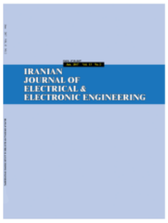
IUST
Iran University of
Science and Technology

Iranian Journal of Electrical and Electronic Engineering

Journal Homepage: ijeee.iust.ac.ir

Research Paper



ASIC Design of Butterfly Unit Based on Non-Redundant and Redundant Algorithm

P. Kulkarni*(C.A.), B. Hogade*, and V. Kulkarni**

Abstract: Fast Fourier Transform (FFT) processors employed with pipeline architecture consist of series of Processing Elements (PE) or Butterfly Units (BU). BU or PE of FFT performs multiplication and addition on complex numbers. This paper proposes a single BU to compute radix-2, 8 point FFT in the time domain as well as frequency domain by replacing a series of PEs. This BU comprises of fused floating point (FP) addition-subtraction (FFAS) and modified booth algorithm based floating point multiplier (FMULT). BU performs all arithmetic operations in floating pointform to overcome the nonlinearities available in fixed word length (FWL). FP arithmetic is slower as compared with FWL. To improve the speed of operation, symmetrical property of twiddle constant is used and they are embedded in the BU. BU outputs two halves of computation simultaneously with a single FFAS and two FMULT. BU design is synthesized, placed and routed for 45nm technology of nangate open cell library. Synthesized results show that proposed BU consumes 23910µm^2 area with latency of 3.44ns which are 5.05% smaller in area, 7.02% faster and replaces a set of two five operand adder and two multipliers by a single FFAS as compared with previously reported smallest work.

Keywords: Binary Signed Digit, Butterfly Unit, Carry Free Addition, Fast Fourier Transform, Fused Floating Point Addition-Subtraction.

1 Introduction

FAST Fourier Transform (FFT) processors are widely used in wireless communications, entertainment devices, biomedical field, image processing, etc. In the past decade, FFT processors were implemented with pipeline architectures. They consist of a series of processing elements (PE). PE has computational and data storing element.

Computational elements known as the butterfly unit (BU) are responsible for performing arithmetic operations.

In most cases, memory is used as a storing element. In N point FFT, $\log_2 N$ computational stages, N load, and N store operations are available. Therefore total $2N\times \log_2 N$ data access for entire dataflow is associated and degrades the performance of FFT computation.

The performance of large Point FFT was improved by splitting into small independent computation [1]. To implement the hardware of BU, it is important to select appropriate word length, data representation form, and arithmetic algorithm. Short word length results in less power consumption, smaller chip area, and faster computation [2]. Fixed point form or finite word length (FWL) is the popular choice to represent the data for its simplicity but introduces the nonlinearities in terms of overflow rounding, aliasing, and coefficient errors [2]. Data represented in floating point number, overcomes these nonlinearities but slows down the processing time. In spite of the sluggish nature of floating point numbers, various BUs were proposed with improved speed and reduced area of consumption in application specific integrated circuit (ASIC) designs. Area reduction by sharing common logic is suggested in [3]. Dual path

Iranian Journal of Electrical and Electronic Engineering, 2021.
Paper first received 14 February 2020, revised 06 April 2020, and accepted 10 April 2020.

* The author is with the Department of Electronics Engineering, Terna Engineering College, Nerul, Navi Mumbai, India.
E-mail: prasad26276@gmail.com and bghogade@gmail.com.
** The author is with the IT Department, Sanpada College of Commerce and Technology, Sanpada, Navi Mumbai, India.
E-mail: 2vidula@gmail.com.
Corresponding Author: P. Kulkarni.

https://doi.org/10.22068/IJEEE.17.1.1809

pipeline [4], merging of the operand/multi operand adder [5], and carry limited addition [6] was mostly suggested in BU architectures. Carry limited addition uses the redundant arithmetic algorithm. To perform the arithmetic operation in the redundant algorithm, non-redundant binary numbers are converted into redundant form. This conversion of non-redundant form to redundant form is free from carry propagation, but redundant to the non-redundant conversion of binary numbers introduces carry propagation. However carry free/carry limited addition [7] increases interconnecting wires by increasing the number of input-output lines of the arithmetic module. These lines also contribute to the latency in design. The multi operand adder introduces the carry propagation.

