INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 4)
Available online at: www.ijariit.com

Radix-4 DIT FFT implementation using Fused Floating-Point Arithmetic Unit

Rajeev Gowda B R
rajeevgowda1234@gmail.com
Banglore Institute of Technology, Bengaluru, Karnataka

Dr. A.B Kalpana
abkalpana@gmail.com
Banglore Institute of Technology, Bengaluru, Karnataka

ABSTRACT

The paper presents a hardware implementation of radix-4 DIT FFT butterfly-unit using Fused Floating-point Arithmetic Unit (FFAU) technique. The proposed FFAU is more efficient in area and delay than the primitive floating-point arithmetic operation. The radix-4 DIT FFT using FFAU is designed and synthesized in cadence using 45nm technology. The non-pipelined conventional architecture of FFT operates in 6 MHz whereas proposed FFT architecture operates on 10 MHz frequency. The outcome area is 46% efficient than the conventional FFT architecture. The 16 point DIT FFT is also implemented on the same proposed FFAU, to ensure the computation speed.

Keywords— *FFT, Radix-4 DIT Butterfly unit, Fused Floating-Point Arithmetic Unit*

1. INTRODUCTION

Now a day's FFT processor as a sub-processor with main-processor on a chip, to ensure a signal computation with fast, minimum area and minimum power. FFT processor with pipeline idea helps to unstop the main processor with the paralleled execution of the sequence of information signals. As of fixed-point computation is not a dynamic range, most of the shifted bits are lost. To avoid such computation by introducing floating –point computation [1]. In [3] IEEE-754 floating-point with a single-precision, having a dynamic range of computation and avoiding information loss. The idea of introducing fused floating-point computational elements in FFT implementation plays an important role in the complexity of computation speed and area and power [2].In [4][5] computational elements of FDP, FAS, and fused multiply and add (FMA) are exist in FFT processor library, but these are sufficient to only when two-term computation requires each stage rounding and alignment. But multi-term computation requires a final rounding and final alignment. The impact of the final rounding and final alignment leads to less area and also the fast execution [7]. Hence for multi-term computation needs an efficient algorithm that is proposed in this paper.

2. DESIGN AND IMPLEMENTATION

2.1 Design and implementation of a radix-4 DIT FFT butterfly unit using FFAU

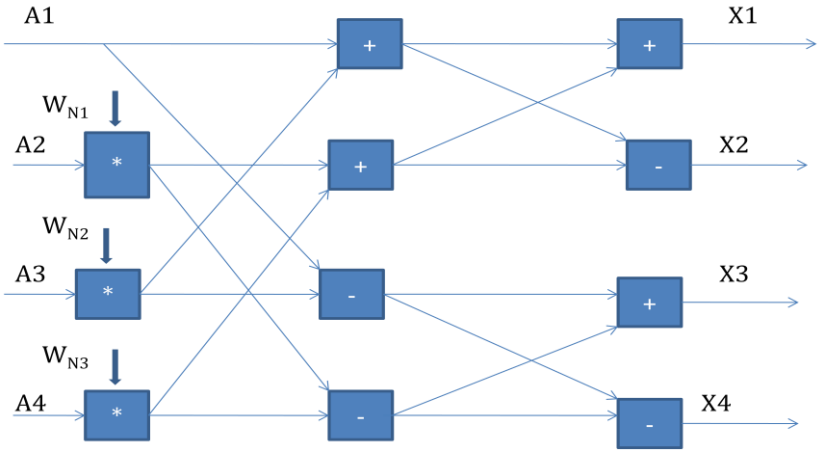

Fig. 1: Radix-4 DIT FFT butterfly unit

In FFT calculation there are two methods, on is decimation in the time domain (DIT) and another one is decimation in the frequency domain (DIF). In DIT and DIF, the serial numbers are grouped into even and odd parts. The Butterfly unit main task is

to perform the calculation of FFT with respect to minimum area, power, and delay. In Fig.1 Radix-4 DIT FFT butterfly unit has inputs are normalized [31:0] of A1,A2,A3,A4 and twiddle factor are W_{N1},W_{N2},W_{N3} and outputs are X1,X2,X3,X4. In [8] the output equations is from Figure 1.

