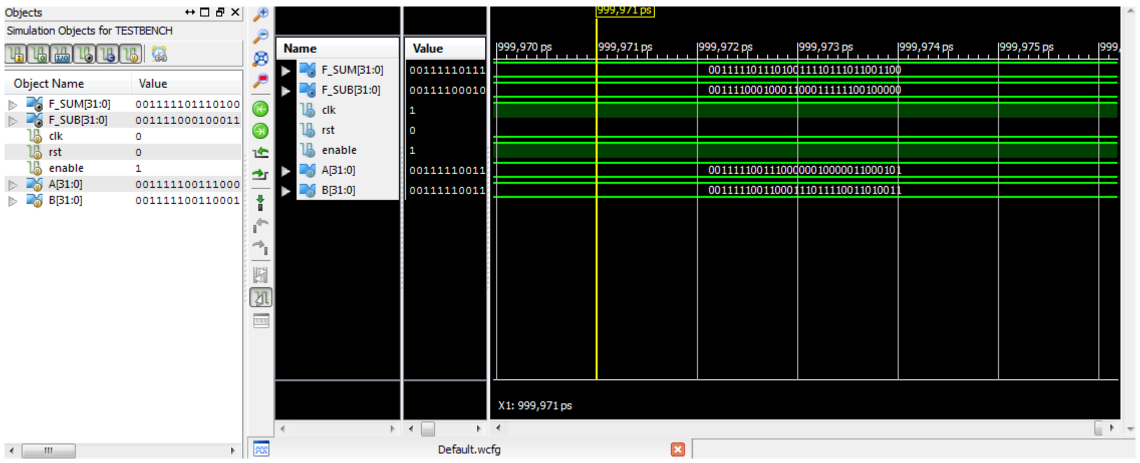
2.2 Algorithm

The steps involved in the proposed fused add sub unit are as follows:

The two operands are represented by means of IEEE floating point standards i.e., exponent, mantissa, and sign bit. The exponents of the two operands are compared and the sign bit is obtained from the highest exponent value. Find the difference between the two exponents to get the value by which the smaller mantissa should be shifted. Add the mantissas of the two operands. Shift the mantissa if required so that the MSB of the resultant mantissa is 1. Add the two exponents to obtain the final exponent value. Obtain the difference between the two operands by taking the 2s complement of the smaller number before step-4.

3. RESULTS AND DISCUSSION

The proposed floating point fused add sub unit is simulated using Isim simulator. The unit is placed and routed on Virtex 6 FPGA and synthesized on Xylinx 14.1. Figure-2 shows the simulation results of single precision floating point fused add sub unit. The two operands A and B are represented using single precision format. The sum and the difference of the two operands are computed and the result is also represented in the same format.


Objects
Simulation Objects for TESTBENCH

Object Name	Value
F_SUM[31:0]	001111101110100
F_SUB[31:0]	001111000100011
clk	0
rst	0
enable	1
A[31:0]	001111100111000
B[31:0]	001111100110001

Name	Value
F_SUM[31:0]	00111110111
F_SUB[31:0]	00111100010
clk	1
rst	0
enable	1
A[31:0]	00111110011
B[31:0]	00111110011

```
999,971 ps
999,970 ps   999,971 ps   999,972 ps   999,973 ps   999,974 ps   999,975 ps   999,
00111110111010011110111011001100
00111100010001100011111100100000
00111110011100000010000011000101
00111110011000111011110011010011
X1: 999,971 ps
```

Figure-2. Simulation result of single preision floating point fused add sub unit.

The RTL schematic of single precision floating point fused add sub unit is shown in Figure-3.

