IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 2, Ver. I (Mar. -Apr. 2016), PP 58-65
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Fused Floating Point Arithmetic Unit for Radix 2 FFT Implementation

Prasanna Palsodkar[1], Ajay Gurjar[2]
[1](Department of Electronics Engineering, Yeshwantrao Chavan College of Engineering, Nagpur, India)
[2](Department of Electronics & Telecomm. Engineering, Sipna's College of Engineering & Technology, Amravati, India)

Abstract : *This paper describes the design and implementation of user defined fused floating-point arithmetic operations that can be used to implement Radix 2 butterfly Fast Fourier Transform (FFT) for complex numbers used in Digital Signal Processing (DSP- C) processors. This paper reports the comparison of area, delay and power of fused floating point modules as compared to discrete floating point with reference to Radix 2 butterfly structure. The design is implemented and simulated by targeting Xilinx vertex 5 FPGA device. Here we have achieved reduction in area (in terms of LUT required) by 27.09%, reduced delay by 17.10%, reduction in power consumption by 11 % and energy is reduced by 26.22% as compared to discrete implementation.*

Keywords: *Floating point, DSP, Area, Power, Delay, FFT*

I. Introduction

In past, many Digital Signal Processing (DSP) applications used fixed point arithmetic due to the high cost (in delay, silicon area, and power consumption) of floating-point arithmetic units [8]. Floating-point arithmetic is much useful in the implementation of various DSP applications as it allows the designer and the user to concentrate on the algorithms and architecture without worrying about the numerical issues [1, 6]. Many applications use floating-point hardware to perform DSP tasks in real time and hence overcome the limitations imposed by the use of fixed- point numeric systems. In realization of modern general purpose processors, fused floating-point multiply add module have become attractive since their delay and silicon area is often less than that of a discrete floating-point multiplier followed by a floating point adder. Further the accuracy is improved by the fused implementation since rounding is performed only once (after the multiplication and addition) [3]. This work extends the consideration of fused floating-point arithmetic operations that are frequently encountered in DSP applications [6]. The Fast Fourier Transform is a case in consideration since it uses a complex butterfly operation. For a radix-2 implementation, the butterfly consists of complex operations of multiplication, addition and subtraction of the same pair of data. This butterfly operations can be implemented with fused primitives, i.e Fused Two-Term Dot-Product Module, and Fused Add-Subtract module [1].

II. Related Work

This section introduces fixed-point computer arithmetic and its limitations, the IEEE-754 floating-point standard, and current usage of combined (fused) arithmetic functions, a quick introduction to the Fast Fourier Transform (FFT), floating- point and FFT error analysis.

II.I DSP processors Arithmetic Overview

DSP processors arithmetic is concerned with the hardware realization of mathematical formulas, algorithms and complex models from a theoretical world. Hardware functions calculate arithmetic's in both fixed-point and floating-point (Scientific Notations) [6].

II.II IEEE-754 Floating Point Standard

The IEEE Standard for Floating-Point Arithmetic (IEEE-754) is a technical standard established by the Institute of Electrical and Electronics Engineers [5]. It is the most ubiquitous standard for floating-point computations representation in today's microprocessors, including Intel-based Processor's, Macintoshes and UNIX platforms. IEEE floating point numbers have three basic components: a sign, an exponent and a significant. The significant is composed of the fraction and an implicit leading digit. The exponent base (2) is implicit and is not stored [4]. This standard specifies the basic types of representation like

- Half Precision (16-bits or 2-bytes)
- Single Precision (32-bits or 4-bytes)

- Double Precision (64-bits or 8-bytes)

TABLE 1 shows the Bit-wise distribution of Floating point number.

TABLE 1:- Floating Point Bit wise distribution

Types of Representation	Sign	Exponent	Fraction
Half Precision	1[15]	5 [14-10]	10 [9-0]
Single Precision	1 [31]	8 [30-23]	23 [22-0]
Double Precision	1 [63]	11[62-52]	52 [51-0]

II.III Overview of Floating-Point Fused Multiply-Add (FMA) operation

In 1990, IBM introduced the floating-point fused multiply-add operation on the RISC System 6000 (IBM RS/6000) chip [1]. IBM recognized that several advanced applications, specifically those with dot products are routinely performed with a floating- point multiplication immediately followed by a floating-point addition i.e. (A x B) + C, ad infinitum. To increase the performance of these applications, a new module was created that merged a discrete floating-point multiplier and floating-point adder into a single hardware block: the floating-point fused multiply-add (FMA) module. This floating-point arithmetic module executes the equation (A x B) + C in a single instruction.

