NOVATEUR PUBLICATIONS
INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY [IJIERT]
ISSN: 2394-3696
VOLUME 2, ISSUE 3 MARCH2015

A Pipelined Fused Processing Unit for DSP Applications

Vinay Reddy N
PG student
Dept of ECE, PSG College of Technology, Coimbatore,

Hema Chitra S
Assistant professor
Dept of ECE, PSG College of Technology, Coimbatore,

Abstract

This paper designs a processing element for FFT processor capable of operating on 32-bit double precision floating point numbers. Pipelining is performed on the computational elements of the DSP processor to enhance the throughput. The performance of the Processing unit is increased by using the concept of fused architecture on the sub modules – the dot product unit and the add sub unit. Pipelining increases the speed of the CE of the processor while fused operations claim area optimization. The DSP applications involve FFT Processors that make use of the butterfly operations consisting of multiplications, additions, and subtractions of complex valued data (data is split into real part and the imaginary part). The radix-2 and radix-4 butterflies are designed using fused architecture. The fused FFT butterflies are to be 20 percent speedier and 30 percent smaller in area compared with the conventional method. The processing unit covers almost all the computations necessary for the processor.

INTRODUCTION

Most of the real time fields like medical fields involving biomedical signal values, communication fields involving transmission and reception of real valued data etc are floating point numbers. These values cannot be neglected, instead should be accurately recorded and processed. Floating point arithmetic serves to give a good dynamic range, freeing special purpose processor designers from the scaling and overflow/underflow concerns that arise with fixed-point arithmetic. Use of the IEEE-754 standard 32-bit floating-point format [1] facilitates the use of fast Fourier transform (FFT) processors as coprocessors in collaboration with general purpose processors. This paper is concerned with the design of a Processing unit that can compute FFT values necessary in almost all the digital signal processing applications. Two fused floating-point primitive operations were developed recently to reduce the delay and area of FFT computation units. The first primitive fused operation is a fused floating-point dot product unit (Fused DP). The Fused DP unit is an extension of the fused multiply-add (FMA) unit which was developed initially for the IBM RS/6000 processor and was recently added to IEEE Std-754 [1].

The FMA unit reduces the latency of a multiplication followed by an addition leading to the enhancement in the throughput. Also, a single FMA can be used to replace the floating-point adder and the floating-point multiplier in a system. Some DSP algorithms have been rewritten to take advantage of the presence of FMA units. For example, a radix-16 FFT algorithm that speeds up FFTs in systems with FMA units is described in[2].

The second of the fused operations is a fused floating-point add subtract unit (Fused AS). In FFT computations, both the sum and the difference of a pair of operands are needed frequently. In a fused implementation, the sum and the difference operations can share a substantial amount of operand alignment logic with the reduction of the circuit area.

In this paper, parallel implementations are considered as the baselines. In view of the limited precision required for most DSP applications, all implementations in this paper (discrete as well as fused) support only the 32-bit IEEE-754 standard format [1].

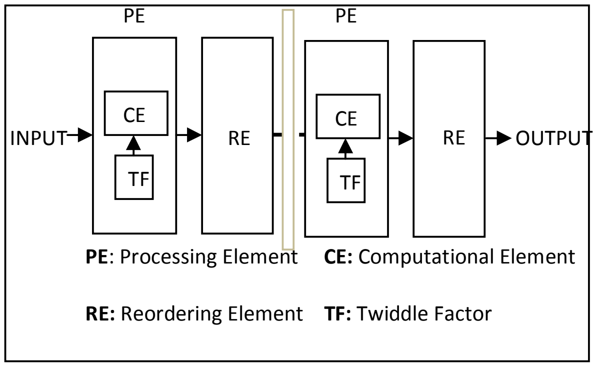


Fig.1. Pipelined FFT processor

FFT PROCESSOR FOR DSP APPLICATIONS

The DSP computations require the computation of the butterfly units, cascading of the butterfly units will build the basic architecture for FFT processing [5]. The basic structure is shown on Fig. 1. It consists of an interleaved cascade of two types of elements. The computational elements (identified as CE on the figure).The data reordering elements [6] (identified as RE on the figure). The data flow is in one direction along the heavy lines, which convey r complex data. The light lines carry r -1 complex twiddle factors from the TF units (identified as TF on the figure) to the computational elements. The twiddle factors may be computed or may be obtained from a memory.

