UNIVERSITÉ DU QUÉBEC

THÈSE PRÉSENTÉE À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DU DOCTORAT EN GÉNIE ÉLECTRIQUE

PAR
MARWAN ALI JABER

FAIBLE COMPLEXITÉ ET HAUTE PERFORMANCE DE LA TRANSFORMÉE DE FOURIER

NOVEMBRE 2013

UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

DOCTORAT EN GÉNIE ÉLECTRIQUE (PH.D.)

Programme offert par l'Université du Québec à Trois-Rivières

TITRE DE LA THÈSE

Faible complexité et haute performance de la transformée de Fourier

PAR

Marwan Ali Jaber

Daniel MASSICOTTE, directeur de recherche	Université du Québec à Trois-Rivières
Pierre SICARD, président du jury	Université du Québec à Trois-Rivières
Messaoud AHMED-OUAMEUR, évaluateur	NUTAQ inc.
Olivier SENTIEYS, évaluateur externe	ENSSAT, France

Thèse soutenue le 15 mars 2013

Dédicace

Ce travail est dédié à mon père Ali Jaber, ma mère Fatima Maatouk et à mes enfants Mohamad-Ali et Marwah Jaber.

Remerciements

Il me fait plaisir d'adresser mes remerciements au professeur Daniel Massicotte, mon directeur de recherche, pour sa patience et sa bonne volonté d'avoir accepté de m'encadrer pour ma thèse de Doctorat.

Résumé

Le travail présenté par cette thèse porte sur l'amélioration de la transformation rapide de Fourier (TRF) et représente une contribution aux progrès dans le traitement numérique du signal et des algorithmes de calcul rapide. La réduction des temps de calcul offerte par la TRF proposée trouve des applications en traitement numérique du signal à temps réel et en analyse spectrale. C'est une contribution bien accueillie dans les domaines du traitement de la parole, les communications par satellite et terrestre, communications numériques avec ou sans fil, traitement du signal multidiffusion, détections et identifications des cibles, radar et systèmes de sonar, machine aux signaux surveillés, sismologie et biomédecine. En outre, les propositions peut être d'intérêt particulier dans les applications de communication sans fil, les cartes DSP (*Digital Signal Processor*) et FPGA (*Field Programmable Gate Array*).

Cette thèse développe et présente un algorithme de la TRF à radice-*r* qui réduit l'effort de calcul (telle que mesurée par le nombre d'opérations arithmétiques) par un facteur de *r* en comparaison avec la plupart des algorithmes de la TRF à radice-*r*. Le problème réside dans la définition du modèle mathématique de la phase de combinaison, dans laquelle la représentation de la TDF en termes de ses TDF partielles devrait être bien structuré pour obtenir le vrai modèle mathématique. L'algorithme qui en résulte, dans lequel les *r* processeurs en parallèles pourraient fonctionner simultanément avec une seule instruction.

La clé conceptuelle du papillon modifié de la TRF à base *r* est la formulation de la TRF à radice-*r* comme *r* éléments de traitement élémentaires (BPE – *Butterfly Processing Element*) avec des structures identiques et un moyen systématique d'accéder les coefficients

multiplicateurs correspondants. Cela permet la conception d'un BPE avec le plus faible taux de multiplicateurs et d'additionneurs complexes, qui utilise r ou $r - 1$ multiplicateurs complexe en parallèle pour calculer chaque sortie du papillon conventionnel. Il y a une association simple entre les trois indices (TRF étape, papillon et élément) et les adresses des coefficients multiplicateurs nécessaires. Pour un environnement de processeur unique, ce type de BPE avec r multiplicateurs en parallèles entraînerait la diminution du délai du calcul de la TRF par un facteur de $O(r)$. Un second aspect des papillons modifiés de la TRF à radice-r est, qu'ils sont également utiles dans les environnements du multitraitement en parallèle où cette structure en parallèle est réalisable au cours de chaque étage de la TRF. Si chaque BPE est exécuté sur le papillon modifié, cela signifie que chacun des r BPE en parallèles exécutera la même instruction simultanément, ce qui est très souhaitable pour la structure d'une seule instruction avec des données multiples (SIMD) sur certains des plus récentes cartes DSP.

En outre, on a développé un générateur d'adresses pour la TRF qui peut réduire la charge de calcul et l'accès aux mémoires en groupant les ensembles de données avec ses multiplicateurs correspondants. L'avantage de regrouper les ensembles de données avec ses correspondants multiplicateurs permettra de réduire les accès aux mémoires où lors de chaque étage la mémoire des coefficients est consulté r^s fois pour la procédure DIT et $r^{(S-s)}$ fois pour la procédure DIF où S = $log_r N - 1$ et s = 0,1,.., $S - 1$. En plus et à l'aide de ce concept, nous pourrions facilement prévoir l'apparition des multiplications triviales. La présence de la multiplication par ± I peut être facilement prédite pendant le processus de la TRF où en faisant donc, l'accès aux mémoires et les multiplications complexes pourraient être réduites ainsi que la multiplication par ± j peut être aussi prédite et qui peut être

incorporée dans les additions par la commutation des parties réelles et imaginaires des données. En plus de cela, la multiplication par $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ peut être prédite aussi où le nombre d'opérations arithmétiques peut être réduit de 6 à 2.

Dans ce domaine, nous avons développé également l'algorithme de la TRF à une seule itération qui est un outil utile pour détecter des fréquences spécifiques dans des signaux surveillés. Une des techniques les plus importantes dans l'analyse des caractéristiques d'un signal est l'extraction des informations utiles d'un signal surveillé. La surveillance des signaux est un domaine en expansion qui visent la détection des changements brusques pour une fréquence spécifique la détection d'un ensemble présélectionné de fréquences tel que RFID (*Radio Frequency Identification*) et dans le système de communication sans fil OFDM (*Orthogonal Frequency Division Multiplex*) dans lequel la TRF est un opérateur clé principal. Notre algorithme proposé a montré un gain significatif en vitesse et en rapport du signal au bruit quantifié (SQNR – *Signal to Quantization Noise Ratio*) en comparaison avec l'algorithme de Goertzel. Enfin, pour ce domaine nous avons développé le *Low Complexity Input/output Pruning FFTs* qui est une méthode utilisée pour calculer une DFT où un sous-ensemble des sorties sont nécessaires.

Abstract

Digital Signal Processing (DSP) is an engineering field that continues to extend its theoretical foundations and practical implications in the modern world from highly specialized aero spatial systems through industrial applications to consumer electronics. The Fast Fourier Transform is one of the most important topics in Digital Signal Processing that generates a map of a signal (called its spectrum) in terms of the energy amplitude over its various frequency components, at regular (e.g. discrete) time intervals, known as the signal's sampling rate. This signal spectrum can then be mathematically processed according to the requirements of a specific application (such as noise filtering, image enhancing, etc...). The quality of spectral information extracted from a signal relies on two major components: the first one is the spectral resolution which means a high sampling rate that will increase the implementation complexity to satisfy the time computation constraints. The second one is the spectral accuracy which is translated into an increasing of the data binary word-length that will increase normally with the number of arithmetic operations.

As a result, the FFTs are typically used to input large amounts of data; perform mathematical transformation on that data and then output the resulting data all at very high rates. The mathematical transformation is executed by arithmetic operations (multiplications, summations or subtractions in complex values) following a specific dataflow structure that should control the systems' input/output. Multiplication and memory accesses are the most significant factors on which the execution time relies. The problem with the computation of an FFT with an increasing N is associated with the straightforward computational structure, the coefficient multiplier memories' accesses and

the number of multiplication that should be performed. In high resolution and better accuracy, this problem will be more and more significant for real time FFT implementation and in order to satisfy the time computation constraints. We should structure the input/output data flow that could reduce the coefficient multipliers accesses and also to reduce the computational load by targeting trivial multiplication. Memories' accesses are major concerns in implementation on DSP cards which on the most cases are costly in DSP cycles. Therefore, in a real time implementation, executing and controlling the data flow structure is important in order to achieve high performance that could be obtained by regrouping the data with its corresponding coefficient multiplier. By doing so, the access to the coefficient multiplier's memory will be reduced drastically and the multiplication by the coefficient multiplier w^0 will be taken out of the equation. In order to maintain lower arithmetic operations within the butterfly critical path (one complex multiplier and certain adders), we will be targeting hardware oriented Radix 2^{α} or 4^{β} which is an alternative way of representing higher radices by mean of less complicated and simple butterflies.

In this thesis, we developed the self-sorting JMFFT (Jaber-Massicotte Fast Fourier Transform) algorithm that could benefit the trivial multiplication ± 1, $\pm j$ and $\pm \sqrt{2}/2 \pm j\sqrt{2}/2$ that will yield to the kernel core JMFFT's computation. One of the most significant impacts of the proposed structure from the hardware point of view is that the coefficient's multiplier memory has been reduced from $N/2$ to $N/8$. The JMFFT was tested on the TMS320C6416 DSP platform using TI's Code Composer Studio which shows a significant gain in clock cycle reduction in comparison to the most recent published method 2^2 kernel core FFT. Furthermore, the JMFFT was benchmarked on the FFTW platform in which our proposed structure revealed a significant gain compared to

FFTW. The FFTW benchmark is an FFT bench platform assembled by Matteo Frigo and Steven G. Johnson at MIT (Massachusetts Institute of Technology). This platform compares the performance of different complex FFT implementations (40 FFT methods) based on speed and accuracy where performance is computed on a single processor environment. The FFTW platform includes an FFT method called FFTW_ESTIMATE that outperforms all other methods and is actually used in Matlab® software. Our results shown a speedup up to 30% compared to FFTW.

One of the most important techniques is the Signal-analysis/feature-extraction techniques which aim to extract useful information from a given monitored signal. Signal monitoring is an expanding domain that deals in detecting any abrupt changes for a special known frequency such as fault detection machine or to scan a pre-selected set of frequencies, as in radio-frequency identification (RFID) tags, the recognition of the dual-tone multi-frequency (DTMF), DNA analysis and in the orthogonal frequency division multiplex wireless communication (OFDM), wherein the Fast Fourier Transform (FFT) is a major key operator, particularly for cognitive radio. In this thesis, we developed the radix-*r* JM-Filter (Jaber Massicotte-Filter) which is a combination of the radix-*r* one iteration FFT algorithm and the Goertzel's algorithm structures. Compared to Goertzel's filter, the proposed first and second order radix-*r* JM-Filter manifested a gain in the computational complexity reduction. The higher radix is; the highest gain is obtained.

Cellular and cordless phones rapidly became mass-market consumer products in which each wireless system has to combat transmission and propagation effects that are substantially more hostile than for a wired system. The advances in signal processing provided methods to overcome the anomalies of the mobile channel by accelerating the

growth of wireless communication and by tackling the channel problems by mean of the diversity reception concept that can substantially improve the link performance. Moreover, advanced digital modulation methods, such as spread spectrum or Multi-Carrier Modulation (MCM) appear suitable for wireless communication where Orthogonal Frequency Division Multiplexing (OFDM), a special form of MCM, will be used extensively in digital terrestrial broadcast systems, e.g. in DAB (Digital Audio Broadcasting) and DTTB. Theoretical aspects of the Pruning FFT (PFFT) have been thoroughly elaborated in past three decades and which was mainly concentrated on sequences that have L_i consecutive non-zero input points at the beginning. In many applications, the percentage of required input/output bins is very small such as the 3GPP (The 3rd Generation Partnership Project) LTE (Long Term Evolution) where the OFMDA's symbol size is 1024 in which 12 users equally share the available 600 sub-carriers. As a result only 50 of the 1024 FFT output bins (5%) are required for each mobile terminal. These partial input/output cases are important for the future wireless systems and because the PFFT can potentially achieve a significant speedup which is made it as a target by many applications such as cognitive radio. In this thesis, we have presented a novel JMIOPFFT (Jaber Massicotte Input/Output Pruning FFT) that shows an important gain in the computational complexity reduction compared to the most relevant published results.

Contents

List of Tables

Chapter 1

Chapter 2

List of Figures

Chapter 1

Chapter 2

Abreviations

AG	Address Generator
BPE	Butterfly Processing Element
Cag	Coefficient Address Generator
CAGDIF	Coefficient Address Generator for the DIF FFT
CAGDIT	Coefficient Address Generator for the DIT FFT
C-MAC	Complex multiply-accumulator
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
DNA	Deoxyribonucleic Acid
DTMF	Dual-tone multi-frequency
FA	Full adder
FFT	Fast Fourier Transform
FFTW	Fast Fourier Transform in the West
IPJMFFT	Input Pruning JMFFT
JMIOPFFT	JM Input Output Pruning FFT
JM	Jaber Massicotte
JMFFT	Jaber Massicotte FFT
LTE	Long Term Evolution
MAC	Multiply–accumulator
MDC	Multi-path Delay Commutator

MIMO-OFDM	Multiple Input Multiple Output – Orthogonal Frequency Division Multiplexing
OFDM	Orthogonal Frequency Divisions Multiplex
OFDMA	Orthogonal Frequency Division Multiplexing Access
OIOO	Ordered Input Ordered Output
OSPE	One Stage Processing Element
pdf	Probability density function
PFFT	Pruning FFT
RAG	Reading Address Generators
RAGDIF	Reading Address Generators for the DIF FFT
RAGDIT	Reading Address Generators for the DIT FFT
RCAG	Reading Coefficients (twiddle factors) Address Generator
RDAG	Reading Data Address Generator
RFID	Radio Frequency IDentification
SDC	Single-path Delay Commutator
SDF	Single-path Delay Feedback structure
SFG	Signal flow graph
SQNR	Signal to Quantization Noise Ratio
TDF	La Transformée Discrète de Fourier
TDSN	Traitement Du Signal Numérique
TRF	La Transformée Rapide de Fourier
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration

WAG	Writing Address Generator
WDAG	The writing data address generator
WFTA	Winograd Fourier Transform Algorithm
3GPP	3rd Generation Partnership Project

List of Symbols

$x_{[n]}$	the input sequence
$X_{[k]}$	the output sequence
N	the transform length
$w_N^{nk} = e^{-j(2\pi/N)nk}$	The twiddle factor in butterfly structure
j^2	The complex number = –1
B_r	The butterfly matrix
T_r	The adder tree matrix
W_N	The twiddle factor matrix
$\lfloor \mathrm{x} \rfloor$	The integer part operator of x
$[\![\mathrm{x}]\!]_N$	The operation x modulo N
$\hat{*}_{(\alpha,\gamma,\beta)}$	Product of radix-α performed on γ column vector of size β

Chapter 1 Introduction

Résumé du Chapitre 1

Dans ce chapitre, nous présentons la transformée discrète de Fourier (TDF) et la transformée rapide de Fourier (TRF). La Transformée Rapide de Fourier est simplement une TDF calculée selon un algorithme permettant de réduire le nombre d'opérations et, en particulier, le nombre de multiplications à effectuer. Une brève description de la dérivation de la transformée rapide de Fourier sera présentée en mettant l'accent sur les deux versions de l'algorithme, avec « entrelacement temporel » et avec « entrelacement fréquentiel ». Par la suite, on présentera une brève description du défi pour réduire la complexité de l'algorithme. Une de notre contribution majeure dans ce domaine, est la réduction de la complexité du passage critique du papillon à radice-*r*. Pour améliorer de manière significative la rapidité de ce traitement, on a suggéré un générateur d'adresses pour la transformée rapide de Fourier dont le but est d'effectuer toutes les opérations de réarrangement des données entrées/sorties. Ce générateur d'adresses nous a permis de regrouper les données avec les coefficients multiplicateurs correspondants tout en prédisant l'apparition des multiplications triviales (8^{th} root of unity). De plus, dans cette thèse on a abordé le sujet de la détection d'une fréquence spécifique tout en proposant un filtre, nommé JM-Filter avec une complexité réduite en comparaison avec celle de Goertzel. Dans de nombreuses applications telles que le Long Term Evolution (LTE) et la radio cognitive, qui est basée sur l'Orthogonal Frequency Division Multiplexing OFDM, on aura besoin de calculer un certains nombres de sorties de la TRF où on a appliqué de manière efficace des zéros sur les entrées. Notre contribution dans ce sujet est basée sur l'introduction d'un

algorithme qui peut réduire la complexité du calcul en se comparant avec les méthodes les plus récemment proposées.

Introduction

Digital Signal Processing (DSP) is an engineering field that continues to extend its theoretical foundations and practical implications in the modern world from highly specialized military systems through industrial applications to consumer electronics. One of the most exciting aspects of DSP use is in new applications such as DNA that are impossible to implement using analog technology where a digital signal processor may be called on to perform one or more of several functions involving algorithmic processing of the digitized signals. So, digital signal processing is one of the most powerful technologies that will shape science and engineering in the upcoming centuries.

For the last decade, the DFT and all resulting algorithms known collectively as Fast Fourier Transform (FFT) showed a great interest for their applications in which revolutionary changes have already been made in a broad range of fields such as: Speech compression [1], Digital filters [2], Image processing [3], Radar [4], OFDM (Orthogonal Frequency Divisions Multiplexing) [5]- [8], Wireless communications [9]-[11], DNA analysis and a lot of other non-cited domains. Therefore, the Discrete Fourier Transform (DFT) is the decomposition of a sampled signal in terms of sinusoidal (complex exponential) components expressed as:

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} w_N^{nk} \quad , \; k \in [0, N-1] \qquad (1),$$

and because of its computational requirements, the DFT algorithm usually is not used for real time signal processing. For the last decencies, the main concern of the researchers was to develop an FFT algorithm in which the number of operations required is minimized. Since Cooley and Tukey presented their approach showing that the number of multiplications required to compute the discrete Fourier transform (DFT) of a sequence

may be considerably reduced by using one of the fast Fourier transform (FFT) algorithms. The basis of the radix-2 FFT is that a DFT can be divided into two smaller DFTs, each of which is divided into two smaller DFTs, and so on, resulting in a combination of two points DFTs kernel. Cooley and Tukey's method, which is known as fast algorithms for DFT computation, is based on the divide-and-conquer approach that was introduced by Danielson and Lanczos in 1942 as shown in Figure 1 [12]. The advantage of appropriately breaking the DFT in terms of its partial DFTs is that the number of multiplications and the number of stages may be controlled. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation, and thus, reduction in the number of stages is beneficial. Several efficient methods are used repeatedly to split the DFTs into smaller (two or four-point) core calculations, where the symmetry and periodicity properties of the DFT are exploited to significantly lower its computational requirements.

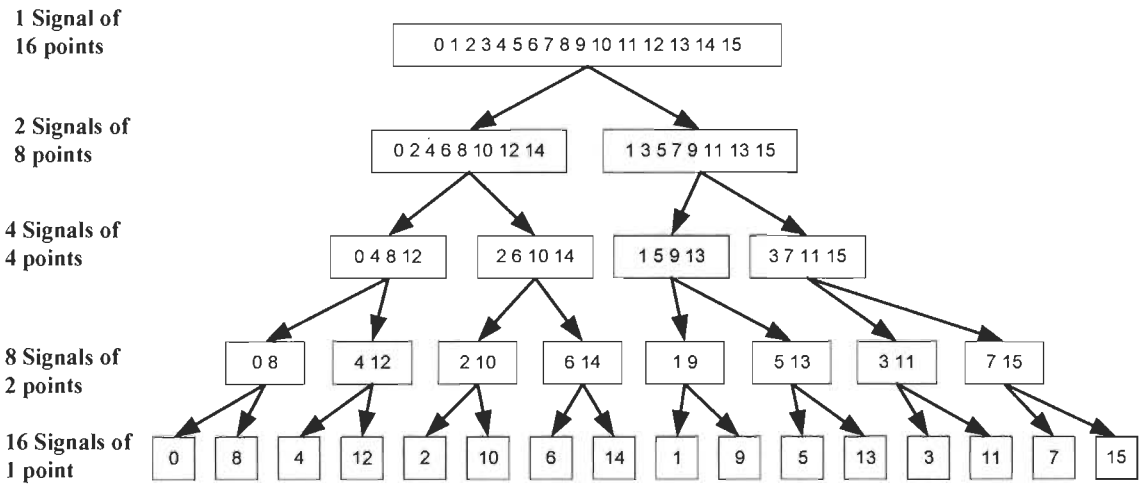


Figure 1: The FFT decomposition. *N* point signal is decomposed into N signals each containing a single point. Each stage uses an interlace decomposition, separating the even and odd numbered samples.

2. Derivation of the FFT

The computational complexity of the DFT increases as the square of the transform length, and thus, becomes expensive for large N. The fast Fourier transform (FFT) provides an effective tool for the calculation of Fourier transforms that is based on the divide-and-conquer approach. In the FFT case, the input data $x_{(n)}$ are divided into subsets on which the DFT is computed. Then the DFT of the initial data is reconstructed from these intermediate results. If this strategy is applied recursively to the intermediate DFTs, an FFT algorithm is obtained. Some of these methods are:

- Common Factor Algorithms (decimation-in-time (DIT) or Cooley-Tukey FFT algorithm [13] and decimation-in-frequency (DIF) or Sande-Tukey FFT algorithm [14]),
- Prime Factor Algorithm (PFA) [15],
- Mixed Radix Algorithm (MRA) [15],
- Winograd Fourier Transform Algorithm (WFTA) [16] and
- Split-Radix Algorithm (SRA) [17].

2.1 Common Factor Algorithms

In the common factor algorithms, the transform length N, is decomposed into arbitrary factors ($N = r_1, r_2, \ldots, r_k$) and if all the factors, r_i, are equal, the algorithm is called radix-r algorithm. Two different versions of the algorithm that are dual of each other, which are always derived depending on how the decimation is performed. The two versions have the same computational complexity and are called decimation-in-time (DIT or Cooley-Tukey) and decimation-in-frequency (DIF or Sande-Tukey).

2.1.1 The DIT and DIF Algorithms

In mid-1960s, J.W. Cooley and J.W. Tukey proposed their first algorithm known as decimation-in-time (DIT) or Cooley-Tukey FFT algorithm, which first rearranges the input elements into bit-reverse order, then builds up the output transform in $\log_2 N$ iterations figure (2) [12], [13] and [18].

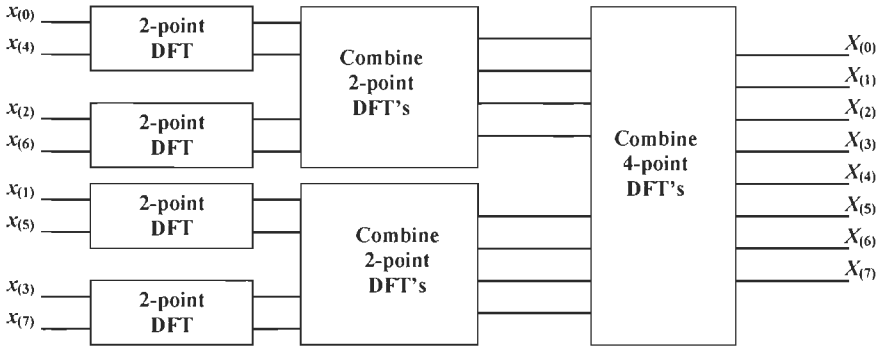


Figure 2: Three stages in the computation of an $N = 8$-point DIT DFT.

It is also possible to derive FFT algorithms that first go through a set of $\log_2 N$ iterations on the input data, and rearrange the output values into bit-reverse order. These are called decimation-in-frequency (DIF) or Sande-Tukey FFT algorithm Figure (3) [19].

