A Floating-Point Fused Dot-Product Unit

Hani H. Saleh
Intel Corporation
5000 Plaza on the Lake Blvd.
Austin, TX 78746
hani.saleh@intel.com

Earl E. Swartzlander, Jr.
ECE Department
University of Texas at Austin
Austin, TX 78712
eswartzla@aol.com

***Abstract*—A floating-point fused dot-product unit is presented that performs single-precision floating-point multiplication and addition operations on two pairs of data in a time that is only 150% the time required for a conventional floating-point multiplication. When placed and routed in a 45nm process, the fused dot-product unit occupied about 70% of the area needed to implement a parallel dot-product unit using conventional floating-point adders and multipliers. The speed of the fused dot-product is 27% faster than the speed of the conventional parallel approach. The numerical result of the fused unit is more accurate because one rounding operation is needed versus at least three for other approaches.**

I. Introduction

Much research has been done on the floating-point fused multiply add (FMA) unit [1]. It has several advantages over discrete floating-point adders and multipliers in a floating-point unit design. Not only can a fused multiply-add unit reduce the latency of an application that executes a multiplication followed by an addition, but the unit may entirely replace a processor's floating-point adder and floating-point multiplier. Many DSP algorithms have been rewritten to take advantage of the presence of FMA units. For example, in [2] a radix-16 FFT algorithm is presented that speeds up FFTs in systems with FMA units. High-throughput and digital filter implementations are possible with the use of FMA units [3]. FMA units are utilized in embedded signal processing and graphics applications [4], used to perform division [5], argument reduction [6], and this is why the FMA has become an integral unit of many commercial processors such as those of IBM [7], HP [8] and Intel [9].

Similar to operations performed by a FMA, in many DSP algorithms and in other fields calculating the sum of the products of two sets of operands (dot-product) is a frequently used operation. For example, this is required in the computation of the FFT and DCT butterfly operations.

In traditional floating-point hardware the dot product is performed with two multiplications and an addition. These operations may be performed in a serial fashion which limits the throughput. Alternatively, the multiplications may be performed in parallel with two independent floating-point multipliers followed by a floating-point adder which is expensive (in silicon area and in power consumption).

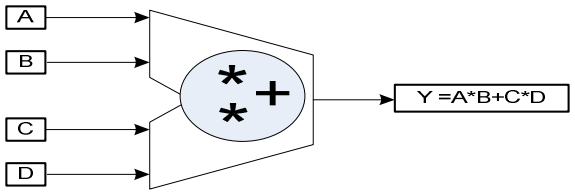

Figure 1. Fused Dot Product Unit Concept.

This paper investigates the implementation of a floating-point fused dot-product unit shown in Fig. 1. It performs the following operation:

$$Y = A * B + C * D \tag{1}$$

The numerical operation performed by this unit can be used to improve many DSP algorithms. Specifically, multiplication of complex operands benefits greatly from the FDP. For example, implementations of the FFT butterfly operation, the DCT butterfly operation, vector multiplication and the wavelet transform could all benefit largely from the speed up offered by this unit.

For example, consider the FFT radix-2 decimation in frequency butterfly shown in Fig. 2. In an implementation with discrete floating-point adders and multipliers ten operations are required (six additions and four multiplications). Alternatively two fused dot product operations and four additions can be used.

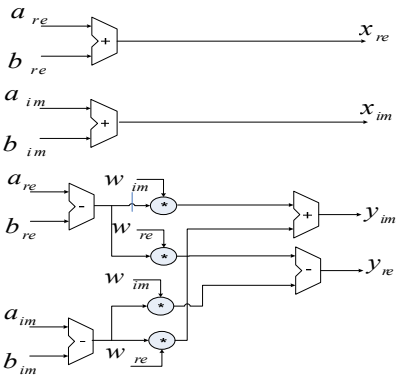

Figure 2. FFT Radix-2 Butterfly Computation.

II. Approach

There are two approaches that can be taken with conventional floating-point adders and multipliers to realize the dot-product. The parallel implementation shown on Fig. 2 uses two multipliers operating in parallel and an adder.

The parallel approach is appropriate for applications where maximizing the throughput is more important than minimizing the area or the power consumption. The serial implementation shown on Fig. 3 uses a single adder and a single multiplier with multiplexers and a register for intermediate results. It has lower throughput and less area and power consumption.

