VOL. 8, NO. 1, JANUARY 2013 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

DISTRIBUTED ARITHMETIC BASED BUTTERFLY ELEMENT FOR FFT PROCESSOR IN 45NM TECHNOLOGY

P. Augusta Sophy[1], R. Srinivasan[2] and J. Raja[3]
[1]AP/VIT Chennai & Research Scholar, Anna University, Chennai, India
[2]SSN College of Engineering, Chennai, India
[3]Anna University of Technology, Trichy, India
E-mail: sophyguna@gmail.com

ABSTRACT

Fast Fourier Transform (FFT) is one of the important signal processing algorithms because of its applications in digital filtering, communication, image processing, spectral analysis and estimation etc. Butterfly computation is the basic operation in the FFT algorithm. In this work, a novel approach is thought of and used in implementing the butterfly element. Distributed Arithmetic Algorithm (DAA) is used to do the butterfly computation instead of using conventional multipliers and adders. This has resulted in a more efficient butterfly element both in terms of area and power. This paper describes the design of such an area efficient butterfly module. Single precision floating point representation is used for the data. This design leads to a lot of area saving and power saving. This butterfly can be used as the basic building block of a low power reconfigurable FFT processor. This finds its application in OFDM based systems and also in software defined radios.

Keywords: Fast Fourier Transform (FFT), butterfly element (BF), Discrete Fourier Transform (DFT), Distributed Arithmetic Algorithm (DAA), twiddle factor.

INTRODUCTION

A Fast Fourier Transform

Fast Fourier Transform (FFT) is one of the most commonly used signal processing algorithms in communication and multimedia systems. It is called as the algorithm for the whole family as it helps in spectral analysis, spectral estimation, interpolation, decimation, convolution, correlation, filtering, etc. FFT is also used in many applications like noise detection and cancellation, speech encryption, sampling rate conversions, applications utilizing the FFT filters, audio signal processing and image processing. So it is the major part of the base band processing. More over FFT and IFFT are also used as demodulation and modulation kernels in the OFDM systems.

OFDM is a special case of Frequency Division Multiplexing, which is also used in the Software Radio (SDR). OFDM is a combination of modulation and multiplexing. OFDM can be easily generated using an IFFT module and demodulated using a FFT module. FFT is an efficient method of computing the discrete Fourier transform (DFT) for a set of data. The Discrete Fourier Transform (DFT) operates on samples of a time domain signal and is defined by the following equation.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad (1)$$

where *X(k)* represents the DFT frequency output at the *k*-th spectral point, *k* ranging from 0 to *N-1*. The quantity *N* represents the number of sample points in the DFT data frame. The quantity *x(n)* represents the *n*-th time sample, where *n* also ranges from 0 to *N-1*. In general, x*(n)* can be real or complex.

The corresponding Inverse Discrete Fourier Transform (IDFT) of the sequence *X(k)* gives a sequence *x(n)* defined only on the interval from 0 to *N-1* as follows:

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} \qquad (2)$$

The DFT equation can be re-written as:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

The quantity W_N^{nk} is defined as:

$$W_N^{nk} = e^{-j2\pi nk/N}$$

This quantity is called twiddle factor. It is the sine and cosine basis functions written in polar form. Examination of equation (1) reveals that the computation of N point DFT requires *N (N-1)* complex multiplications and *N (N-1)* complex additions.

The FFT algorithm was developed by Cooley and Turkey in the year 1965,by taking advantage of the symmetry and periodicity properties of the twiddle factors.

$$W_N^{r+N/2} = -W_N^r$$
$$W_N^{r+N} = W_N^r \qquad (3)$$

Out of the two FFT algorithms, DIT (decimation in time) and DIF (decimation in frequency), DIF is mostly used in the implementations of FFT. In the DIF algorithm, the inputs are given in the natural order and the output

frequency points are obtained in the bit reversed manner i.e the output frequency points are found to be regrouped or subdivided. This radix-2 DIF FFT algorithm starts by dividing the samples into two groups, the first half and the last half. The division goes till the DFT computation is done between only two samples. This 2-point DFT block is called as the butterfly operation. The signal flow graph of FFT is constructed with N/2 number of butterflies in each stage and there are $\log_2 N$ number of stages.