$$X1= (A1 + W_{N2} \times A3) + (W_{N1} \times A2 + W_{N3} \times A4) \quad (a)$$

$$X3= (A1+ W_{N2} \times A3) - (W_{N1} \times A2 + W_{N3} \times A4) \quad (b)$$

$$X2= (A1 - W_{N2} \times A3) + (W_{N1} \times A2 - W_{N3} \times A4) \quad (c)$$

$$X4= (A1 - W_{N2} \times A3) - (W_{N1} \times A2 - W_{N3} \times A4) \quad (d)$$

2.2 Proposed flow chart for radix-4 DIT FFT implementing using FFAU

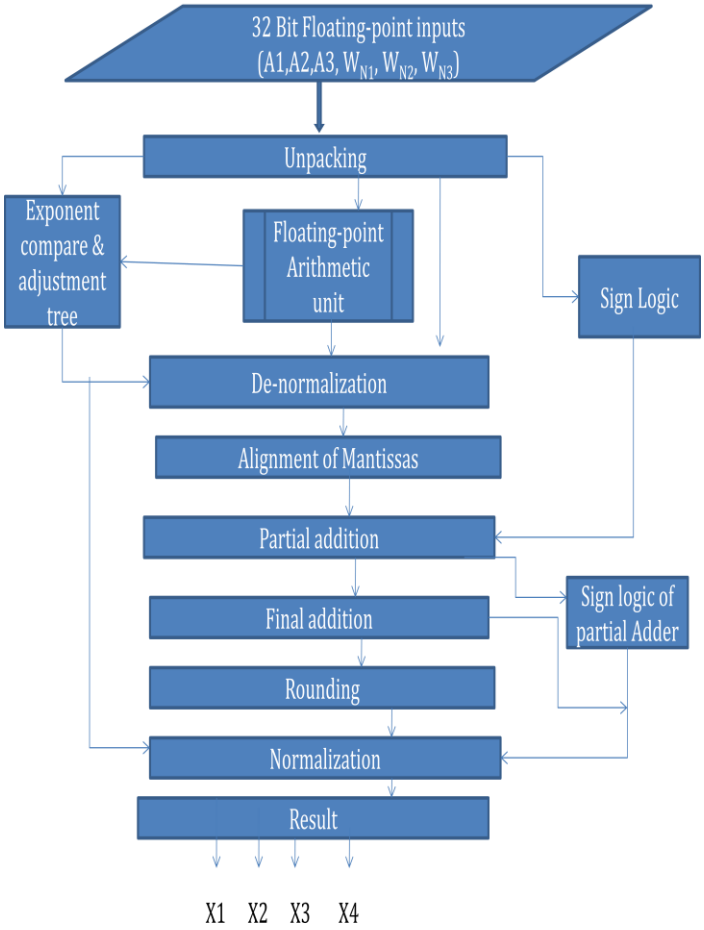

Fig. 2: Proposed flow chart for radix-4 SDIT FFTS implementing USINGS FFAU

Figure 2 The flow chart of the proposed FFAU is applied to the Sradix-4 DIT FFT butterfly unit. The flowchart consists of the number of steps to designing of FFAU. There are Unpacking, Floating-point multiplier, Exponent compare, De-normalization and alignment of mantissa, Addition of Mantissa, and Alignment rounding and normalization of the result.