Fig. 5 shows the architecture of a conventional single path floating-point multiplier [10]. Much of the multiplier is devoted to operations that can be shared if it is extended to perform the dot-product.

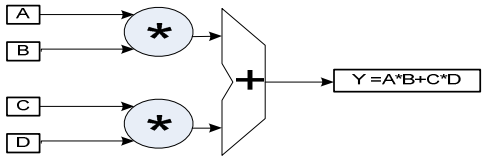

Figure 3. Conventional Parallel Dot-Product Unit.

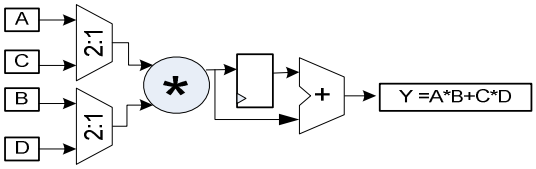

Figure 4. Conventional Serial Dot-Product Unit.

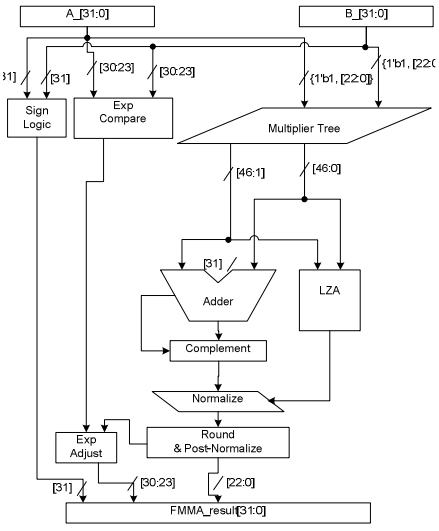

Figure 5. Conventional Floating-Point Multiplier.

In a parallel conventional implementation of the dot-product (such as that shown in Fig. 2) two floating-point multipliers are used in addition to a floating-point adder, thus three rounding operations are performed in generating the result. In the FDP unit that is shown in Fig. 6, a multiplier tree, an aligner in addition to 4:2 reduction tree are added to a conventional FPM to perform the dot-product operation, the remaining components of the FPM are used as is which results in a significant area reduction compared to the conventional implementation. The exponent compare circuit is shown in Fig. 7.

Although it is not especially attractive, a system could use this unit to replace a floating-point adder and a floating-point multiplier. If operands B and D are set to one, then the unit will perform addition only, with simple data forwarding multiplexers for operands A and C to skip the multiplication trees, the speed of the addition will be one multiplexer delay more than a discrete floating-point adder.

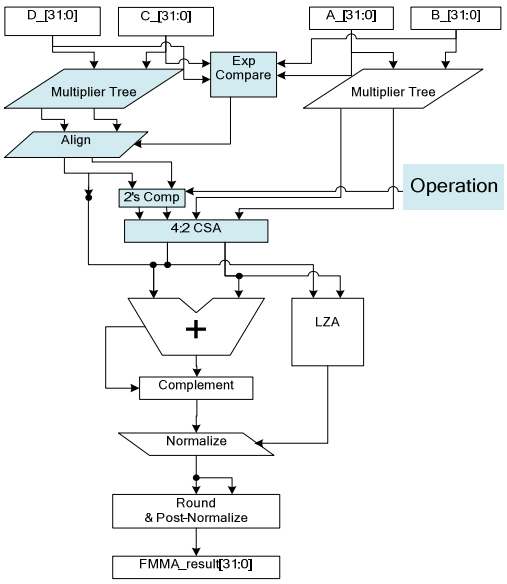

Figure 6. Floating-Point Fused Dot-Product Unit.

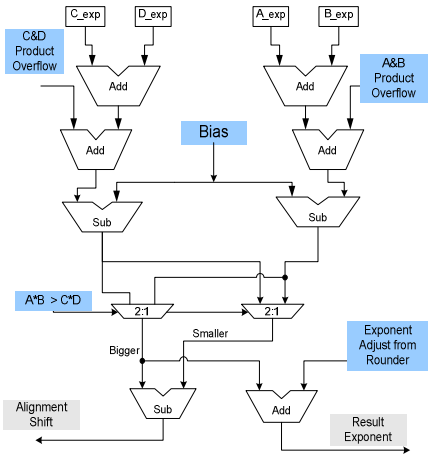

Figure 7. Exponent Compare Circuit.