With the continued demand for 3D graphics, multimedia applications and new advanced processing algorithms, IEEE has included the fused multiply-add operation into the IEEE 754-2008 standard [5]. Even though the fused multiply-add architecture has troublesome latencies, high power consumption and performance degradation with single-instruction execution, more and more microprocessor designs implement floating- point fused multiply-add module in their silicon.

II.IV Fast Fourier Transform (FFT) Algorithm

Fourier analysis is a family of mathematical techniques, based on decomposing signals into sinusoids. The Discrete Fourier Transform (DFT) is used with digitized signals [13]. The DFT of a sequence of N complex numbers is given by

$$Xk = \sum_{n=0}^{N-1} x_n e^{2\pi i/N}, k = 0,1,....N-1 \tag{1}$$

Fast Fourier Transform (FFT) is an efficient method for calculating the DFT. Even if it produces the same result as the other approaches, it often reduces the computation time by a factor of ten or more for large sequences [14].

There are two types of the FFT algorithm: Decimation in Time (DIT) where the time domain sequence is split into even and odd parts for processing and Decimation in Frequency (DIF) where the frequency components are divided into even and odd parts for processing. Both the DIT and DIF can accept inputs either in order or in bit reversed order to produce bit reversed or in order outputs respectively.

Fig. 1 shows the radix-2 DIT FFT and DIF FFT butterflies, which are the basic computation element in performing the FFT. Fig. 2 shows the data flow diagram for performing a radix-2 DIT FFT and Fig. 3 shows the data flow diagram for performing a radix-2 DIF FFT.

The X0-X8 are the input data samples, W_N^K is the twiddle factor for butterflies which is given by equation:

$$W_N^K = e^{-i2\pi K/N} = \cos(2\pi K/N) - i\sin(2\pi K/N) \tag{2}$$

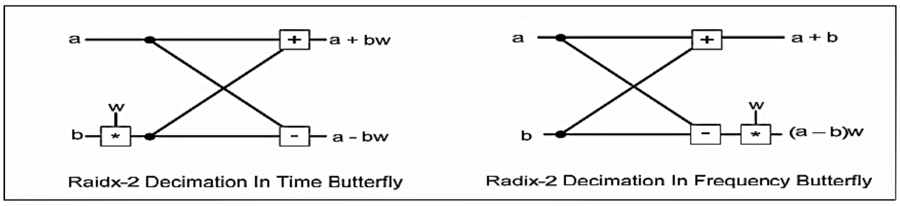

a
b
w
*
+
-
a + bw
a - bw
Raidx-2 Decimation In Time Butterfly
a
b
+
-
*
w
a + b
(a – b)w
Radix-2 Decimation In Frequency Butterfly