In the data flow, the computational element is used first, then the two types of elements (RE and CE) alternate ending with a computational element. There are two types of computational elements (also called butterfly units). Decimation In Time (DIT) computational elements consist of complex multiplication(s) followed by a sum and difference network. Decimation In Frequency (DIF) computational elements consist of a sum and difference network followed by complex multiplication(s).

FUSED FLOATING POINT TWO-TERM DOT - PRODUCT UNIT

The floating-point two-term fused dot product (Fused DP) unit computes a two-term dot product

$$X = AB \pm CD \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Although a conventional dot product adds the two products, the Fused DP unit also allows forming the difference of the two products, which is useful in implementing complex multiplication. The Fused DP unit is based on the fused multiply-add unit. The Fused dot product unit is shown in Fig. 2. It adds a second multiplier tree, and a revised exponent comparison circuit to the conventional FMA. From the carry save adder onward the FDP and FMA are identical. There is a significant area reduction compared to a conventional parallel discrete implementation of two multipliers and an adder, since the rounding and normalization logic of both of the multipliers are eliminated.

In addition, the Fused DP unit provides slightly more accurate results because only one rounding operation is performed compared to the three rounding operations (one at the output of each multiplier and the other at the output of the adder) of the discrete implementation.

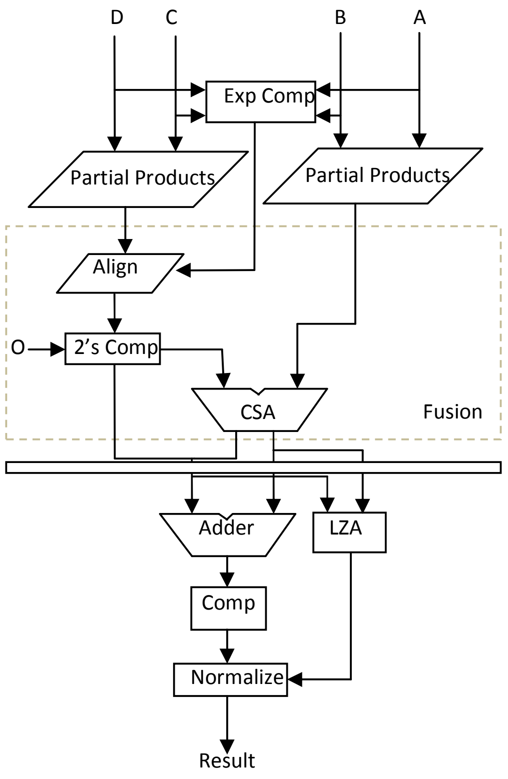


Fig.2. Floating-point fused dot-product unit

FUSED FLOATING POINT ADD-SUB UNIT

The floating-point fused add-subtract unit (Fused AS) performs an addition and a subtraction in parallel on the same pair of data

$$X=A+B \text{ and}$$
$$Y=A-B \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

The fused add-subtract unit is based on a conventional floating point adder [8]. A block diagram of the fused add subtract unit is shown in Fig.3. The exponent difference calculation, significand swapping, and the significand shifting for both add and subtract operations are performed with a single set of hardware and the results are shared by both the operations. This significantly reduces the required circuit area. The significand swapping and shifting is done based solely on the values of the exponents (i.e., without comparing the significand). As a result, if the exponents are equal, the smaller significand may be misidentified as the larger operand. Within the unit, it is advantageous to treat both operands as positive, in order to simplify the addition and the subtraction operation