Also the unit could be used to perform multiplication of C * D only by setting A or B to zero and use data forwarding multiplexers to skip the alignment circuit. In this case, there will be an extra delay of two multiplexer operations.

III. Error Analysis

Computing using floating-point (FP) representations provides a wide dynamic range, freeing programmers from writing the manual scaling code required for fixed-point representation. Nevertheless, floating-point computation suffers from two types of errors: propagation error, which is determined by the errors of input data and the operation type only, and rounding error, which is caused by the rounding of the operation result [12].

The propagation error is derived as follows:

$$f(x,y) \approx f(\hat{x},\hat{y}) + \frac{\partial f(\hat{x},\hat{y})}{\partial x}(x-\hat{x}) + \frac{\partial f(\hat{x},\hat{y})}{\partial y}(y-\hat{y}) \tag{2}$$

$$\varepsilon_{prop} = \frac{|f(x,y) - f(\hat{x},\hat{y})|}{f(x,y)} \approx \frac{f'(\hat{x},\hat{y})\hat{x}}{f(\hat{x},\hat{y})}\varepsilon_x + \frac{f'(\hat{x},\hat{y})\hat{y}}{f(\hat{x},\hat{y})}\varepsilon_y = k_x\varepsilon_x + k_y\varepsilon_y \tag{3}$$

Where: K is the amplification factor, determined based on the operation type and data. For floating-point multiplication, the propagation error amplification factors are:

$$k_x = \frac{f'(\hat{x},\hat{y})\hat{x}}{f(\hat{x},\hat{y})} = \frac{\hat{x}\hat{y}}{\hat{x}\hat{y}} = 1.0 \tag{4}$$

$$k_y = \frac{f'(\hat{x},\hat{y})\hat{y}}{f(\hat{x},\hat{y})} = \frac{\hat{x}\hat{y}}{\hat{x}\hat{y}} = 1.0 \tag{5}$$

While for floating-point addition, the amplification factors are given by:

$$k_x = \frac{f'(\hat{x},\hat{y})\hat{x}}{f(\hat{x},\hat{y})} = \frac{\hat{y}}{\hat{x}+\hat{y}} \tag{6}$$

$$k_y = \frac{f'(\hat{x},\hat{y})\hat{y}}{f(\hat{x},\hat{y})} = \frac{\hat{x}}{\hat{x}+\hat{y}} \tag{7}$$

The second component of a floating-point operation overall error is rounding error, a formula for it is derived as shown in the following equations, where the precious value of a floating-point significand is given by:

$$z = (1.0 + a_1 2^{-1} + a_2 2^{-2} + \cdots + a_b 2^{-b} + a_{b+1} 2^{-b-1} + \cdots + a_{22} 2^{-22} + a_{23} 2^{-23}) \times 2^e \tag{8}$$

The floating-point representation is given by:

$$\hat{z} = (1.0 + a_1 2^{-1} + a_2 2^{-2} + \cdots + a_b 2^{-b}) \times 2^e \tag{9}$$

So the rounding error will be:

$$\varepsilon_{round} = \frac{z-\hat{z}}{z} = \frac{(a_{b+1}2^{-b-1} + \cdots + a_{22}2^{-22} + a_{23}2^{-23})}{(1.0 + a_1 2^{-1} + a_2 2^{-2} + \cdots + a_{23}2^{-23})} \approx \frac{p_{b+1}2^{-b-1} + \cdots + p_{22}2^{-22} + p_{23}2^{-23}}{1.0 + p_1 2^{-1} + p_2 2^{-2} + \cdots + p_{23}2^{-23}} \tag{10}$$

The arithmetic model for any floating-point add or multiply operation is the sum of these two errors given in Equations (3) and (10).

In the FDP unit the overall error is given by:

$$\varepsilon_{prop} = 3 \times \varepsilon_{prop} + \varepsilon_{round} \tag{11}$$

If the same operation is performed using discrete floating-point adders and multipliers, then the overall error will be given by:

$$\varepsilon_{prop} = 3 \times \varepsilon_{prop} + 3 \times \varepsilon_{round} \tag{12}$$

The above analysis shows that the FDP unit has one-third of the rounding error of the discrete execution.