Figure 1: Radix -2 Butterfly Structure

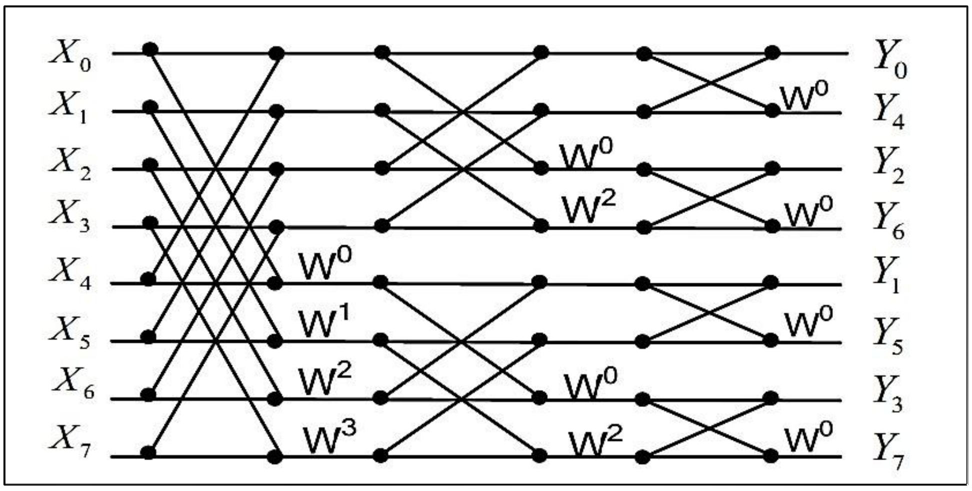


Figure 2: 8-Point Radix -2 DIT FFT Using Butterfly Structure

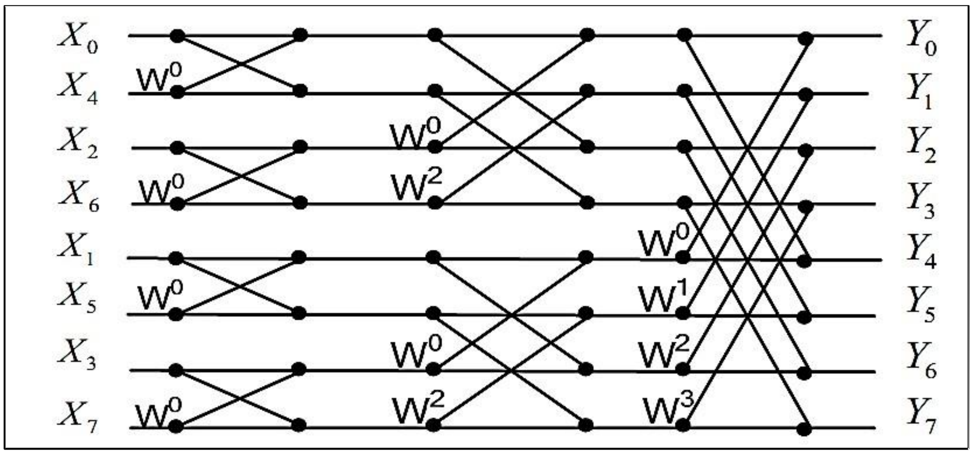


Figure 3: 8-Point Radix -2 DIF FFT Using Butterfly Structure

III. Fused Floating Point Modules

This section describes three proposed Fused Floating Point Arithmetic (FFPA) modules i.e. Fused Floating Point Add-Subtract (FFPAS) module, Fused Floating Point Multiply-Add (FFPMA) module and Fused Floating Point Two-Term Dot-Product (FFP2TDP) module.

III. I Fused Floating Point Add-Subtract (FFPAS) Module

Discrete Floating-Point Adder (FPA) and Floating-Point Subtractor (FPS) can be designed using two approaches as follows:

- Parallel implementation where two adders operate in parallel
- Serial implementation where a single adder is used twice with the same operands.

The architecture of the proposed fused floating point add-subtract (FFPAS) module is derived from the floating-point add module. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both add and the subtract operations. Fig. 4 shows the flowchart of Proposed FFPAS module.

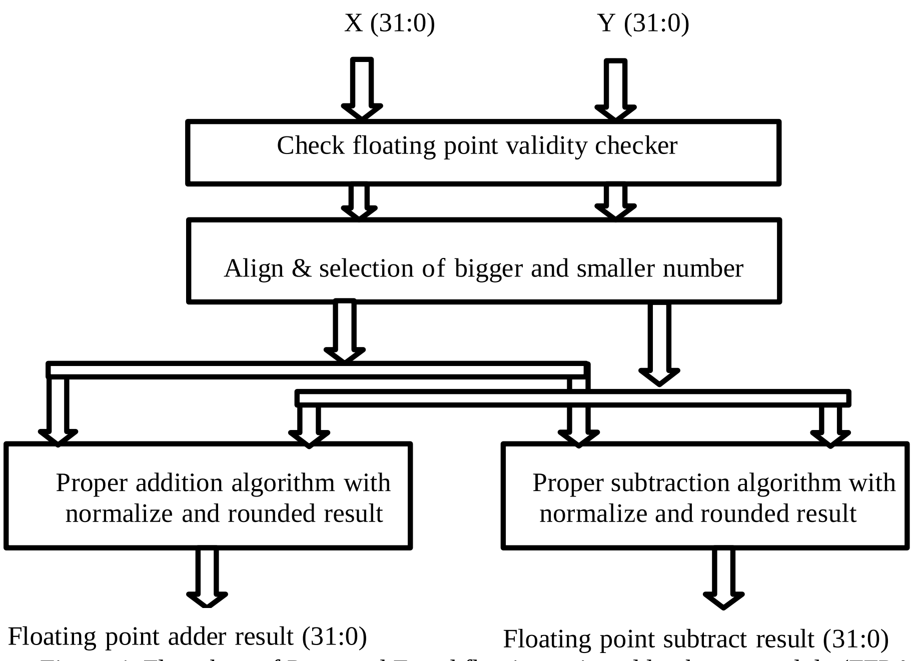

X (31:0)
Y (31:0)
Check floating point validity checker
Align & selection of bigger and smaller number
Proper addition algorithm with normalize and rounded result
Proper subtraction algorithm with normalize and rounded result
Floating point adder result (31:0)
Floating point subtract result (31:0)