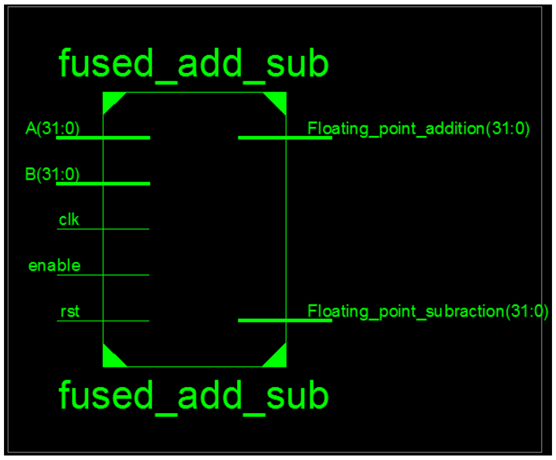
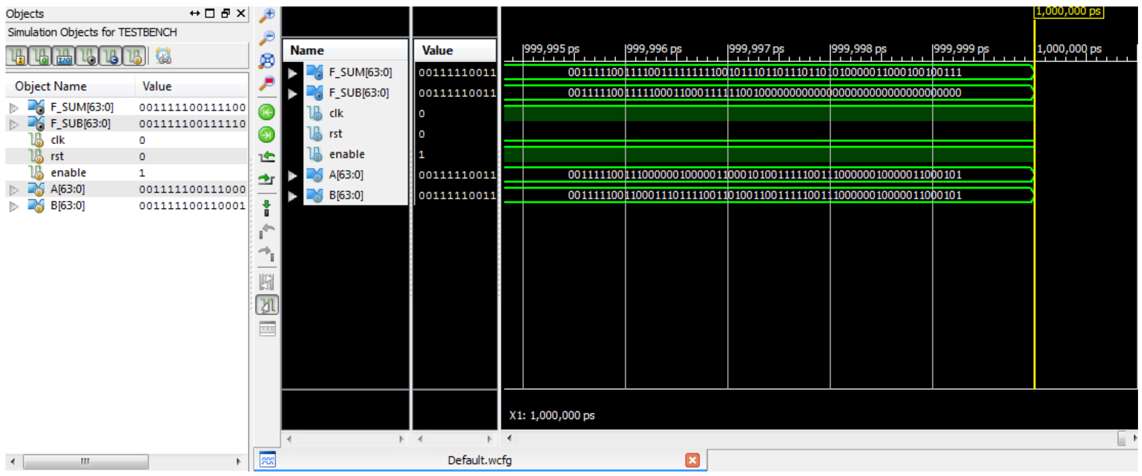

fused_add_sub
A(31:0)
B(31:0)
clk
enable
rst
Floating_point_addition(31:0)
Floating_point_subraction(31:0)
fused_add_sub

Figure-3.RTL schematic of single precision floating point FAS unit.

Figure-4 shows the simulation results of double precision floating point fused add sub unit. The two operands A and B are represented using double precision format. The sum and the difference of the two operands are computed and the result is also represented in the same format.



```
Objects
Simulation Objects for TESTBENCH
Object Name      Value
F_SUM[63:0]      001111100111100
F_SUB[63:0]      001111100111110
clk              0
rst              0
enable           1
A[63:0]          001111100111000
B[63:0]          001111100110001

Name             Value
F_SUM[63:0]      00111110011   0011111001111001111111111001011101101110110101000001100010010 0111
F_SUB[63:0]      00111110011   0011111001111100011000111111001000000000000000000000000000000000
clk              0
rst              0
enable           1
A[63:0]          00111110011   0011111001110000000100000110001010011111001110000001000001100 0101
B[63:0]          00111110011   0011111001100011101111001101001100111110011100000010000011000101

999,995 ps  999,996 ps  999,997 ps  999,998 ps  999,999 ps  1,000,000 ps
X1: 1,000,000 ps
Default.wcfg
```

Figure-4. Simulation result of double preision floating point fused add sub unit.

The RTL schematic of the double precision floating point fused add sub unit is shown in Figure-5.

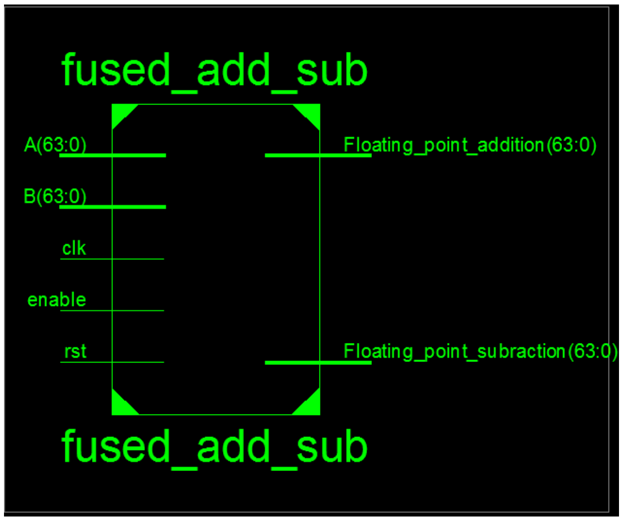

fused_add_sub
A(63:0)
B(63:0)
clk
enable
rst
Floating_point_addition(63:0)
Floating_point_subraction(63:0)
fused_add_sub