The FDP unit and discrete floating point adder and multiplier were modeled in Matlab. The FFT butterfly operation was simulated using the single-precision FDP unit and then using discrete single-precision floating-point operations. Both were compared to the Matlab implementation using the built-in double precision operations. Fig. 8 and Fig. 9 show the error plots of the two approaches. The error values using the FDP unit were in the range of –1.7x10-5 to 1.6x10-5, while the error values using the discrete floating-point operations were in the range of –2.4x10-5 to 2.3x10-5 (about 40% higher).

IV. Verilog Modeling and Synthesis

To confirm the benefits of the fused dot-product unit, the following floating-point units were implemented in synthesizable Verilog-RTL:

- Conventional Floating-Point Adder
- Conventional Floating-Point Multiplier
- Fused Floating-Point Dot-Product Unit

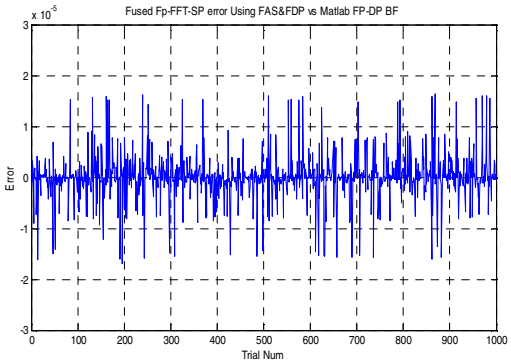

Figure 8. Error in FFT Butterfly Using the Fused Dot Product.

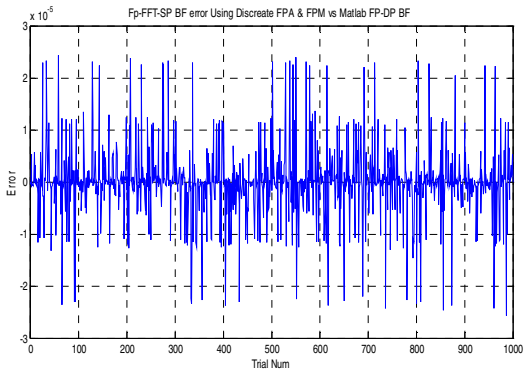

Figure 9. Error in FFT Butterfly Using Discrete FPMs and FPA.

The Verilog models were synthesized using 45nm libraries. The area and the critical timing paths were evaluated. All the units were designed to operate on single-precision IEEE Std-754 operands [11].

V. Place and Route

The floating-point adder, multiplier and the fused dot-product unit were implemented using an automatic synthesize, place and route approach. A high performance 45 nm process was used for the implementation with a standard cell library designed for high speed applications. Fig. 10 shows the floorplan of the FDP unit.

Table 1 shows the implementation data. The conventional floating-point multiplier occupies an area of 102 µm by 103 µm, while the fused dot-product unit occupies an area of 140 µm by 141 µm.

Table 1. Implementation Data.

	F-P Multiplier	Fused Dot Product
Format	IEEE 754 Single-Precision	
Standard Cell area	9,482 µm2	16,104 µm2
Height	102 µm	140 µm
Width	103 µm	141 µm
Utilization	90.4%	81.7%
Total wire length	70 mm	127 mm
Critical Timing Path	1,804 ps	2,721 ps
Dynamic Power	5068 µW	5839 µW
Leakage Power	808 µW	1366 µW
Total Power	5876 µW	7205 µW

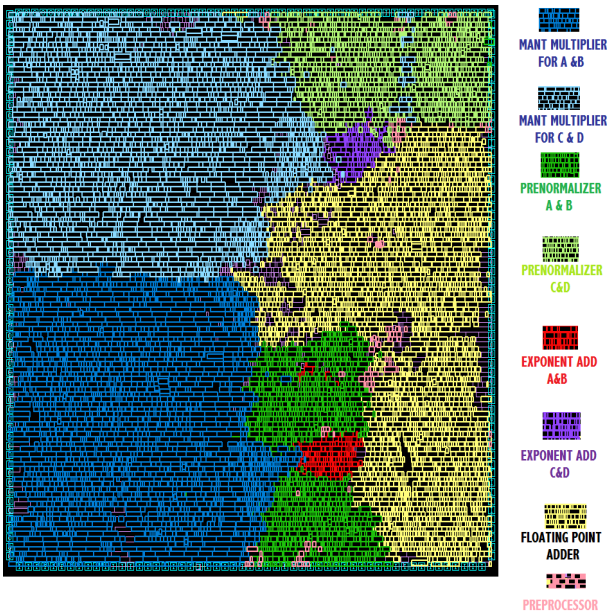

Figure 10. FDP Floorplan.