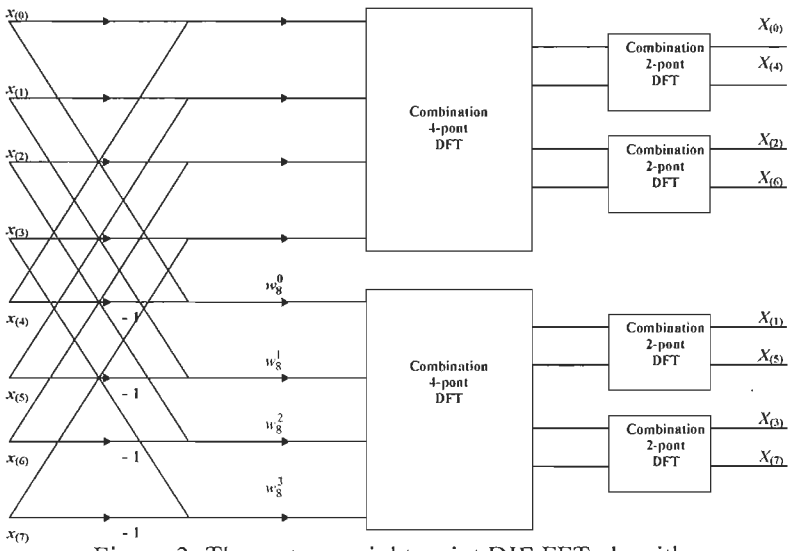


Figure 3: Three stages eight-point DIF FFT algorithm.

2.1.2 The Radix-2 Algorithm

In this section, the development of the radix-2 DIF FFT will be described in detail. The integers ***n*** and ***k*** in equation (1) (for the case $N = 2^{\gamma}$) can be expressed in binary numbers as [15],

$$n = 2^{\gamma-1}n_{\gamma-1} + 2^{\gamma-2}n_{\gamma-2} + \ldots\ldots + n_0 \qquad (2),$$

$$k = 2^{\gamma-1}k_{\gamma-1} + 2^{\gamma-2}k_{\gamma-2} + \ldots\ldots + k_0 \qquad (3).$$

in which n and k can take the values 0 and one only. Equation (1) can be rewritten as

$$X_{(k_{\gamma-1},k_{\gamma-2},\ldots,k_0)} = \sum_{n_0=0}^{1}\sum_{n_1=0}^{1}\sum_{n_2=0}^{1}\cdots\sum_{n_{\gamma-1}=0}^{1} x_{(n_{\gamma-1},n_{\gamma-2},\ldots,n_0)} w_8^{\left(2^{\gamma-1}n_{\gamma-1}+2^{\gamma-2}n_{\gamma-2}+\cdots+n_0\right)\left(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+k_0\right)}. \qquad (4)$$

Now, the γ^{ple} sum can be divided into γ separate summations

$$X_{1(k_0,n_{\gamma-2},\ldots,n_0)} = \sum_{n_{\gamma-1}=0}^{1} x_{(n_2,n_{\gamma-1},\ldots,n_0)} w_N^{\left(2^{\gamma-1}n_{\gamma-1}+2^{\gamma-2}n_{\gamma-2}+\cdots+n_0\right)k_0}, \qquad (5)$$

$$X_{2(k_0,k_1,n_{\gamma-3},\ldots,n_0)} = \sum_{n_{\gamma-2}=0}^{1} X_{1(k_0,n_{\gamma-2},\ldots,n_0)} w_N^{\left(2^{\gamma-2}n_{\gamma-2}+2^{\gamma-3}n_{\gamma-3}+\cdots+n_0\right)2k_1}, \qquad (6)$$

-

-

$$X_{\gamma(k_0,k_1,\ldots,k_{\gamma-2},n_0)} = \sum_{n_{\gamma-2}=0}^{1} X_{\gamma-1\left((k_0,k_1,\ldots,k_{\gamma-2},n_0)\right)} w_N^{n_0 2^{\gamma-1}k_{\gamma-1}}, \qquad (7)$$

The computation of equation (1) has been divided into $\log_2 N = \gamma$ stages, each having a computational complexity of N, and thus, the result of the manipulations is that the total computational complexity has decreased from N^2 to $N \log_2 N$. If the result needs to be in the natural order, an unscrambling stage for X_γ is needed:

The signal flow graph for an 8-points radix-2 DIF FFT is shown in figure (4) in which the butterfly has been introduced as the primitive operation of the FFT. The radix-2 butterfly consists of two complex additions and one complex multiplication and it is shown in figure (5).

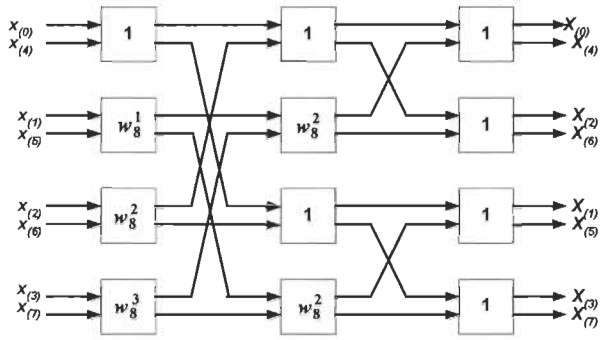


Figure 4: Eight-point DIF FFT Signal Flow Graph (SFG).

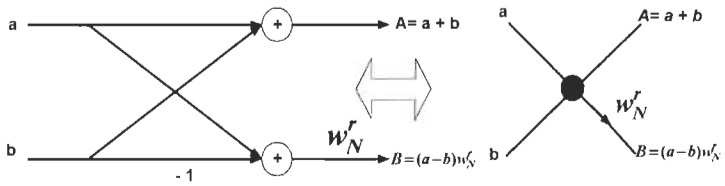


Figure 5: Basic butterfly computation for the DIF FFT algorithm.

2.1.3 Higher Radix

Another class of FFTs subdivides the initial data of length N not all the way down to the trivial transform of length 2, but rather only down to some other small power of two, four or eight (Figure 6). The result obtained in the above section could be extended to any base r that can be developed using the same techniques as for radix-2 case.

$$n = r^{\log_r(N-1)} n_{\log_r(N-1)} + r^{\log_r(N-2)} n_{\log_r(N-2)} + \ldots + n_0 , \qquad (8)$$

$$k = r^{\log_r(N-1)} k_{\log_r(N-1)} + r^{\log_r(N-2)} k_{\log_r(N-2)} + \ldots + k_0 . \qquad (9)$$

The derivation of such algorithm will not be discussed, but some key points are listed below.

- Transform length $N = r^m$ where m is an integer.
- The arithmetic complexity is $N \times \log_r N \times$ the arithmetic complexity of the radix r butterfly.
- The number of stages is equal to $\log_r N$.
- The number of butterflies in each stage is equal to N/r.
- Each butterfly has r-inputs and r-outputs and $r - 1$ twiddle factor multiplication.

The advantage of using higher radix is that the number of multiplications and the number of stages decrease. The number of stages often corresponds to the amount of global communication and/or memory accesses in an implementation, and thus, the reduction in the number of stages is beneficial if communication is expensive as is the case in most hardware implementation [15], [20] and [21]. Up to date, the most disadvantage of using a higher radix is that the butterfly becomes more complicated and can be difficult to implement. Most subsequent authors have directed their attention to the special case of $N = 2^m$ due to its simplicity in programing and the restricted choice of values of N is adequate for a majority of applications.

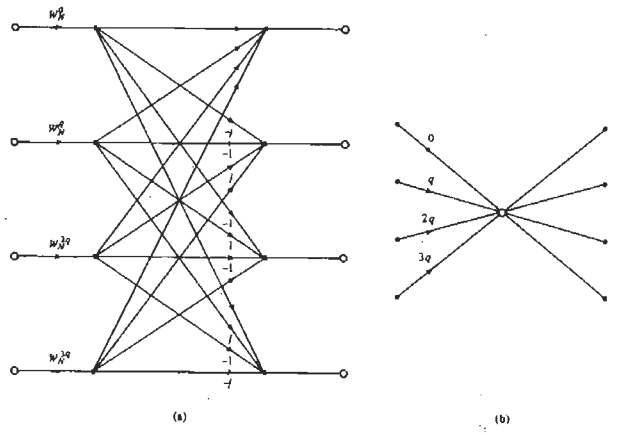


Figure 6: Basic butterfly computation in a radix-4 FFT algorithm.

2.2 The Mixed radix and Prime Factor Algorithm (PFA)

There are, however, some applications in which a wider choice of values of N is needed, especially in speech's spectral analysis and economic series data. Since the transform length is a problem, the mixed radix approach makes it possible [22]-[23]. This approach was achieved by decomposing the matrix T_r into a radix-3 stage and a radix-5 stage in order to compute a 15 points DFT [12].

Gentleman and Sand have extended the development of the general case by mentioning the existence of a mixed radix FFT program written by sand [14]. So, R. C. Singleton proposed an improved method by decomposing N into its prime factor, yielding an algorithm for computing the mixed radix Fast Fourier Transform [21]. The larger the prime factor of N is the worse this methods works because if N is prime, then no subdivision is possible, and the complexity would be of order N^2.

Since the twiddle factors dominate the arithmetic workload in the common factor algorithm, it seems logical to try to remove them and that is exactly what prime factor algorithms are all about. This approach can only be used if N can be decomposed into factors relatively prime, i.e.

$$N = N_1 \times N_2 \tag{10}.$$

Therefore, the greatest common divisor for N_1 and N_2 is equal to one and for this case, $\boldsymbol{n}$ and $\boldsymbol{k}$ can be expressed as [15],

$$n = ((N_2 \times n_1 + N_1 \times n_2))_N \; n_1 \in [0, N_1 - 1],\; n_2 \in [0, N_2 - 1] \tag{11},$$

$$k = \left[\!\left[N_2 \left[\!\left[N_2^{-1} \right]\!\right]_{N_1} \right]\!\right]_{N_1} \times k_1 + \left[\!\left[N_1 \left[\!\left[N_1^{-1} \right]\!\right]_{N_2} \right]\!\right]_{N_2} \times k_2 \tag{12}$$

$k_1 \in [0, N_1 - 1]$, $k_2 \in [0, N_2 - 1]$

where $[\![x]\!]_N$ denotes x modulo N and $[\![x^{-1}]\!]_N$ denotes the multiplicative inverse of x reduced modulo N.

Using these expressions equation (1) can be represented as

$$X_{\left(\left[N_2\left[N_2^{-1}\right]\right]_{N_1}\times k_1+\left[N_1\left[N_1^{-1}\right]\right]_{N_2}\times k_2\right)}=\sum_{n_2=0}^{N_2-1}\left[\sum_{n_1=0}^{N_1-1}x_{\left([\![N_2n_1+N_1n_2]\!]_N\right)}w_{N_1}^{k_1n_1}\right]w_{N_2}^{k_2n_2} \tag{13}$$

which means that the one-dimensional transform has been expressed as a true two-dimensional transform without inverting the twiddle factor. The signal flow graph of a 12 point prime factor algorithm is shown in figure (7) for $N_1 = 4$ and $N_2 = 3$.

These are essentially three drawbacks of the prime factor algorithm:

- Since N to be decomposed into factors that are relatively primes, these factors will be large if N is large. This means that the short-length DFTs, which have a length, equal to these factors, becomes expensive to implement.
- There are more severe restrictions on the transform length than in the case of the common factor algorithm.

These problems were dealt with and developed for the first time by S. Winograd and were referred as the Winograd Fourier Transform Algorithm (WFTA) [16]. Winograd used the indexing scheme from the prime factor algorithm, expressed in small DFTs in terms of convolution, and combined this with efficient methods for computing periodic convolution [24]. The price paid for this decrease in multiplication compared to common factor algorithms, is a significant increase in the number of addition and the irregular structure [24].

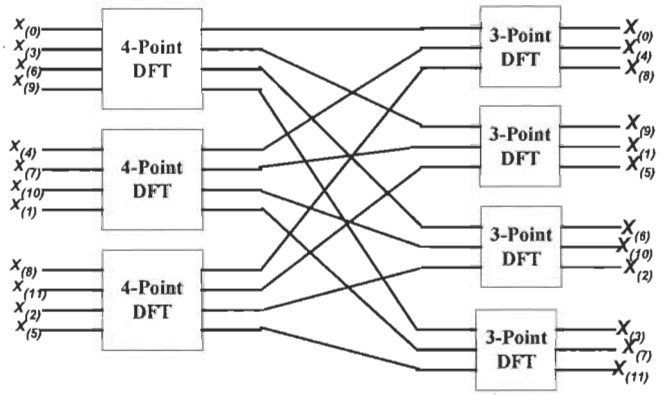


Figure 7: 12 point PFA (N_1 =4, N_2 =3).

2.3 Winograd Algorithm (WA).

Winograd algorithms are in some way analogous to the base-4 and base-8 FFTs, where he has derived a highly optimized coding for taking small-*N* discrete Fourier transforms. The method involves a reordering of the data both before the hierarchical processing and after it, but it allows a significant reduction in number of multiplication in the algorithm. For some especially favorable values of *N*, the Winograd algorithm can be significantly (up to factor 2) faster than the simple FFT algorithms, however, this advantage must be weighed against the considerably more complicated data indexing involved in these transforms. So, in 1977 based on Winograd method, Silverman proposed an improved method to calculate the FFT [25].

2.4 The Split Radix Algorithm (SRA)

The split radix algorithm is an FFT based algorithm which was introduced by R. Yavne [26] and then developed by Duhamel and Hollmann [17]. It seems that the split-radix has achieved the lowest published arithmetic operation count (total exact number of required real additions and multiplications) on a data of size *N* which is a power of two [27]-[32]. The basic idea behind the SRFFT as derived by Duhamel and Hollman [17] is the

application of a radix 2-index map to the even-indexed terms and a radix-4 map to the odd-indexed terms. Under these indexing scheme equation 1 can be rewritten as

$$X_{(2k)} = \sum_{n=0}^{\frac{N}{2}-1} \left(x_{(n)} + x_{\left(n+\frac{N}{2}\right)} \right) w_N^{2nk} \tag{14}$$

for the even index terms, and

$$X_{(4k+1)} = \sum_{n=0}^{\frac{N}{4}-1} \left(\left(x_{(n)} - x_{\left(n+\frac{N}{2}\right)} \right) - j \left(x_{\left(n+\frac{N}{4}\right)} - x_{\left(n+\frac{3N}{4}\right)} \right) \right) w_N^{n} w_N^{4nk} \tag{15}$$

$$X_{(4k+3)} = \sum_{n=0}^{\frac{N}{4}-1} \left(\left(x_{(n)} - x_{\left(n+\frac{N}{2}\right)} \right) + j \left(x_{\left(n+\frac{N}{4}\right)} - x_{\left(n+\frac{3N}{4}\right)} \right) \right) w_N^{3n} w_N^{4nk} \tag{16}$$

for the odd index terms. This results in an "L-shaped butterfly" figure (8), which relates a length N DFT to one length $N/2$ DFT and two-length $N/4$ DFT [33]. Such algorithms are known by having the lowest of both multiplication and addition for length 2^m FFTs, but it involves significantly more butterfly computations than radix 4 Cooley-Tukey Algorithms [34], [52] and [58].

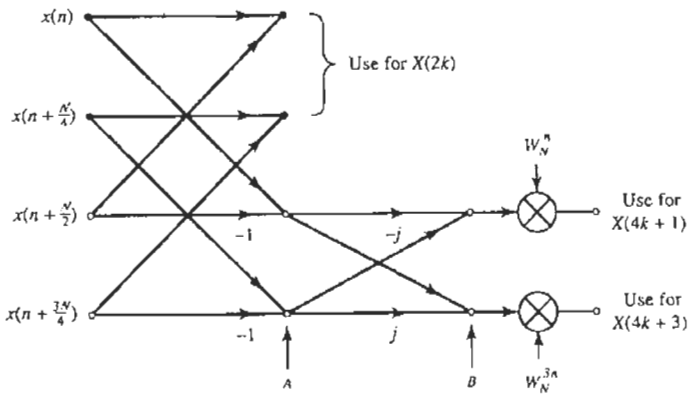


Figure 8: Butterfly for SRFFT algorithm.

In the previous sub sections, different algorithm and butterfly structures for the dedicated FFT were cited. The main objective of these proposals was a reduction in computation and particularly the reduction in the number of multiplications where the most common way to compare algorithms is based on the number of arithmetic operations.

In Table 1 and Table 2 the number of real multiplications and real additions required for different algorithms are listed [15] and [35]. In these tables; it is assumed that non-trivial complex multiplication is implemented using 3 real multiplications and 3 real additions. It is clear that the split radix and the Winograd algorithms offer the lowest number of multiplications for small and medium length FFTs whereas the Winograd algorithm require less multiplications than every other algorithm for long FFTs. From Table 2, it can be seen that the split radix algorithm offers the lowest number of additions.

Table 1: Number of non-trivial real multiplications for various FFTs

N		Radix-2	Radix-4	SRA	PFA	WA
16		24	20	20		
	30				100	68
32		88		68		
	60				200	136
64		264	208	196		
	120				460	276
128		712		516		
	140				1100	632
256		1800	1392	1284		
	504				2524	1572
512		4360		3076		
	1008				5804	3548
1024		10248	7856	7172		
	2048	23560		1638		
2520					17660	9492

In conclusion it is hard to make a fair and general comparison between the different algorithms because the importance of different properties of the algorithms is depending on the implementation. In the case of hardware implementation of FFT processors there are

number of other algorithm's properties that should be dealt with such as: regularity, modularity, parallelism and simplicity which are mostly offered by the common and prime factor algorithms.

Table 2: Number of non-trivial real additions for various FFTs

N		Radix-2	Radix-4	SRA	PFA	WA
16		152	148	148		
	30				384	384
32		408		388		
	60				888	888
64		1032	976	964		
	120				2076	2076
128		2504		2308		
	140				4812	5016
256		5896	5488	5380		
	504				13388	14540
512		13566		12292		
	1008				29548	34668
1024		30728	28336	27652		
	2048	68616		61444		
2520					84076	99628

Finally, in this chapter we will define the problematic and major challenges, to finally demonstrate clearly our methodology, originality and scientific contribution.

3. Problematic and major challenges

The FFTs are typically used to input large amounts of data; perform mathematical transformation on that data and then output all the resulting data at very high rates. The mathematical transformation is translated into arithmetic operations (multiplications, summations or subtractions in complex values) following a specific dataflow structure that should control the systems' input/output. Multiplication and memory accesses are the most significant factors on which the execution time relies therefore; the major challenge is to reduce the multiplication load in a simple dataflow structure to facilitate the parallel and

pipeline implementation in one hand and on the other hand to reduce the coefficient multiplier memories' accesses by regrouping the data with its corresponding coefficient multiplier. The quality of spectral information extracted from a signal relies on two major components:

- Spectral resolution which means high sampling rate that will increase the implementation complexity to satisfy the time computation constraints.
- Spectral accuracy which is translated into an increasing of the data binary word-length that will increase normally with the number of arithmetic operations.

The problem with the computation of an FFT with an increasing N is associated with the straightforward computational structure, the coefficient multiplier memories' accesses and the number of multiplication that should be performed. In high resolution and better accuracy this problem will be more and more significant for real time FFT implementation and in order to achieve our objective we should address the problems with the mathematical structure of the FFT that could be summarized as follow:

1. An FFT of size N ($N = r^n$) is computed in n stages therefore, for larger N the number of stages will increase which could be translated into an increase of the communication load and the computational load. So, increasing r will decrease n but it has been shown (e.g. [15]) that the adder tree simplification method did not provide a complete solution for the FFT problem due to the increasing complexity of the butterflies. For higher radices, the complexity of the butterfly implementation increases due to the added complex multipliers on its data path [15], and [36]-[39].

2. Another attempt to speed up the FFT process, that does not necessarily involve computational reduction, is the parallel multiprocessing. One of the most significant problem in FFT implementation resides in its data's parallel multiprocessing. This difficulty arises in finding a feasible algorithm that could meet the following objectives [40]- [47] and [64]:

    i) To build an algorithm, which could be easily implemented on VLSI technologies (DSP, FPGA and ASIC)

    ii) The r parallel processors should execute a single instruction simultaneously.

    iii) Reduce the NOP (no operations) to its minimum value.

    iv) Reduce the communication load between the r processors.

    v) Reduce the computational load.

    vi) No Pipeline break (or "pipeline stall"): the delay caused on a processor using pipelines when a transfer of control is taken (is absent).

    vii) Straightforward design for real time FFT implementation.

3. Memories' accesses are major concerns in implementation on DSP cards which on the most cases are costly in DSP cycles. Therefore, in a real time implementation, executing and controlling the data flow structure is important in order to achieve high performance that could be obtained by regrouping the data with its corresponding coefficient multiplier [48]. By doing so, the access to the coefficient multiplier's memory will be reduced drastically and the multiplication by w^0 ($w_N^{nk} = e^{-j(2\pi/N)nk}$) will be taken out of the equation.

4. The scope of work in this thesis is to target the wireless communication such as OFDM therefore; I will be paying more attention to pipelined and pruning FFTs

algorithms where pruning FFTs are used to monitor specific frequencies outputs [49]-[51]. Consequently, for such types of signals' monitoring FFTs we will be investigating two types of algorithms:

i. Goertzel Algorithm

ii. Input/output pruning FFT [59]-[63].

4. Methodology, Originality and Scientific Contributions

In order to address the higher radices butterflies' problem, our main objective is to reduce the complexity of the butterfly's critical path that could be achieved in two ways:

- The proposed structure in [51] has reduced the complexity of the butterfly's critical path as a result our objective is to minimize the resources needed to implement[1] higher radices butterflies.
- A hardware oriented Radix 2^{α} or 4^{β} which is an alternative way of representing higher radices by mean of less complicated and simple butterflies [29] in which we used the symmetry and periodicity of the root unity to further lower down the coefficient multiplier memories' accesses [53] and [54].

Up to date there was no attempt to reduce the computational load by incorporating the twiddle factors and the adder tree matrices into a single stage of calculation. So, if we pay attention to the elements of the adder tree matrix $\mathbf{T}_r$ and to the elements of the twiddle factor matrix $\mathbf{W}_N$, we notice that both of them contain twiddle factors. So, by controlling the variation of the twiddle factor's exponent during the complete FFT calculation, we can incorporate the twiddle factors and the adder tree matrices into a single stage of calculation

[1] This originality has been approved by several patents filed since 2004.

which will represent the originality of our proposed method and based on this, we will propose new concepts of the FFT implementation[2].

This was the origin of our mathematical model for the butterfly computation that will be detailed in the paper of Chapter 2 "A New FFT Concept for Efficient VLSI Implementation: Part I Butterfly Processing Element DSP'09, Santorini, Greece, 5-7 July 2009", where we have introduced a novel approach for the Discrete Fourier Transform (DFT) factorization by redefining the butterfly computation, which is more suitable for efficient VLSI implementation. The proposed factorization motivated us to present a new concept of the radix-*r* Fast Fourier Transform (FFT), in which the radix-*r* butterfly was formulated as composite engines to implement each of the butterfly computations. This concept enables the radix *r* butterfly-processing element (BPE) to be designed by maintaining only one complex multiplier in the butterfly critical path for any given *r*. Once this article was published Kim and al proposed in [55] a proper multiplexing scheme that reduces the usage of complex multiplier for the radix-8 butterfly from 11 to 5. The proposed method for the radix-8 case was implemented on FPGA where we have targeted in our comparison the Spartan-3, Virtex-E, Virtex-4 and Virtex 5 families. The proposed method's implementation results achieved better performance in terms of the throughput per area ratio (Msamples/s/slice) as shown in the paper of the same Chapter "A Higher Radix FFT FPGA Implementation Suitable for OFDM Systems ICECS 2011, Beirut Lebanon".