This paper proposes a BU for radix-2, 8 point small independent computation in time domain as well in the frequency domain. The novelty of this BU is that it replaces a series of PEs and computes two halves of the computation simultaneously. This paper demonstrates architecture of BU based on redundant, non-redundant algorithm and combination of them. The redundant algorithm is used to perform carry free addition. Here the non-redundant operand is converted into redundant form before addition. The sum of this adder is again converted into the non-redundant form before applying to the next logic. This conversion saves the width of the internal data bus and also helps to reduce the width of the storing element.

This paper compares BU design based on the redundant and non-redundant algorithms and proposes a BU suitable to compute radix-2, 8 point FFT. All floating-point arithmetic operations described here are based on the algorithm stated in [8]. 16 bit simple 2’s complement form is used to represent the data as stated in [9]. The paper briefs on the following:

1. Floating point addition-subtraction (FFAS) using the non-redundant and redundant approach.
2. Floating point multiplier (FMULT) using the non-redundant and redundant approach.
3. BU designs for radix-2, 8 point FFT computation based on the non-redundant algorithm, redundant algorithm, and combination of them.

2 Fused Floating Point Addition-Subtraction Using Redundant and Non-Redundant Algorithm

To perform the addition on two floating numbers, the following steps are followed.

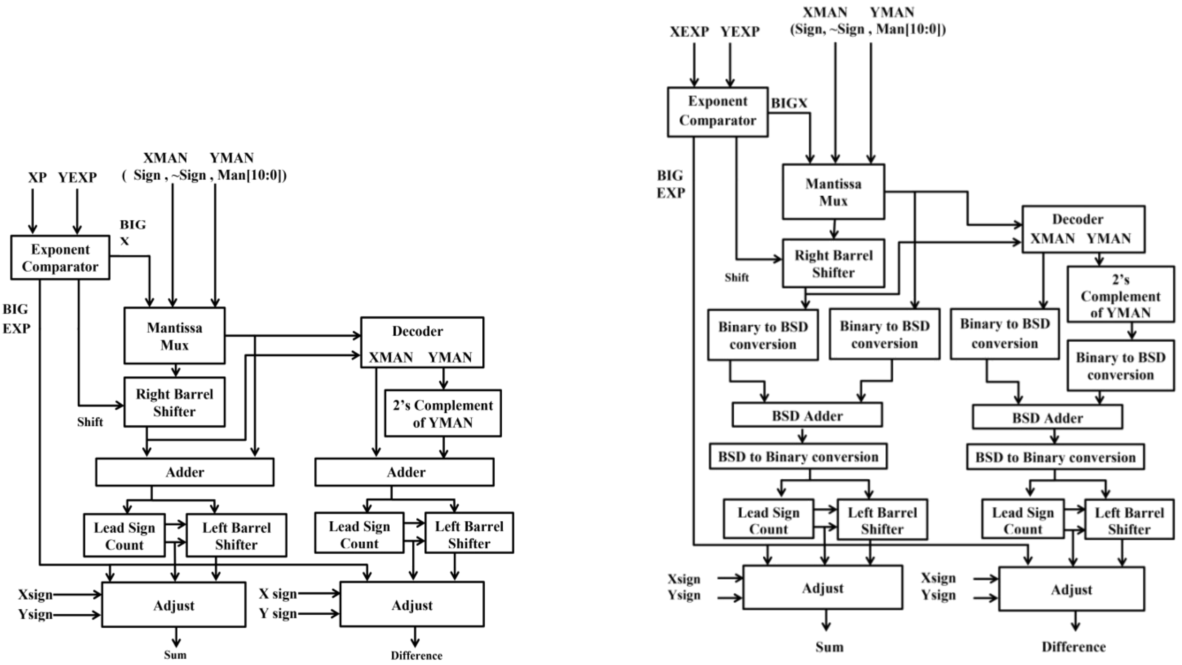
1. Exponent comparator logic compares exponent of two floating point numbers, *X* and *Y*. XEXP, YEXP represents exponents and XMAN, YMAN represents the mantissa of *X* and *Y* respectively. Comparator takes the difference of two exponents and records the difference in terms of shift count. Comparator also asserts the BIGX to indicate that XEXP is greater than YEXP. Greater exponent is assigned as exponent of sum in case of inequality of the exponents.
2. Mantissa multiplexer (mux) insert the implied bit (~sign bit) next to the leading sign bit of mantissa. The implied bit is zero when the floating point number is zero. Mantissa mux passes the mantissa of the number having smaller exponent to right barrel shifter and mantissa of number having greater exponent to the binary adder.
3. Right barrel shifter, shifts the mantissa of number having smaller exponent to right by shift count provided by exponent comparator.
4. The binary adder adds the two mantissas and outputs the sum.
5. The leadsign logic checks the sign of sum. It also counts the numbers of leading sign bits.
6. Left barrel shifter aligns the sum by shifting it to left equal to the count of leading sign bits.
7. Adjust logic, normalizes the result by verifying underflow, overflow, and zero conditions. In case of overflow of sum, the sum is set to the most positive/negative value. The underflow of the sum is set to zero. Adjust logic packs the exponent and mantissa to result the sum of floating point number.

