Title: A New Architecture For floating-point fused add-subtract Unit Design

Volume o6, Issue 03, Pages: 52 – 59.

Paper Authors

G.SIVA KUMAR, Y.B.SHABBIR HUSSIAN.
Dept of Electronics and Communication Engineering, CVRT, AP, India.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

A New Architecture For floating-point fused add-subtract Unit Design

G.SIVA KUMAR[1], Y.B.SHABBIR HUSSIAN [2]
[1]PG Scholar, Electronics and Communication Engineering, CVRT, AP, India
[2]Assistant Professor, Electronics and Communication Engineering, CVRT, AP, India
kamakshi.n14@gmail.com

Abstract—This paper presents improved architectures for a fused floating-point add–subtract unit. The fused floating-point add–subtract unit is useful for digital signal processing (DSP) applications such as fast Fourier transform (FFT) and discrete cosine transform (DCT) butterfly operations. To improve the performance of the fused floating-point add–subtract unit, a dual-path algorithm and pipelining are employed. The proposed designs are implemented for both single and double precision and synthesized with a 45-nm standard-cell library. The fused floating-point add–subtract unit saves 40% of the area and power consumption compared to a discrete floating-point add–subtract unit. The proposed dual-path design reduces the latency by 30% compared to the discrete design with area and power consumption between that of the discrete and fused designs. Based on a data flow analysis, the proposed fused dual-path floating-point add–subtract unit can be split into two pipeline stages. Since the latencies of two pipeline stages are fairly well balanced, the throughput is increased by 80% compared to the non pipelined dual-path design.

Index Terms—Digital signal processing (DSP), floating-point arithmetic, fused floating-point operation, high-speed computer arithmetic.

I. INTRODUCTION

Current digital signal processing (DSP) systems are making the transition from fixed-point arithmetic (used initially because of its simplicity) to floating-point arithmetic. The latter has several advantages including the freedom from overflow and underflow and ease of interfacing to the rest of the system (which generally will use IEEE-754 Standard floating-point arithmetic [1]). To improve the performance of floating-point arithmetic, several fused floating - point operations have been introduced: Fused Multiply–Add (FMA) [2], Fused Add–Subtract [7], and Fused Two-Term Dot-Product [8].This paper presents improved architecture designs and implementations for a fused floating-point add–subtract unit. Many DSP applications such as fast Fourier transform (FFT) and discrete cosine transform (DCT) butterfly operations can benefit from the fused floating point add–subtract unit [3]. There-fore, the improved fused floating-point add–subtract unit will contribute to the next generation floating-point arithmetic and DSP application development. The proposed fused floating-point add–subtract unit takes two normalized floating-point operands and generates their sum and difference simultaneously. It supports all five rounding modes specified in IEEE-754 Standard [1]. Several techniques are applied to achieve low area and high speed: Instead of executing two identical floating-point adders, the fused floating-point add–subtract unit shares the common logic to generate the sum and difference simultaneously. Therefore, it saves much of the area compared to a discrete floating-point add–subtract unit. Also, it reduces the latency by simplifying the control signals.

1) A dual-path algorithm can be applied to increase speed. The dual-path logic consists of a far path and a close path. In the far path, the addition, subtraction and rounding logic are performed in parallel. By aligning the significands to the minimal number of bits, the addition, subtraction and rounding logic are simplified. There are three cases for the close path depending on the difference of the exponents. For each case, addition, subtraction and leading zero anticipation (LZA) are performed in parallel and rounding is not required. Therefore, the dual-path design reduces the latency of the critical path.

2) To increase the throughput, pipelining can be applied. Based on data flow analysis, the proposed dual-path de-sign is split into two pipeline stages. By properly arranging the components, latencies of the two pipeline stages are balanced so that the throughput of the entire design is increased. Section II describes the traditional discrete floating point add–subtract unit with two identical floating point adders. The next three sections present improved architectures for a fused floating-point add– subtract unit design. In Section III, the fundamental concepts of the fused floating -point add–subtract unit and its implementation are presented. Improved architectures for applying the dual-path algorithm and implementation details are described in Section IV.

II. TRADITIONAL FLOATING-POINT ADD-SUBTRACT UNIT

A direct way to implement the floating-point add–subtract operation is to use two identical floating-point adders in parallel. One of the adders performs an addition and the other performs a subtraction to produce the sum and difference simultaneously. A traditional floating-point adder such as that of Fig. 1 can be used for each operation. The steps to execute the floating-point addition are as follows:

1) Exponent compare logic compares the exponents of the two operands A and B to determine which exponent is greater and calculates their difference.