Another attempt to speed up the FFT process, that does not necessarily involve computational reduction, is the parallel multiprocessing. Based on the reformulation of the

[2] This originality has been approved by several patents filed since 2004.

radix-r factorization we defined the concept of the parallel pipelined FFT that boosted the execution time by a factor of r (Chapter 2 paper "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing" DSP 2009, Santorini, Greece, 5-7 July 2009") in which one stage (or iteration) will be eliminated for each break down of the DFT [36].

One of the recent strategies to reduce the computational load is to target the trivial multiplication that could be achieved by grouping the data with its corresponding coefficients multipliers [15] and [56]. By achieving this, all trivial multiplications by ± 1 or $\pm j$ have been excluded from the process and adding to that the accesses to the coefficient multipliers have been also reduced. With a proper indexing scheme which will be based on the radix-r DFT factorization, the bit reversing techniques for accessing the data including the coefficient (twiddle factor) multiplier would be replaced by simple counter as address generators to boost the execution time of the FFT [56]. The originality of the indexing scheme is detailed in the paper "A Novel Approach for the FFT Data Reordering, Int. Symp. On Circuit and System (ISCAS), Paris, May 2010" of chapter 3 in which this concept was tested on the FFTW platform [57] that shows a significant improvement on the execution time of the FFT process. With further prediction of trivial multiplication such as $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ will lead to a reduction in the memory accesses and where the number of arithmetical operation required for the complex multiplication can be reduced from 6 to 4 arithmetical operations. By using the symmetry and periodicity of the root unity to lower down the computational effort, we developed an algorithm that could eliminate the trivial multiplication (eightth root of unity) which was elaborated in the paper of chapter 5 "The Self-Sorting JMFFT Algorithm with Automatic Elimination of Trivial Multiplication,

NEWCAS 2012 Montreal, Canada" that will yield to the radix-2^3 kernel core FFT computation in which the kernel core is composed of 4 butterfly radix-2 that will access one coefficient multiplier per 8 inputs. The method is elaborated in the paper of the same Chapter "A Novel Radix-2^3 JMFFT Suitable for Embedded DSP Processor, to be submitted to a Journal", where the memory requirement to stock the coefficient multiplier is reduced from $N/2 - 1$ to $N/8 - 1$ and this will be highly desirable for a small and low power special purpose FFT processor.

Further decomposition of the DFT in term of its partial DFT will yield to the radix-r one stage FFT kernel computation which is used to compute a specific frequency output that is widely used in monitored signals. The proposed algorithm showed a significant gain by a factor of $log_r N$ compared to the conventional radix-r in order to compute a specific frequency as shown in the paper of Chapter 4 "The Radix-r One Stage FFT Kernel Computation, ICASSP 2008, April First Las Vegas Nevada USA 2008". By keeping all twiddle factors in memory in order to compute one frequency, the proposed algorithm has been compared to Goertzel Algorithm that revealed a substantial gain in speed and SQNR as described in the paper of Chapter 4 "Fast Method to Detect Specific Frequencies in Monitored Signal, International Symposium on Communications, Control and Signal Processing (ISCCSP 2010), Cyprus, March 2010". Furthermore, this algorithm has been optimized in order to obtain the proposed method named JM-Filter (Jaber-Massicotte Filter) that would have approximately the same structure as the Goertzel Filter with a reduction in computation by a factor of r as demonstrated in the paper of the same Chapter "The JM-filter to Detect Specific Frequencies in Monitored Signal, to be submitted to a Journal".

Finally and based on this strategy by incorporating the twiddle factors and the adder tree matrices into a single stage of calculation we manifested a significant gain in the reduction of the computational load in the input/output pruning FFT which is widely used in the OFDM, OFDMA and MIMO-OFDM as shown in the paper of Chapter 6 "Low Complexity Input/output Pruning JMFFTs; to be submitted to a Journal".

Chapter 2 The Radix-*r* Butterfly Processing Element

Paper I: M. Jaber and D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I Butterfly Processing Element", *International Conference on Digital Signal Processing* (DSP), Santorini, Greece, 5-7 July 2009.

Paper II: M. Jaber and D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing", *International Conference on Digital Signal Processing* (DSP), Santorini, Greece, 5-7 July 2009.

Paper III: M. Jaber, D. Massicotte, and Y. Achouri, "A Higher Radix FFT FPGA Implementation Suitable for OFDM Systems", *IEEE International Conference on Electronics, Circuits, and Systems* (ICECS), Beirut Lebanon, December 2011.

Résumé Du Chapitre 2

La transformée rapide de Fourier (TRF) est l'un des quelques algorithmes dont la publication a révolutionné le domaine de l'analyse spectrale. La TRF a permis de réduire la complexité de la transformée discrète de Fourier de N^2 à $(N/r)\ log_r\ N$ opérations complexes. Les accès aux mémoires sont aussi un facteur important influençant sur la rapidité de l'exécution de la TRF où la TRF à radice élevée peut réduire les accès mémoires et la complexité arithmétique de l'algorithme. Les désavantages de la structure du papillon à radice élevée sont;

1) La complexité de la connectivité
2) Nombres d'entrées en parallèles
3) L'augmentation de la profondeur du passage critique du papillon.
4) L'ajout des multiplicateurs complexes dans le passage critique du papillon.

Ce chapitre décrit une nouvelle approche pour le calcul en papillon à base élevée dédiée pour la structure en pipeline de la Transformée rapide de Fourier. En se basant sur le concept introduit par Cooley-Tukey, nous introduisons une nouvelle approche de factorisation de la DFT (Discrete Fourir Transform) en redéfinissant le calcul en papillon qui nous a permis de concevoir l'élément du traitement en papillon BPE (Butterfly Processing Element). Cette structure nous a permis de maintenir un seul multiplicateur complexe dans le chemin critique du papillon pour tout r donné. La description algorithmique, la performance, et la complexité de la méthode sont considérées dans ce chapitre. Ce chapitre présente également une solution au problème du multitraitement en parallèle de la FFT et implique de nouveaux concepts dans lesquels la réalisation, des pipelines à plusieurs étages en parallèle, est possible. La contribution la plus importante dans notre proposition, c'est que notre structure de BPE proposée maintient un seul

multiplieur complexe dans le chemin critique peu importe le radix 2,4 ou 8, rendant ainsi notre solution des BPE fort avantageux pour l'implantation en technologies VLSI

Paper I: M. Jaber and D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I Butterfly Processing Element", *International Conference on Digital Signal Processing* (DSP), Santorini, Greece, 5-7 July 2009.

A New FFT Concept for Efficient VLSI Implementation: Part I Butterfly Processing Element

Marwan A. Jaber and Daniel Massicotte

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations
{marwan.jaber, daniel.massicotte}@uqtr.ca

Abstract – **This article describes a new approach of higher radices butterflies which is suitable for pipeline implementation. Based on the butterfly computation introduced by Cooley-Tukey [1], we redefined the DFT factorization by introducing the new concept of the butterfly computation that is suitable for efficient VLSI implementation. Finally, we will present a new concept of a radix-*r* Fast Fourier Transform (FFT), in which the concept of the radix-*r* butterfly computation has been formulated as composed engines with identical structures and a systematic means of accessing the corresponding multiplier coefficients. This concept enables the design of the butterfly processing element (BPE) with the lowest rate of complex multipliers and adders that utilizes *r* or *r* – 1 complex multipliers in parallel to implement each of the butterfly computations. Furthermore, the parallel pipelined FFT is also considered in Parallel Pipelined FFT Processing [15].**

1. Introduction

The Discrete Fourier Transform (DFT) is a fundamental digital signal-processing algorithm used in many applications, including frequency analysis and frequency domain processing. Frequency analysis provides spectral information for signals that are examined or used in further processing, such as speech compression, meanwhile the frequency domain processing allows for the efficient computation of the convolution integral (for

linear filtering), the computation of the correlation integral (for correlation analysis), and in wireless communication system based on the Orthogonal Frequency Division Multiplexing (OFDM) where the FFT is a major key operator [2], [3]. It is not unusual to find more than a dozen structures to complete a given task, so, finding the best structure is a crucial engineering problem, where in reality the class of the most efficient structures rests on a real time analysis of the signals to be processed. Thus, this paper proposes a new FFT algorithm which is able to increase the computation speed for an equivalent implementation complexity.

The definition of the DFT is represented by the following equation

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \quad , \quad k \in [0, N-1] \tag{1}$$

where $x_{[n]}$ is the input sequence, $\mathbf{X}_{[k]}$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ called the twiddle factor in the butterfly structure, and $j^2 = -1$. Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex number sequences.

From Eq. (1), it can be seen that the DFT computational complexity increases according to the square of the transform length, and thus becomes expensive for large N. Some algorithms used for efficient DFT computation, known as fast DFT computation algorithms, are based on the divide-and-conquer approach. The principle of this method is dividing a large problem into smaller sub-problems that are easier to solve. In the FFT case, dividing the work into sub-problems means that the input data $x_{[n]}$ can be divided into subsets from which the DFT is computed, and then the DFT of the initial data is reconstructed from these intermediate results. Some of these methods are known as the Cooley-Tukey algorithm [1], Split-Radix Algorithm [4], Winograd Fourier Transform

Algorithm (WFTA) [5] and others, such as the Common Factor Algorithms [2], [5], and [6].

The overall arithmetic operations deployed in the computation of an N-point FFT decreases with increasing r as a result, the butterfly complexity increases in term of complex arithmetic computation, parallel inputs, connectivity, and number of phases in the butterfly's critical path delay. The higher radix butterfly involves a non-trivial VLSI implementation problem (i.e. increasing butterfly critical path delay), which explains why the majority of FFT VLSI implementations are based on radix-2 or 4, due to their low butterfly complexity. The advantage of using a higher radix is summarized by decreasing the number of multiplications and stages to execute an FFT [2]. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation, and thus the reduced number of stages becomes beneficial if communication is expensive, as is the case in most hardware implementations. Fewer attempts to reduce the computational load have failed, due to the added multipliers in the butterfly's critical path for higher radices [7], [8].

The most significant impact in our proposed BPE structure is by maintaining lower arithmetic operations within its critical path (one complex multiplier and few adders). Based on this proposition, the design of higher radices BPE with low butterfly complexity in terms of complex multipliers in the butterfly critical path is always feasible, which is the key component in the FFT implementation. By doing so, the VLSI butterfly implementation for higher radices would be feasible since it maintains approximately the same complexity of the radices 2 and 4 butterflies.

The paper is organized as follows; Section 2 details the radix-r DFT factorization while Section 3 defines the prior art butterfly operation. Section 4 provides a detailed description of the proposed FFT and the modified radix-*r* FFT methods. Section 5 presents the butterfly processing elements based on the modified radix-*r* FFT. Section 6 provides a performance evaluation while Section 7 reports the conclusions.

2. The Radix-*r* DFT Factorization – Demystified

Eq. (1) could be factorized as follow

$$\mathbf{X}_{(k)} = \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_N^{rnk} + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_N^{(rn+1)k} + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_N^{(rn+2)k} + \cdots + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_N^{(rn+(r-1))k} \quad , \tag{2}$$

for $k = 0,1,\cdots,N-1$. In the summations, the variables r and k are independents of n therefore, we can rewrite (2) as:

$$\mathbf{X}_{(k)} = \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_N^{rnk} + w_N^k \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_N^{rnk} + w_N^{2k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_N^{rnk} + \cdots + w_N^{(r-1)k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_N^{rnk} \quad , \tag{3}$$

Knowing that $w_N^{rnk} = w_{N/r}^{nk}$ as a result we can express (3) as follow

$$\mathbf{X}_{(k)} = w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{nk} + w_N^k \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{nk} + w_N^{2k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{nk} + \cdots + w_N^{(r-1)k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{nk} \quad . \tag{4}$$

To subdivide the axis k in Eq. (4) in 2 new axis p and q, we pose $k = p + qN/r$ with $p = 0,1,....,(N/r)-1$ and $q = 0,1,....,r-1$. By using the new indices p and q, $\mathbf{X}_{(k)}$ is replaced by:

$$\mathbf{X}_{(k)} = \mathbf{X}_{(p+q(N/r))}, \tag{5}$$

with $k = 0,1,\cdots,N-1$, $p = 0,1,....,(N/r)-1$ and $q = 0,1,....,r-1$. Following the axis v, equation (4) is expressed in r equations as shown in Eq. (6) or in a compact form Eq. (7).

$$\left.\begin{aligned}
\mathbf{X}_{(p)} &= w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{np} + w_N^p \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{np} + w_N^{2p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{np} + \cdots + w_N^{(r-1)p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{np} \\
\mathbf{X}_{(p+N/r)} &= w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+N/r)} + w_N^{(p+2N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+N/r)} + w_N^{2(p+N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{n(p+N/r)} + \\
&\quad \cdots + w_N^{(r-1)(p+N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+N/r)} \\
\mathbf{X}_{(p+2N/r)} &= w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+2N/r)} + w_N^{(p+2N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+2N/r)} + w_N^{2(p+2N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{n(p+2N/r)} \\
&\quad + \cdots + w_N^{(r-1)(p+2N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+2N/r)} \\
&\quad \vdots \\
\mathbf{X}_{(p+(r-1)N/r)} &= w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+(r-1)N/r)} + w_N^{(p+(r-1)N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+(r-1)N/r)} + \\
&\quad \cdots + w_N^{(r-1)(p+(r-1)N/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+(r-1)N/r)}
\end{aligned}\right\} \tag{6}$$

$$\begin{aligned}
\mathbf{X}_{(p+qN/r)} &= w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+qN/r)} + w_N^{(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+qN/r)} + w_N^{2(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{n(p+qN/r)} + \\
&\quad \cdots + w_N^{(r-1)(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+qN/r)}
\end{aligned} \tag{7}$$

Considering $w_{N/r}^{\alpha N/r} = \left(w_{N/r}^{N/r}\right)^{\alpha} = 1^{\alpha} = 1$, therefore Eq. (7) becomes:

$$\mathbf{X}_{(p+qN/r)} = w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{np} + w_N^p w_N^{qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{np} + w_N^{2p} w_N^{2qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{np} + \cdots + w_N^{(r-1)p} w_N^{q(r-1)N/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{np} \quad . \tag{8}$$

Finally, based on Eq. (8), Eq. (1) could be formulated in a matrix-vector equation as:

$$\begin{bmatrix} \mathbf{X}_{(p)} & \mathbf{X}_{(p+N/r)} & \mathbf{X}_{(p+2N/r)} & \cdots & \cdots & \mathbf{X}_{(p+(r-1)N/r)} \end{bmatrix}^T = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix} \times \begin{bmatrix} w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{np} \\ w_N^p \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{np} \\ w_N^{2p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{np} \\ \vdots \\ w_N^{(r-1)p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{np} \end{bmatrix}, \tag{9}$$

or

$$\begin{bmatrix} \mathbf{X}_{(p)} \\ \mathbf{X}_{(p+N/r)} \\ \mathbf{X}_{(p+2N/r)} \\ \vdots \\ \vdots \\ \mathbf{X}_{(p+(r-1)N/r)} \end{bmatrix} = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix} \times \begin{bmatrix} w_N^0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & w_N^p & 0 & \cdots & 0 & 0 \\ 0 & 0 & w_N^{2p} & 0 & \vdots & 0 \\ \vdots & 0 & 0 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & 0 & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & w_N^{(r-1)p} \end{bmatrix} \begin{bmatrix} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{np} \\ \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{np} \\ \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{np} \\ \vdots \\ \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{np} \end{bmatrix}. \tag{10}$$

We recognize the first and the second matrix, the well-known adder tree matrix $\mathbf{T}_r$ and the twiddle factor matrix $\mathbf{W}_N$, respectively. Equations (2)-(10) can be expressed in a compact form as:

$$\mathbf{X} = \mathbf{T}_r \mathbf{W}_N col\left(\sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \,\middle|\, q = 0,1,\cdots,r-1 \right), \tag{11}$$

for $k = 0,1,2,\cdots,N-1$, $p = 0,1,2,....,(N/r)-1$ and $q = 0,1,2,....,r-1$, with $\mathbf{X} = \left[\mathbf{X}_{(p)}, \mathbf{X}_{(p+N/r)}, \mathbf{X}_{(p+2N/r)}, \ldots, \mathbf{X}_{(p+(r-1)N/r)} \right]^T$, $\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p} \right)$, *col* refers to the column vector and

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix}. \tag{12}$$

From Eq. (11), we can write the well-known butterfly matrix $\mathbf{B}_r$, which can be expressed as

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N. \tag{13}$$

2. In Place (or Butterfly) Computation

The basic operation of a radix-*r* PE is the so-called butterfly computation in which *r* inputs are combined to give the *r* outputs via the operation:

$$\mathbf{X} = \mathbf{B}_r \mathbf{x}, \tag{14}$$

where $\mathbf{x} = [x_{(0)}, x_{(1)}, x_{(2)}, \cdots, x_{(r-1)}]^T$ and $\mathbf{X} = \left[X_{(0)}, X_{(1)}, X_{(2)}, \ldots, X_{(r-1)} \right]^T$ are, respectively, the BPE's input and output vectors. $\mathbf{B}_r$ is the butterfly matrix (dim ($\mathbf{B}_r$) = $r \times r$) which can be expressed as

$$\mathbf{B}_r = \mathbf{W}_N^r \mathbf{T}_r \tag{15}$$

for decimation in frequency (DIF) process, and

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N^r \tag{16}$$

for decimation in time (DIT) process. Therefore, the column vector, *col*(•), in (11) can be expressed as:

$$col\left(\sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \,\middle|\, q = 0,1,\cdots,r-1 \right) = [X_{(0)}, X_{(1)}, \cdots, X_{(r-1)}]^T \tag{17}$$

In both cases the twiddle factor matrix, $\mathbf{W}_N$ is a diagonal matrix defined by $\mathbf{W}_N = diag\left(1, w_N^p, w_N^{2p}, \ldots, w_N^{(r-1)p}\right)$ with $p = 0,1,\ldots,N/r^s - 1$, $s = 0,1,\ldots,\log_r N - 1$ and $\mathbf{T}_r$ is the adder-tree matrix within the butterfly structure (12).

Knowing that the higher radix will decrease the number of complex multiplications and the number of stages needed to execute a total *N*-point FFT, we were consequently oriented to the implementation of higher radices butterflies. Since the higher radix automatically reduces the communication load, the only problem remaining was the butterfly's complexity (computational load and phase number). The best-known technique for reducing the computational load is factoring the adder-tree matrix $\mathbf{T}_r$.

Fig. 1 shows the signal flow graph (SFG) for the radix-4 and the radix-8 butterflies. The computational reduction is achieved by incorporating the trivial multiplications j or $-j$ into the summation by switching the real and imaginary parts of the data. Factoring the summation matrix is the best-known method of computation reduction. As we move to higher radices, the complexity of such butterfly implementation increases and the amount of the non-trivial multiplications increases as shown in Fig. 1-b [2].

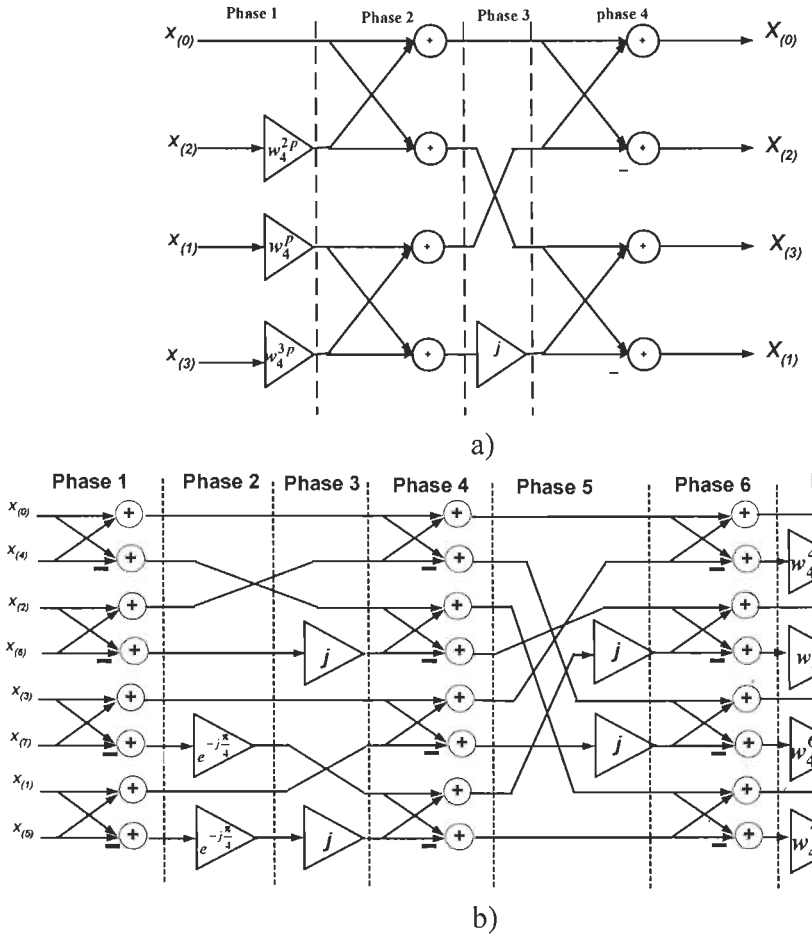


Figure 1: SFG of the a) radix-4 and b) radix-8 butterflies [2]

3. Proposed FFT Method

It has been shown that the adder tree simplification method did not provide a complete solution for the FFT problem due to the increasing complexity of the butterflies for higher radices [2]. The problem's solution resides in the structure of the adder tree matrix and the twiddle factor matrix. Thus, if we pay attention to the elements of the adder tree matrix $\mathbf{T}_r$ and to the elements of the twiddle factor matrix $\mathbf{W}_N$, we notice that both of them contain twiddle factors. So, by controlling the variation of the twiddle factor during the calculation of a complete FFT, we can incorporate the twiddle factors and the adder tree matrices into a single stage of calculation.

According to Eq. (15), $\mathbf{B}_r$ is the product of the twiddle factor matrix $\mathbf{W}_N$ and the adder-tree matrix $\mathbf{T}_r$. By defining the element of the l^{th} line and the m^{th} column in the matrix $\mathbf{T}_r$ as $\left[T_r\right]_{l,m}$:

$$\left[\mathbf{T}_r\right]_{l,m} = w_N^{\left\lfloor (lmN/r) \right\rfloor_N}, \tag{18}$$

and by defining $\mathbf{W}_{N(u,v,s)}$ the set of the twiddle factor matrix as

$$\mathbf{W}_{N(u,v,s)} = \mathrm{diag}\left(w_{N(0,v,s)}, w_{N(1,v,s)}, \ldots, w_{N(r\text{-}1,v,s)} \right), \tag{19}$$

in which each element of the diagonal matrix for the DIF process is represented as:

$$\left[\mathbf{W}_N\right]_{l,m(u,v,s)} = \begin{cases} w_N^{\left\lfloor \lfloor v/u^s \rfloor l\, u^s \right\rfloor_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}, \tag{20}$$

therefore, the proposed modified radix-r butterfly computation $\mathbf{B}_r$ is expressed, for the DIF process, as:

$$\mathbf{B}_r = \mathbf{W}_{N(u,v,s)} \mathbf{T}_r \tag{21}$$

will be simplified as:

$$\left[\mathbf{B}_r\right]_{l,m(v,s)} = w_N^{\left\lfloor lmN/r + \lfloor v/r^s \rfloor l r^s \right\rfloor_N}, \tag{22}$$

where the indices are $u = l = m = 0,1,\ldots,r-1$, $v = 0,1,\ldots,V-1$, $s = 0,1,\ldots,S$, r is the radix-r, V is the number of words ($V = N/r$), $\lfloor \mathrm{x} \rfloor$ represents the integer part operator of x, $\left[\!\left[\mathrm{x} \right]\!\right]_N$ represents the operation x modulo N and S is the number of stages ($S = \log_r N - 1$).