Similarly the subtraction of two floating point number is performed by using 2’s complement. The decoder logic separates the minuend and subtrahend. 2’s complement of subtrahend is obtained before the addition.

In order to save area, common logic i.e. exponent comparator, mantissa mux and right barrel shifter are shared in FFAS. Authors have used the non-redundant and redundant algorithms to perform the addition only. Fig. 1(a) shows FFAS based on the non-redundant algorithm and Fig. 1(b) shows the redundant algorithm used for addition in FFAS. The redundant algorithm performs the carry free addition using binary signed digit (BSD) adder as stated in [6]. Each binary bit in non-redundant form is encoded into redundant form (BSD) before the addition. Binary to BSD converter entity encodes non-redundant number into redundant number using neg-pos encoding similar to [7, 10]. The BSD adder results the sum in redundant form. This sum is again converted into non-redundant form using the same neg-pos decoding as stated in [7, 10]. This conversion process restricts the number of interconnecting wires (internal bus width) to carry the redundant number in the entire flow. The redundant to non-redundant converter is shown in Fig. 2. Radix-2 binary number *X* is represented by (1).

$$X = \sum_{i=0}^{i-1} d_i \times 2^i \qquad d_i = \{-1, 0, 1\} \tag{1}$$

where X^+ and X^- are the bits of redundant number and



(a) (b)

Fig. 1 FFAS using a) non-redundant algorithm and b) redundant algorithm.

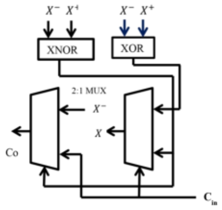


Fig. 2 Redundant to non-redundant converter.

Table 1 Comparative statistics of floating point addition and subtraction.

Parameter	Discrete addition–subtraction	FFAS	FFAS
Algorithm	Non-redundant	Non-redundant	Redundant
Technology	Nangate open cell library 45nm	Nangate open cell library 45nm	Nangate open cell library 45nm
Area (A) [μm^2]	11422	10330	11944
Delay (T) [ns]	1.47	1.56	1.57
AT^2 [mm^2 ns^2]	0.024	0.025	0.029

its equivalent non-redundant binary X is represented by (2), where C_{in} is the previous carry. This converter propagates the carry C$_O$ for the next leading bit conversion. This contributes the overhead of $O(\log n)$.