VI. Timing

The placed, routed and tapeout ready floating-point multiplier and fused dot-product units timing were analyzed using industry standard STA tools, with an extracted and back-annotated netlist. The floating-point multiplier performed a multiply operation in 1804 ps while the fused dot-product unit needed 2721 ps to perform the dot-product operation.

VII. Conclusions

The area and latency of the two conventional approaches (ignoring the multiplexers and register) and the FDP unit are compared in Table 2 and plotted in Fig. 11. The fused dot product is intermediate in area between the conventional serial and the conventional parallel approaches. Its latency is about 80% of that of the conventional parallel approach and about half that of the conventional serial approach.

Table 2. Comparison of Dot-Product Approaches.

Unit		Area (µm2)	Latency (ps)
F-P Adder		3,811	1,644
F-P Multiplier		9,482	1,804
Dot-Product	Conv. Parallel	22,775	3,448
	Conv. Serial	13,293	5,252
	Fused	16,104	2,721

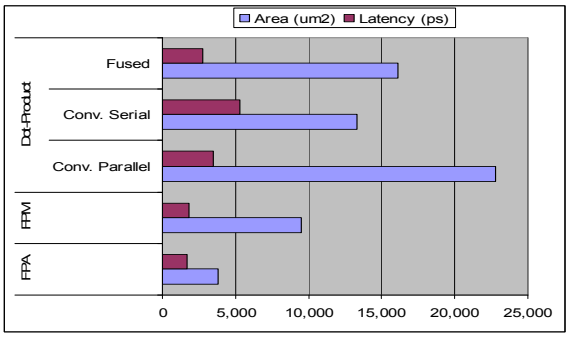

Figure 11. Area and Latency Comparison.

REFERENCES

[1] E. Hokenek, R. K. Montoye and P. W. Cook, "Second-generation RISC floating point with multiply-add fused," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1207-1213, 1990.

[2] D. Takahashi, "A radix-16 FFT algorithm suitable for multiply-add instruction based on Goedecker method," *Proceedings 2003 International Conference on Multimedia and Expo*, vol. 2, July 2003, pp. II-845-II-848.

[3] C. Xu, C. Y. Wang and K. K. Parhi, "Order-configurable programmable power-efficient FIR filters," *Proceedings 3rd International Conference on High Performance Computing*, December 1996, pp. 357-361.

[4] C. Hinds, "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications," *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, pp. 147-151, 1999.

[5] A. D. Robison, "N-Bit Unsigned Division Via N-Bit Multiply-Add," *Proceedings of the 17th IEEE Symposium On Computer Arithmetic*, pp. 131-139, 2005.

[6] R.-C. Li, S. Boldo and M. Daumas, "Theorems on Efficient Argument Reductions," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 129-136, 2003.

[7] F. P. O'Connell and S. W. White, "POWER3: The Next Generation of PowerPC Processors," *IBM Journal of Research and Development*, vol. 44, pp. 873-884, 2000.

[8] A. Kumar, "The HP PA-8000 RISC CPU," *IEEE Micro Magazine*, vol. 17, Issue 2, pp. 27-32, April, 1997

[9] B. Greer, J. Harrison, G. Henry, W. Li and P. Tang, "Scientific Computing on the Itanium Processor," *Proceedings of the ACM/IEEE SC2001 Conference*, pp. 1-8, 2001.

[10] M. P. Farmwald, *On the Design of High Performance Digital Arithmetic Units*, Ph.D. Thesis, Stanford University, 1981.

[11] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985.

[12] Z. Zhao and M. Leeser, "Precision Modeling and Bit-width Optimization of Floating-Point Applications," *MIT HPEC*, 2003.