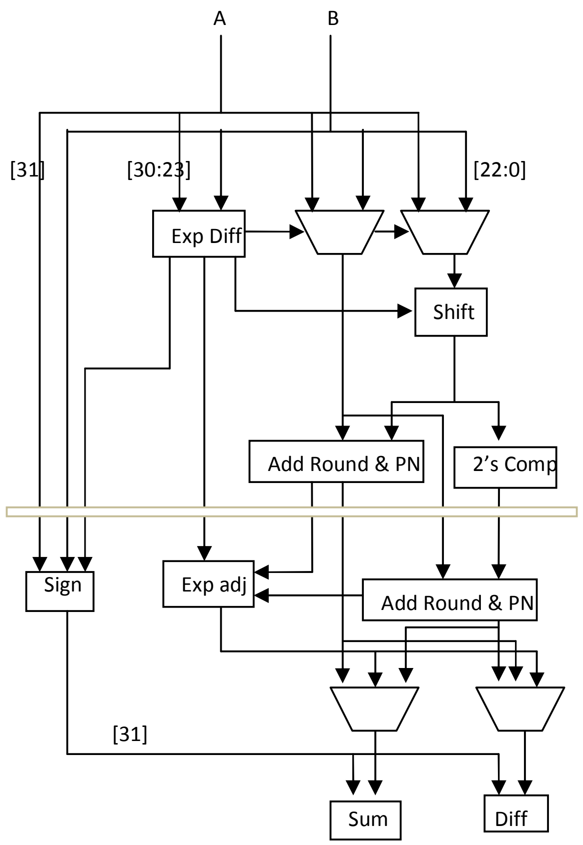


Fig.3. Floating-point fused add-subtract unit

RADIX – 2 FFT BUTTERFLY

To demonstrate the utility of the Fused DP and Fused AS units for FFT implementation, FFT butterfly unit designs using both the discrete and the fused units have been made. Firstly Radix-2 decimation in frequency FFT butterfly was designed, it is shown in Fig.4. All lines carry complex pairs of 32-bit IEEE-754 numbers and all operations are complex.

The complex add, subtract, and multiply operations shown in Fig.5 is realized with a discrete implementation that uses two real adders to perform the complex add or subtract and four real multipliers and two real adders to perform the complex multiply. The complete butterfly consists of six real adders and four real multipliers as shown on the figure. In this and the following figure, all lines are 32-bits wide for the IEEE-754 single-precision data. Alternatively, as shown in Fig.6, the complex add and subtract can be performed with two fused add-subtract units (marked as FAS in the figure) and the complex multiplication can be realized with two fused dot product units (marked as FDP).