$$X = X^+ - X^- - C_{in} \tag{2}$$

Verilog code of FFAS using non-redundant and redundant algorithm is synthesized and placed using Mentor-Graphics-Oasys for 45nm technology of nangate open cell library. Authors have also synthesized the verilog code of the discrete design of floating point addition and subtraction using non-redundant algorithm. In discrete logic, common logic i.e. exponent comparator, mantissa mux, and right barrel shifter are not shared. Operating conditions are set to typical values. Table 1 shows the synthesized results of comparative statistics of non-redundant and redundant algorithms of floating point addition and subtraction. Discrete addition-subtraction design consumes 11422μm^2 area with a delay of 1.47ns. Similarly FFAS design using non-redundant and redundant algorithm contributes area 10330μm^2 and 11944μm^2 respectively. FFAS design using non-redundant algorithm causes a delay of 1.56ns and FFAS in redundant algorithm contributes a delay of 1.57ns. FFAS design using non-redundant algorithm saves the area by 9.56% as compared with discrete addition-subtraction performed in the non-redundant algorithm. However FFAS adds a penalty of 5.76% in latency. The FFAS design using the redundant algorithm contributes an additional 13.5% area as compared to the non-redundant algorithm.

3 Floating Point Multiplier Using Redundant and Non-Redundant Algorithm

Multiplication of two floating point numbers is performed as per the steps mentioned below. The signed multiplication of mantissas is performed using the modified booth algorithm.

1. Unpack logic separates the exponents and mantissas of floating point numbers. Exponents of these numbers are added separately. Also, implied bit is inserted in mantissas as described in FFAS.
2. Booth encoder, booth decoders, and partial product formations are similar to [11].
3. Partial products are aligned to performed addition

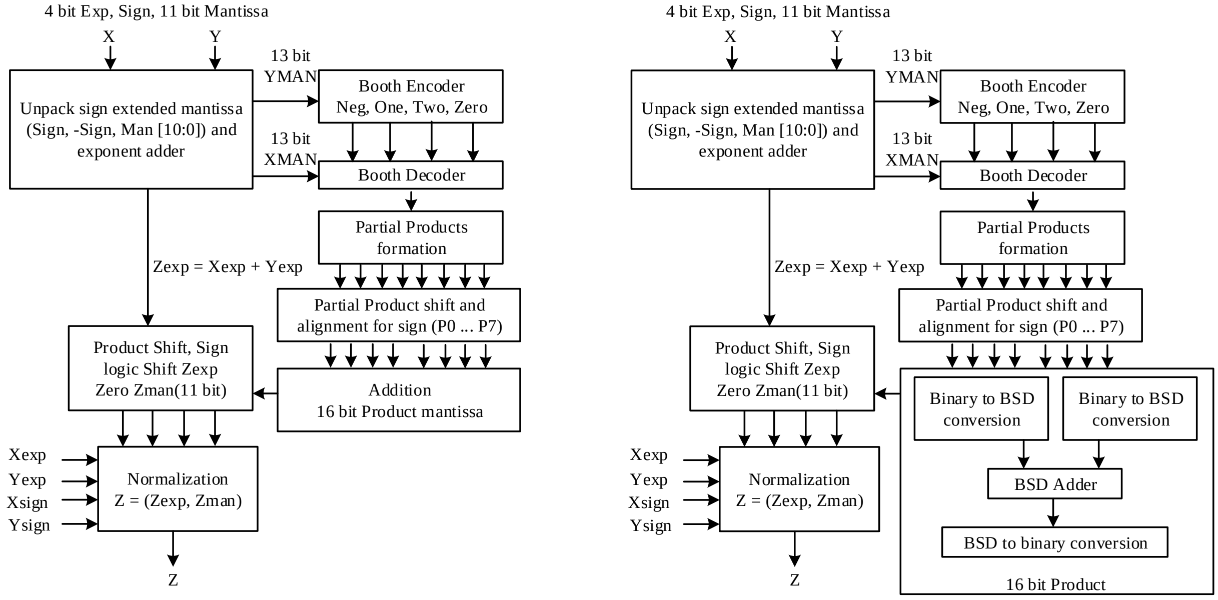
on them.