2) The exponent comparison results are used for the significand swap logic. When the exponents are equal, the significands are compared to identify the smaller significand. The significand of the smaller operand is shifted by the amount of the exponent difference (if any) for the alignment and the guard, round and sticky bits are attached to the LSB.

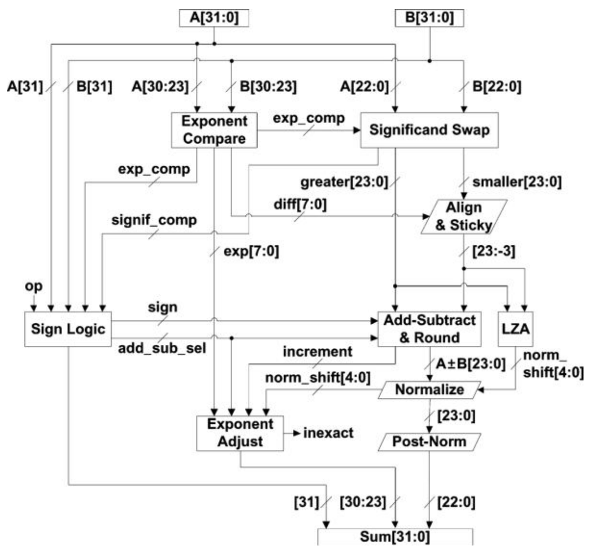


Fig. 1. Traditional floating-point adder

3) Since some of rounding modes specified in IEEE-754 Standard [1] require knowing the sign (i.e., round to positive and negative infinity), the sign logic must be performed prior to the round logic. The sign logic provides the sign of the sum and the operation decision bit to the round logic and significand adders,respectively.

4) The two significands are passed to the significand add–sub-tract unit and LZA simultaneously. The add–subtract unit performs the addition or subtractionof the two significands depending on the operation. It produces rounded and un-rounded results and the round logic selects one of them for a fast rounding. The LZA generates the amount of cancellation during the subtraction in a constant time so that the subtraction result is immediately normalized. The overflow of the significand adder and the shift amount from the LZA are passed to the exponent adjust logic. Using the shift amount, the exponent adjust logic generates the exponent of the sum. In this step, inexact, overflow and underflow of the exponent (if any) are detected for setting the exception flags.

III. FUSED FLOATING-POINT ADD-SUBTRACT UNIT

The discrete floating-point add–subtract unit produces the sum and difference simultaneously by executing two identical floating-point additions. However, much of the logic such as exponent comparison, significand swap and alignment in the two floating-point adders is nearly the same for the two operations.

In order to reduce the overhead, a fused floating-point add–subtract unit shares the common logic for the two operations. Fig. 2 shows the design of a fused floating point add–subtract unit. The fused floating-point add– subtract unit produces the sum and difference results simultaneously by executing the shared logic such as the exponent comparison, significand swap and alignment. Also, the fused floating-point add–subtract unit performs only one significand addition and subtraction for each operation.

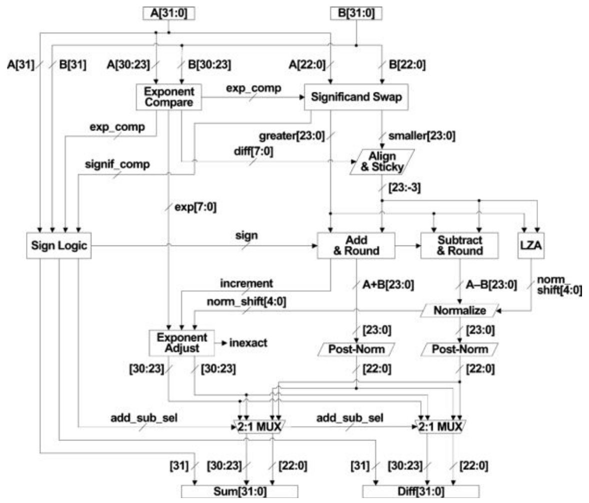


Fig. 2. Fused floating-point add–subtract unit

Since two operations are explicitly performed for sum and difference results (e.g., if the addition is used for the sum, the subtraction is used for the difference), the addition and subtraction are separately placed and only one LZA and normalization (for the subtraction) is required. Assuming both sign bits are positive, the addition and subtraction are performed separately. Then, two multiplexers select the sum and difference with the operation decision bit, which is the XOR of the two sign bits. More details of the logic are described in next section. This approach simplifies the addition and subtraction operations. It also reduces the control signals for distinguishing the signs and final results. Thus, the fused floating-point add–subtract unit achieves low area and high speed.