Figure-5.RTL schematic of double precision floating point FAS unit.

Table-2 summarizes the device utilization of the fused unit in single and double precision representations. The logic utilization of double precision is more by 49% when compared to single precision format. However double precision fused unit exhibits high level of precision when compared to single precision representation.

Table-2. Device utilization summary of single double precision floating point FAS unit.

	Proposed fused add sub unit	
	Single precision	**Double precision**
Number of slice registers	197	387
Number of slice LUTs	388	878
Number of occupied slices	182	353
Number of bonded IOBs	131	257

Table-3. Delay of fused add sub unit and floating point adder unit.

	Proposed Fused add sub unit		**Floating point add sub unit**	
	Single precision	**Double precision**	**Single precision**	**Double precision**
Delay(ns)	5.485	7.038	9.262	13.532

Table-3 compares the delay of the proposed fused add sub unit with a conventional floating point add sub unit. The results show that the delay of the proposed fused unit is 41% less than the floating point adds sub unit in single precision format and 47% less in the case of double precision format.

4. CONCLUSIONS

This paper has presented a single precision and double precision fused floating point add sub unit. The modules have been written in Verilog HDL. The proposed FAS unit has been synthesizedand implemented on Virtex6 FPGA in single precision and double precision IEEE 754 floating point representation. Comparing the delay, the fused unit exhibits better performance than the floating point add sub unit. Moreover the proposed fused add sub unit supports all rounding modes and also handles special values. The proposed fused add-subtract unit can be used to realize DSP algorithms, including the basic butterfly computation of FFT and DCT.

REFERENCES

[1] P. -M. Seidel, 2003, Multiple Path IEEE Floating-Point Fused Multiply-Add. Proceedings of the 46thIEEE International Midwest Symposium on Circuits and Systems. pp. 1359-1362.

[2] 2008. IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, New York: IEEE, Inc.

[3] Sohn*et al*. 2012. Improved Architectures for a Fused Floating point add-subtract unit. IEEE Transactions on Circuits and Systems-1: Regular Papers. 59: 2285-2291.

[4] William Stallings, 2010, Computer Organization and Architecture, 8th edition Pearson, 345-361.ISBN 978-81-317-3245-8.

[5] A. Amaricai, O. Boncalo, C.E. Gavriliu, 2013, Low Precision DSP Based Floating Point Multiply-Add Fused for FPGAs. IET Computing AND Digital Techniques. 8:187-197.

[6] JongwookSohn and Earl E. Swartzlander. 2012. Improved Architectures for a Fused Floating-Point Add-Subtract Unit. IEEE transactions on Circuits and Systems. 59:2285-2291.

[7] Hani Saleh and Earl Swartzlander, Jr. 2008. A Floating-Point Fused Dot Product Unit, IEEE International Conference on Computer Design (ICCD), Lake Tahoe, CA, 427-431, 2008.

[8] E. E. Swartzlander, Jr. and H. H. Saleh. 2012. FFT implementation with fused floating-pointoperations, IEEE Trans. Comput.61: 284-288.

[9] A. Akkas, M.J. Schulte. 2011. A decimal floating-point fused multiply-add unit with a novel decimal Leading -zero anticipator, Application- Specific Systems, Architectures and Processors (ASAP),IEEE International Conference. 43: 11-14.

[10] R Samy, H.A.H. Fahmy, Raafat, R, A. Mohamed, T. El Deeb, Y. Farouk. 2010. A decimal Floating-point fused-multiply-add unit, Circuits and Systems (MWSCAS), IEEE International Midwest Symposium. 529:1-4.