As a result, the operation of a radix-r BPE for the DIF FFT is formulated by the column vector:

$$\mathbf{X}_{(u,v,s)} = \mathbf{B}_r \mathbf{x}, \tag{23}$$

will be formulated with respect to the butterfly's l^{th} output as

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left\lfloor lmN/r + \lfloor v/r^s \rfloor l r^s \right\rfloor_N}. \qquad (24)$$

With the same reasoning as above, a radix-r DIT FFT operation can be derived.

4. BPE Structures

The conceptual key to the modified radix–***r*** **FFT** butterfly is the formulation of the radix–***r*** as composed engines with identical structures and a systematic means of accessing the corresponding multiplier coefficients [9]. This enables the design of an engine with the lowest rate of complex multipliers and adders, which utilizes r or $r - 1$ complex multipliers in parallel to implement each of the butterfly computations.

There is a simple mapping from the three indices *u, v,* and *s* (FFT stage, butterfly, and element) to the addresses of the multiplier coefficients needed (Fig. 2). This mapping is offered by the proposed FFT address generator [10]. For a single processor environment, this type of PE with r parallel multipliers would result in decrease in time delay for the complete FFT by a factor of $O(r)$.

It was shown in [11] that with further development of the proposed structure in Fig. 2 could yield to the one iteration FFT in which a specific frequency is computed in S cycles. Such implementation is very desirable for detecting the presence of a special known frequency in a monitored signal. This must be very efficient because most of the computed results with a conventional FFT are ignored.

A second aspect of the modified radix – r FFT butterfly, is that they are also useful in parallel multiprocessing environments as shown in Fig 3. In essence, the precedence relations between the engines in the radix– r FFT are such that the execution of r engines in

parallel is feasible during each FFT stage. If each engine is executed on the modified PE, it means that each of the r parallel processors would always be executing the same instruction simultaneously, which is very desirable for SIMD implementation on some of the latest DSP cards.

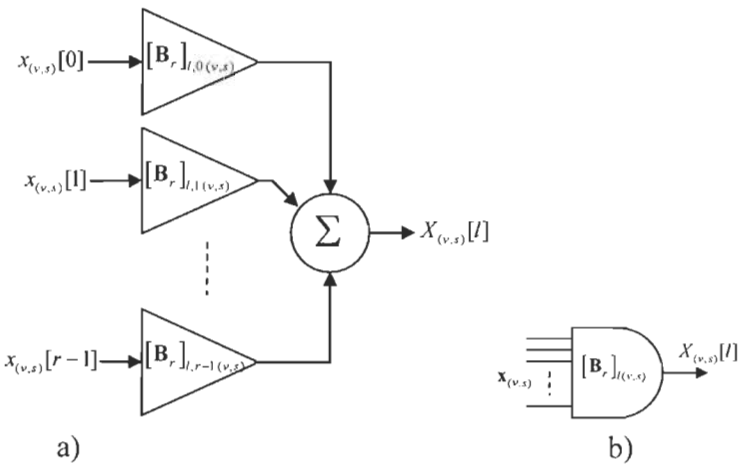


Figure 2: Radix-r partial BPE (l^{th} output) (a) using B_r in Eq. (16), and the symbol b).

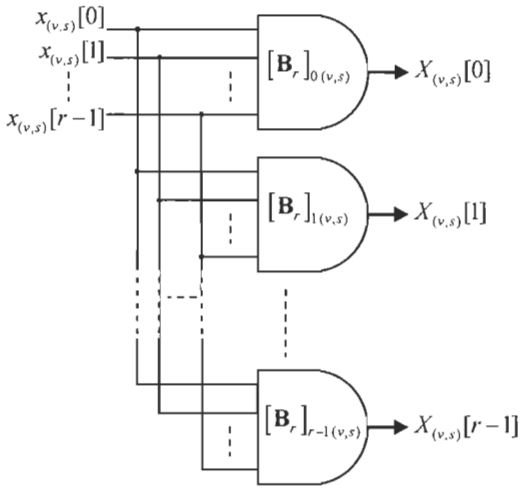
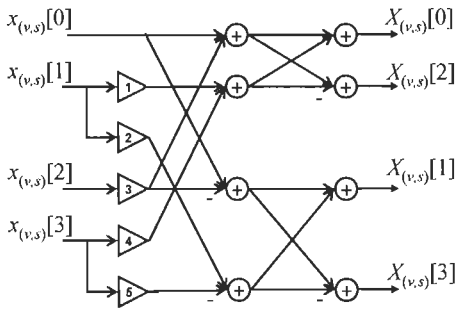


Figure 3: Maximize the data throughput using r BPE in parallel.

Based on this concept, an alternative hardware implementation could be achieved as shown in Fig. 4 and 5 for the radix-4 and 8, respectively. The hardware reductions in complex multipliers and adders are obtained by increasing the connectivity complexity of the butterfly structures.

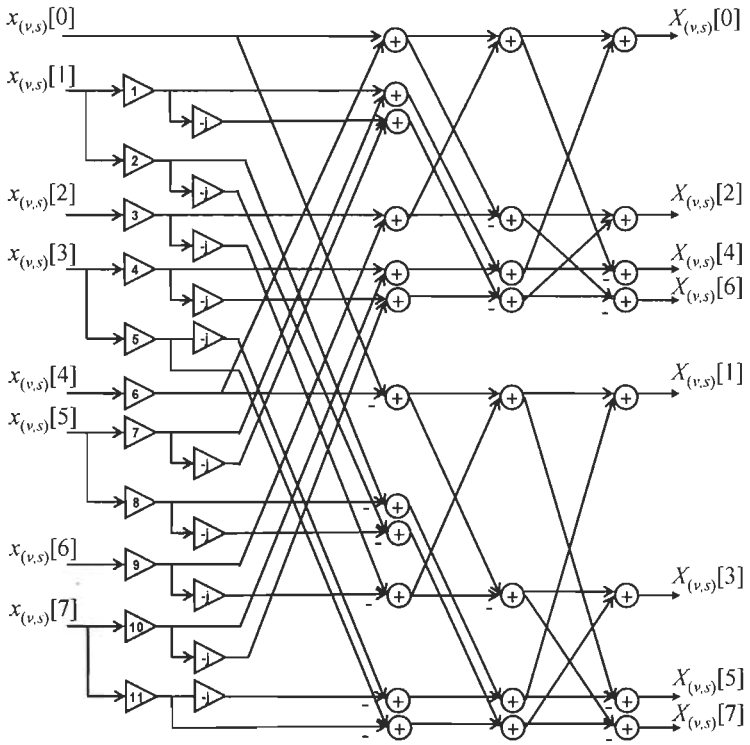


$$M_1 = w_N^{\left\lfloor \left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N}, \quad M_2 = w_N^{\left\lfloor N/4 + \left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N},$$

$$M_3 = w_N^{\left\lfloor 2\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N}, \quad M_4 = w_N^{\left\lfloor 3\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N},$$

$$M_5 = w_N^{\left\lfloor N/4 + 3\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N}$$

Figure 4: SFG of the proposed radix-4 BPE and the value of the multipliers M_i with i=1,2,…,5.



$$M_1 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)} \right\rfloor_N}, M_2 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)} + \frac{N}{8} \right\rfloor_N}, M_3 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 2\times 8^{(S-s)} \right\rfloor_N},$$

$$M_4 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)} \right\rfloor_N}, M_5 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)} + \frac{N}{8} \right\rfloor_N}, M_6 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 4\times 8^{(S-s)} \right\rfloor_N},$$

$$M_7 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)} \right\rfloor_N}, M_8 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)} + \frac{N}{8} \right\rfloor_N}, M_9 = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 6\times 8^{(S-s)} \right\rfloor_N},$$

$$M_{10} = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)} \right\rfloor_N}, M_{11} = w^{\left\lfloor \left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)} + \frac{N}{8} \right\rfloor_N}$$

Figure 5: SFG of the proposed radix-8 BPE and the value of the multipliers M_i with i = 1,2,…,11

5. Performance Evaluation

A recursive application of Eq. (16) can convert the DFT computation of length $N=r^S$ into S steps in order to compute r^S DFTs of length r. In each step N/r words have to be processed. Therefore, the sequential time computation, t_c, of the algorithm is given by:

$$t_c = \left(\frac{N}{r}\right) t_{BPE} \log_r N = \left(\frac{N}{r}\right) t_{BPE} S \tag{25}$$

where t_{BPE} is the BPE time computation which is depicted by its critical path.

As for computation time and in order to compare our proposed radix-r with the conventional radix-r structures, we assume that all data is present at the input, in a pre-ordered manner. We quantified this comparison based on the critical path delay for one BPE in order to obtain the first r outputs for an FFT of size r for the following radices 2, 4, 8 and 16. The conventional radix-r structures shown in Fig. 1, in which the BPE's critical paths are defined as:

i) Radix-4 needs 2 complex multiplications and 2 complex additions, and

ii) Radix-8 needs 4 complex multiplications and 3 complex additions. The proposed radix-4 BPE structure (Fig. 4) requires one complex multiplication and two complex additions, while radix-8 (Fig. 5) requires one complex multiplication and 3 complex additions. Our radix-2 BPE is the same as the conventional BPE.

Given that the time delay for real addition (T_A) is 4 times less than real multiplication (T_M), therefore, Table 1 summarizes the critical path delay based on T_M for each BPE.

By also assuming that the time delay $T_M = 4T_A$, Table 2 shows the performance result in terms of computation time t_c, between the proposed structures and the

conventional one for different radices and for N = 4096. The gain obtained by the conventional and the proposed radix-r is compared to the Cooley-Tukey (radix-2) algorithm which is also shown.

Table 1: Critical path delay of BPE for conventional and proposed BPE with Radix-2, 4, 8 and 16 in order to obtain the first r outputs.

Radices	Critical Path Delay [$\times T_M$]		Speed Gain
	Conventional	Proposed FFT	
Radix-2	4.75	4.75	1
Radix-4	5.25	5.00	1.05
Radix-8	10.25	5.25	1.95
Radix-16	18.0	6.0	3.0

One of the most FFT powerful implementation is the pipelined FFT (Fig. 6). An $N = r^S$ length FFT is implemented within S stages where each stage performs a radix-r butterfly. The switch blocks correspond to the data communication buses from the $(s-1)^{th}$ to the s^{th} stage where we considered the proposed switching concept in [14]. Since r data paths are used, the BPE pipeline achieves a data rate of S times the inter-module clock rate.

Table 2: Time computation results in terms of T_M between the proposed structures and the conventional one and the speed gain comparison with Cooley-Tukey (radix-2) for N=4096.

Radices	Conventional		Proposed FFT	
	t_c [$\times T_M$]	Speed Gain	t_c [$\times T_M$]	Speed Gain
Radix-2	116 736	1	116 736	1
Radix-4	32 256	3.6	30 720	3.8
Radix-8	20 992	5.6	10 752	10.8
Radix-16	13 824	8.4	4 608	25.3

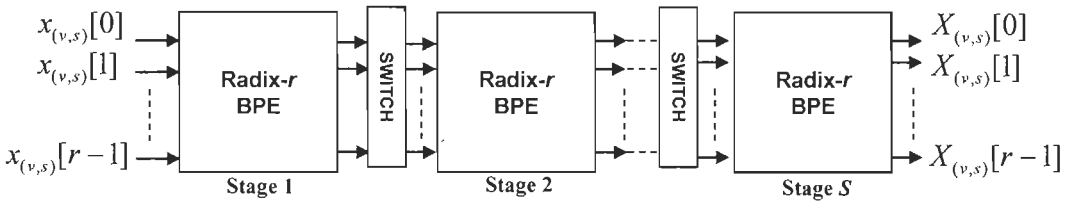


Figure 6: S Stages Radix-r Pipelined FFT.

Compared to [12] and [13], in which the radix-16 butterfly was represented as cascaded radix-2 butterfly known as very fast Fourier transforms (VFFT), the cited method's critical path will contain 5 cascaded complex multipliers and 4 complex adders which will make it slower than our proposed model by a factor of 5.

Fig. 7 shows the comparison in computation time for both pipelined structures in order to compute pipelined FFTs for different FFT-length (N). According to this figure we observe that:

i) For the conventional case the critical path increases when the $r > 4$;

ii) The critical path of the proposed FFT maintain one complex multiplier when r increases; and

iii) The critical path increases exponentially with N for the conventional radix-r when $r>4$ meanwhile the critical path of the proposed FFT remains invariant with increasing r.

Table 3 presents the comparison in terms of implementation resources needed to execute the respective BPE, in terms of:

i) Number of real number multiplications and additions,

ii) Number of full adders (FA) needed to implement both fixed-point arithmetic operators in the VLSI.

We assumed that a complex multiplication can be implemented using 3 real multiplications and 3 real additions. We considered 16-bit and 32-bit global lengths for the real multiplication and addition, respectively. Since our proposed model will have only one multiplier in the butterfly's critical path therefore, we can implement the real multiplication

with less overall bit word length than the conventional BPE, in order to obtain the equivalent result.

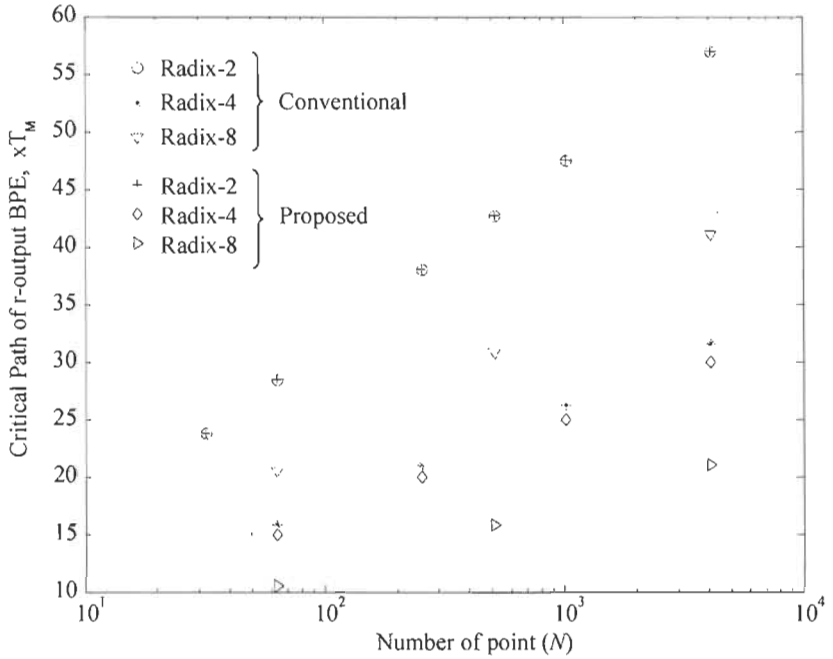


Figure 7: Critical path delay for *r*-output pipeline FFT conventional and proposed FFT Radix-2, 4 and 8.

Table 3: Number of real multiplications and additions for BPE and in term of FA (multiplier on 16-bit and adder on 32-bit).

Butterflies	Conventional			Proposed FFT		
	Mult	Add	FA	Mult	Add	FA
Radix-2	3	9	1056	3	9	1056
Radix 4	9	31	3296	15	41	5152
Radix 8	27	93	9888	33	111	12000

6. Conclusion

This article has presented an efficient implementation method for the FFT algorithm, where various issues concerning FFT implementation processors were discussed, placing the emphasis on butterfly processing elements (BPE) implementation. It can be argued that the higher radix FFT algorithms are advantageous for the hardware implementation, due to the reduced quantity of complex multiplications and memory access

rate requirements. In this paper, we showed that the implementation of a radix-r PE for the FFT is feasible. For the radix-8 and 16 cases, we have shown an improvement in the critical path delay for the proposed BPE by a factor of 2 and 3, respectively, compared to conventional BPE.

Multistage parallel pipelined FFT Architectures are presented in the companion paper entitle –."Parallel pipelined processing" – [15].

Acknowledgment

The authors would like to thank the financial support from the Natural Sciences and Engineering Research Council of Canada.

References

[1] W. Cooley, J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. Comput., 19, pp. 297-301 , April 1965.

[2] T. Widhe, "Efficient Implementation of FFT Processing Elements" Linkoping studies in Science and Technology, Thesis No. 619, Linkoping University, Sweden, June 1997.

[3] G. Klang, "A Study of OFDM for Cellular Radio Systems, M.Sc. Thesis, Linkoping University, Sweden 1994.

[4] P. Duhamel, and H. Hollman, "Split Radix FFT Algorithm", Electronics Letters, Vol. 20, No. I, pp. 14- 16, Jan. 1984.

[5] S. Winograd, "On Computing The Discrete Fourier Transform", Proc. Nat. Acad. Sci. USA, Vol. 37, pp 1005-1006,April 1976.

[6] J. Melander, "An FFT processor based on the SIC architecture with asynchronous PE", IEEE Midwest symposium on Circuits and Systems, Vol. 3, pp. 1313- 1316, Aug. 1996.

[7] Y. Wang et al., "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors", IEEE Trans. on signal Processing, Vol. 55, No.5, pp. 2338-2349, May 2007.

[8] S.G. Johnson and M. Frigo, "A Modified Split-Radix FFT with Fewer Arithmetic Operations", IEEE Trans. on signal Processing, Vol. 55,No. I,pp. 111-119,May2007.

[9] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6,751,643,2004 .

[10] M Jaber "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 82 and European patent application Serial no: PCT/USOI /07602

[11] M. Jaber and D. Massicotte "The Radix-r One Stage FFT Kernel Computation" ICASSP2008, April First Las Vegas Nevada USA.

[12] A. Despain, "Very Fast Fourier Transform Algorithms Hardware for Implementation", IEEE Trans. on Computers, Vol. C-28, No.5, May 1979.

[13] E. Wold, A. Despain, "Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementations", IEEE Trans. On Computers, Vol. C-33, No.5, May 1984.

[14] V. Szwarc et al., "A Chip Set for Pipeline and Parallel Pipeline FFT Architectures",1. VLSI Signal Processing,8, 1994,253-265.

[15] M. Jaber and D. Massicotte "A New FFT Concept for Efficient VLSI Implementation: Part II - Parallel Pipelined Processing", accepted at DSP'2009,July 2009.

Paper II: M. Jaber and D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing", *International Conference on Digital Signal Processing* (DSP), Santorini, Greece, 5-7 July 2009.

A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing

Marwan A. Jaber and Daniel Massicotte

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations (LSSI)
{marwan.jaber, daniel.massicotte}@uqtr.ca

This paper presents a solution to the FFT's parallel multiprocessing problem, and involves novel concepts wherein the realization of parallel pipelines and multistage parallel pipelines are possible. The problem resides in defining the mathematical model of the so-called combination phase, in which the concept of representing the Discrete Fourier Transform (DFT) in terms of its partial DFTs should be well structured to obtain the right mathematical model. The resulting implementation in which *r* parallel processors operate simultaneously within a single instruction reduces the number of communications phases and the no-operation states (NOP) to their minimum values. The two papers, Butterfly Processing Element (BPE) and Parallel Pipelined Processing, provide a new FFT concept for efficient VLSI implementation.

1. Introduction

The computational of the fast Fourier transforms (FFTs) of size $N = r^S$ is the cornerstone of many super-computer applications. These include not only the common ones such as digital signal processing, speech recognition, image processing, communication systems (e.g. OFDM) and petroleum seismic analysis, but also other less obvious applications, such as in computational fluid dynamics, medical technology, multiple precision arithmetic and computational number theory. Computations worthy of a parallel computer generally fall into four categories:

1) One or a few very long 1-D FFTs;

2) Many small or moderate-sized 1-D FFTs;

3) One or a few large 2-D FFTs; or

4) One or a few large 3-D FFTs.

The most significant problem in spectral analysis resides in its data's parallel multiprocessing. This difficulty arises in finding a feasible algorithm that could meet the following objectives:

i) Build an algorithm that can be easily implemented on DSP cards using the newest technology;

ii) Choose r parallel processors able to execute a single instruction simultaneously;

iii) Maintain the number of NOP (no operations) at a minimum value;

iv) Maintain the communication load between r processors at a minimum value;

v) Maintain the computational load at its minimum value;

vi) Allow no pipeline break (or "pipeline stall"): the delay caused on a processor using pipelines when a transfer of control takes place (or is absent); and

vii) Ensure simplicity in VLSI design.

To meet the growing demand for high-speed processing, highly efficient parallel pipeline FFT processors have been deployed. In order to keep these processors busy, the simultaneous distribution of all data samples ($N = r^S$) is necessary. This problem, which involves serial word data coupled with limited I/O resources in FPGAs complicates the implementation of high performing parallel pipelined radix-r FFT processors [1]-[4]. From a mathematical point of view the representation of the DFT in terms of its partial DFTs has not been well structured to date. The problem resides in finding a mathematical model of the combination phase, in which the concept of butterfly computation proposed in [6],

should be well structured in order to obtain the right mathematical model. This problem can be addressed by discovering how $\mathbf{X} = \mathbf{T}_N x$ (a vector with n components) can be recovered from r vectors that are r times shorter.

This paper is organized as follows: Section 2 provides a detailed description of the proposed DFT factorization. Section 3 shows how various aspects of the parallel pipelined FFT and the multistage parallel pipelined FFT are proposed, while Section 4 describes the performance evaluation and Section 5 is devoted to the conclusion.

2. DFT Factorization

The DFT definition is shown as:

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \quad , \quad k \in [0, N-1] \tag{1}$$

where $x_{[n]}$ is the input sequence, $X_{[k]}$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ is called the twiddle factor in the butterfly structure, and $j^2 = -1$. Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex number sequences.

Equation (1) could be factorized as follow [4] and [5]:

To subdivide the axis k into 2 new axes p and q, we place $k = p + qN/r$ with $p = 0,1,\ldots, N/r^s - 1$, $s = 0,1,\ldots,\log_r N - 1$ and $q = 0,1,....,r-1$. Therefore, $\mathbf{X}_{(k)}$ is replaced using new indices p and q

$$\mathbf{X}_{(k)} = \mathbf{X}_{(p+qN/r)}, \tag{3}$$

with

$$\mathbf{X}_{(p+qN/r)} = w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+qN/r)} + w_N^{(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+qN/r)} + \cdots + w_N^{(r-1)(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+qN/r)} .$$

(4)

Equation (4) could be formulated in a matrix-vector equation, using the column vector operator, *col*(•), as follow

$$\mathbf{X} = \mathbf{T}_r \, \mathbf{W}_N \, col\left(\left. \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \right| q = 0,1,\cdots,r-1 \right), \tag{5}$$

for $p = 0,1,2,....,(N/r)-1$, $s = 0,1,\ldots,\log_r N - 1$ and $q = 0,1,2,....,r-1$, with

$\mathbf{X} = \left[X_{(p)}, X_{(p+N/r)}, X_{(p+2N/r)}, \ldots, X_{(p+(r-1)N/r)} \right]^T$, $\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p} \right)$ and

$$\mathbf{T}_r = \begin{pmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{pmatrix}.$$

From Eq. (5), we can write the well-known butterfly matrix $\mathbf{B}_r$, which can be expressed as:

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N \tag{6}$$

and for the particular case when $N=r$ input-output data elements therefore, the column vector in Eq. (5) becomes

$$col\left(\left. \sum_{n=0}^{r-1} x_{(rn+q)} w_r^{np} \right| q = 0,1,\cdots,r-1 \right) = [x_{(0)}, x_{(1)}, \cdots, x_{(r-1)}]^T, \tag{7}$$

and the BPE output is expressed as

$$\mathbf{X} = \mathbf{B}_r \mathbf{x}, \tag{8}$$

where $\mathbf{x} = [x_{(0)}, x_{(1)}, \cdots, x_{(r-1)}]^T$ and $\mathbf{X} = \left[X_{(0)},\ X_{(1)},\ \ldots,\ X_{(r-1)}\right]^T$ are, the BPE input and output vectors respectively.