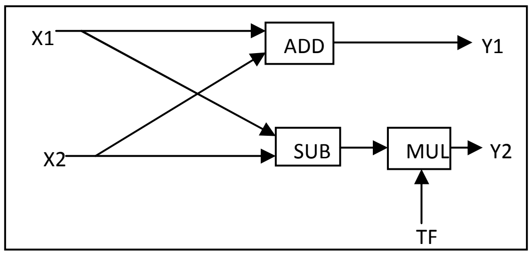


Fig.4. Radix-2 DIF FFT butterfly unit

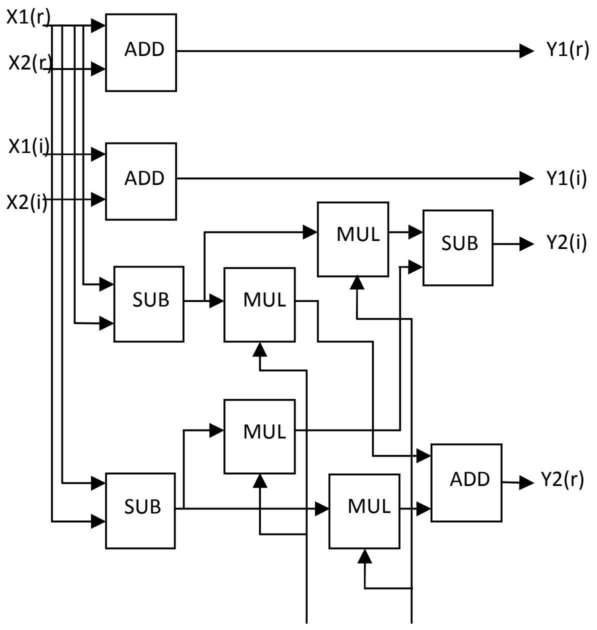


Fig.5. Discrete Implementation Of The Radix-2 Dif Fft Butterfly

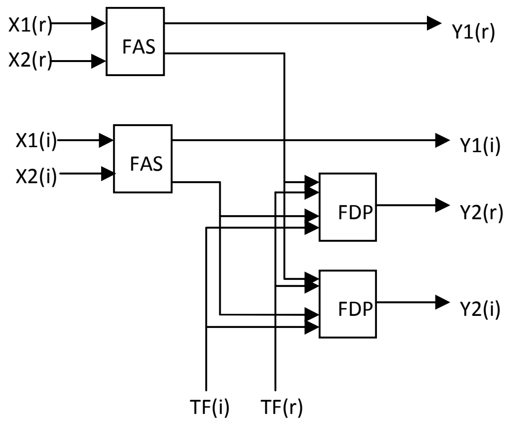


Fig.6. Fused implementation of the radix-2 DIF FFT butterfly

PROPOSED PROCESSING UNIT

The computational elements ie, the fused dot product unit and the fused add sub unit of the FFT processor is pipelined in order to perform the operations to achieve a higher throughput compared to the conventional architecture. The processing unit designed is able to perform 26 different computations and is shown in the Fig.7 below

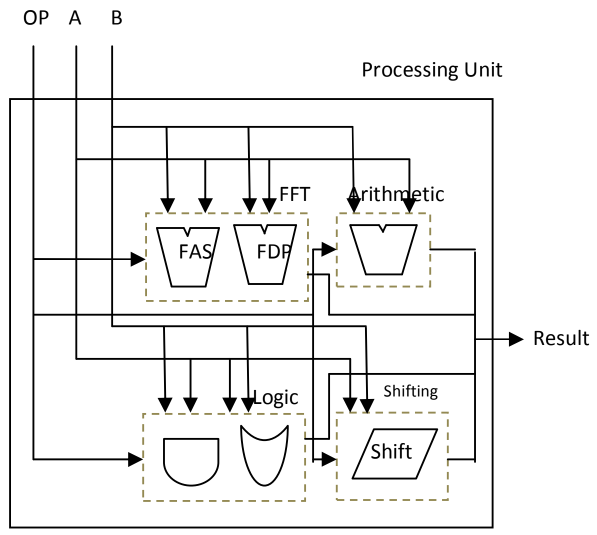


Fig.7. Processing Unit Design of the FFT Processor

The instructions handled by the processing unit is tabulated below

Table.1. Instructions handled by the Processing Unit

No	Opcode	Operation
1	00000	Reg <= a and b
2	00001	Reg <= a or b
3	00010	Reg <= not a
4	00011	Reg <= not b
5	00100	Reg <= a nor b
6	00101	Reg <= a nand b
7	00110	Reg<= a xor b
8	00111	Reg <= a xnor b
9	01000	Reg <= shft_l
10	01001	Reg<= shft_r;
11	01010	Reg <= rota_r
12	01011	Reg<= rota_l
13	01100	Reg<= a + b
14	01101	Reg<= a - b
15	01110	mlt <= a * b
16	01111	Reg <= a; ---- load Reg,A
17	10000	Reg <= b; ---- load Reg,B
18	10001	Temp_reg <= a; -load Reg_T,A
19	10010	Temp_reg <= b; -load Reg_T,b
20	10011	Temp_reg<= reg;-- mov temp,reg
21	10100	reg <=Temp_reg;-- mov reg,temp
22	10101	Reg <=f_add(a,b)
23	10110	Reg<=f_sub(a,b)
24	10111	Reg<=mult
25	11000	Two term Dot-product
26	11001	Radix-2 FFT