A. Unpacking

Consider the proposed FFAU is implementing butterfly-unit inputs are A1, A2, A3, A4 and their twiddle factors are W_{N1}, W_{N2}, W_{N3}. These input and output are normalized 32-bit single-precision format. The unpacking splits the three parts as a sign, exponent, and mantissa. The inputs and twiddle factors of sign bit are stored in a one-bit register in respective as S_A1, S_A2, S_A3, S_A4 and S_W_{N1}, S_W_{N2}, S_W_{N3}. The zero bit indicates the positive number and one bit indicates the negative number. The inputs and twiddle factor of exponent bits are stored in an 8-bit register in respective as E_A1, E_A2, E_A3, E_A4 and E_W_{N1}, E_W_{N2}, E_W_{N3}.The inputs and twiddle factor of mantissa bits are stored in 24-bit register respective as M_A1, M_A2, M_A3, M_A4 and M_W_{N1}, M_W_{N2}, M_W_{N3}.

B. Floating point multiplier

As per the equations mentioned earlier, to obtain radix 4 DIT FFT, the multiplications involved are $W_{N1} \times A2$, $W_{N2} \times A3$, and $W_{N3} \times A4$. To carry out a multiplication say, first biased exponents of the two multiplicands should be added, the result has to be normalized by subtracting (127)10. Result exponent is denoted by E_W_{N1}A2, E_W_{N2} A3, and E_W_{N3} A4. Mantissas of the multiplicand is multiplied as per the Wallace multiplication method, which gives faster result and reduced which leads into 48-bits of the result. Sign of the multiplication is decided by doing ex-or operation between two operands and the resulting sign is denoted by S_W_{N1} A2, S_W_{N2} A3, S_W_{N3} A4.

C. Exponent compare

To perform floating point expansion, mantissa ought to be adjusted by the contrast between most elevated type and each other example. So here, 3 types from multiplier squares i.e., E_W1A2, E_W2A3, E_W3A4 and E_A1 is contrasted with get the most astounding type and afterward distinction between most elevated to other type mantissas are to be adjusted. The type contrast tree

which is shaped to choosing the most noteworthy example. Essentially < and > administrators perform subtraction activity to discover which one is more noteworthy. At that point, another subtraction is required to get the correct distinction. So this strategy requires two subtract which builds the region and basic way. Thus another strategy is received to ascertain more prominent type. Type of the second operand is subtracted from the principal operand utilizing two's compliment strategy. In the event that the MSB of the outcome is set then the outcome is sure. It implies that the principal example is more prominent than the second type. Outright contrast can be acquired to right move the mantissa of slower type by disposing of MSB bit of the outcome. Generally, if MSB bit of result is "h0n", at that point it is reasoned that the second example is more prominent than the first. The outcome is again experiencing b2's complementation to get the outright distinction between examples to right move the mantissa of the lower type.

D. De-normalisation and alignment of mantissa

De-standardization of mantissa i.e., extended interior width is chosen with the end goal that it incorporates add up to flood and an incomplete flood of all the four operands. As per prior discourse, while clarifying the floating point turn in the flowchart, the inward width is chosen to be "3f+6". At that point every one of the mantissas is correct moved as the outright distinction between their examples and most elevated types.

E. Addition of mantissas

To get the butterfly unit yields, A1, A2, A3 and A4, expansion of four operands are partitioned into two sections. The initial one is an incomplete expansion which gives the yields P1, P2, P3, P4 as appeared in (1) to (4).

$$P1 = X1 + (W_{N2} \times A3) \quad (1)$$
$$P2 = X1 - (W_{N2} \times A3) \quad (2)$$
$$P3 = (W_{N1} \times A2) + (W_{N3} \times A4) \quad (3)$$
$$P4 = (W_{N1} \times A2) - (W_{N3} \times A4) \quad (4)$$

Bit widths of incomplete yields are "3f+7". As indicated by singular sign piece, these conditions will give the sign for halfway yield, which later chooses the indication of the butterfly yields. The bit width of the last expansion moves toward becoming "3f+8". The yields after the last expansion are appeared in appeared in (i) to (iv).

$$X1 = P1 + P3 \quad (i)$$
$$X3 = P1 - P3 \quad (ii)$$
$$X2 = P2 + P4 \quad (iii)$$
$$X4 = P2 - P4 \quad (iv)$$