3. Multistage architecture of parallel pipelined FFT

An FFT of length r^S is implemented in S stages where each stage performs a radix-r butterfly (Fig. 1). The switch blocks correspond to the data communication buses from the $(S-1)^{\text{th}}$ to S^{th} stages where $S = \log_r N$ and $s = 0,1,...,S-1$. Since r data paths are used, the pipelined BPE achieves a data rate S times the inter-module clock rate.

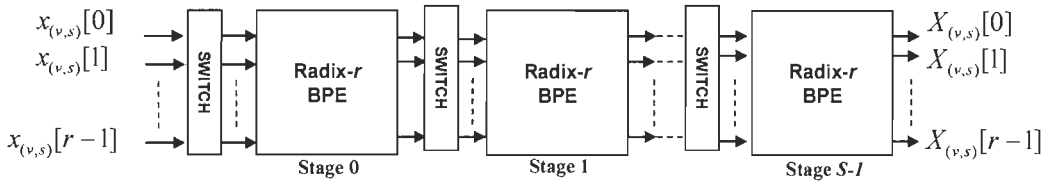


Figure 1: S Stages Radix-r Pipelined FFT

If $\mathbf{X}_{(k)}$ is the N^{th} order Fourier transform $\sum_{n=0}^{N-1} x_{(n)}\ w_N^{nk}$ then, $\mathbf{X}_{(0)}$, $\mathbf{X}_{(1)}$,... and $\mathbf{X}_{(r-1)}$ will be the N^{th}/r order Fourier transforms given respectively by the following expressions, $\sum_{n=0}^{(N/r)-1} x_{rn}\ w_{N/r}^{np}$, $\sum_{n=0}^{(N/r)-1} x_{rn+1}\ w_{N/r}^{np}$,... and $\sum_{n=0}^{(N/r)-1} x_{rn+(r-1)}\ w_{N/r}^{np}$.

In this section we refer to these smaller order DFT's as partial DFTs. As a result, Equation (5) can now be expressed as:

$$\mathbf{X} = \mathbf{T}_r \mathbf{W}_N col\left(\mathbf{X}_{(q)} \middle| q = 0,1,\cdots,r-1\right) \tag{9}$$

Finally Eq. (9) reveals that a DFT of size N can be decomposed into r shorter DFTs of size N/r which could be computed in parallel, with no data dependency among these r DFTs pieces, and then combined according to the same equation to obtain DFTs of size N.

By doing so, a complete iteration would be replaced by a single multiplication for each parallelized stage. According to Fig. 1, a radix-r pipelined FFT of size N will produce the first r outputs in S cycles where $S = \log_r N$. The number of cycles needed to complete a FFT is therefore $S + \left(N/r - 1\right)$ cycles.

Fig. 2 illustrates the parallel implementation of r radix-r pipelined FFTs of size N/r, which are interconnected with r radix-r butterflies in order to complete an FFT of size N.

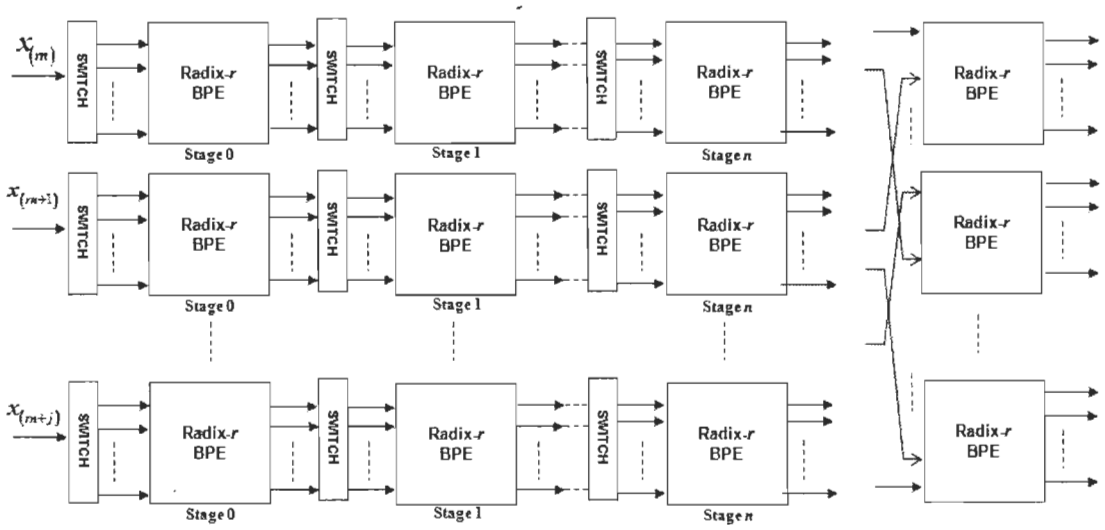


Figure 2: r-parallel pipelined radix-r for JBPE structure.

In DSP Layman's language, the factorization of an FFT can be interpreted as a dataflow diagram (or Signal Flow Graph) depicting the arithmetic operations and their dependencies. Thus by labeling the s^{th} stage's r outputs of each pipeline by $\mathrm{OUT}_{(j,p)}$, which are interconnected according to Eq. (5) to r butterfly processing elements (BPE) labeled as $\mathrm{BPE}_{(p,j)}$ in which $j = 0,1,\ldots,r-1$ and $p = 0,1,\ldots,r-1$. This interconnection is achieved by feeding the j^{th} output of the p^{th} pipeline to the p^{th} input of the j^{th} butterfly. For instance the output labeled zero of the second pipeline will be connected to the second input of the butterfly labeled zero.

Compared to Fig. 1, the first r outputs will be also provided by S cycles. Meanwhile a complete FFT will require:

$$S+\left(N/r^2-1\right), \tag{10}$$

and the gain G would be

$$G=\frac{S+r^{s-1}-1}{S+r^{s-2}-1}, \tag{11}$$

which for large N, could be approximated by:

$$G\approx\frac{r^{s-1}}{r^{s-2}}\approx r. \tag{12}$$

With the same reasoning as above, further DFT decomposition in terms of its partial DFTs yield to the multistage parallel pipelined FFT as shown in Figure 3 for the mixed radix-4 and radix-2 case. Finally the decomposition of the DFT in terms of its partial DFTs will lead to the FFT array structure which will be executed in S cycles as shown in Fig. P6.

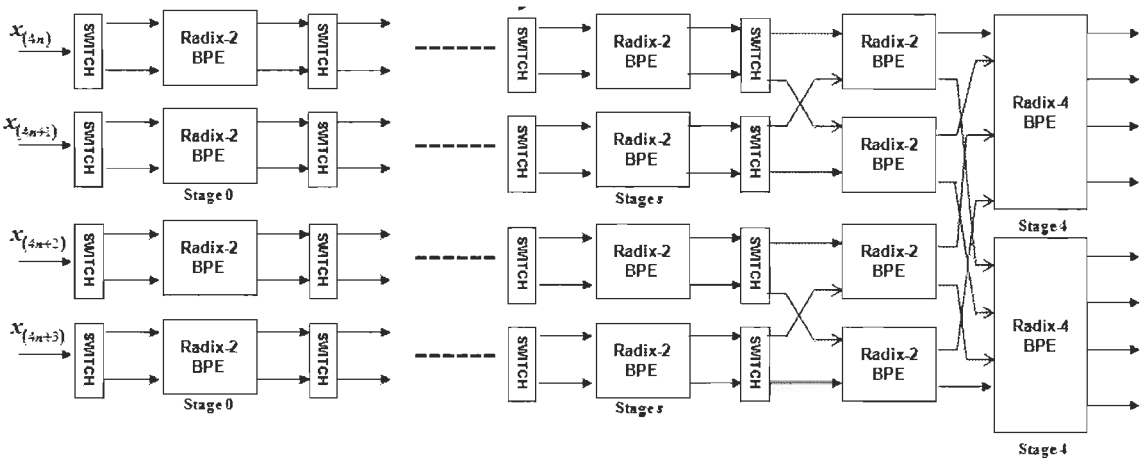


Figure 3: Two parallels radix-2 pipelined BPEs connected to two radix-4 BPEs.

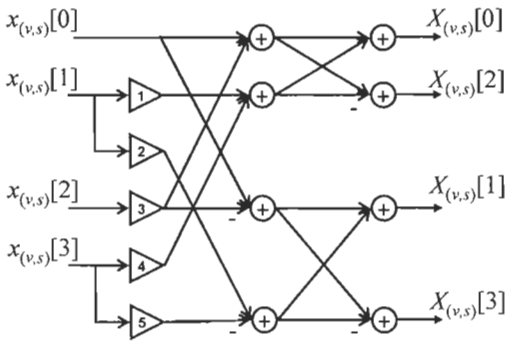
4. Performance Results

Our proposed parallel pipelined FFT will be targeting the OFDM application that mostly uses 64 complex points (N=64). This type of 2^6 FFT is implemented in 6 stages

(S=6) where each stage performs a radix-2 butterfly. The switch blocks correspond to the data communication buses from the $(S-1)^{th}$ to the s^{th} stages ($s = 0,1,2,...,5$) in which we will consider the switching concept proposed in [7]. Since two data paths are used, the BPE pipeline achieves a data rate of 6 times the inter-module clock rate. Our comparison study will be based on the computational time of the 2^6 FFT implemented on a pipelined radix-2 structure versus the proposed structures where we have assumed that all data is present at the input, in a pre-ordered manner. We quantified this comparison based on the BPE's critical path delay in order to compute the 2^6 FFT with the proposed structures for radices 4 and 8 as illustrated in Figures 4 and 5 of the cited reference [6]. By assuming that the time delay for real value addition (T_A) is 4 times less than a real value multiplication (T_M); Table 2-4 summarizes the critical path delay based on the delay time T_M of each BPE.

Table 1: Critical path delay for the conventional BPE and the proposed BPE for Radix-2, 4, 8 and 16 in order to obtain the first r outputs.

Radices	Critical Path Delay [$\times T_M$]		Speed Gain
	Conventional	Proposed FFT	
Radix-2	4.75	4.75	1
Radix-4	5.25	5.00	1.05
Radix-8	10.25	5.25	1.95
Radix-16	18.0	6.0	3.0

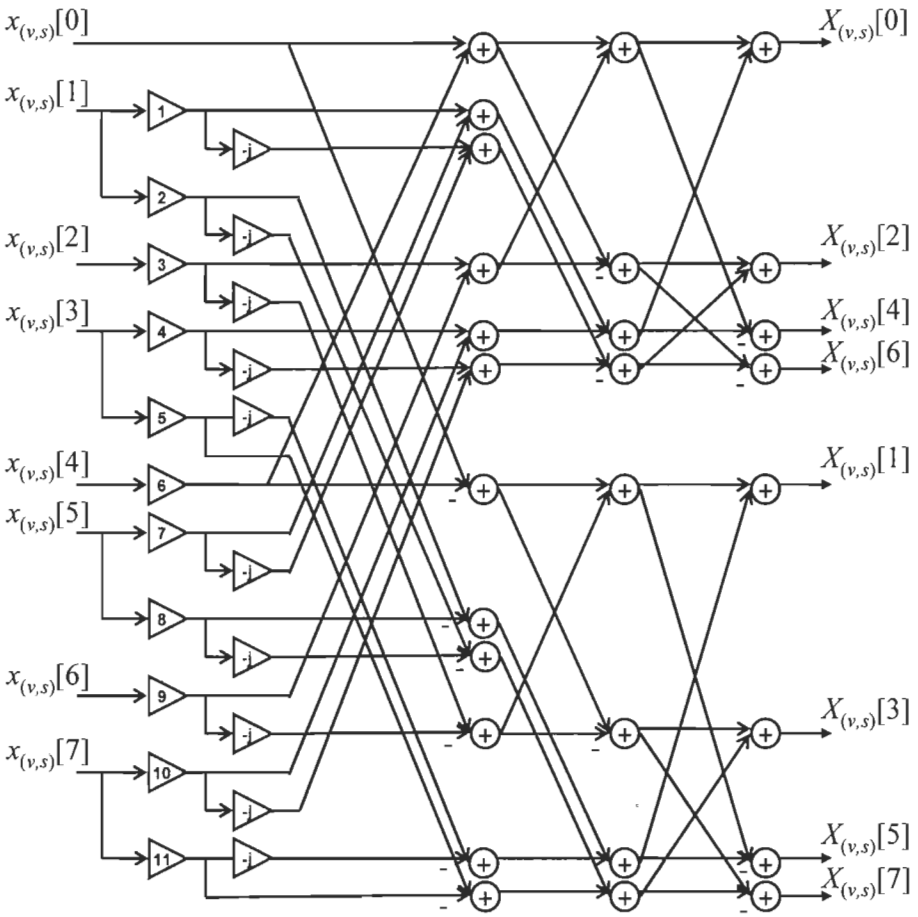


$$M_1 = w_N^{\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)}}, \quad M_2 = w_N^{\left\lfloor N/4+\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N},$$

$$M_3 = w_N^{\left\lfloor 2\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N}, \quad M_4 = w_N^{\left\lfloor 3\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N},$$

$$M_5 = w_N^{\left\lfloor N/4+3\left\lfloor (v/4^{(S-s)}) \right\rfloor 4^{(S-s)} \right\rfloor_N}$$

Figure 4: SFG of the proposed radix-4 BPE and the value of the multipliers M_i with i=1,2,…,5.



$$M_1 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)}}_N, M_2 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)} + \frac{N}{8}}_N, M_3 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 2\times 8^{(S-s)}}_N, M_4 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)}}_N$$

$$M_5 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)} + \frac{N}{8}}_N, M_6 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 4\times 8^{(S-s)}}_N, M_7 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)}}_N, M_8 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)} + \frac{N}{8}}_N$$

$$M_9 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 6\times 8^{(S-s)}}_N, M_{10} = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)}}_N, M_{11} = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)} + \frac{N}{8}}_N$$

Figure 5: Fig. 5 SFG of the proposed radix-8 BPE and the value of the multipliers M_i with i=1,2,...,11.

Tables 1 and 2 compares the critical path delay and the resources needed for the respective BPE, in terms of:

i) Number of real multiplications and additions,

ii) Number of elementary unit as full adders (FA) needed to implement both fixed-point arithmetic operators in VLSI.

Table 2: Number of real multiplications and additions for BPEs in term of FA (Mult. on 16-bit and adder on 32-bit).

Butterflies	Conventional			Proposed FFT		
	Mult	Add	FA	Mult	Add	FA
Radix-2	3	9	1056	3	9	1056
Radix 4	9	31	3296	15	41	5152
Radix 8	27	93	9888	33	111	12000

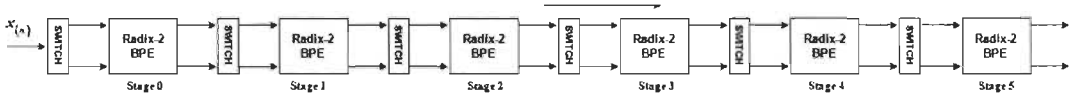


Figure P1: Pipelined radix-2 structure.

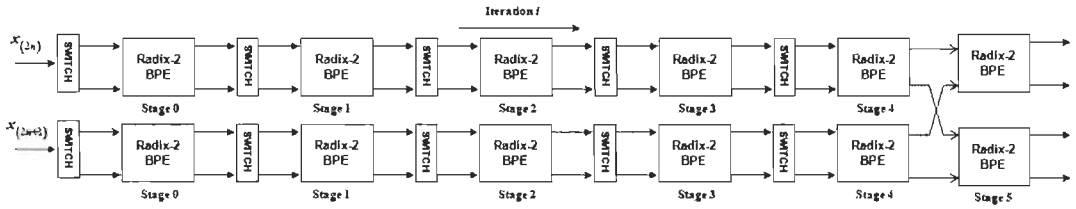


Figure P2: Two-parallel pipelined radix-2 structure.

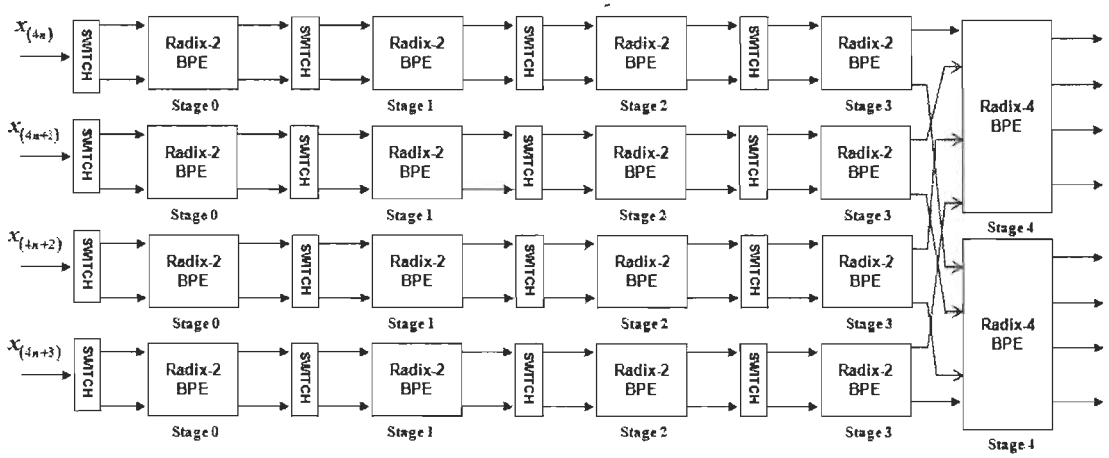


Figure P3: Four-parallel pipelined radix-2 structure.

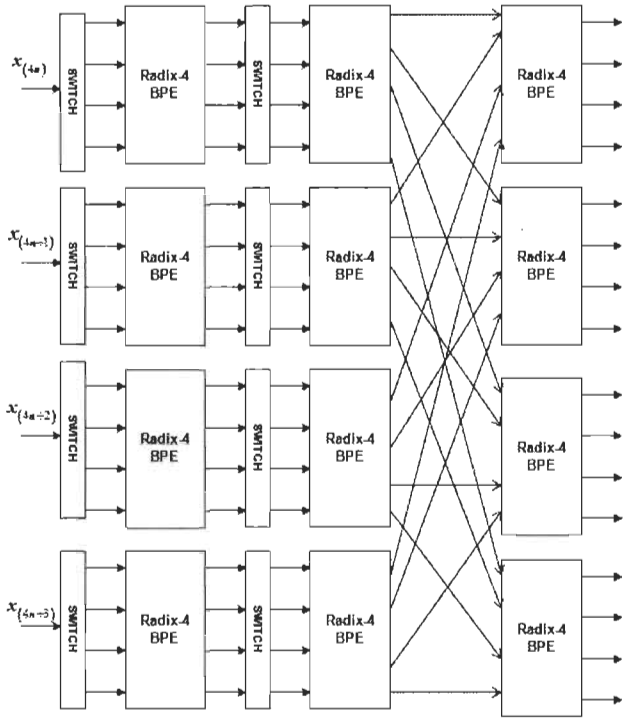


Figure P4: Four-parallel pipelined radix-4 structure.

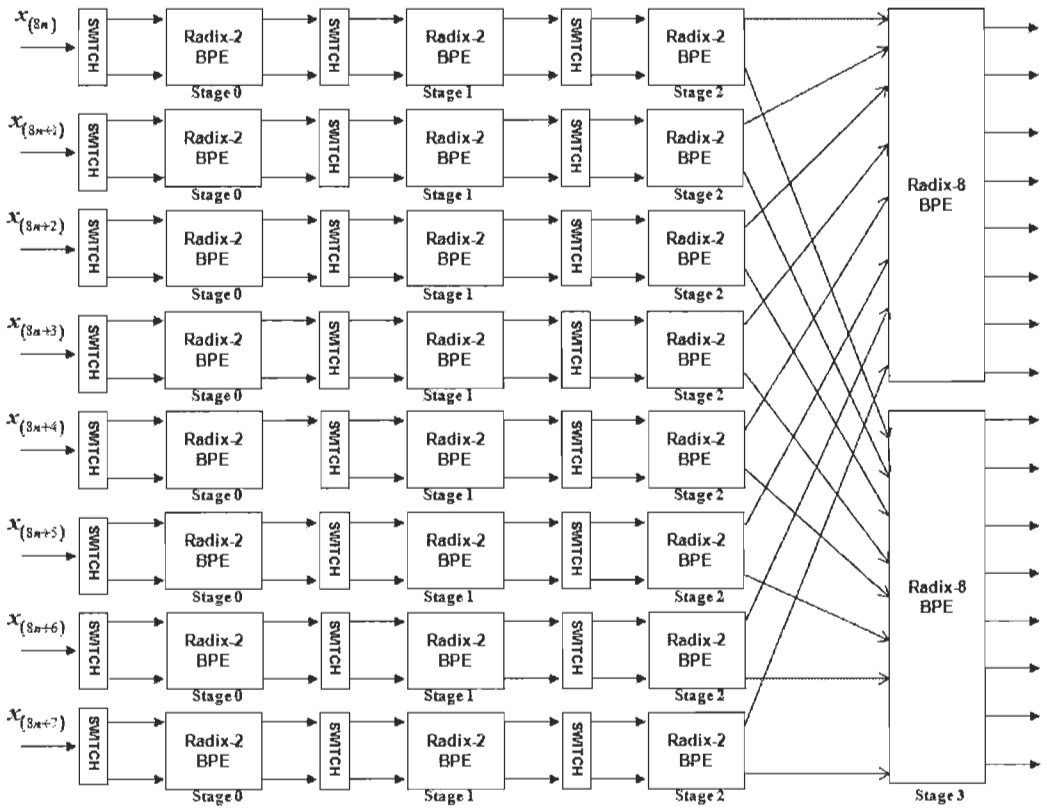


Figure P5: Eight-parallel pipelined radix-2 structure.