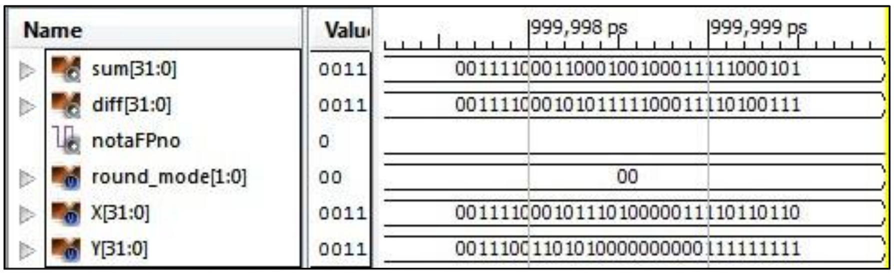
Figure 4: Flowchart of Proposed Fused floating point add-subtract module (FFPAS)

TABLE 2:- Testing Input for Proposed FFPAS module

Input	32 Bit Floating Point	Equivalent Floating Point Number
X	32'b0__0111_1000__1011_1010_0000_1111_0110_110	1.3490607e-2
Y	32'b0__0111_0011__0101_0000_0000_0011_1111_111	3.2044944e-4

X + Y = (+1.3490607e-2)+(+3.2044944e-4)= +1.381105644e-2
X – Y = (+1.3490607e-2)-(+3.2044944e-4)= +1.317015756e-2

TABLE 2 indicates the testing inputs for the proposed FFPAS module and the manual calculations done for the inputs. Figure 5 shows the timing simulation of FFPAS module. From the timing simulation of FFPAS module the output sum is (X + Y) = +1.3811056e-2 and difference is (X - Y) = +1.3170158e-2, which are nearly same as that of manual calculation



```
Name              Valu   |999,998 ps          |999,999 ps
sum[31:0]         0011   00111100011000100100011111000101
diff[31:0]        0011   00111100010101111100011110100111
notaFPno          0
round_mode[1:0]   00     00
X[31:0]           0011   00111100010111010000011110110110
Y[31:0]           0011   00111001101010000000000111111111
```

Figure 5: Timing Simulation waveform of Proposed FFPAS module

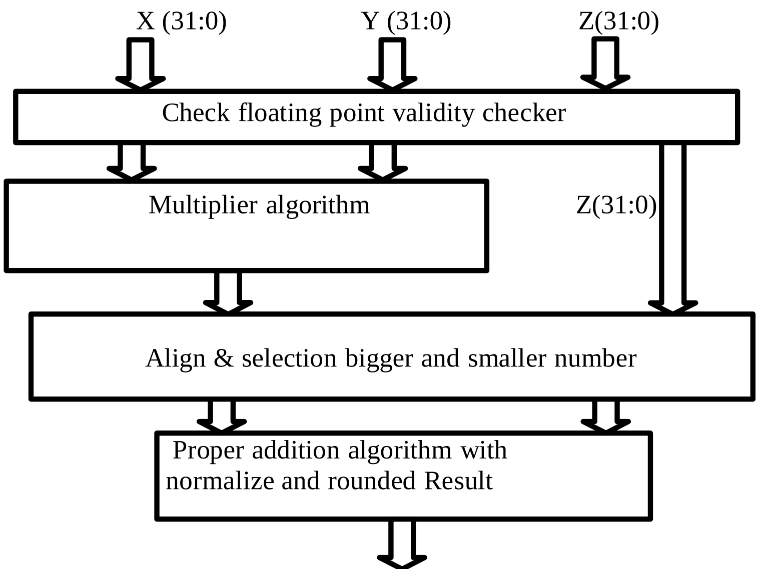
III. II Fused Floating Point Multiply-Add (FFPMA) Module

The architecture of the proposed Fused Floating Point multiply-add (FFPMA) module is derived from the floating-point add and multiplier module. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both the multiply and the add operations. Fig. 6 shows the flowchart of Proposed Fused Floating Point Multiply-Add Module (FFPMA). TABLE 3 shows the testing input for proposed FFPMA module.