F. Alignment, rounding, and normalization of result

After the expansion of mantissa, standardization must be done from "3f+8" bits to 23 bits. In any case, some of the time {(In the instance of littler size information sources) MSB bit of the outcome isn't set. Consequently, we have to move the outcome left, which implies duplication by 2 in the parallel organization. The type of the outcomes is additionally to be diminished by one for every left move to the MSB of the outcome move toward becoming "1". This must be done in light of the fact that while unloading the mantissa bits, we have thought about the concealed "1" bit according to the IEEE 754 arrangement. Presently we have to expel the concealed bit from MSB of the outcome again to frame the standardize 23-bit mantissa. At that point as indicated by the adjusting strategy chose, barring the set MSB of the mantissa, 23 bits are grabbed for definite outcome pressing with the individual 8-bit type and the sign bit

2.3 Design and implementation of sixteen point radix-4 DIT FFT using FFAU.

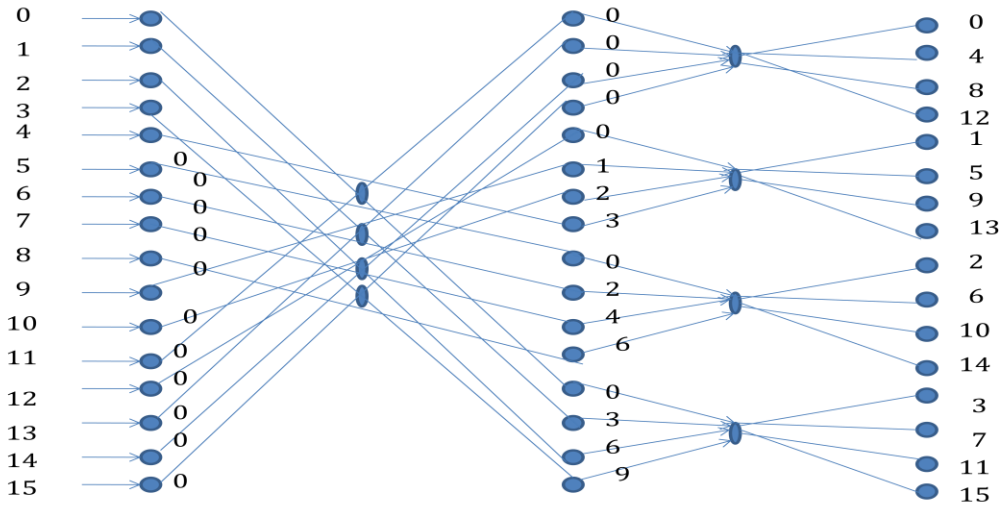

Fig. 3: 16-Point Radix-4 DIT FFT using FFAU

The sixteen point FFT calculation is done by using radix-4 butterfly unit. The proposed fused floating-point arithmetic unit improves the performance of the FFT calculation. The Performance is nothing but the time-saving methodology is the main advantage for high-speed application in communication such as telecommunication and DSP application. The sixteen point N=16

of input sequence which is decimation in time that is to be calculated by radix-4 butterfly unit. It has two stages, in the first stage requires 4 radix-4 butterfly unit is the calculation of DFT and the second stage requires the other four radix-4 butterfly unit in the calculation of DFT output. The input is taken the normal order and the output of the sequence is 16 which are in the format of bit-reversal 0,4,8,12 and 1,5,9,13 and 2,6,10,14 and 3,7,11,15.