SIMULATION RESULTS

Simulation is performed using Modelsim 6.3E and the results are being projected

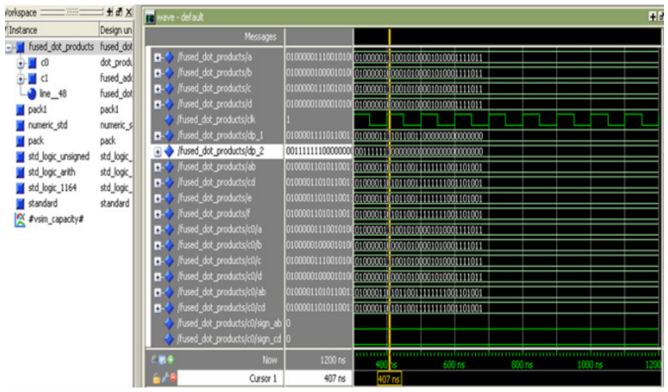

```
/fused_dot_products/a          01000001110010100
/fused_dot_products/b          01000001000010100
/fused_dot_products/c          01000001110010100
/fused_dot_products/d          01000001000010100
/fused_dot_products/clk        1
/fused_dot_products/dp_1       01000011110111001
/fused_dot_products/dp_2       00111111110000000
/fused_dot_products/ab         01000011010110011
/fused_dot_products/cd         01000011010110011
/fused_dot_products/e          01000011010110011
/fused_dot_products/f          01000011010110011
/fused_dot_products/c0/a       01000001110010100
/fused_dot_products/c0/b       01000001000010100
/fused_dot_products/c0/c       01000001110010100
/fused_dot_products/c0/d       01000001000010100
/fused_dot_products/c0/ab      01000011010110011
/fused_dot_products/c0/cd      01000011010110011
/fused_dot_products/c0/sign_ab 0
/fused_dot_products/c0/sign_cd 0
Now                            1200 ns
Cursor 1                       407 ns
```

Fig.8. Simulation result for Fused Dot – product unit

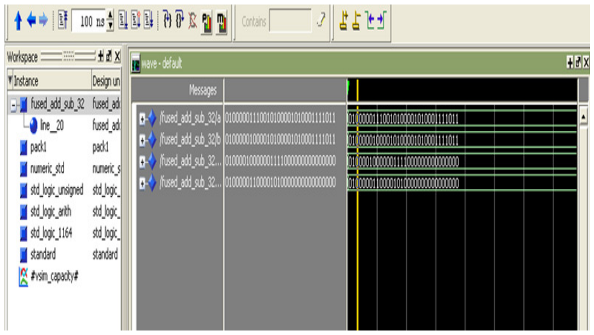

```
/fused_add_sub_32/a  01000001110010100001010001111011
/fused_add_sub_32/b  01000001000010100001010001111011
/fused_add_sub_32... 01000010000001111000000000000000
/fused_add_sub_32... 01000001100001010000000000000000
```