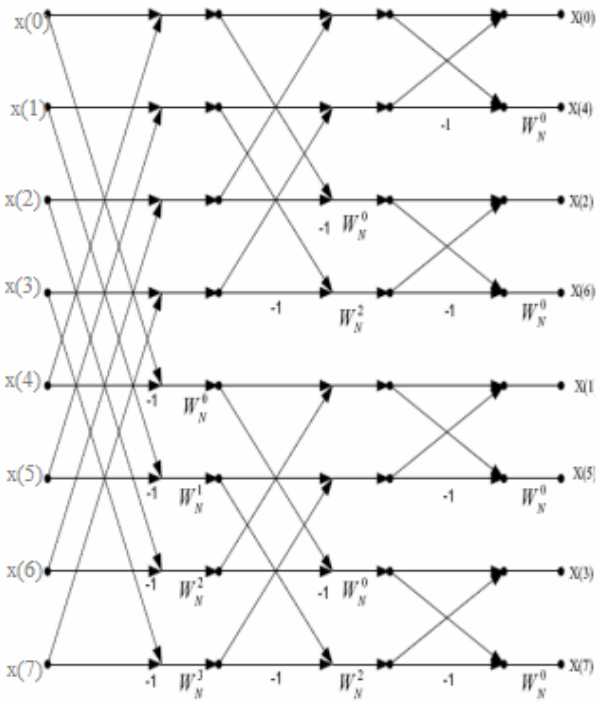

Figure-1: Signal flow graph of an 8 point DIF FFT.

The signal flow graph of an 8 point DIF FFT algorithm is shown in Figure-1. The Basic butterfly operation of the DIF algorithm is shown in Figure-2.

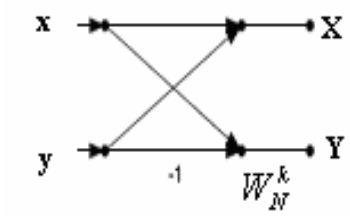

Figure-2: Basic butterfly operation of DIF algorithm

For any DIF butterfly, there are two inputs x and y (two input samples). The outputs X and Y are given by:

$$X = x + y$$

$$Y = (x - y)W_N^k$$

The inputs x, y, W_N^k and as well as the outputs X and Y are complex values.

As the butterfly operation is the basic computation in FFT algorithm, designing a highly efficient butterfly element would lead to a lot of saving in silicon area and/or in power.

The objective of this work is to design and implement an area and power efficient butterfly element, which could be used in building a reconfigurable FFT/IFFT. Here we have presented a new design for the butterfly element, based on the distributed arithmetic algorithm (DAA). DA can be used for the computation of sum of products. The DFT equation given in equation (1) is in the sum of products form to which DA algorithm can be applied. It can be noted that, in this work, the DAA is not applied for the computation of the N point DFT as a whole, but used for a single Butterfly which is a 2 point DFT. This design shows saving in the silicon area.

Applying DAA to a N point DFT as sum of product restricts the expansion of the design to any value of N. This makes the design rigid and the modularity is lost. Using the DAA approach to a single butterfly makes the design modular and it becomes easy to expand the design to any value of N. By using this butterfly element, we can build an efficient reconfigurable FFT processor.

More over, in this project, IEEE single precision 32 bit floating point notation is used for representing the samples. This enhances the suitability of the design for industrial applications.

This paper presents this novel DAA based butterfly design. Section 2 explains the general DA algorithm. Section 3 presents the DA algorithm applied for the floating point butterfly element. Section 4 presents the hardware design. Section 5 presents the synthesis results obtained from Synopsys Design compiler.

DISTRIBUTED ARITHMETIC ALGORITHM - AN OVERVIEW

There are numerous signal processing applications in which DA algorithm is used. They are listed out in journals as early as 1989, where as DAA is as old as 1974. DAA is easily used for implementing DSP algorithms, because most of the signal processing algorithms are in the form of sum of products or inner product of two vectors.

DA is basically, a bit-serial computation that forms an inner product of a pair of vectors in a single direct step. In computations of sum of products, if one of the vector operands is known, distributed arithmetic can be used. It uses look up tables and adders instead of multipliers. So there is a considerable reduction in the power consumption also, whenever we employ this DAA. A frequently argued disadvantage is its apparent slowness because of its bit serial nature. This disadvantage is not real if the number of samples in each vector commensurate with the number of bits in each vector element. i.e., the time required to input eight, 8-bit words one at a time in a parallel fashion is exactly the same as the time required to input all eight words serially.