4. All bits of partial products are not contributing in generation of the final product. Hence they are ignored while performing addition of partial product. Only leading 16 bits out of 26 bits of the sum of the partial product are necessary.
5. Product shift logic performs XOR operation on the 16th and 15th bit of product as well as on the 15th and 14th bit of product to determine the number of bit position of the product to be shifted right.
6. Normalization logic checks the overflow, underflow, and zero conditions similar to FFAS. It also packs the exponent and mantissa to give the final product of two floating point numbers.

Figs. 3(a) and 3(b) show the FMULT using the non-redundant and redundant algorithm. The alignment of partial products and their addition are shown in Fig. 4 and Fig. 5 respectively. The addition is performed using Wallace tree method. The adders are also tailored to ignore bits which are not contributing to the final product.

Verilog codes of FMULT using non-redundant and redundant algorithms are synthesized and placed using Mentor-Graphics-Oasys for 45nm technology of nangate open cell library. Table 2 shows the synthesized results comparative statistics of non-redundant and redundant algorithms of FMULT.

FMULT design using non-redundant and redundant algorithms contributes 13904µm^2 and 18631µm^2 area, respectively. FMULT design using the non-redundant algorithm causes a delay of 1.94ns and FMULT in redundant algorithm causes a delay of 0.40 ns. In FMULT partial products are added using seven adders. Adder based on non-redundant algorithm contributes carry propagation delay, whereas adder based on redundant algorithm performs the carry free addition to save the delay. Hence the FMULT design using the redundant algorithm saves the delay by 79.38% as compared with FMULT based on the non-redundant algorithm with 25.37% additional area. The area increases due to the conversion process of non-redundant to redundant and vice-versa. Power analysis is also performed on both designs. FMULT using the redundant algorithm dissipates 2.734mW power whereas FMULT using the non-redundant algorithm dissipates 0.592mW power.


4 bit Exp, Sign, 11 bit Mantissa
X
Y
Unpack sign extended mantissa (Sign, -Sign, Man [10:0]) and exponent adder
13 bit YMAN
13 bit XMAN
Booth Encoder Neg, One, Two, Zero
Booth Decoder
Partial Products formation
Zexp = Xexp + Yexp
Partial Product shift and alignment for sign (P0 ... P7)
Product Shift, Sign logic Shift Zexp Zero Zman(11 bit)
Addition 16 bit Product mantissa
Xexp
Yexp
Xsign
Ysign
Normalization Z = (Zexp, Zman)
Z
4 bit Exp, Sign, 11 bit Mantissa
X
Y
Unpack sign extended mantissa (Sign, -Sign, Man [10:0]) and exponent adder
13 bit YMAN
13 bit XMAN
Booth Encoder Neg, One, Two, Zero
Booth Decoder
Partial Products formation
Zexp = Xexp + Yexp
Partial Product shift and alignment for sign (P0 ... P7)
Product Shift, Sign logic Shift Zexp Zero Zman(11 bit)
Binary to BSD conversion
Binary to BSD conversion
BSD Adder
BSD to binary conversion
16 bit Product
Xexp
Yexp
Xsign
Ysign
Normalization Z = (Zexp, Zman)
Z

(a) (b)

Fig. 3 FMULT using a) non-redundant algorithm and b) redundant algorithm.

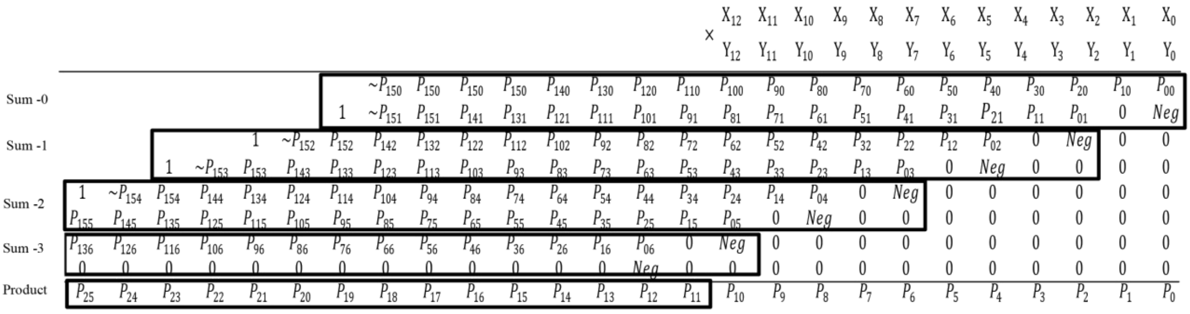


Fig. 4 Partial products.