Fig.9. Simulation result for Fused Add - Sub unit

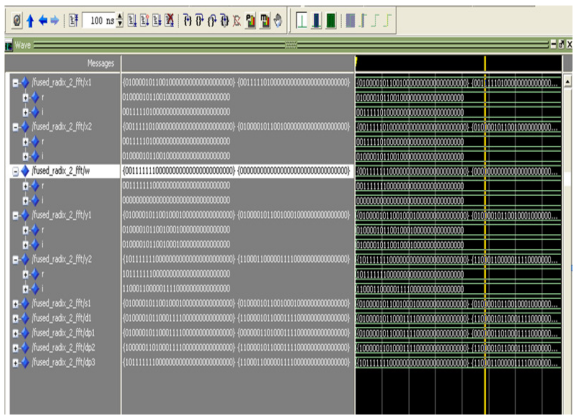
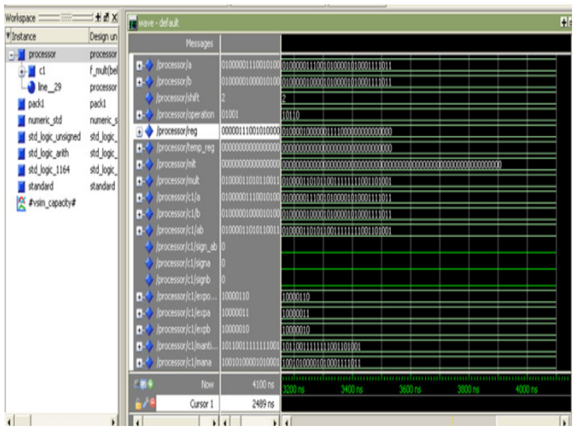

```
/fused_radix_2_fft/x1  {01000010110010000000000000000000} {00111110100000000000000000000000}
  r                    01000010110010000000000000000000
  i                    00111110100000000000000000000000
/fused_radix_2_fft/x2  {00111110100000000000000000000000} {01000010110010000000000000000000}
  r                    00111110100000000000000000000000
  i                    01000010110010000000000000000000
/fused_radix_2_fft/w   {00111111100000000000000000000000} {00000000000000000000000000000000}
  r                    00111111100000000000000000000000
  i                    00000000000000000000000000000000
/fused_radix_2_fft/y1  {01000010110010001000000000000000} {01000010110010001000000000000000}
  r                    01000010110010001000000000000000
  i                    01000010110010001000000000000000
/fused_radix_2_fft/y2  {10111111100000000000000000000000} {11000110000011110000000000000000}
  r                    10111111100000000000000000000000
  i                    11000110000011110000000000000000
/fused_radix_2_fft/s1  {01000010110010001000000000000000} {01000010110010001000000000000000}
/fused_radix_2_fft/d1  {01000010110001111000000000000000} {11000010110001111000000000000000}
/fused_radix_2_fft/dp1 {01000010110001111000000000000000} {00000011010001111000000000000000}
/fused_radix_2_fft/dp2 {10000011010001111000000000000000} {11000010110001111000000000000000}
/fused_radix_2_fft/dp3 {10111111100000000000000000000000} {11000110000011110000000000000000}
```

Fig.10. Simulation result for Radix- 2 FFT unit using the fused architecture



```
Workspace
Instance             Design un
processor            processor
  c1                 f_mult(be
  line__29           processor
pack1                pack1
numeric_std          numeric_s
std_logic_unsigned   std_logic_
std_logic_arith      std_logic_
std_logic_1164       std_logic_
standard             standard
#vsim_capacity#

wave - default
Messages
/processor/a              0100000111001010100  01000001110010100001010001111011
/processor/b              0100000100001010100  01000001000010100001010001111011
/processor/shift          2                    2
/processor/operation      01001                10110
/processor/reg            000001110010100000   01000010000001111000000000000000
/processor/temp_reg       0000000000000000     00000000000000000000000000000000
/processor/mlt            0000000000000000     0000000000000000000000000000000000000000000000000000000000000000
/processor/mult           0100001101011001 1   01000011010110011111111001101001
/processor/c1/a           0100000111001010100  01000001110010100001010001111011
/processor/c1/b           0100000100001010100  01000001000010100001010001111011
/processor/c1/ab          0100001101011001 1   01000011010110011111111001101001
/processor/c1/sign_ab     0
/processor/c1/signa       0
/processor/c1/signb       0
/processor/c1/expo...     10000110             10000110
/processor/c1/expa        10000011             10000011
/processor/c1/expb        10000010             10000010
/processor/c1/manti...    1011001111111 1001   10110011111111001101001
/processor/c1/mana        1001010000101 0001   10010100001010001111011
Now                       4100 ns              3200 ns   3400 ns   3600 ns   3800 ns   4000 ns
Cursor 1                  2489 ns
```