As an example, take the computation of the following sum of products.

$$y(n) = \sum_{K=o}^{N-1} A_k X_k \quad (4)$$

Let A_k be the known vector and X_k be the input data represented in 2's complement binary number, scaled such that $| X_k | < 1$ then X_k may be expressed as:

$$X_k = -X_{k0} + \sum_{b=1}^{B-1} X_{kb} 2^{-b} \quad (5)$$

where X_{kb} is a binary variable 0 or 1 and- X_{k0} indicates the sign bit

Substituting equation (5) in equation (4)

$$y(n) = \sum_{k=0}^{N-1} A_K [-X_{k0} + \sum_{b=1}^{B-1} X_{kb} 2^{-b}]$$

And now this can be expanded as

$$y(n) = -\sum_{k=0}^{N-1} A_K X_{k0} + \sum_{k=1}^{N-1} \sum_{b=1}^{B-1} X_{kb} 2^{-b} A_K \quad (6)$$

the above equation is rearranged as follows:

$$\begin{aligned} y(n) = &-[x_{10}A_1 + x_{20}A_2 + x_{30}A3 + \cdots\cdots\cdots + x_{N0}A_N] + \\ &[x_{11}A_1 + x_{21}A_2 + x_{31}A_3 + \cdots\cdots\cdots + x_{N1}A_N]2^{-1} + \\ &[x_{12}A_1 + x_{22}A_2 + x_{32}A_3 + \cdots\cdots\cdots + x_{N2}A_N]2^{-2} \\ &\vdots \\ &\vdots \\ &\vdots \\ &[x_{1(B-1)}A_1 + x_{2(B-1)}A_2 + x_{3(B-1)}A_3 + \cdots\cdots\cdots + x_{N(B-1)}A_N]2^{-(B-1)} \end{aligned} \quad (7)$$

Each term within the brackets denotes a binary AND operation involving a bit of an input variable and all bits of the coefficient (Stanley A. White, 1989). The plus signs denote arithmetic sum operations. The exponential factors denote the scaled contributions of the bracketed partial sum, to the total sum. A look-up-table which stores all the possible 'sums' of the coefficients is constructed. The look up table will be addressed by the nth bit from all input samples. Thus the arithmetic operations have now been reduced to addition, subtraction, binary scaling and accumulation.

DAA APPLIED TO A FLOATING POINT BUTTERFLY MODULE

In this work, DAA is applied to do the butterfly computation (Stanley A. White) and not for computing the whole DFT. The signal flow graph of the DIF butterfly that has been implemented is shown in Figure-3.

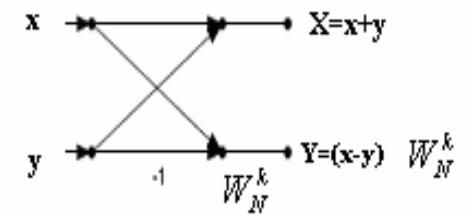

Figure-3: Signal flow graph of a DIF butterfly.

Here the inputs x, y and W_N^k are complex and the outputs X and Y are also complex. i.e.

$$x = x\_real + jx\_img$$

$$y = y\_real + jy\_img$$

$$X = X\_real + jX\_img$$

$$Y = Y\_real + jY\_img$$

$$W_N^k = W\_real + jW\_img$$

And each and every, real and imaginary part of the data is represented by a 32 bit IEEE single precision, floating point number.

The outputs X and Y are as given below.

$$\begin{aligned} X &= x + y \\ &= (x\_real + y\_real) + j(x\_img + y\_img) \end{aligned} \quad (8)$$

$$Y = (x - y)W_N^k$$

Let $(x\_real - y\_real) = G_1$

and $(x\_img - y\_img) = G_2$

Then $Y = (G_1 + jG_2) \times (W\_real + jW\_img)$

$$\begin{aligned} &= (G_1 W\_real - G_2 W\_img) + \\ &\quad j(G_1 W\_img + G_2 W\_real) \end{aligned} \quad (9)$$

$$Y\_real = (G_1 W\_real - G_2 W\_img) \quad (10)$$

$$Y\_img = (G_1 W\_img + G_2 W\_real) \quad (11)$$

From the above equations, it is obvious that, six floating point adders and four floating point multipliers are required to implement a butterfly element in the conventional way. But in this work, we have suggested a technique using distributed arithmetic, to obtain the real part and imaginary part of the output Y. Thus equations (10) and (11) are realized using DAA, instead of using floating point multipliers and adders. Here the W vector (twiddle factor) is a known value. The numbers are represented in single precision floating point notation.