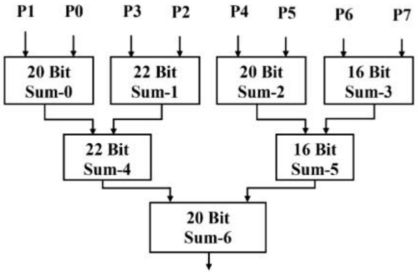


Fig. 5 Addition of partial product.

Table 2 Comparative statistics of FMULT.

Parameter	FMULT	FMULT
Algorithm	Non-redundant	Redundant
Technology	Nangate Open cell library 45nm	Nangate open cell library 45nm
Area (A) [µm^2]	13904	18631
Delay (T) [ns]	1.94	0.40
AT^2 [mm^2.ns^2]	0.052	0.002
Power (D_p) [mW]	0.592	2.734

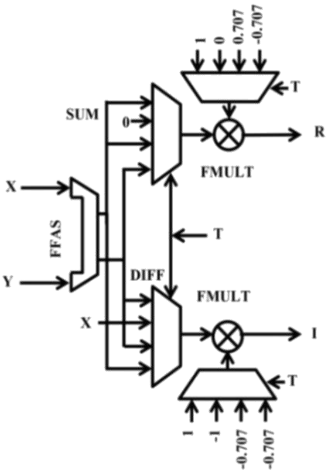


Fig. 6 Radix-2 Butterfly Unit using FFAS and Modified Booth FMULT.

Table 3 Comparison of BU

FMULT	FFAS	Delay [ns]	Area [µm^2]	AT^2 [mm^2.ns^2]	Power (D_p) [mW]	AT^2D_p [mm^2.ns^2.mW]
Non-redundant	Non-redundant	3.44	23910	0.282	2.80	0.792
Non-redundant	Redundant	3.46	25038	0.299	3.48	1.043
Redundant	Non-redundant	3.13	26669	0.261	4.71	1.230
Redundant	Redundant	3.06	27606	0.258	5.11	1.320

4 Butterfly Unit

Authors had already presented BU design in [9]. Discrete addition, subtraction unit in [9] is replaced by FFAS. The complex multiplier in [9] is replaced by FMULT using the modified booth algorithm. With the same operational methodology as stated in [9], Fig. 6 shows the proposed BU design. For stage 1 computation, W_8^0 is 1. Sample inputs are added and subtracted. This operation is initiated by T = 00. Sum is available at output R and difference at output I.

For stage 2 computation, $W_8^2 = (e^{-j2\pi/8})^2 = -j$ input sample is subtracted from zero and multiplied with -1 to produce computation at output I. This multiplication is selected by asserting T = 01. Similarly at stage 3, computation with $W_8^1 = (e^{-j2\pi/8})^1 = 0.707 - j0.707$ and $W_8^3 = (e^{-j2\pi/8})^3 = -0.707 - j0.707$ is computed by asserting T=10 and T=11 respectively. The real part of the complex multiplication is available at output R and imaginary part of multiplication at output I.The combination of non-redundant and redundant algorithm based FFAS and FMULT are used in BU to gives the four design types. These BU designs are synthesized and placed using Mentor-Graphics-Oasys for 45nm technology of nangate open cell library. The area and delay comparison of these designs are tabulated in Table 3.