IV. DUAL PATH FUSED FLOATING-POINT ADD-SUBTRACT UNIT

To achieve a high-performance fused floating-point add–sub-tract unit, this paper proposes a dual-path approach. Most high-speed floating-point adders employ the dual-path algorithm [4]. Fig. 3 shows the dual-path fused floating-point add–subtract unit. The dual-path algorithm skips the normalization step depending on the exponent difference. Since the normalization after the subtraction is one of the bottlenecks in the fused floatingpoint add–subtract unit, the dual-path approach improves the performance.

The dual-path approach consists of far path and close path logic. The far path logic takes the significands if the difference of the exponents is greater than 1. In this case,,massive cancellation does not occur during the subtraction so that the LZA is un-necessary. The far-path logic is implemented similar to the front end of the traditional floating-point adder as shown in Fig. 3.

The greater and smaller signifi cands are determined by swap-ping two significands based on the exponent comparison:

A sign	B sign	Comp.	Sum	Difference
+	+	$\|A\| < \|B\|$	$\|A\| + \|B\|$	$-(\|B\| - \|A\|)$
+	+	$\|A\| > \|B\|$	$\|A\| + \|B\|$	$\|A\| - \|B\|$
+	−	$\|A\| < \|B\|$	$-(\|B\| - \|A\|)$	$\|A\| + \|B\|$
+	−	$\|A\| > \|B\|$	$\|A\| - \|B\|$	$\|A\| + \|B\|$
−	+	$\|A\| < \|B\|$	$\|B\| - \|A\|$	$-(\|A\| + \|B\|)$
−	+	$\|A\| > \|B\|$	$-(\|A\| - \|B\|)$	$-(\|A\| + \|B\|)$
−	−	$\|A\| < \|B\|$	$-(\|A\| + \|B\|)$	$\|B\| - \|A\|$
−	−	$\|A\| > \|B\|$	$-(\|A\| + \|B\|)$	$-(\|A\| - \|B\|)$

TABLE I SIGN DECISION TABLE

The two significands are aligned with a 1 attached to the MSB end to make 24- bit normalized significands. By aligning the two significands to 24-bits, significand addition and subtraction are simplified, re-sulting in a reduction in the logic area and delay. The significand of the smaller operand is right shifted by amount of the exponent difference and aligned to 24 -bits. The sticky bit is set if at least one bit of the 22 LSBs is a 1 and the 23rd and the 24th LSBs become the round and guard bits, respectively. Since the significand of the larger operand is not shifted, the 24-bit significand is kept as it is without guard, round and sticky bits. The greater and smaller significands are passed to the addition and subtraction units.

The round logic takes the LSBs, guard, round and sticky bits of the two significands and performs 4-bit addition and subtraction to determine if the result is rounded up or not for each operation. Also, it requires the sign bits of the addition and subtraction to support all five round modes specified in IEEE-754 Standard. Since the far path requires at most a 1-bit normalization shift for both addition and subtraction, it avoids a large normalization procedure.

The close path takes the significands if the difference of the two exponents is 0 or 1. Fig. 5 shows the close path
logic. There are three cases for the close path depending on the difference of the exponents:

For each case, addition, subtraction and LZA are performed simultaneously. LZA with concurrent correction is used for a fast normalization [5], [6]. One of the three results is selected based on the small exponent comparison, which compares the two LSBs of the exponents. In contrast to the far path, the significands are not swapped to avoid a large significand comparison.
When the subtraction result is negative, a two's complement operation is performed to convert the result to a positive value. The carry-out of the subtraction indicates a significand comparison, which is passed to the sign logic, to determine the sign bits when the two exponents are equal. Since the significands in the close path are mis-aligned by at most 1-bit, rounding is not required. The addition result is normalized by 1-bit overflow, while the subtraction result is normalized by up to 23-bits using the shift amount from the LZA. The remaining logic for the dual-path fused floatingpoint add–subtract unit are the exponent compare logic shown in Fig. 6 calculates the difference of the two exponents and determines which is greater, these are the same functions required for the traditional logic. In addition to thisthe path decision between the far and close paths based on the exponent difference is required:

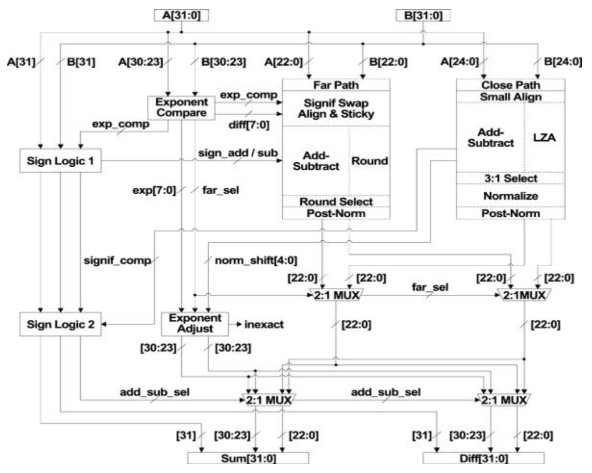


Fig. 3 . Dual-path fused floating-point add–subtract unit

The path decision flag is passed to the two multiplexers for selecting the addition and subtraction results between the far and close paths. The exponent adjust logic shown in Fig.7 performs addition and subtraction to adjust the exponents by the amount that the significands are shifted. The exponent adjust logic produces two exponent results simultaneously. In the case of addition, one of the increment values is added depending on the path decision that is the overflow from the significand addition. In the case of subtraction, if the far path is selected, the decrement value is subtracted that is the underflow from the significand subtraction. If the close path is selected, the

normalization shift value is subtracted that is the shift amount of the massive cancellation that occurred during the subtraction. The two adjusted exponents are passed to the exception logic. Since underflow does not occur in default exception handling, the exception logic supports abrupt Underflow, an alternate exception handling specified in IEEE-754 Standard [1] to detect three exception cases:

where round up is the rounding decision of the significand result. The overflow flag is set if the exponent exceeds the maximum value that can be represented such as positive and negative infinity. The underflow flag is set if the exponent is too small to be represented such as zero and subnormal values. Overflow only occurs in addition and underflow only occurs in subtraction.

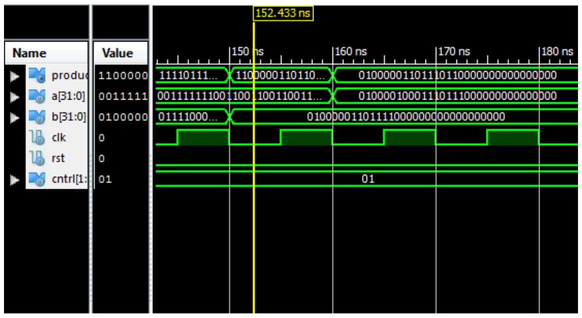
V. PIPELINED FUSED FLOATING-POINT ADD–SUBTRACT UNIT

As is well known, proper pipelining increases the throughput of floating-point adders [4]. In order to achieve a proper pipelined fused floating-point add–subtract unit, the latencies of the components in the proposed design are investigated. Each component is implemented in Verilog-

HDL, simulated with XILINX ISE design suite and synthesized with Leonardo spectrum ASIC standard library. Since several components are executed in parallel, they are combined to a stage and the sum of the component delays determines the latency of the stage. Considering the latencies of components and their parallel execution, the proposed design is split into two pipeline stages. Each pipeline stage is executed every cycle so that the largest latency determines the throughput of the design. Fig. 8 shows the data flow, the latency of each component, and the critical path. The first pipeline stage consists of unpacking logic and the two data paths: the far path and the close path. The two data paths are the first half of the dual path, which is described in Figs. 4 and 5. The far path in the first pipeline stage contains the exponent compare, sign logic 1, significand swap, align and sticky logic. The close path in the first pipeline stage contains the small exponent compare, small significand align, three additions, subtractions and LZAs, and 3:1 select logic. Among the two data paths, the close path takes the larger latency so that it becomes the critical path. The series of components in the close path determines the latency of the first pipeline stage, which is 4.37 ns. The second half of the dual path and the remaining logic comprise the second pipeline stage. The far path in the second pipeline stage contains the addition, subtraction, round logic, and round select logic. The close path in the second pipeline stage contains the sign logic 2, complement and normalization logic. Among the two data paths, the far path takes the larger latency so that the second half of the far path logic and the remaining logic (path select, exponent adjust, and operation select logic) comprise the critical path, which adds up to 3.9 ns. The latencies of the two pipeline stages are fairly well balanced so that the throughput of the design is increased. Since the latency of the first pipeline stage is slightly larger than that of the second pipeline stage, it determines the throughput of the entire design.