A 32 bit number X in single precision floating point is represented as $X = (-1)^S 2^E \times 1.M$

Where

S = Sign bit (1 bit)
E = Biased exponent (8 bits)
M = Mantissa (23 bits)
And its value V is given by the expression

$$V = (-1)^S \left[2^E \times \sum_{n=0}^{23} M_n 2^{-n} \right] \quad (12)$$

In the equations (9), (10) and (11), the values of the W_real and W_img are taken as a 32 bit word and the values of G_1 and G_2 are going to be treated bit wise as required in the distributed arithmetic algorithm technique.

Therefore

$$\begin{aligned} Y\_real &= (G_1 W\_real - G_2 W\_img) \\ &= (-1)^{S_{11}} \left(2^{E_1} \sum_{n=0}^{23} M1_n 2^{-n} \right) W\_real - \\ &(-1^{S_2}) \left(2^{E_2} \sum_{n=0}^{23} M2_n 2^{-n} \right) W\_img \end{aligned} \quad (13)$$

where M1 and M2 are the 24 bit mantissa of G_1 and G_2 including the first hidden msb bit '1'. If $E_1 = E_2$ then the same value can be taken as the common exponent value. If $E_1 \neq E_2$ then, the exponent is made equal to the higher exponent value of the two and thereby shifting its mantissa to the left by the required number of positions.

Then

$$Y\_real = 2^E \left\{ \begin{array}{l} (-1)^{S_1} \left(\sum_{n=0}^{23} M1_n 2^{-n} \right) \times W\_real - \\ (-1)^{S_2} \left(\sum_{n=0}^{23} M2_n 2^{-n} \right) \times W\_img \end{array} \right\} \quad (14)$$

Assuming that both the sign bits are zero i.e. G_1 and G_2 are positive values, and then equation (14) can be expanded as:

$$Y\_real = 2^E \left\{ \begin{array}{l} \left[M1_0 2^0 + M1_1 2^{-1} + M1_2 2^{-2} + ... + M1_{23} 2^{-23} \right] W\_real - \\ \left[M2_0 2^0 + M2_1 2^{-1} + M2_2 2^{-2} + ... + M2_{23} 2^{-23} \right] W\_img \end{array} \right\} \quad (15)$$

Here $M1_0$ is the most significant bit and $M1_{23}$ is the least significant bit of the mantissa part of G_1. Now the equation can be rearranged as per the distributed arithmetic technique as follows.

$$Y\_real = 2^E \left\{ \begin{array}{l} (M1_0 W\_real - M2_0 W\_img) 2^0 + \\ (M1_1 W\_real - M2_1 W\_img) 2^{-1} + ... + . \\ (M1_{23} W\_real - M2_{23} W\_img) 2^{-23} \end{array} \right\} \quad (16)$$

$$Y\_img = 2^E \left\{ \begin{array}{l} (M1_0 W\_img + M2_0 W\_real) 2^0 + \\ (M1_1 W\_img + M2_1 W\_real) 2^{-1} + ... + . \\ (M1_{23} W\_img + M2_{23} W\_real) 2^{-23} \end{array} \right\} \quad (17)$$

Now in equation (16), each term in the braces is a binary AND operation involving a bit of the mantissa of G1 and the real or imaginary part of the twiddle factor. So its value can be one of the four combinational values of the real and imaginary parts of the twiddle factor as shown below. The exponential factors denote shifting or scaling of the value inside the braces. The "+" sign indicates arithmetic addition and here it's a floating point addition.

The value inside the braces can be calculated from the following look up table as the value of twiddle factor is already known.

Table-2. LUT for $S_1 = S_2 = 0$.

$M1_n$	$M2_n$	LUT Value
0	0	0
0	1	- W_img
1	0	W_real
1	1	W_real - W_img

The LUT for while the sign bits of both G_1 and G_2 are '0' is given in table 2. If the sign bits change then the value of the LUT gets a sign change accordingly. So together with the sign bits of G_1 and G_2, a 16 value LUT is

used. The real part of output Y is calculated by using a scaled accumulator which accumulates the values from the LUT for each bit of the mantissa.

The same algorithm can be followed to compute the imaginary part of output Y. Another LUT of size 16 is used and the similar DAA module is used.