BU design based on redundant algorithm causes a delay of 3.06ns, an area of 27606µm^{2,} and dissipates 5.11mW power. BU design based on non-redundant algorithm causes a delay of 3.44ns, an area of 23910µm^{2,} and dissipates 2.80mW power. BU design comprising FMULT based on non-redundant and FFAS based on redundant algorithm cause the delay of 3.46ns, an area of 25038µm^{2,} and dissipates 3.48mW power. BU design comprising FMULT based redundant and FFAS based on non-redundant algorithm causes the delay of 3.13ns, an area of 26669µm^2, and dissipate 4.71mW power. The hardware cost metric is represented by the product of AT^2 and power D_p. The comparative table shows that the BU design based on the redundant algorithm gives the minimum latency whereas BU design based on the non-redundant algorithm exhibits minimum area. Also the power dissipation is lower in BU design based on the non-redundant algorithm with the lowest hardware cost metric of 0.792.

5 Evaluation and Comparison

The proposed BU designs are verified for functionality using Xilinx 14.7. Table 4 shows the logical hierarchical details of placement in Oasys. BU Designs based on the non-redundant algorithm are reported in [3, 5, 9]. The work reported in [3] has a two fused dot product and two fused addition-subtraction unit. This work has 47489µm^2 area and latency of 4.00ns. Later on, the smallest BU based on the non-redundant algorithm is presented in [5] with a five operand adder and a two dot products for one halve of computational element. Hence for entire BU

Table 4 Logical hierarchical details of BUs placement in Oasys Comparison of BU

Module	Non-redundant algorithm		FMULT: Non-redundant FFAS: Redundant		FMULT: Redundant FFAS: Non-redundant		Redundant algorithm	
	No. of cells	Cell area [μm^2]	No. of cells	Cell area [μm^2]	No. of cells	Cell area [μm^2]	No. of cells	Cell area [μm^2]
Top (BU)	3971	5373	4429	5791	7957	8538	8471	9015
FFAS	915	1040	1373	1458	915	1040	1429	1518
FMULT x 2	2712	4026	2712	4026	6698	7190	6698	7190
Mux 4:1 x 4	344	308	344	308	344	308	344	308

Table 5 Comparison of BU.

	Proposed BU	Proposed BU	[3]	[5]	[6]	[9]
Technology	Nangate open cell 45nm lib.	Nangate open cell 45nm lib.	45nm Bulk CMOS standard lib.	Nangate open cell 45nm lib.	STM CMOS 90nm lib.and scaled to 45nm	Nangate open cell 45nm lib.
Algorithm	Non-redundant	Redundant	Non-redundant	Non-redundant	Redundant	Non-redundant
Area (A) [μm^2]	23910	27606	47489	25182	93836	22364
Delay (T) [ns]	3.44	3.06	4.00	3.70	2.59#	4.56
AT^2 [mm^2.ns^2]	0.282	0.261	0.759	0.344	0.629	0.465
Power (D_p) [mW]	2.80	5.11	–	–	–	2.43
AT^2D_p [mm^2.ns^2.mW]	0.792	1.32	–	–	–	1.13

# Extra logic is required to convert non-redundant form into redundant form and vice versa. The delay of this logic is not included in design.

– Power not given.

computational flow two sets of five operand adder and two multipliers are required. This work consumes 25182μm^2 area with the latency of 3.70ns. Authors have reported the BU design in [9] with two floating point adder and three floating point multiplier. This BU design is also synthesized and placed using Mentor-Graphics–Oasys for 45nm technology of nangate open cell library. This design has delay of 4.56ns, consumes 22364μm^2 area, and dissipates 2.43mW power. The proposed BU design reduces the latency by 24.56% at the additional cost of 6.46% in area and dissipates 13.21% more power as compared with the authors' previous work [9]. However it is worth mentioning that the proposed BU design has lower hardware cost metric and AT^2 complexity. The redundant algorithm based BU is reported in [6] with three operand adder and a two dot products for one halve of computational element. Hence for the entire BU computational flow two sets of three operand adder and two multipliers are required. This work consumes 93836μm^2 (scaled to 45nm) area with the latency of 2.59ns. The output of this work is in redundant form. Hence the redundant to non-redundant converter is required which propagates the carry and additional delay of $O(\log n)$. Also redundant logic doubles the space of the storing element. However the proposed BU design has single FFAS and two FMULT to compute two halves of computational stage.