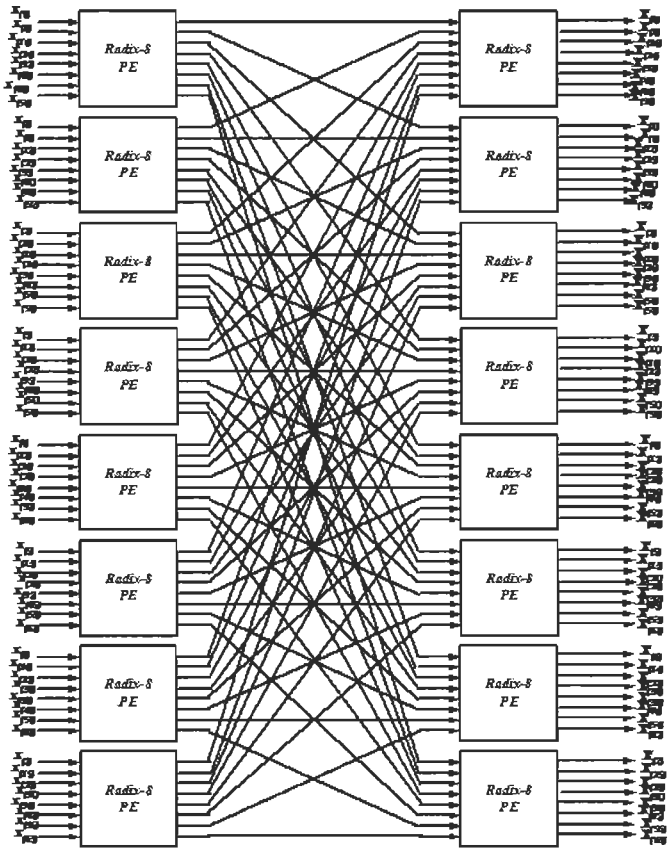


Figure P6: Eight-parallel pipelined radix-8 structure.

Based on the partial DFT concept presented in the previous section; Fig. P1 to P6 show the proposed multistage parallel pipelined FFT architectures. Table 3 compares all the proposed structures for N=64 (N=2^6) which is currently used in OFDM wireless applications such as WIMAX and LTE that uses up to N = 2048. The computation time to execute the total 64 points FFT is based on Table 1 and the area in terms of FA is based on Table 2. Table 3 shows the area and computation times for various parallel pipelined FFT structures, where according to this table, the use of parallel resources result in higher gains due to the fact that our proposed butterfly proposed in [6] will have one multiplier in its critical path. In order to achieve the same fixed-point arithmetic precision compared to the conventional butterfly that uses more than one multiplier in its critical data path; this will

result in reduction in bit terms of the data's word length. By doing so, our proposed models reach a closed area in term of FA as the conventional one, yet with better performance.

Table 3: Area and computation time of the proposed multistage parallel pipelined FFT architectures as shown in Fig. P1-P6.

Fig.	Radix-r used	Nb of BPE (Area)	Area (FA)	Increasing Area	Nb of iterations	Time ($\times T_M$)	Speed Gain
P1	radix-2	6 radix-2	6336	1.00	6	176	1.00
P2	2 parallels radix-2	12 radix-2	12672	2.00	5	100	1.76
P3	4 parallels radix-2	16 radix-2 2 radix-4	27200	4.29	4	57	3.08
P4	4 parallels radix-4	12 radix-4	61824	9.76	2	29	6.11
P5	8 parallels radix-2	24 radix-2 2 radix-8	49344	7.79	3	34	5.24
P6	8 parallels radix-8	16 radix-8	192000	30.3	1	10.5	16.7

As stated earlier, further DFT decomposition in terms of its partial DFTs will lead to the FFT array structure, which will be executed in S cycles as shown in Figure P6. Knowing that the first iteration of the DIT FFT process the coefficient multipliers are equal to 1 therefore, 64 complex DFT points will be executed in two cycles. If the proposed radix-8 butterflies shown in Figure 5 are used, 88 complex multipliers will be required for such implementation. Compared to the array structure of the conventional radix-2 FFT, 64 complex points DFT will be executed in 6 cycles and will require the implementation of 160 complex multipliers. For the same amount of complex data the conventional radix-8 will require 72 complex multipliers and will be executed in two clock cycles, but the conventional radix-8 butterfly clock cycle is two times slower than our proposed butterfly.

5. Conclusion

High performance parallel computing is essential for solving very large and complex scientific and engineering problems within a reasonable amount of computation time. These two mains tasks must be carried out in order to deliver a proper parallel computing solution to a specific problem, and they involve choosing the appropriate

parallel VLSI implementation. In this article we proposed a solution to the FFT's parallel multiprocessing problem, wherein the mathematical model described the global philosophy and the detailed strategy, and its resolution method was presented in chronological order. We have clearly shown that our proposed butterfly processing element structure in [5] is an effective solution for higher radix pipelined FFT implementation. This objective was achieved by reducing the complexity of the critical path compared to the conventional radix-r that uses the adder tree simplification. On the other hand, we clearly showed that the number of stages S in a pipelined architecture could be reduced through implementation our parallel method which could boost the FFT's execution time. Future work will consist of implementing our proposals on FPGA and ASIC by using the switching concept proposed in [7].

Acknowledgment

The authors would like to thank the financial support from the Natural Sciences and Engineering Research Council of Canada.

References

[1] M. Torkelson, "A New Approach to Pipeline FFT Processor", Proc. IPPS, p. 766 (1996)

[2] E. Carr, L. G. Cuthbert, "A Radix 4 Delay Commutator For Fast Fourier Transform Processor Implementation", IEEE 1. Solid-State Circuits, SC-19(5), Oct. 1984, pp. 702-709.

[3] R. Storn. "Radix-2 FFT Pipeline Architecture With Reduced NoiseTo- Signal Ratio", lEE Proc. Vis. Image Signal Process., vol. 141, no. 2, Apr. 1994, pp. 81-86.

[4] M Jaber and D. Massicotte "The Radix-r One Stage FFT Kernel Computation" Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP'08), April, Las Vegas Nevada USA, 2008.

[5] M Jaber "Parallel Multiprocessing for the Fast Fourier transform with Pipeline Architecture" US Patent No.6, 792, 441.

[6] M. Jaber and D. Massicotte "A New FFT Concept for Efficient VLSI Implementation: Part I - Butterfly Processing Element", accepted at DSP'2009, July 2009.

[7] V. Szwarc et al., "A Chip Set for Pipeline and Parallel Pipeline FFT Architectures", 1.VLSI Signal Processing, 8, 1994, pp. 253-265.

Paper III: M. Jaber, D. Massicotte, and Y. Achouri, "A Higher Radix FFT FPGA Implementation Suitable for OFDM Systems", *IEEE International Conference on Electronics, Circuits, and Systems* (ICECS), Beirut Lebanon, December 2011.

A Higher Radix FFT FPGA Implementation Suitable for OFDM Systems

Marwan A. Jaber, Daniel Massicotte and Youssef Achouri

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations
{marwan.jaber, daniel.massicotte, achouri}@uqtr.ca

Abstract – This article describes a new approach for higher radix butterflies suitable for pipeline implementation. Based on the butterfly computation introduced by Cooley-Tukey [1], we will introduce a novel approach for the Discrete Fourier Transform (DFT) factorization, by redefining the butterfly computation, which is more suitable for efficient VLSI implementation. The proposed factorization lead us to present a new concept of a radix-*r* Fast Fourier Transform (FFT), in which the radix-*r* butterfly computation concept was formulated as composite engines to implement each of the butterfly computations. This concept enables the radix *r* butterfly-processing element (BPE) to be designed by maintaining only one complex multiplier in the butterfly critical path for any given *r* [2]. Algorithmic description and performance of low complexity FFT methods are considered in this paper where the speed and accuracy evaluations of the proposed method in fixed point are also elaborated.

1. Introduction

The Discrete Fourier Transform (DFT) is a fundamental digital signal-processing algorithm used in many applications such as the orthogonal frequency division multiplexing wireless communication (OFDM), wherein the Fast Fourier Transform (FFT) is a major key operator [3], [4]. Most of the digital signal processing is treated in floating point in order to avoid the accuracy problem that is associated with a high cost in

implementation. However, in order to satisfy the cost constraints, these algorithms must be converted into fixed point algorithm where the computing accuracy losses must be contained in order to ensure algorithm integrity and applications performance. We can find a lot of structures to complete a given task, but finding the best structure is not a trivial problem. This paper thus proposes a new FFT algorithm able to increase the computation speed, preserving the accuracy with an affordable cost of implementation.

The definition of the DFT is represented by the following equation

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} , \quad k \in [0, N-1] \tag{1}$$

where $x_{[n]}$ is the input sequence, $\mathbf{X}_{[k]}$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ called the twiddle factor in butterfly structure, and $j^2 = -1$. Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex number sequences.

From Eq. (1), it can be seen that the DFT computational complexity increases according to the square of the transform length, and thus becomes expensive for large N. Some algorithms are used for efficient DFT computation, that are collectively known as Fast Fourier Transform (FFT) such as Cooley-Tukey algorithm [1], Split-Radix Algorithm, Winograd Fourier Transform Algorithm (WFTA) and others, such as the Common Factor Algorithms. The overall arithmetic operations involved in the computation of N-point FFT decrease with increasing r, however the complexity of the butterfly processing element (BPE) increases in term of complex arithmetic computation, parallel inputs, connectivity, number of phases and critical path delay in the butterfly. The higher radix butterfly involves a non-trivial VLSI implementation problem (i.e. increasing the butterfly's critical path delay), which explains why the majority of FFT VLSI implementations are based on

radix-2 or 4, due to their low butterfly complexity. The advantage of using a higher radix is that the number of multiplications and the number of stages to execute FFTs decreases [3]. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation, and thus reducing the number of stages becomes beneficial if communication is expensive, as is the case in most hardware implementations. Fewer attempts to reduce the computational load have failed, due to the added multipliers in the butterfly's critical path for higher radices [8], [9].

The most significant contribution in our proposition is that our proposed BPE structure maintains low arithmetic operations within its critical path (one complex multiplier and certain adders). Consequently, we propose a solution for higher radices BPE with low butterfly complexity in terms of complex multipliers in the butterfly critical path which is the key component in FFT implementation. By doing so, the butterfly VLSI implementation for higher radices would be feasible since it maintains approximately the same complexity of the radices 2 and 4 butterflies.

The paper is organized as follows; Section 2 the radix-r DFT factorization-demystified while Section 3 provides a detailed to description of the proposed FFT and the modified radix-*r* FFT methods. Section 4 provides a performance evaluation of the proposed butterfly structure while Section 5 reports the conclusions.

2. The Radix-r DFT Factorization

Eq. (1) can be expressed in compact form as [2], [5] and [6]:

$$\mathbf{X}_{(p)} = \mathbf{T}_r \mathbf{W}_N col\left(\sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \middle| q = 0,1,\cdots,r-1 \right), \tag{2}$$

For $k = 0,1,\cdots,N-1$, $p = 0,1,....,(N/r)-1$ and $q = 0,1,....,r-1$, with

$$\mathbf{X}_{(p)} = \left[X_{(p)},\ X_{(p+N/r)}, X_{(p+2N/r)},\ \ldots,\ X_{(p+(r-1)N/r)} \right]^T, \tag{3}$$

$$\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p} \right), \tag{4}$$

and

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix}. \tag{5}$$

A recursive application of Eq. (2) can convert the DFT computation of length $N = r^S$ into S steps in order to compute r^S DFTs of length r, where in each step N/r words have to be processed and the whole process is known as the FFT algorithm.

The twiddle factor matrix $\mathbf{W}_N$ is a diagonal matrix which is defined by $\mathbf{W}_N = diag\left(1, w_N^p, w_N^{2p}, \ldots, w_N^{(r-1)p}\right)$ with $p = 0,1,\ldots,r^S - 1$, $s = 0,1,\ldots,S$ where $S = \log_r N - 1$ and $\mathbf{T}_r$ is the well-known *DFT matrix* within the butterfly structure.

Since the higher radix automatically reduces the communication load, the only remaining problem was reducing the computational complexity in the butterfly structure. The best-known technique for reducing the computational load is factoring the DFT matrix $\mathbf{T}_r$.

Fig. 1 shows the signal flow graph (SFG) for the conventional radix-8 butterfly (named FFT Conv.), where the computational reduction is achieved by incorporating the trivial multiplications j or $-j$ into the summation by switching the real and imaginary parts of the data. Factoring the DFT matrix is the best-known method for computation reduction.

As we move to higher radices, the implementation complexity of such butterfly increases and the amount of non-trivial multiplications increases. In the radix-8 algorithm case, the butterfly complexity is significantly higher, and non-trivial multiplications are shown in Fig. 1 [3].

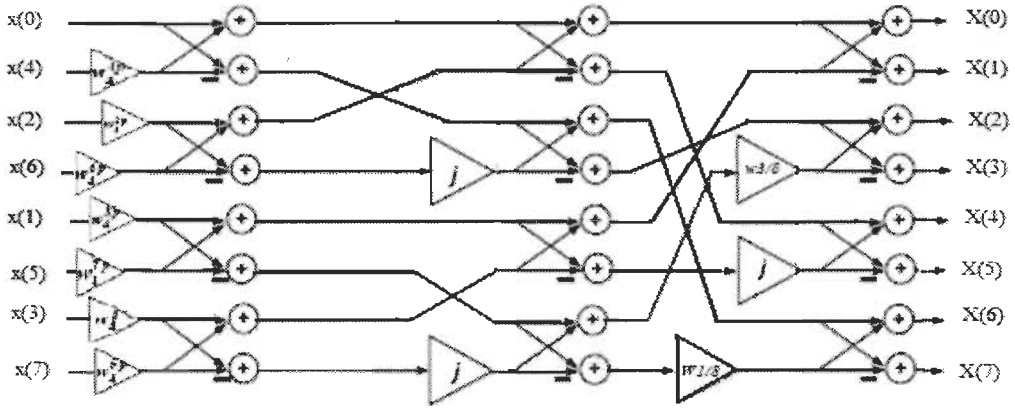


Figure 1: SFG of the radix-8 DIT butterflies [3] where the highlighted red portion represents the butterfly critical path used in FFT conventional.

3. Proposed FFT Method

It has been shown that the DFT matrix simplification method did not provide a complete solution for the FFT problem due to the increasing complexity of the butterflies for higher radices [3]. The problem's solution resides in the structure of the DFT matrix and the twiddle factor matrix. Thus, if we pay attention to the elements of the adder tree matrix $\boldsymbol{T}_r$ and to the elements of the twiddle factor matrix $\boldsymbol{T}_r$, we notice that both of them contain twiddle factors. So, by controlling the variation of the twiddle factor during the calculation of a complete FFT, we can incorporate the twiddle factors and the DFT matrix matrices into a single stage of calculation.

By defining $\left[\mathbf{T}_r\right]_{l,m}$ as the l^{th} and m^{th} element of the matrix $\mathbf{T}_r$, we can rewrite equation (5) as:

$$\left[\mathbf{T}_r\right]_{l,m} = w_N^{\left[\!\left[(lmN/r)\right]\!\right]_N}, \tag{6}$$

with l=0,1,…,r-1, m=0,1,…,r-1 and $\left[\!\left[x\right]\!\right]_N$ represents the operation x modulo N and By defining $\mathbf{W}_{N\,(r,v,s)}$ the set of the twiddle factor matrix (Eq. 4) as:

$$\left[\mathbf{W}_N\right]_{l,m\,(r,v,s)} = \begin{cases} w_N^{\left[\!\left[\lfloor v/r^s \rfloor\, l\, r^s\right]\!\right]_N} & for\ l = m \\ 0 & elsewhere \end{cases}, \tag{7}$$

where l=0,1,…,$r-1$, m=0,1,…,$r-1$, v = 0,1,…,N/$r-1$ and $\lfloor x \rfloor$ represents the integer part operator of x. Therefore, the proposed modified radix-r DIF butterfly computation $\boldsymbol{B}_r$ for the l^{th} output is expressed as:

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[\!\left[lmN/r + \lfloor v/r^s \rfloor\, l\, r^s\right]\!\right]_N} \tag{8}$$

With the same reasoning as above, the l^{th} output of the radix-r DIT FFT operation can be derived as:

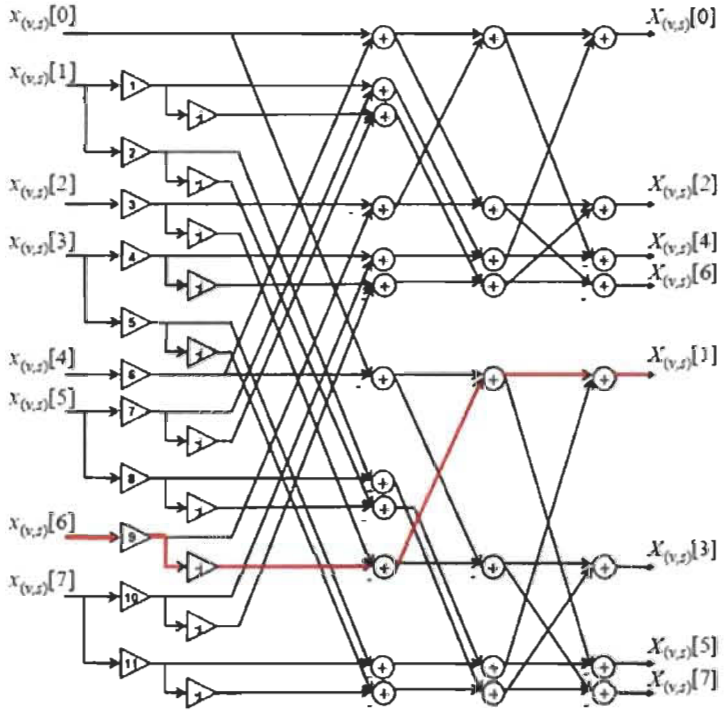
$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[\!\left[lmN/r + \lfloor v/r^{s-s} \rfloor\, m r^{s-s}\right]\!\right]_N} \tag{9}$$

Equations (8) and (9) yield to the BPE JFFT structures that maintain one complex multiplier in their critical data path [2] as shown in Fig. 2.

4. Performance Evaluation

FFTs are the most powerful algorithms that are used in communication systems such as OFDM. Their implementation is very attractive in fixed point due to the reduction in cost compared to the floating point implementation. One of the most powerful FFT implementation is the pipelined FFT which is highly implemented in the communication systems such as the Single-path Delay Feedback structure (SDF Fig. 3a), the Single-path Delay Commutator (SDC Fig. 3b) and the Multi-path Delay Commutator (MDC Fig. 3c)

from which the MDC structure will be the target in our performance evaluation. Our performance study will be conducted in two parts: the quantization effect and FPGA implementation.



$$M_1 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)}}_N, M_2 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 8^{(S-s)} + \frac{N}{8}}_N, M_3 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 2\times 8^{(S-s)}}_N,$$

$$M_4 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)}}_N, M_5 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 3\times 8^{(S-s)} + \frac{N}{8}}_N, M_6 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 4\times 8^{(S-s)}}_N,$$

$$M_7 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)}}_N, M_8 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 5\times 8^{(S-s)} + \frac{N}{8}}_N, M_9 = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 6\times 8^{(S-s)}}_N,$$

$$M_{10} = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)}}_N, M_{11} = w^{\left\lfloor \frac{v}{8^{(S-s)}} \right\rfloor 7\times 8^{(S-s)} + \frac{N}{8}}_N$$

Figure 2: SFG of the proposed radix-8 BPE and the value of the multipliers M_i are defined in [2] where the highlighted red portion represents the butterfly critical path (named BPE JFFT).

A) Fixed-Point Accuracy

In our fixed point comparative study, we used the native OFDM principle as shown in Fig. 4 [9]. The Signal to Quantization Noise Ratio (SQNR) is used to measure the FFT

accuracy at the receiver output versus the transmitted signal and in order to make a fair comparison between the two methods we implemented the FFT address generator proposed in [6] that could exclude the only trivial multiplication by one (i.e. w_N^0) from the FFT process. Furthermore, in this comparison we will be elaborating two scenarios where the input/output data word-length is fixed to 16-bit and the twiddle factor word-length will vary (Fig 5). In the second scenario, the twiddle factor word-length is fixed to 8-bit and the input/output data word-length varies between 8 and 24 bits (Fig 6).

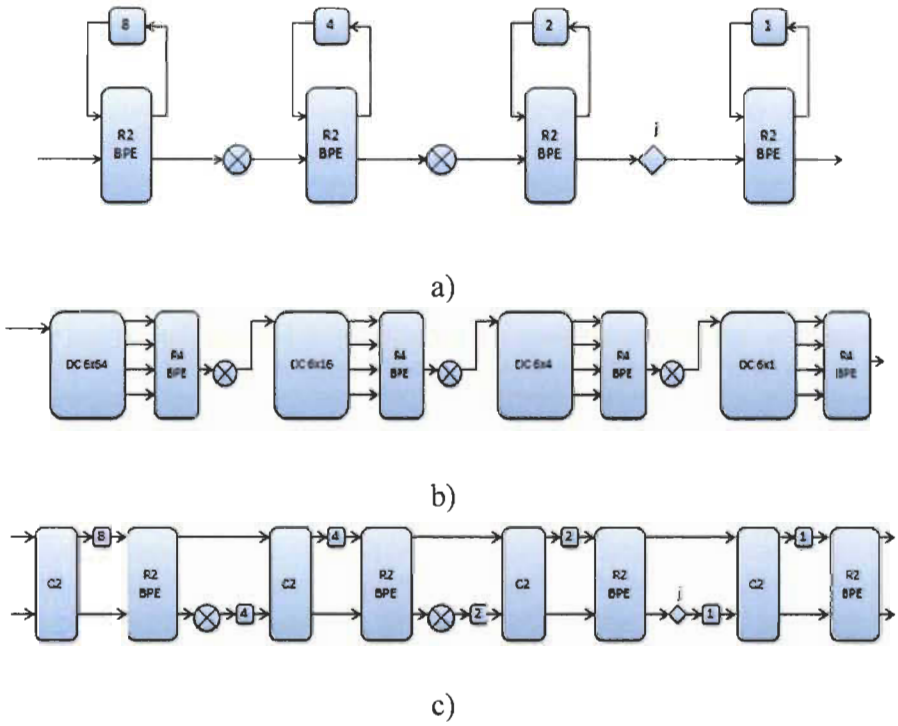


Figure 3: Pipelined FFT structures: a) Radix-2 SDF structure (R2SDF) for N = 16, b) Radix-4 SDC structure (R4SDC), and c) Radix-2 MDC structure (R2MDC).

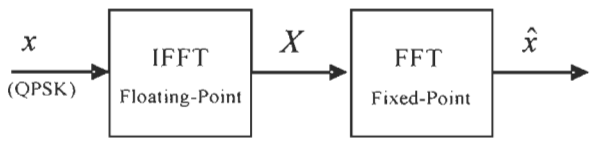


Figure 4: Fixed-point simulation with QPSK signals.

Fig. 5 reveals that the proposed method (called JFFT) has significant gains of 3 dB in the first scenario for a twiddle's factor word-length of 7-bit which is translated into reduction by 0.5-bit to obtain the same SQNR. In the second scenario, a gain of 3 dB is observed where the input/output's data word-length is greater than 16-bit as shown in Fig. 6. In these Figures, FFT2, 8 and 16 correspond to radix 2, 8 and 16, respectively.

B) FPGA Implementation

For the FPGA implementation, we have targeted in our comparison the Spartan-3 family, Virtex-E, Virtex-4 and Virtex-5 families and since the complex multiplication is a major concern in the FFT process; our performance study will be based on 4 different structures of the complex multiplier illustrated in Fig. 7 for the different cases labeled as: case 0 to case 3.