3. RESULTS AND DISCUSSION

3.1 Proposed radix-4 butterfly with Floating Point Fused Arithmetic Unit (FFAU) implementation

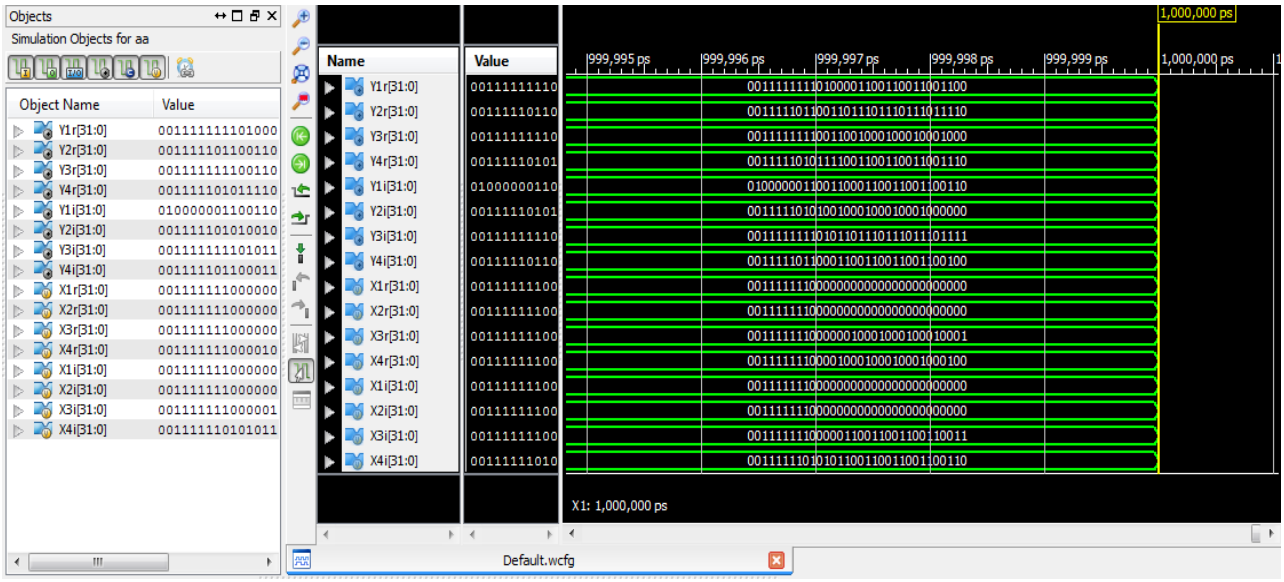

Fig. 4: Simulation results of radix-4 butterfly with Floating Point Fused Arithmetic Unit (FFAU) implementation