HARDWARE IMPLEMENTATION OF THE DAA BUTTERFLY

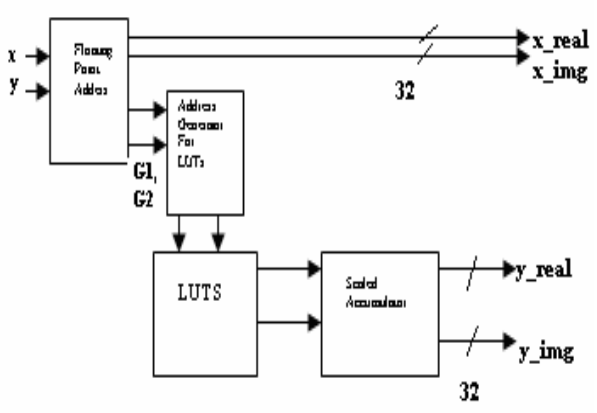

Figure-4. Architecture of a DAA butterfly.

Equations (16) and (17) are implemented using the block diagram shown in Figure-4. This is the general architecture of a bit serial DA. The inputs x and y are added and subtracted in the floating point adders. The real and imaginary parts of output X are obtained immediately, whereas for calculating the output Y the address generator blocks, LUTs and scaled accumulators are used. These blocks eliminate the need for multipliers. Thus it reduces the area requirement and also consumes less power.

The top level block diagram of our design is viewed as in Figure-5. Here X1 and X2 are the inputs, and O1 and O2 are the outputs. There are more control inputs for loading, reading LUT, shifting, resetting etc.

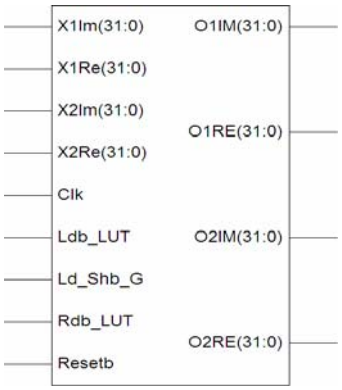

Figure-5. Top view of DAA BF.

EXPERIMENTAL RESULTS

The RTL modeling of the above design is done with Verilog HDL and the formal verification is done. The area reports of the butterfly module implemented in the conventional method using 6 floating point adders and 4 floating point multipliers is compared with butterfly implemented using DAA technique which requires only 4 floating point adders and two DAA units.

The design is synthesized using Synopsys Design Complier with 45 nm technology node and the net list is generated. Also the area report is obtained and this design is found to be area efficient. Table-3 shows the comparison between the conventional Butterfly and DAA Butterfly.

Table-3. Area report in 45nm technology.

Parameter	Conventional BF	DAA BF
No. of standard cells	7	5
Combinational area	34362 sq. micron	29446 sq. micron
Total area	37633 sq. micron	36446 sq. micron

CONCLUSIONS

In this paper we have presented a novel design for the butterfly element. The efforts are to make the basic butterfly module more efficient in terms of area, power and also speed, as 'butterfly' is the basic building block of FFT and IFFT cores. The design can be used for reconfigurable FFT processor also. This butterfly element is designed for floating point numbers. Usually the DAA algorithm and also the butterfly/FFT processors are designed for data represented in the 2's complement form. This floating point DAA BF is found to be area efficient as expected and also shows reduction in the critical path. So it can be used for designing high speed, reconfigurable FFT processors.

In the near future a reconfigurable FFT/IFFT floating point processor will be designed using this novel area efficient 'butterfly' element.

REFERENCES

L. Wanhammar. 1999. DSP Integrated Circuits. Academic Press.

K. K. Parhi. 1999. VLSI Digital Signal Processing Systems: Design and Implementation. John Wiley and Sons.

A. V. Oppenhein and R. W. Schaffer. 1980. Discrete - time Signal Processing. Prentice Hall.

Stanley A White. 1989. Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Overview. IEEE, ASSP Magazine, July.

Stanley A White. A Simple Butterfly Arithmetic Unit. IEEE Transactions on Circuits and Systems. Vol. Cas28, No.4, April 81.

Sansaloni T, Pérez-Pascual A and Valls. J. Digit-serial Distributed Arithmetic Butterflies for FPGA.

Christophe Bobda, Ali Ahmadinia and J¨urgen Teich. Generation of Distributed Arithmetic Designs for Reconfigurable Applications.

Wayne P Burleson and Louis L Scharf. 1991. A VLSI Design Methodology for Distributed Arithmetic. Journal of VLSI Signal Processing.

Amit Sinha and Mahesh Mehendale. 1997. Improving Area Efficiency of FIR filters implemented using Distributed Arithmetic. IEEE.