TABLE 3:- Testing Input for Proposed FFPMA module

Input	**32 Bit Floating Point**	**Equivalent Floating Point Number**
X	32'b 0 0110_0000 1010_0000_1101_1110_0000_111	7.5827738e-10
Y	32'b 0 0111_0000 0101_1111_0000_0010_0000_101	4.1843410e-5
Z	32'b 0 0101_0001 0111_0001_0000_1010_0110_100	2.0485871e-14

X*Y = (+7.5827738e-10) * (+ 4.1843410e-5) = +3.172891131e-14
(X*Y) + Z = (+3.172891131e-14) + (+ 2.0485871e-14) = +5.221478231e-14


X (31:0)
Y (31:0)
Z(31:0)
Check floating point validity checker
Multiplier algorithm
Z(31:0)
Align & selection bigger and smaller number
Proper addition algorithm with normalize and rounded Result

Floating point multiply-adder result (31:0)
Figure 6 Flowchart of Proposed Fused floating multiply- add module (FFPMA)

Fig. 7 shows the timing simulation of proposed FFPMA module. Also the output (Mul-Add = +5.2413355e-14) from the timing simulation and manual calculations are same.

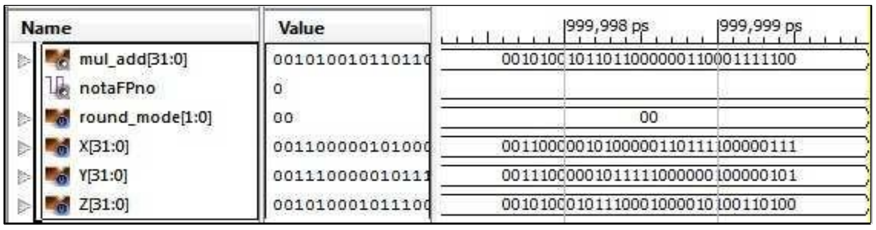

```
Name              Value              |999,998 ps     |999,999 ps
mul_add[31:0]     001010010110110    0010100 10110110000001100 01111100
notaFPno          0
round_mode[1:0]   00                 00
X[31:0]           001100000101000    0011000 0010100000110111 100000111
Y[31:0]           001110000010111    0011100 0001011111000000 100000101
Z[31:0]           001010001011100    0010100 0101110001000010 100110100
```

Figure 7 Timing Simulation waveform of Proposed FFPMA module

III.III Fused Floating Point Two-Term Dot Product (FFP2TDP) Module

The architecture of the proposed fused Floating Point two term dot product (FFT2TDP) module is derived from the floating-point add and multiplier module. Input A and input B multiply and input C and input D multiply after adjusting result of multiplication. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both the multiply and the add operations. Figure 8 shows Flowchart of Fused Floating Point two-term Dot-Product (FFP2TDP) module.

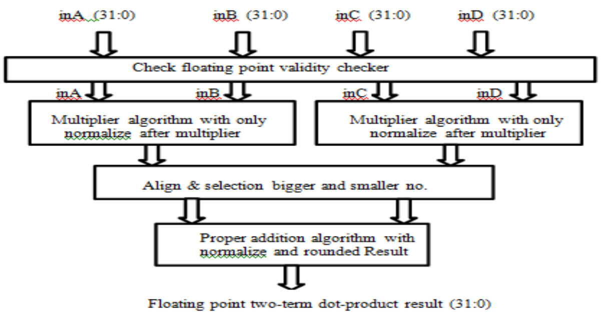

inA (31:0)
inB (31:0)
inC (31:0)
inD (31:0)
Check floating point validity checker
inA
inB
inC
inD
Multiplier algorithm with only normalize after multiplier
Multiplier algorithm with only normalize after multiplier
Align & selection bigger and smaller no.
Proper addition algorithm with normalize and rounded Result
Floating point two-term dot-product result (31:0)