Fig.11. Simulation result for floating point multiplication operation (Opcode = 10111) – Processing unit

Table.2. Performance summery of the design

Module	Parameter	Conventional	Pipelined
Fused Add-SUB unit	Area (gates)	40,147	37,291
	Throughput (1/ns)	0.083	0.22
Fused Dot-Product unit	Area (gates)	63,704	53,608
	Throughput (1/ns)	0.067	0.31

CONCLUSION

This paper describes the design of two new fused floating-point arithmetic units and their application to the implementation of FFT butterfly operations. Although the fused add-subtract unit is specific to FFT applications, the fused dot product is applicable to a wide variety of signal processing applications. Both the fused dot product unit and the fused add-subtract unit are smaller than parallel implementations constructed with discrete floating-point adders and multipliers. The fused dot product is faster than the conventional implementation, since rounding and normalization is not required as a part of each multiplication. Due to longer interconnections, the fused add-subtract unit is slightly slower than the discrete implementation. The fused FFT butterflies were found to be 20 percent speeder and 30 percent smaller in area compared with the conventional method from table 2. Also the processing unit covers almost all the computations necessary for the processor.

REFERENCES

[1] *IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, Aug. 2008.*

[2] R.K. Montoye, E. Hokenek, and S.L. Runyon, *"Design of the IBM RISC System/6000 Floating-Point Execution Unit," IBM J. Research and Development, vol. 34, pp. 59-70, 1990.*

[3] E. Hokenek, R.K. Montoye, and P.W. Cook, "*Second-Generation RISC Floating Point with Multiply-Add Fused," IEEE J. Solid-State Circuits, vol. 25, no. 5, pp. 1207-1213, Oct. 1990.*

[2] D. Takahashi, "*A Radix-16 FFT Algorithm Suitable for Multiply-Add Instruction Based on Goedecker Method," Proc. Int'l Conf. Multimedia and Expo, vol. 2, pp. II-845-II-848, July 2003.*

[5] J.H. McClellan and R.J. Purdy, "*Applications of Digital Signal Processing to Radar," Applications of Digital Signal Processing, A.V. Oppenheim, ed., pp. 239-329, Prentice-Hall, 1978.*

[6] B. Gold and T. Bially, "*Parallelism in Fast Fourier Transform Hardware," IEEE Trans. Audio and Electroacoustics, vol. AU-21, no. 1, pp. 5-16, Feb. 1973.*

[7] H.H. Saleh and E.E. Swartzlander, Jr., "*A Floating-Point Fused Dot-Product Unit," Proc. IEEE Int'l Conf. Computer Design (ICCD), pp. 427-431, 2008.*

[8] M.P. Farmwald, "*On the Design of High-Performance Digital Arithmetic Units," PhD thesis, Stanford Univ., 1981.*

[9] P.-M. Seidel and G. Even, "Delay-*Optimized Implementation of IEEE Floating-Point Addition," IEEE Trans. Computers, vol. 53, no. 2, pp. 97-113, Feb. 2004.*

[10] H. Saleh and E.E. Swartzlander, Jr., "A *Floating-Point Fused Add-Subtract Unit," Proc. IEEE Midwest Symp. Circuits and Systems (MWSCAS), pp. 519- 522, 2008.*

[11] H.H. Saleh, "*Fused Floating-Point Arithmetic for DSP," PhD dissertation, Univ. of Texas, 2008.*

[12] Earl E. Swartzlander Jr and Hani H.M. Saleh, "*FFT Implementation with Fused Floating-Point Operations" , 2012.*