3.2 Proposed l6-point radix-4 DIT BFFT implemented with FFAU

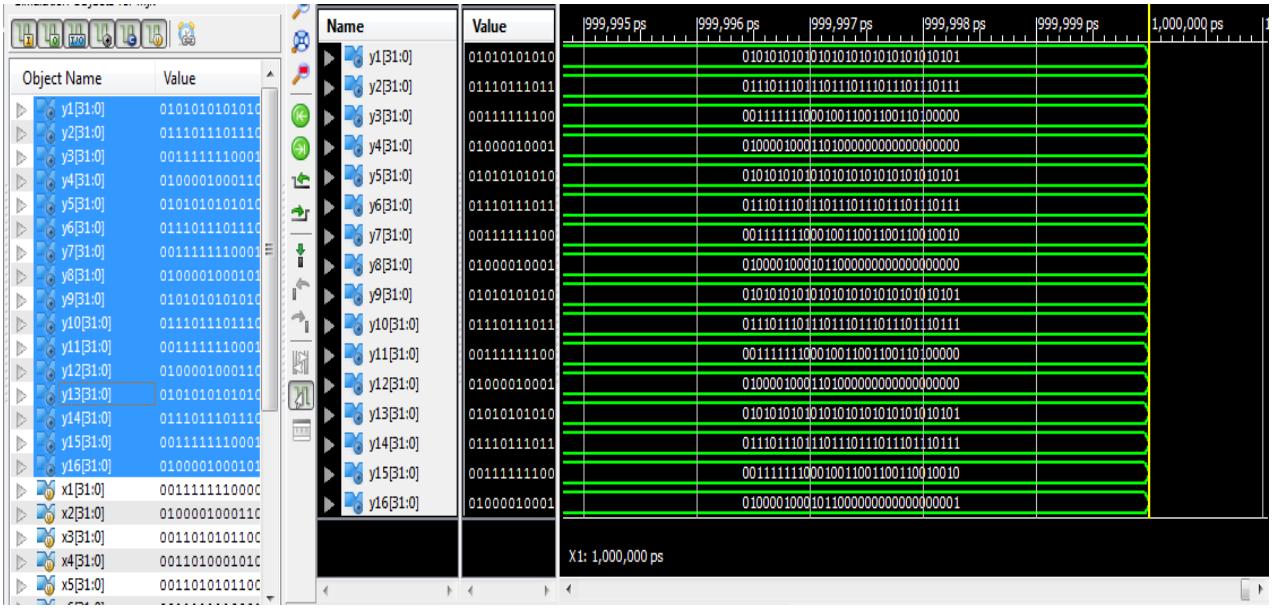

Fig. 5: Simulation results of l6-point radix-4 DIT FFT implemented with FFAU

The proposed architecture for the hardware implementation of radix-4 butterfly DIT FFT using FFAU unit is synthesized using CADENCE RC COMPILER in CMOS 45nm technology. The area report of the multi-term adder is given in table 1.

Tables 1: Area report for multi term adder

Fused arithmetic unit	Previous fused	Proposed
Standard cell area(μ $m^{2)}$	101258	27114

The result of Table I reveals that the multi-term adder implemented using FFAU here is 35% area efficient than adder proposed in conventional architecture [8].

Table 2: Comparative report of FFT

Radix-4 DIT FFT	Previously fused	Proposed FFAU
Standard-cell Area	47489μm^2	45668μm^2
Worst-case delay	4ns	3.5ns
Power	302.1mW	137.83mW

The result of Table II reveals that use of fused floating point computation unit (FFAU) in the implementation of Radix-4 DIT FFT butterfly unit has reduced area by 46% and delay by 33.33%. It operates with a frequency of l0MHz which his higher than the architecture in conventional architecture [4], which operates at 6MHz.Hence it can be concluded that the design is suitable for high-speed applications.