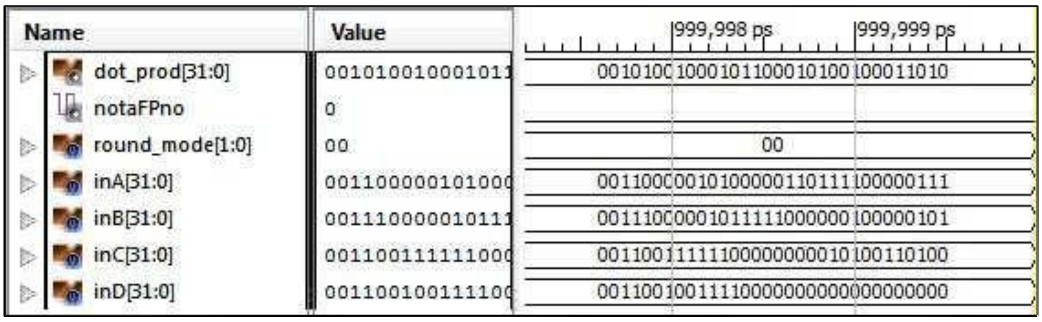
Figure 8 Flowchart of Proposed Fused floating point Two-term Dot Product (FFP2TDP) module

TABLE 4:- Testing Input for Proposed FFP2TDP module

Input	32 Bit Floating Point	Equivalent Floating Point Number
in A	32'b0__0110_0000__1010_0000_1101_1110_0000_111;	7.5827738e-10
in B	32'b0__0111_0000__0101_1111_0000_0010_0000_101;	4.1843410e-5
in C	32'b0__0110_0111__1110_0000_0000_1010_0110_100;	1.1176817e-7
in D	32'b0__0110_0100__1111_0000_0000_0000_0000_000;	1.4435499e-8

inA * inB = (+7.5827738e-10) * (+4.1843410e-5) = +3.172891131e-14
inC * inD = (+1.1176817e-7) * (+1.4435499e-8) = +1.613429306e-15
(inA*inB) + (inC*inD) = (+3.172891131e-14) + (+1.613429306e-15) = +3.334234061e-14

TABLE 4 shows the testing input for the proposed fused floating point 2 term dot product module. Fig. 9 shows the timing simulation for the proposed fused floating point 2 term dot product module and the calculation from the simulations is Two Term Dot Product = +3.3342340e-14 which appears to be equal to the manual calculations.



```
Name              Value               |999,998 ps              |999,999 ps
dot_prod[31:0]    00101001000101l     0010100 1000101100010100 100011010
notaFPno          0
round_mode[1:0]   00                  00
inA[31:0]         001100000010100     001100 0010100000110111 100000111
inB[31:0]         001110000001011     001110 0001011111000000 100000101
inC[31:0]         001100111111000     001100 1111100000000010 100110100
inD[31:0]         001100100111100     001100 1001111000000000 000000000
```

Figure 9 Timing Simulation waveform of proposed FFP2TDP module

III.IV Analysis of Proposed Fused Modules

TABLE 5 shows the analysis of the proposed fused models in terms of area, delay, power and energy.

TABLE 5:- Analysis of proposed fused modules

Name of module	LUT (Area)	DSP4 8E	IOBS	DELAY (ns)	POWER (mW)	ENERGY (nJ)
ud_beh_str_add_sub	975	--	131	29.02	719.94	20.89
ud_beh_str_mul_add	739	2	131	39.96	692.27	27.67
ud_beh_str_2term_dot_prod	926	4	163	39	715.68	27.91

IV. Proposed Fused Floating Point Radix 2 FFT Butterfly Structure

In this section the implementation of radix-2 FFT butterfly structure is discussed. This Radix 2 butterfly structure is designed by using Proposed fused floating point add-Subtractor (FFPAS), multiply-add (FFPMA) and two-term dot-product (FFP2TDP) module as show in Fig 10. After analysis of Discrete Implementation [13] and Proposed Fused implemented modules as shown in Table 6 it is observed that discrete radix-2 butterfly structure require three Adders, three Subtractor and four Multiplier whereas fused floating point radix-2 butterfly structure require only two Fused FP Add-Subtract and two Fused FP Two-Term Dot Product module.

TABLE 6:- Comparison of Fused and discrete FP modules for radix-2 butterfly structure

Name of Module	**LUT (Area)**	**DSP 48E**	**IOBs**	**Delay (ns)**	**Power (mW)**	**Energy (nJ)**
Discrete Radix2 Butterfly Structure	4799	8	323	73.089	1138.37	83.20
Proposed Fused Radix2 Butterfly Structure	3499	8	323	60.594	1013.15	61.39

TABLE 6 shows the comparative analysis of proposed fused floating point radix-2 FFT butterfly structure and the discrete implementation of radix-2 butterfly structure. From the table 6, it is clear that the proposed fused FFT structure is efficient in terms of area, delay, power and energy required for processing the signals.