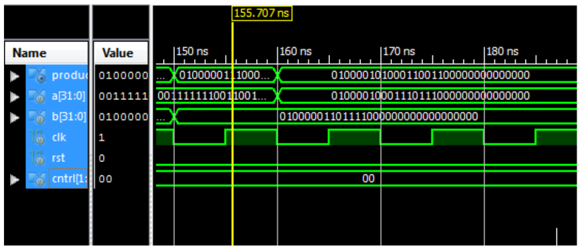
VI. Results

The previous sections have introduced various designs for the fused floating-point add–subtract unit. Each design is implemented in Verilog-HDL and synthesized with xilinix.



```
Name        Value
produ       1100000   11110111...   1100000110110...   01000001101110110000000000000000
a[31:0]     0011111   001111111001100110011 0011...      01000010001110111000000000000000
b[31:0]     0100000   01111000...   01000001101111000000000000000000
clk         0
rst         0
cntrl[1:    01        01
```

Fig 4 Simulation Result



```
Name        Value
produ       0100000   ...   0100000111000...   01000010100011001100000000000000
a[31:0]     0011111   0011111110011001...      01000010001110111000000000000000
b[31:0]     0100000   ...   01000001101111000000000000000000
clk         1
rst         0
cntrl[1:    00        00
```

Fig 5 Simulation Result

VII. CONCLUSION

Improved architectures for the design and implementation of a fused floating-point add–subtract unit are presented. The floating-point add–subtract unit is useful for digital signal processing applications such as FFT and DCT butterfly operations. This paper presents improved architectures which apply the dual-path algorithm and pipelining to the fused floating-point add–subtract unit and compares the area, latency, throughput with the traditional parallel implementation. The fused floating-point add–subtract unit saves area compared to the traditional discrete floating-point add– subtract unit by sharing the common logic. Also, the fused floating-point add–subtract unit reduces the latency due to its simplified control logic. The dualpath fused floatingpoint add–subtract unit reduces the latency compared to the discrete design by performing several add–subtract operations for each case in parallel. Additionally, a pipelined implementation to increase the throughput of the dual-path fused floating-point add–subtract unit is described. It uses two pipeline stages and the latencies are well balanced so that the throughput is increased compared to the non-pipelined dual-path design.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, New York: IEEE, Inc., Aug. 29, 2008.

[2] T. Lang and J. D. Bruguera, “Floating-point fused multiply-add with reduced latency,” IEEE Trans. Comput., vol. 53, no. 8, pp. 988– 1003, Aug. 2004.

[3] E. E. Swartzlander, Jr. and H. H. Saleh, “FFT implementation with fused floating-point operations,” IEEE Trans. Comput., vol. 61, no. 2,pp. 284–288, Feb. 2012.

[4] P. M. Seidel and G. Even, “Delay-optimized implementation of IEEE floating-point addition,” IEEE Trans. Comput., vol. 53, no. 2, pp.97– 113, Feb. 2004.

[5] J. D. Bruguera and T. Lang, “Leading—One prediction with concurrent position correction,” IEEE Trans. Comput., vol. 48, no. 10, pp. 1083–1097, Oct. 1999.

[6] R. Ji, Z. Ling, X. Zeng, B. Sui, L. Chen, J. Zhang, Y. Feng, and G. Luo, “Comments on ‘Leading one prediction with concurrent position correction’,” IEEE Trans. Comput., vol. 58, no. 12, pp. 1726–1727,Dec. 2009.

[7] H. H. Saleh and E. E. Swartzlander, Jr., “A floating-point fused add–subtract unit,” in Proc. 51st IEEE Midwest Symp. Circuits Syst., 2008, pp. 519–522.

[8] H. H. Saleh and E. E. Swartzlander, Jr., “A floating-point fused dotproduct unit,” in Proc. IEEE Int. Conf. Comput. Design, 2008, pp. 427–431.