Table 5 shows the comparison of proposed BU based on THE non-redundant and redundant algorithm with the previous reported BU designs [3, 5, 6, 9]. The proposed BU design based on non-redundant algorithm reduces area and delay by 5.05% and 7.02% respectively as compared with the previously reported work [5]. Similarly BU design based on redundant algorithm reduces delay by 17.29% with the additional cost of the increased area by 9.26%. However AT^2 complexity of both proposed BUs is lower as compared with previous work [3, 5, 6].

6 Conclusion

The BU design proposed in this paper is used to compute all stages of radix-2, 8 point FFT with the reduced arithmetic elements. Twiddle constants are embedded in the design and selected as per the computation stage which saves the loading time from the lookup table or the memory element. Sharing logic is used to save the area. The BU design based on the non-redundant algorithm consumes less area and dissipates less power as compared with BU design based on the redundant algorithm at additional latency. The BU design based on the non-redundant algorithm saves the area by 5.05% and reduces the latency by 7.02% and BU design based on redundant algorithm reduces the latency by 17.29% at the cost of the additional area of 9.62% as compared with the previous fastest and smallest reported BU design.

References

[1] X. Guan, Y. Fei, and H. Lin, "Hierarchical design of an application-specific instruction set processor for high-throughput and scalable FFT processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 20, No. 3, pp. 551–563, Mar. 2012.

[2] L. Wanhammar, *DSP integrated circuits*. San Diego, CA, USA: Academic, 1999.

[3] E. E. Swartzlander and H. H. Saleh, "FFT implementation with fused floating-point operations," *IEEE Transactions on Computers*, Vol. 61, No. 2, pp. 284–288, Feb. 2012.

[4] J. Sohn and E. E. Swartzlander, "Improved architectures for a fused floating-point add-subtract unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 59, No. 10, pp. 2285–2291, Oct. 2012.

[5] A. Kaivani and S. B. Ko, "Area efficient floating-point FFT butterfly architectures based on multi-operand adders," *Electronics Letter*, Vol. 51, No. 2, pp. 895–897, Jun. 2015.

[6] A. Kaivani and S. B. Ko, "Floating-point butterfly architecture based on binary signed-digit representation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 24, No.3, pp. 1208–1211, Mar. 2016.

[7] K. Schneider and A. Willenbücher, "A new algorithm for carry free addition of binary signed digit numbers," in *IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 44–51, 2014.

[8] K. C. Chang, *Digital systems design with VHDL and synthesis: An integrated approach.* IEEE Computer Society, 1999.

[9] P. Kulkarni, B. G. Hogade, and V. Kulkarni, "Designing of radix-2 butterfly for digital signal processor for FFT computation," in *Information and Communication Technology for Intelligent Systems*, Springer, Singapore, pp 603–610.

[10] K. K. Parhi, *VLSI digital signal processing systems: Designs and implementation.* John Wiley & Sons, 1999.

[11] S. R. Kang, J. P. Wang, and C. Y. Guo, "Modified booth multipliers with regular partial product array," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, Vol. 56, No. 55, pp. 404–408, May 2009.

P. Kulkarni received B.E. (Electronics) degree from NMU Jalgoan in 1997 and completed his M.E. (Electrical with Control System) degree from V.J.T.I Mumbai, University of Mumbai in 2005. He is currently working towards the Ph.D. degree in Electronics at Terna Engineering College affiliated to University of Mumbai. His present research interest is in VLSI to design DSP processors.

B. Hogade received the B.E. (Electronics) degree from Marathwada University in 1991, M.E. (Power Electronics) degree from Gulbarga University in 1999, and Ph.D. degree in 2014 from NMIMS, Mumbai. His Ph.D. research focused on the smart antenna for wideband wireless communication. He is also the supervisor for research scholars in the University of Mumbai. His present research interest is in antennas, wireless communication, and power electronics.

V. Kulkarni received the B.E. (Electronics) degree from NMU Jalgoan in 2002 and completed M.E. (Electronics) degree at Terna Engineering College affiliated to the University of Mumbai. Her present research interest is in the embedded system.