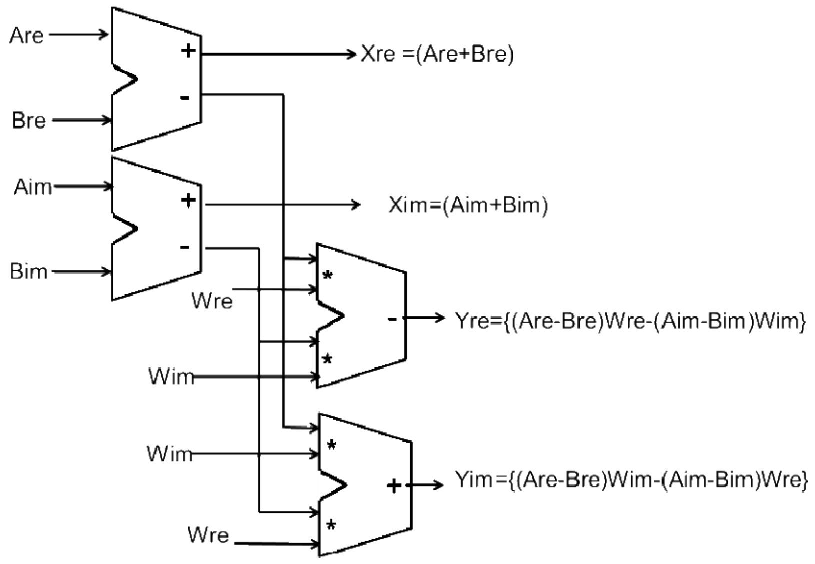

Are
Bre
Aim
Bim
Wre
Wim
Wim
Wre
+
-
+
-
*
*
-
*
*
+
Xre =(Are+Bre)
Xim=(Aim+Bim)
Yre={(Are-Bre)Wre-(Aim-Bim)Wim}
Yim={(Are-Bre)Wim-(Aim-Bim)Wre}

Figure 10 Proposed Fused Floating Point implementation of Radix 2 FFT butterfly structure

V. Conclusion

The Prior art to realize floating-point DSP hardware falls into one of the two categories: a serial approach used for applications with low area, power and energy budgets, while for applications that need to achieve high speed processing, the parallel approach is used with large increase in the area and power consumption. The proposed fused architectures have been specifically designed to address the problems of high latency, area and power consumption for the floating- point implementation of DSP algorithms. The implementation results using industry standard process and an automatic synthesis FPGA implementation flow shows that the fused primitives are faster, smaller, uses less power and energy than discrete implementation of Radix2 butterfly and provide more accurate result. In implementation of the proposed fused modules, area required is reduced by 27.09%, delay is reduced by 17.10%, it consumes 11 % less power and requires 26.22% less energy as compared to discrete implementation of Radix 2 butterfly structure.

References

[1]. E.E. Swartzlander, Jr. and H. H. Saleh, "FFT Implementation with Fused Floating-Point Operations," IEEE Transactions on Computers, *, 61(2),* 2012, 284-288.

[2]. JongwookSohn and Earl E. Swartzlander, Jr., "Improved Architectures for a Fused Floating- Point Add-Subtract Unit" IEEE Transactions on circuits and systems-I, Vol. 59, no. Nov, 2012

[3]. 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, Dec 2008.

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

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

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

[7]. Stuart Franklin Oberman, Design Issues in High Performance Floating Point Arithmetic Units, Ph. D. Dissertation, Stanford University, 1996.

[8]. R.K.Montoye, E.Hokenek and S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit,"

IBM Journal of Research and Development, Vol. 34, pp. 59-70, 1990.
[9]. IEEE Standard for Verilog Hardware Description Language, IEEE 1364-1995.
[10]. IEEE Standard for Verilog Language, IEEE 1364-2001.
[11]. http://www.xilinx.com/FPGA_series[Online]
[12]. Lecture notes - Chapter 7 - Floating Point Arithmetic[Online] http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html
[13]. The FFT Demystified, Engineering Productivity Tools Ltd., [Online]: Available: http://www.engineeringproductivitytools.com/
[14]. IEEE 754 Standard - Binary Floating Point Number Calculator Convert a 32 Bit Word to Decimal Value [Online] http://www.ajdesigner.com/fl_ieee_754_word/ieee_32_bit_word.php