3.3 Detailed area view for synthesis of radix-4 DIT FFT implemented using FFAU.

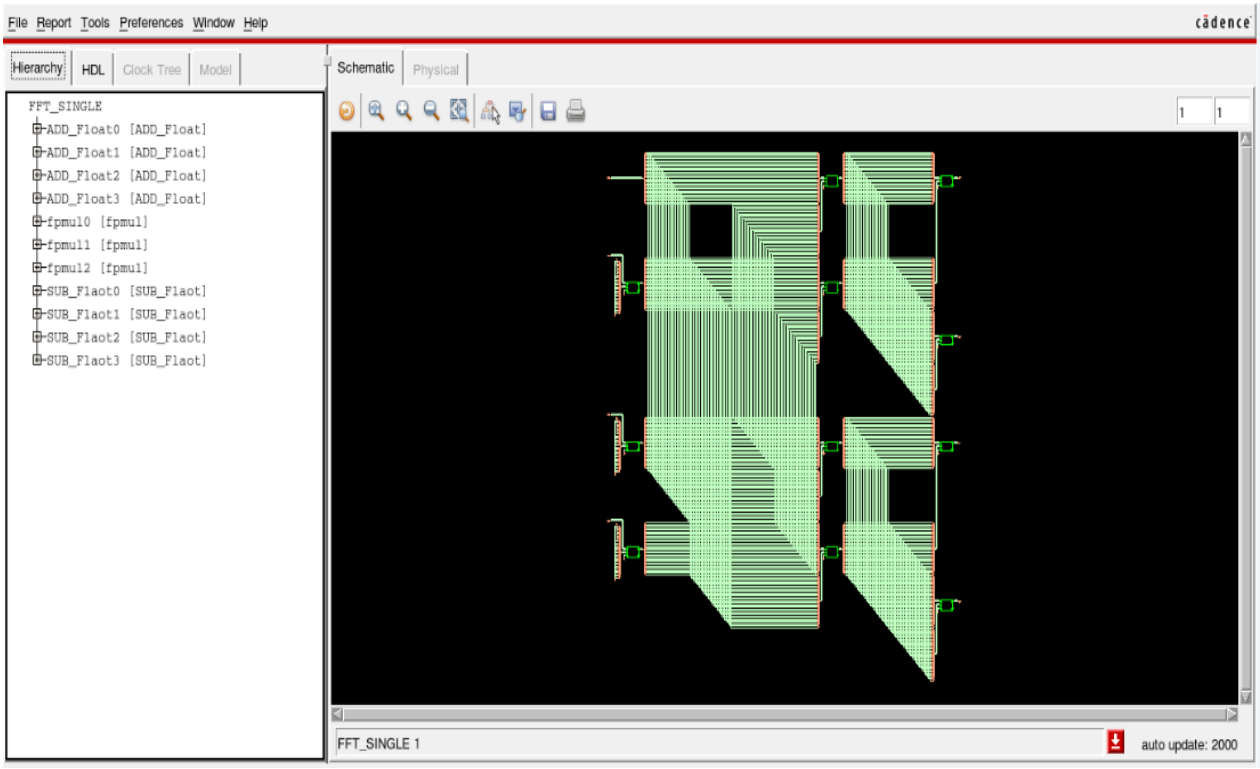

Fig. 6: Detailed Area view for the synthesis of Radix-4 DIT FFT Implemented using FFAU.

4. CONCLUSION

Hardware realization of fast Fourier transform (FFT) function consists of multiple complex arithmetic operations. Floating point implementation of FFT provides wider dynamic range than their fixed point counterparts and fusing the floating point arithmetic operations inside the Butterfly unit of FFT improves the speed of operation. This paper presents an FFT implementation using a fused floating-point arithmetic unit (FFAU) technique. The proposed FFT with FFAU is area efficient and operates at a higher frequency. Radix-4 decimation in time butterfly FFT unit is designed and synthesized in Cadence using 45nm technologies with. Based on the results it is analyzed that FFT with FFAU is 46% area efficient. The non-pipelined conventional architecture of FFT operates on 6MHz whereas proposed FFT architecture operates on 10MHz frequency. The proposed unit will applicable to DSP, Signal Processing, and Telecommunications.

5. REFERENCES

[1] Earl E. Swartzlander Jr., Hani H. M. Saleh, "FFT Implementation with Fused Floating-point Operations," IEEE Transactions on Computers, vol. 61, No. 2, February 2012.
[2] P.M. Seidel and G. Even. "Delay-optimized implementation of IEEE floating-point addition". IEEE Transactions on Computers, 49(7), pp.638 – 650, July 2000.
[3] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754- 2008, Aug. 2008.
[4] 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.
[5] Hani H. M. Saleh, "FAM," IEEE Transactions on Computers, vol. 61, No. 2, February 2012.
[6] Y. Tao, G. Deyuan, F. Xiaoya, and R. Xianglong, "Three-operand floating – point adder," in Proc. 12th IEEE Int. Conf. Comput. Inf. Technol., 2012, pp.192 – 196.
[7] JongwookSohn, Earl E. Swartzlander, "A Fused Floating-point Three-Term Adder", IEEE Transactions on Circuits and Systems – 1: Regular Papers, vol. 61, no. 10, October 2014.
[8] A. Tenca, "Multi – operand floating – point addition," in Proc. 21st Symp. Computer Arithmetic, 2009, pp. 161-168.
[9] Bringham, E. O.The Fast Fourier Transform.Prentice-Hall, Englewood Cliffs, NJ, 1974.
[10] Liu, D. Embedded DSP Processor Design. Morgan Kaufmann 1st edition, june 13, 2008.