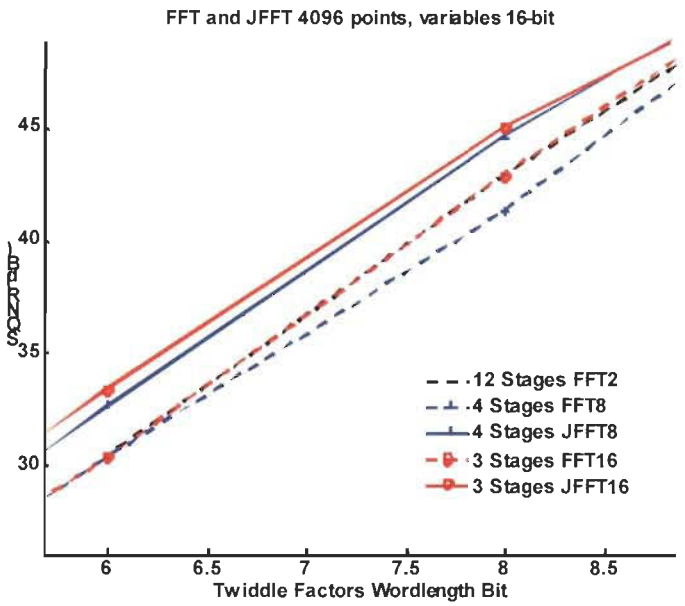


Figure 5: Scenario 1: SQNR comparison for coefficients' word-length 6 to 9-bit and input/output data's word-length is fixed to 16-bit where N=4096.

Our comparison will be based on the cost in term of MS/s/Slice as used in [7] where the comparative study was conducted on 1024 FFT. The FFT size was extended to 4096

points which is a multiple of 8 in order to elaborate the comparison between the proposed structures versus the R2^2SDF pipelined FFT structure. The cited structures [7] are based on the pipelined complex multiplier where it has been demonstrated that the R2^2SDF pipelined FFT structure performs better than the R4SDC structure.

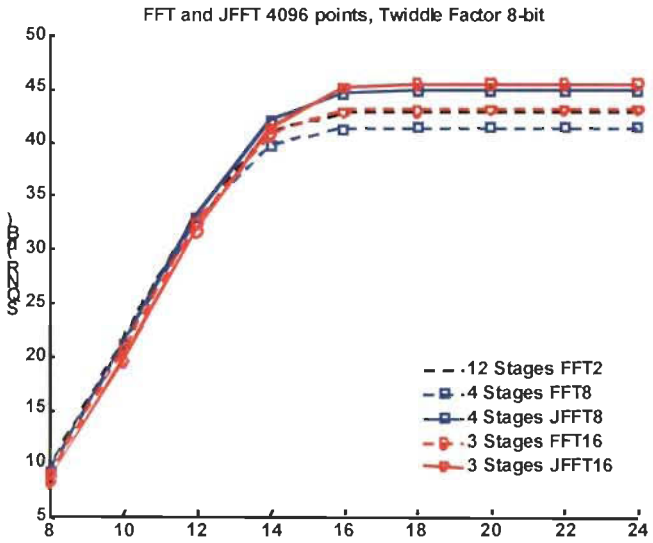


Figure 6: Scenario 2: SQNR comparison for input/output data's word-length 8 to 24-bit and coefficients' word-length is fixed to 8-bit where N=4096.

Table 1 reveals that the cost is maximized for the case 1 where our proposed radix-8 butterfly on the MDC structure reveals a gain of 270%, 64% and 132% on the Spartan-3, Virtex-E and Virtex-4 respectively, compared to the cited method in [7].

Table 2 shows that our proposed radix-8 JFFT on the MDC structure for case 1 reduces the latency time by a factor of 10 on Spartan-3, 8 times faster on Virtex-4 and 6 times faster on Virtex-E compared to the method cited in [7].

Table 3 reveals that the proposed radix-8 structure on the MDC structure performs the best in comparison to the conventional FFT butterfly and the cited method [7] due to the fact that the proposed method maximizes its use of the complex multipliers on Spartan-3 and Virtex-E.

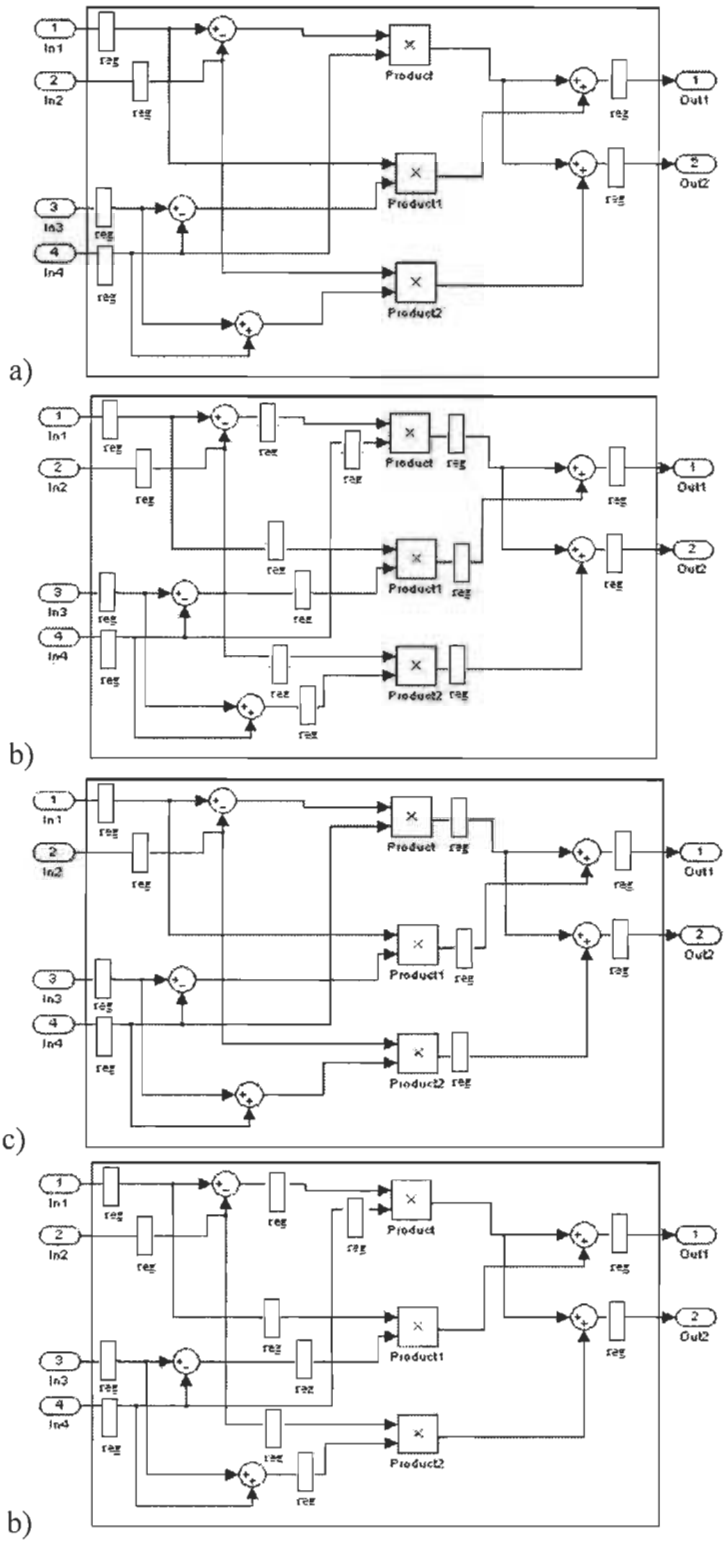


Figure 7: Complex Multiplier case studied: a) Case 0, b) Case 1, c) Case 2, and d) Case 3.

In conclusion and based on the results, we conclude that our proposed JFFT for case 1 outperform the results obtained versus the conventional BPE (Fig. 2). Table 4 shows the resource's comparison in term of the total number of embedded multipliers involved in this comparative study. Our proposed JFFT structure uses more embedded multipliers than the other cited methods but less than the conventional butterfly for Virtex 4 and 5.

Table 1: Cost Evaluation in MS/s/Slice for an FFT of size 4096

Method		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [07]		0,0283	0,0158	0,0870	-
R4SDC [07]		0,0234	0,0111	0,0596	-
Radix-8 Conv.	Cas 0	0,0098	0,0022	0,0219	0,0336
	Cas 1	0,0486	0,0229	0,2014	0,1817
	Cas 2	0,0408	0,0152	0,1583	0,1513
	Cas 3	0,0398	0,0139	0,1493	0,1483
Radix-8 JFFT	Cas 0	0,0064	0,0026	0,0272	0,0301
	Cas 1	0,1048	0,0259	0,2018	0,2447
	Cas 2	0,0859	0,0171	0,1466	0,1779
	Cas 3	0,1001	0,0200	0,1510	0,1832

Table 2: Latency time for an FFT of size 4096 (in μs)

Method		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [07]		86,2	86,4	34,8	-
R4SDC [07]		66,3	87,1	37,4	-
Radix-8 Conv.	Cas 0	101,6	154,7	57,0	45,3
	Cas 1	17,3	15,7	4,9	4,1
	Cas 2	23,7	23,0	7,5	6,2
	Cas 3	26,6	25,2	9,0	7,5
Radix-8 JFFT	Cas 0	98,1	104,8	38,9	30,5
	Cas 1	9,5	15,6	4,9	4,1
	Cas 2	12,8	22,9	7,5	6,2
	Cas 3	10,8	20,2	7,2	5,9

Table 3: Computational Time of the FFT execution for a size 4096 (in µs)

Method		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [07]		43,0	43,1	17,4	-
R4SDC [07]		33,1	43,5	18,7	-
Radix-8 Conv.	Cas 0	46,4	70,7	26,0	20,7
	Cas 1	7,8	7,1	2,2	1,8
	Cas 2	10,8	10,5	3,4	2,8
	Cas 3	12,1	11,5	4,1	3,4
Radix-8 JFFT	Cas 0	44,9	47,9	17,8	13,9
	Cas 1	4,3	7,1	2,2	1,8
	Cas 2	5,8	10,4	3,4	2,8
	Cas 3	4,9	9,2	3,3	2,7

Table 4: Number of embedded multiplier used for devices and methods

	Embedded multipliers		DSP48	
Method	Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [07]	18,0	18,0	19,0	-
R4SDC [07]	18,0	18,0	19,0	-
Radix-8 Conv.	108,0	108,0	108,0	108,0
Radix-8 JFFT	132,0	132,0	88,0	88,0

5. Conclusion

This article has presented an efficient implementation method for the FFT algorithm, where various issues concerning FFT implementation processors were discussed, placing the emphasis on the butterfly processing elements (BPE) implementation. It can be argued that the higher radix FFT algorithms are advantageous for the hardware implementation, due to the reduced quantity of complex multiplications and memory access rate requirements. In this paper, we showed that the implementation of a radix-*r* PE for the FFT is feasible. In radix-8, we have shown an improvement of critical path delay for BPE by a factor of 2 and 3, respectively, and by maintaining higher SQNR compared to the radix-2.

References

[1] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. Comput., 19, pp. 297-301, April 1965.

[2] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009.

[3] T. Widhe, "Efficient Implementation of FFT Processing Elements" Linköping studies in Science and Technology, Thesis No. 619, Linköping University, Sweden, June 1997.

[4] G. Klang, "A Study of OFDM for Cellular Radio Systems, M.Sc. Thesis, Linköping University, Sweden 1994.

[5] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6751643, 2004.

[6] M Jaber, "Address Generator for the Fast Fourier Transform Processor", US-6,993,547B2 and European PCT/US01/07602.

[7] B. Zhou, Y. Peng et D. Hwang, "Pipeline FFT Architectures, Optimized for FPGAs", Journal of Reconfigurable Computing, Hindawi, 2009, doi:10.1155/2009/219140.

[8] W. Chang and Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms", IEEE Transactions on Signal Processing, vol. 56, no. 10, October 2008, pp. 4673-4682.

Chapter 3 A Novel Approach for the FFT Data Reordering

Paper IV: M. Jaber and D. Massicotte, "A Novel Approach for the FFT Data Reordering", *International Symposium on Circuits and Systems* (ISCAS), Paris, May 2010.

Résumé Du Chapitre 3

La transformée rapide de Fourier (TRF) avait et a toujours été une méthode de base dans les applications du traitement du signal et surtout dans l'analyse fréquentielle des signaux. Le calcul de la TRF nécessite pour chaque étape un schéma d'indexage pour contrôler d'une manière appropriée les données d'entrée / sortie et les coefficients multiplicateurs. La plupart des schémas d'indexages connues sont basées sur l'inversion des bits qui sont des techniques basés sur une table de consultation où on devra stocker ces indexes dans une mémoire supplémentaire. Ce chapitre décrit une nouvelle technique pour réordonner les données en se basant sur trois compteurs simples. Ces trois compteurs calculent les adresses de données avec les adresses de ses coefficients multiplicateurs correspondants qui doivent alimenter l'entrée du papillon. Un autre générateur d'adresse permet de stocker les données de sortie du papillon dans leur emplacement approprié de la mémoire. La méthode proposée peut réduire énormément l'accès mémoire des coefficients multiplicateurs en regroupant les données avec ses coefficients multiplicateurs correspondants tout en réduisant le temps d'exécution de la TRF. En agissant ainsi, toutes les multiplications triviales par ± 1 ou $\pm j$ ont pu être exclus du processus et en ajoutant à cela les accès à ces coefficients multiplicateurs ont été également réduits.

Paper IV: M. Jaber and D. Massicotte, "A Novel Approach for the FFT Data Reordering", *International Symposium on Circuits and Systems* (ISCAS), Paris, May 2010.

A Novel Approach for the FFT Data Reordering

Marwan A. Jaber and Daniel Massicotte

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations
{marwan.jaber, daniel.massicotte}@uqtr.ca

Abstract – **The Fast Fourier Transform (FFT) had and always been a key role in signal processing applications which has been useful for the frequency domain analysis of the signals. The FFT computation requires for each stage an indexing scheme to address the input/output data and the coefficient multipliers in an appropriate way. Most of the indexing schemes to address the input/output data are based on the bit reversing techniques which will be boosted by a look up table that will need extra storage memory. This paper describes a novel technique in reordering the data based on three simple counters that computes the addresses of the butterfly's input data with the addresses of its corresponding coefficient multipliers and store the butterfly's output data into their proper memory location. FFTs are considered to be in place algorithms (or in situ) which transform a data structure by using a constant amount of memory storage. We showed that our proposed method reduces the memory usage by eliminating the look-up table traditionally used in the computation of the bit reversal indexes.**

1. Introduction

The FFT algorithm is especially memory access and storage intensive, where the communication burden of an algorithm is a measure of the amount of data (written and read) that must be moved to or from the computing elements. Therefore, FFTs are typically used to input large amounts of data, perform mathematical transformation on that data, and

then output the resulting data all at very high rates. In a real time system, data flow must be understood and controlled in order to achieve the high performance of a future wireless communication system based on orthogonal frequency division multiplexing wireless communication (OFDM) wherein the FFT is a major key operator [1]. Since the butterfly computation consists of a simple multiplication of the input data with an appropriate coefficient multiplier, the idea arises to have simple address generators (AG) that compute such address sequences from a small parameter set that describes the address pattern. Because the butterfly's CPU should only be used to compute mathematical transformation, it is preferable for dataflow to be controlled by an independent device; if not, the system may incur performance degradation. Such peripheral devices, which can control data transfers between an I/O (Input/Output) subsystem and a memory subsystem in the same manner that a processor can control such transfers, reduce CP interrupt latencies and leave precious DSP cycles free for other tasks leading to increased performance [2].

Thus, given that dataflow control is a major concern in the FFT process, inadequate AG burden the memory interface with additional load and slow down computations [3].

One "rediscovery" of the FFT, that of Danielson and Lanczos in 1942, provides one of the clearest derivations of algorithms [4] and [5]. Danielson and Lanczos showed that a discrete Fourier transform could be written as the sum of two discrete Fourier transforms, each of length $N/2$. In the mid-1960s, J.W. Cooley and J.W. Tukey proposed their first algorithm, known as the decimation-in-time (DIT) or Cooley-Tukey FFT algorithm, which first rearranges the input elements into bit-reverse order, then builds up the N-data output transform in $\log_2 N$ iterations; in other words, the radix-2 DIT algorithm first computes the transform of even-indexed and odd-indexed data, then combines these two results to

produce the Fourier transform of the entire data sequence [6]. Since that time, several techniques have been proposed for ordering and accessing the data at each stage of the FFT; the best known reordering technique is the bit-reversal in which the data at index n are written in binary digits that are permuted in reversed order.

In this paper we propose an innovative AG structure that is faster than the bit-reversal technique most frequently proposed and compared with the most recent published bit-reversal techniques.

The paper is organized as follows: Section 2 gives a brief outline of the bit-reversal technique; Section 3 describes the proposed method, Section 4 provides the performance results of that method and Section 5 contains the conclusion.

2. The Bit Reversing Techniques

Many FFT users prefers the natural order outputs of the computed FFT and that is why they concentrated their efforts in reducing the computational time impact in the bit reversal stage which is the first stage of the DIT process known as the bit reversal data shuffling technique. The DIT FFT was attractive in fixed point implementation since Chang and Nguyen showed in [7] that DIT process executed in fixed-point arithmetic is more accurate than the decimation-in-frequency (DIF). Furthermore, it is highly recommended to reorder the intermediate stage of the FFT algorithm in order to facilitate the operation on consecutive data element which is required for many hardware architectures. To these ends, a number of alternative implementation has been proposed where one of which has greatly simplified this problem by adopting the out-of-place algorithm where the output array is distinct from the input one. Therefore, this section will be devoted in reviewing the existing current architecture of the bit reversing technique

which is needed at the first stage of the DIT process where a number of bit reversal algorithms have been published in recent years [8], [9] and [10]. The vector calculation method proposed by Pei & Chang for bit reversing technique as the "fastest known technique" is also used in our comparative study [10].

The operation count of the proposed algorithm in [8] by excluding the index calculations for each stage is

$$\begin{aligned} &N-2 \text{ integer additions,} \\ &2(N-2) \text{ integer increments,} \\ &(\log_2 N)-1 \text{ multiplications by 2,} \\ &(\log_2 N)-1 \text{ divisions by 2,} \end{aligned} \tag{1}$$

plus two more divisions by $N/2$ and $N/4$. In Eq. (1), multiplications and divisions can be efficiently implemented using bit-shift operations. On the top of that, this algorithm will require a storage table of $N/2$ index numbers [8].

On the other hand the proposed method in [10] showed a significant improvement in the operation count which will require N shifts, N additions and an adjusting index that will require the use of $O(N)$ memories.

3 The Proposed Method

The proposed method is based on the radix–r DFT factorization proposed in [11]-[15]. The definition of the DFT is represented by the following equation

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \quad , \quad k \in [0, N-1] \tag{2}$$

which could be factorized as follow:

$$\mathbf{X}_{(k)} = \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_N^{rnk} + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_N^{(rn+1)k} + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_N^{(rn+2)k} + \cdots + \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_N^{(rn+(r-1))k} , \tag{3}$$

where after simplification equation (3) could be expressed as

$$\mathbf{X}_{(k)} = w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{nk} + w_N^k \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{nk} + w_N^{2k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{nk} + \cdots + w_N^{(r-1)k} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{nk}, \tag{4}$$

for $k = 0,1,\cdots,N-1$.

Finally, from Eq. (4), could be formulated in a matrix-vector notation as

$$\begin{bmatrix} X_{(p)} & X_{(p+N/r)} & X_{(p+2N/r)} & \cdots & \cdots & X_{(p+(r-1)N/r)} \end{bmatrix}^T = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix} \begin{bmatrix} w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{np} \\ w_N^p \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{np} \\ w_N^{2p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{np} \\ \vdots \\ w_N^{(r-1)p} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{np} \end{bmatrix} \tag{5}$$

which could be expressed in a compact form as:

$$\mathbf{X} = \mathbf{T}_r \mathbf{W}_N col\left(\left. \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \right| q = 0,1,\cdots,r-1 \right), \tag{6}$$

for $p = 0,1,....,(N/r)-1$ and $q = 0,1,....,r-1$ with

$$\mathbf{X} = \left[X_{(p)}, X_{(p+N/r)}, X_{(p+2N/r)}, \ldots, X_{(p+(r-1)N/r)} \right]^T, \tag{7}$$

$$\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p} \right), \tag{8}$$

and

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix}. \tag{9}$$

In DSP Layman language, the factorization of an FFT can be interpreted as dataflow diagram (or Signal Flow Graph), which depicts the arithmetic operations and their dependencies. When the equation (6) is read from left to right we will obtain the decimation in frequency algorithm, meanwhile if the dataflow diagram is read from right to left we will obtain the decimation in time algorithm

By examining equation (5) or (6), we could easily conclude that first we have to compute the transform of r sets of data of size N/r and then combines these r results to produce the Fourier transform of the whole data sequence. For the all stages ordered input ordered output (OIOO) FFT algorithm, there is N/r vector sets of size r that has to be processed in each stage therefore, r specific data should be fed to the butterfly's input which are provided by the DIT Reading Address Generators (RAG). For this version of the FFT, the m^{th} butterfly's input $x_{(m)}$ of the p^{th} word at the s^{th} stage (i^{th} iteration) is fed by the OIOO DIT RAG $r_{m(p,s)}$ [3] as

$$r_{m(p,s)} = m\left(\frac{N}{r^{(s+1)}}\right) + [\![p]\!]_{r^{n-s}} + \left\lfloor \frac{p}{r^{(n-s)}} \right\rfloor r^{(n+1-s)} \quad , \tag{10}$$

for $p = 0,1,....,(N/r)-1$ and m=0,1,…,r-1 where $[\![x]\!]_N$ represents the operation x modulo N and $\lfloor x \rfloor$ represents the integer part operator of x.

It is clearly evident that for the first iteration/stage (i.e. s = 0) equation (4) will be equal to

$$r_{m(p,s)} = m\left(\frac{N}{r}\right) + p \tag{11}$$

On the other hand equation (7) reveals that the transformed outputs are in a bit reverse order which means that the transformed outputs of each set of the input data are

obtained at a stride N/r, which means that in order to obtain ordered output, the l^{th} processed butterfly's output $X_{(l,k,s)}$ for the p^{th} word at the s^{th} stage should be stored into the memory address location given by:

$$A_{(l,p)} = l\,(N/r) + p\,, \tag{12}$$

for $l=q= 0, 1,\ldots, r-1$, and $k=p= 0, 1, \ldots, (N/r)-1$.

Equation (11) represents the bit reversal stage in the DIT process meanwhile equation (12) which is identical to equation (11) represents the bit reversal stage that is required at the end of the DIF process. In an OIOO radix-2 DIT FFT process, the two butterfly's input will be labeled by $m = 0$ for the first input and by $m = 1$ for the second input. As a result and according to equation (11), the first butterfly's input will be driven by the data located at the memory address (Fig. 1)

$$r_{0(p,s)} = 0\times\left(\frac{N}{r}\right)+p = p \tag{13}$$

and the second butterfly's input will be driven by the data located at the memory address

$$r_{1(p,s)} = 1\times\left(\frac{N}{r}\right)+p = \left(\frac{N}{r}\right)+p \tag{14}$$

```
#define WordLimit(N)\
  WordLimit=(N>>1);

#define Reading(p, WordLmit)\
   register UINT32 Sum;\
   Sum= p+WordLimit;\
  SrcMemory [p].Real=In0.Real;\
  SrcMemory [p].Imaginary=In0.Imaginary;\
   SrcMemory [Sum].Real=In1.Real;\
  SrcMemory [Sum].Imaginary=In1.Imaginary;\
```

Figure 1: C Function of the proposed method.

The DIT process that only requires:

$$(r-1)\,N/r \text{ additions,} \tag{15}$$

which could be used in an OIOO for each stage/iteration of the FFT process where In0, In1 are the butterflies' inputs and SrcMemory refers the Source memory from which the data is picked up. By replacing SrcMemory in this Figure by DestMemory and In0, In1 by Out0, Out1 we will obtain the proposed method for the DIF process where DestMemory is the sink memory in which the output data is stored and Out0, Out1 are the butterfly's outputs.

4 Performance Results

Increment operator is a unary operator that operates on single operand; but + is a binary operator which needs at least 2 operands to execute. So, logically unary operators are always faster than binary operators and the main reason for this is that an increment-instruction that should be supported by the hardware is often a lot faster than an addition-instruction. The addition-instruction that requires the access to two operands makes it slower than the increment instruction which is not always true on many of the RISC systems where the time spent on these will be the same (one clock cycle).

By also assuming that the addition and bit-shift operations take only one clock cycle, as a result and based on these assumptions we will consider the operation count of the proposed method would be:

$$(r-1)N/r \quad \text{cycles.} \tag{16}$$

By ignoring the divisions by $N/2$ and $N/4$ the operation count of one bit reversing technique of the radix-2 FFT as described in [9] become:

$$\left(3N-6\right)+2\left(\log_2(N)-1\right) \text{ Cycles,} \tag{17}$$

where we considered that multiplication by 2 and division by 2 can be materialized using a one cycle bit-shift operation.

The operation count of the vector calculation method proposed in [10] is N shifts and N integer additions, we have

$$2N \text{ cycles} \tag{18}$$

The performance gain for the radix-2 FFT in terms of operation cycles between the proposed structure and the reference methods for the bit reversing technique is

$$G = \frac{6N + 4\log_2 N - 16}{N}, \tag{19}$$

for Rius & de Porrata-Doria [8]

and

$$G = 4. \tag{20}$$

for the vector calculation method proposed by Pei & Chang [10]. Fig. 2 shows the performance gain for $N = 2^S$.

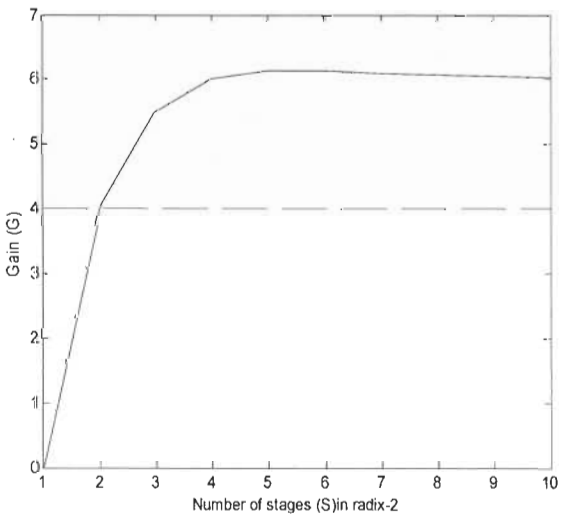


Figure 2: Performance gain of proposed method compared to Rius & de Porrata-Doria [8], in solid line, and Pei & Chang [10], in dash line, for radix-2.

The FFTW benchmark [16] is an FFT bench platform assembled by Matteo Frigo and Steven G. Johnson at MIT (Massachusetts Institute of Technology) that compares the

performance of different complex FFT implementations (40 FFT implementations) in terms of speed and accuracy where the performance in this benchmark is computed on a single processor environment even though this benchmark will be run on multi-processors systems [17] and [18]. This bench platform is internationally recognized where the complex FFT performance is plotted in terms of "mflops" (Efficiency axis) and the FFT size N which is a scaled version of the speed defined by:

$$mflops = \left(5N \log_2 N\right)/t, \tag{21}$$

where t is the computational time in μs to execute the N-point FFT [64]. The FFTW benchmark of Fig. 3 shows the significant improvement on the FFT execution time by implementing our proposed method on a conventional radix-4 butterfly.

Adding to that, our proposed method does not need extra memory needed for index storage and by doing so; we have reduced the memory usage at least by $N/2$ which is used as storage table of $N/2$ index numbers as shown in Table 1.

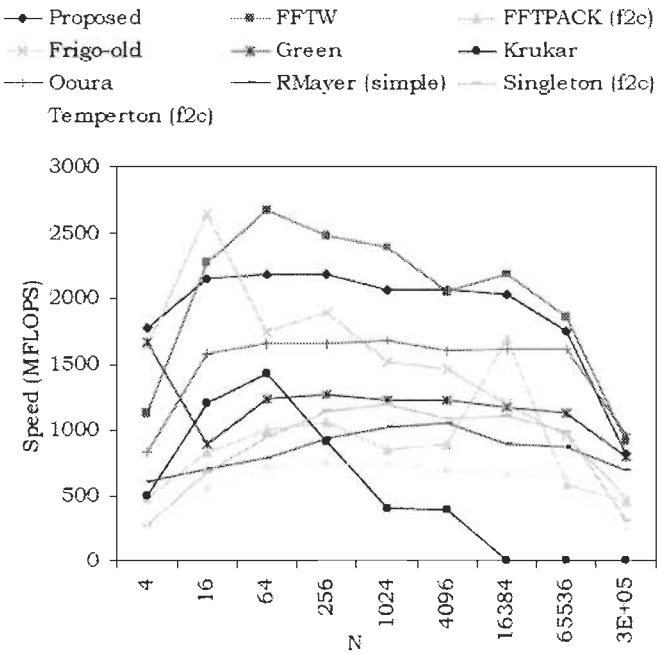


Figure 3: FFTW benchmark results of the proposed method (JFFT) compared to reference methods for radix-4.

Table 1: Memory for table index number

Methods	Memory
Rius [59]	$N/2$
Rius+Yong [59]	$N/8$
Prado [60]	S even, $\sqrt{N}$ S odd, $\sqrt{N/2}$
Pei [61]	N
Proposed	0

5 Final Remarks and Conclusion

As we have seen that the FFT algorithm is especially memory access and storage intensive where most studious task in this process is the data flow control. As we know that the butterfly main function is to multiply the input data with its corresponding coefficient multipliers in order to compute the transform. As a result, an efficient tool that could control efficiently the data flow would increase the overall system's performance. The FFT Address Generator presented in [3], has detailed an embodied address generator for use with a variety of FFT algorithms in which the address generator is typically used to compute the addresses (locations in memory) where input data, output data and twiddle coefficients will be stored and retrieved from memory. In addition to its structure's simplicity, the speed of the address generators is greatly increased as shown in Figure 3 where in [19], we proposed a fast method to detect specific frequencies in monitored signal that is useful for OFDM communication systems.

The present paper has presented a novel approach for the FFT data reordering algorithms that boosted the FFT execution. Compared to recent bit reversing techniques proposed in [8] and [10], we presented, respectively, speedups of 6 and 4 in terms of

operation cycles. The implementation of this method would be highly recommended on low power DSP processors and this is achieved by reducing the memory usage by $N/2$ which is used as storage table of $N/2$ index numbers. By doing so the size and the power consumption of such processor will be reduced which are highly desirable for portable devices?

Acknowledgment

The authors wish to thank the Natural Sciences and Engineering Research Council of Canada and Jabertech Canada Inc. for their financial and technical support.

References

[1] C.L. Hung, S.S. Long, and M.T. Shiue, "A Low Power and Variable-Length FFT Processor Design for Flexible MIMO OFDM Systems", Int. Symposium on Circuit and Systems, Taiwan, May 2009, pp. 705-708.

[2] A. Huang, J. Shen, "The Intrinsic Bandwidth Requirements of Ordinary Programs", Proceedings of the 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, ASPLOS-VII, October, 1996.

[3] M Jaber "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 B2 and European patent application Serial no: PCT/US01/07602.

[4] S. W. White S. Dhawan, " Power 2: Next Generation of the RISC System/6000 family", Copyright IBM Corporation, 1994.

[5] R. C. Agarwal, F. G. Gustavson, and M. Zubair, IBM J. Res. 38, 563, 1994.

[6] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematical Computer 19, pp. 297-301, April 1965.

[7] W.-H Chang and T.Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms", IEEE Trans. on Signal Processing, vol.56, no. 10, Oct. 2008, pp. 4673-4682.

[8] J. M. Rius and R. De Porrata-Doria "New FFT Bit-Reversal Algorithm", IEEE Transactions On Signal Processing, Vol. 43, No.4, April 1995, pp. 991-994.

[9] J. Prado "A New Fast Bit-Reversal Permutation Algorithm Based on Symmetry", IEEE Signal Processing Letters, Vol. 11, No.12, Dec. 2004, pp. 933-936.

[10] S. Pei, K. Chang "Efficient Bit and Digital Reversal Algorithm Using Vector Calculation" IEEE Transactions on Signal Processing, Vol. 55, No. 3, March 2007, pp. 1173-1175.

[11] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6751643, 2004.

[12] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009.

[13] M. Jaber "Parallel Multiprocessing for the Fast Fourier transform with Pipeline Architecture" US Patent No. 6, 792, 441

[14] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009.

[15] M Jaber and D. Massicotte "The Radix-r One Stage FFT Kernel Computation", IEEE Int. Conf. on Speech, and Signal Processing, Las Vegas Nevada, April 2008, pp. 3585-3588.

[16] M. Frigo and S.G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT", IEEE Int. Conf. on Speech, and Signal Processing, Seattle, 1998, pp. 1381-1384.

[17] M. Frigo and S.G. Johnson, "The Design and Implementation of FFTW3", Proceeding of IEEE, vol. 2, no. 2, Feb. 2005, pp. 216- 231.

[18] FFTW, http://www.fftw.org, (visited in 2009).

[19] M. Jaber and D. Massicotte, "Fast Method to Detect Specific Frequencies in Monitored Signal", accepted at International Symposium on Communications, Control and Signal Processing (ISCCSP 2010), Cyprus, March 2010.

Chapter 4 The JM-filter to Detect Specific Frequencies in Monitored Signal

Paper V: M. Jaber and D. Massicotte, "The Radix-r One Stage FFT Kernel Computation", *International Conference on Acoustic, Speech, and Signal Processing* (ICASSP), Las Vegas Nevada USA, April 2008.

Paper VI: M. Jaber and D. Massicotte, "Fast Method to Detect Specific Frequencies in Monitored Signal", *International Symposium on Communications, Control and Signal Processing* (ISCCSP), Cyprus, March 2010.

Paper VII: M. Jaber and D. Massicotte, The JM-filter to Detect Specific Frequencies in Monitored Signal, *to be submitted to a Journal after the end of the confidentiality.*

Résumé du Chapitre 4

Une des techniques les plus importantes dans l'analyse des caractéristiques d'un signal est l'extraction des informations utiles d'un signal donné surveillé. La surveillance des signaux est un domaine en expansion qui visent la détection des changements brusques pour une fréquence spéciale comme :

- dans la détection de panne dans les machines à roulement de billes
- la détection d'un ensemble présélectionné de fréquences tel que Radio Frequency Identification (RFID)
- la reconnaissance du double-ton multifréquence (DTMF)
- dans le système de communication sans fil orthogonal frequency division multiplex (OFDM) dans lequel la TRF est un opérateur clé principal, particulièrement pour la radio cognitive
- et un grand nombre de domaines non cités.

Ce chapitre introduit une méthode de calcul à base r qui permet de calculer une fréquence spécifique d'un signal, que nous nommons JM-filtre. Ce filtre permet une réduction d'opération arithmétique par un facteur tendant vers r soit le radice utilisé en comparaison avec le filtre de Goertzel.

Paper V: M. Jaber and D. Massicotte, "The Radix-r One Stage FFT Kernel Computation", *International Conference on Acoustic, Speech, and Signal Processing* (ICASSP), Las Vegas Nevada USA, April 2008.

The Radix-*r* One Stage FFT Kernel Computation

Marwan A. Jaber and Daniel Massicotte

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations, www.uqtr.ca/lssi
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
{marwan.jaber, daniel.massicotte}@uqtr.ca

Abstract – **The FFT process is an operation that could be performed through different stages. In each stage, the only operation that occurs is the butterfly computation in which the accessed data is multiplied by certain w^{α} then, added or subtracted and finally it will be stored or it will be held for further processing. In the next stage, the processed data is accessed, multiplied by certain w^{β} then, added or subtracted and finally it will be stored or it will be held for further processing till the final stage where the processed data is driven to the output. So, by finding an appropriate indexing or mapping schemes between the input data and the coefficient multipliers through the different stages will yield to a single stage of computation in which those different stages will collapse into a single stage of computation. Therefore, this paper will elaborate the state of the art of computing the FFT in a single stage of computation by proposing the radix-*r* one iteration FFT kernel computation.**

Index Terms– **Discrete Fourier transforms Frequency domain analysis and parallel processing.**

1. Introduction

The Discrete Fourier Transform (DFT) is a fundamental digital signal-processing algorithm used in many applications, including frequency analysis and frequency domain processing, such as speech compression, in wireless communication system based on Orthogonal frequency division multiplexing (OFDM) in which the FFT is an operator key

[4], meanwhile the frequency domain processing allows for the efficient computation of the convolution integral (for linear filtering) and of the correlation integral (for correlation analysis).

The definition of DFT is shown in equation (1), $x_{[n]}$ is the input sequence, $\mathbf{X}_{[k]}$ is the output sequence, N is the transform length and w_N is the N^{th} root of unity ($w_N = e^{-j2\pi/N}$). Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex sequences.

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \quad , \quad k \in [0, N-1] \tag{1}$$

DFT is the decomposition of a sampled signal in terms of sinusoidal (complex exponential) components, and because of its computational requirements, the DFT algorithm, which requires N^2 complex multiplication plus a smaller number of operations to complete a complex addition or subtraction, usually is not used for real time signal processing. Several efficient methods have been developed to compute the DFT, "Cooley and Tukey presented their approach showing a number of multiplications required to compute the DFT of a sequence may be considerably reduced to $N\log_2 N$ by using one of the fast Fourier transform (FFT) algorithms [1]". One of the bottlenecks in most applications, where high performance is required, is the FFT/IFFT processor.

In this paper, the structure of the one stage algorithm for the dedicated FFT will be elaborated. The main objective of this proposal is reduction in communication load, reduction in computation and particularly reduction in the number of multiplications. The advantage of appropriately breaking the DFT in terms of its partial DFTs is that the number of multiplications and the number of stages may be controlled. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation, and thus, reduction in the number of stages is beneficial. Minimizing the

computational complexity may be done at the algorithmic level of the design process, where the minimization of operations depends on the number representation (word length in bit) in the implementation.

Despite of the Cooley-Tukey's clear definition stating that the DFT is a combination of its partial DFTs, researchers used to express the DFT in terms of its partial DFTs as:

$$X_{[k]} = \sum_{n=0}^{(N/r)-1} x_{[rn]} w^{rnk} + \cdots + \sum_{n=0}^{(N/r)-1} x_{[rn+(r-1)]} w^{(rn+(r-1))k} \tag{2}$$

As a result the mathematical representation of the DFT into its partial DFTs is not well defined yet. The problem resides in finding the mathematical model of the combination phase, in which the concept of butterfly computation should be well structured in order to obtain the right mathematical model.

The paper is organized as follows; in Section 2 a butterfly operation is defined. Section 3 is devoted to describe in details the proposed FFT method and the modified radix-*r* FFT. The implementation aspects are given to Section 4, while Section 5 draws conclusions.

2. The Butterfly Processing Element

The basic operation of a radix-*r* BPE is the so-called butterfly in which *r* inputs are combined to give the *r* outputs via the operation [2]:

$$\mathbf{X} = \mathbf{B}_r \mathbf{x} \tag{3}$$

where $\mathbf{x} = \left[x_{[0]}, x_{[1]}, \ldots, x_{[r-1]}\right]^T$ is the BPE's input vector and $\mathbf{X} = \left[X_{[0]}, X_{[1]}, \ldots, X_{[r-1]}\right]^T$ is the BPE's output vector. $\mathbf{B}_r$ is the butterfly matrix, dim ($\mathbf{B}_r$) = $r \times r$, which can be expressed as

$$\mathbf{B}_r = \mathbf{W}_N^r \mathbf{T}_r \tag{4}$$

for the decimation in frequency (DIF) process, and

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N^r \tag{5}$$

for the decimation in time (DIT) process. In both cases the twiddle factor matrix $\mathbf{W}_N$, is a diagonal matrix defined by $\mathbf{W}_N = diag\left(1, W_N^p, W_N^{2p}, \ldots, W_N^{(r-1)p}\right)$ with $p = 0,1,\ldots,\log_r N - 1$ and $\mathbf{T}_r$ is the adder-tree matrix in the butterfly structure

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \cdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix} \tag{6}$$

where dim $(\mathbf{T}_r) = r \times r$.

If we pay attention to the elements of the adder-tree matrix $\mathbf{T}_r$ and to the elements of the twiddle matrix $\mathbf{W}_N$, we could notice that both of them contain twiddle factors. So, by controlling the variation of the twiddle factor during the calculation of a complete FFT, results in incorporating the twiddle factors and the adder matrix in a single-stage of calculation.

By defining $\left[\mathbf{T}_r\right]_{l,m}$ as the l^{th} line and the m^{th} column element of the matrix $\mathbf{T}_r$ therefore, equation (6) can be written as

$$\left[\mathbf{T}_r\right]_{l,m} = w_N^{\left[(lmN/r)\right]_N}, \tag{7}$$

and by defining $\mathbf{W}_{N(r,v,s)}$ the set of the twiddle factor matrix $\mathbf{W}_N$ as:

$$\mathbf{W}_{N(r,v,s)} = diag\left(w_{N(0,v,s)}, w_{N(1,v,s)}, \ldots, w_{N(r-1,v,s)}\right), \tag{8}$$

in which each element of the diagonal matrix for the DIF process is represented as:

$$[\mathbf{W}_N]_{l,m(r,v,s)} = \begin{cases} w_N^{\left[\left\lfloor v/r^s \right\rfloor l\, r^s\right]_N} & for\ l = m \\ 0 & elsewhere \end{cases}, \tag{9}$$

therefore, the modified radix-r butterfly computation $\boldsymbol{B}_r$ expressed as:

$$\mathbf{B}_r = \mathbf{W}_{N(r,v,s)}\mathbf{T}_r, \tag{10}$$

that could be simplified as

$$[\mathbf{B}_r]_{l,m(v,s)} = w_N^{\left[lmN/r + \left\lfloor v/r^s \right\rfloor l\, r^s\right]_N} \tag{11}$$

with $l = 0,1,\ldots,r-1$, $m = 0,1,\ldots,r-1$, $v = 0,1,\ldots,V-1$, $s = 0,1,\ldots,S-1$, r is the radix-r, V is the number of words (set of r inputs), $V = N/r$, S is the number of stages (or iteration), $S = \log_r N$, $[\![x]\!]_N$ represents the operation x modulo N and $\lfloor x \rfloor$ is defined as the integer part operator of x.

As a result, the operation of a radix-r for the DIF FFT is formulated by, the column vector:

$$\mathbf{X}_{(r,v,s)} = \mathbf{B}_{r\,DIF}\mathbf{x}, \tag{12}$$

where the l^{th} output is

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m] w_N^{\left[lmN/r + \left\lfloor k/r^s \right\rfloor l\, r^s\right]_N}. \tag{13}$$

With the same reasoning as above, the operation of a radix-r DIT FFT can be derived.

The conceptual key of the modified radix-r FFT butterfly is the formulation of the radix-r as composed butterflies with identical structures and a systematic means of accessing the corresponding multiplier coefficients. This enables the design of processing element (PE) which is referred as Butterfly PE (BPE) shown in Fig. 1 and in order to

maximize the data throughput; we can utilize r or $r-1$ complex multipliers in parallel to implement each of the radix-r butterfly computations.

Fig. 2 shows the radix-r BPE that is used to compute the overall butterfly's output where if implemented on a single processor environment; this would decrease the FFT time delay by a factor of $O(r)$. A second aspect of the modified radix $-r$ FFT butterfly, is that they are also useful in parallel multiprocessing environments (see (Fig. 2)). In essence, the precedence relations between the engines in the radix– r FFT are such that the execution of r engines in parallel is feasible during each FFT stage. If each engine is executed on the modified PE, it means that each of the r parallel processors would be always executing the same instruction simultaneously, which is very desirable for SIMD implementation on some of the latest DSP cards [43].

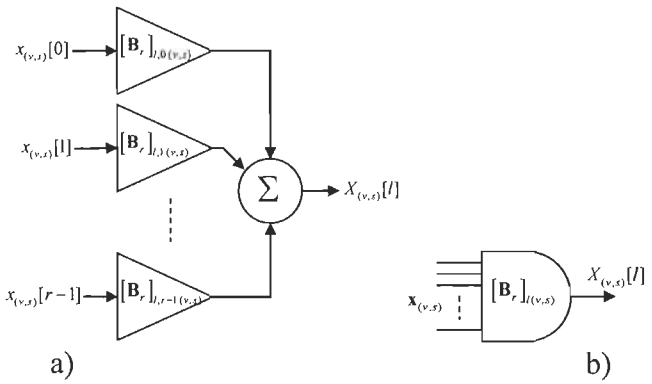


Figure 1: BPE for the FFT Radix-r (a) using Br in (11) and the symbol (b).

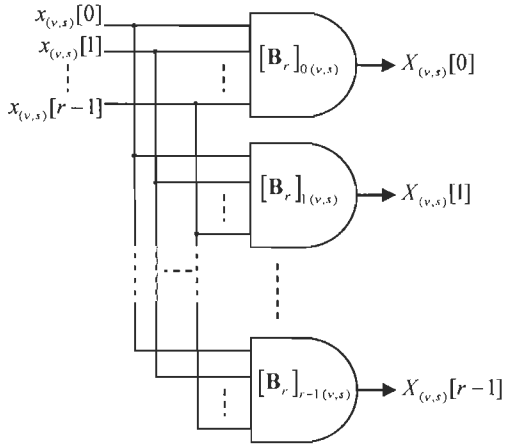


Figure 2: Maximize the data throughput using r BPEs in parallel [3].

3. DFT Factorization

For a given $r \times r$ square matrix $\mathbf{T}_r$ and for a given column vector $\boldsymbol{x}_{[n]}$ of size N, we define a special product expressed with the operator $\hat{*}_{(\alpha,\gamma,\beta)}$ (product of radix-α performed on γ column vector of size β) by the following operation, where the γ column vectors are subsets of $\boldsymbol{x}_{[n]}$ picked up at a stride α.

$$\mathbf{X}_{[k]} = \hat{*}_{(r,r,N/r)}\left(\mathbf{T}_r, \begin{bmatrix} x_{[rn]} \\ x_{[rn+1]} \\ \vdots \\ x_{[rn+(r-1)]} \end{bmatrix}\right) = \mathbf{T}_r \begin{bmatrix} x_{[rn]} \\ x_{[rn+1]} \\ \vdots \\ x_{[rn+(r-1)]} \end{bmatrix} \tag{14}$$

$$\mathbf{X}_{[k]} = \begin{bmatrix} T_{0,0} & T_{0,1} & \cdots & T_{0,r-1} \\ T_{1,0} & T_{1,1} & \cdots & T_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ T_{r-1,0} & T_{r-1,1} & \cdots & T_{r-1,r-1} \end{bmatrix} col\left(x_{[rn+j_0]}\right)_{j_0=0,1,\ldots,r-1} \tag{15}$$

$$\mathbf{X}_v[l] = \left[\sum_{j_0=0}^{r-1} [\mathbf{T}]_{l,j_0}\, x_{[rn+j_0]}\right] \tag{16}$$

for $v = 0,1,...,N/r-1$ and j_0=0,1,…,r-1; $\mathbf{X} = \left[X_{[0]}, X_{[1]},\ldots, X_{[r-1]}\right]^T$ is a column vector. This can be generalized to a r column vectors of length $\lambda\beta$ where λ is a power of r in which the l^{th} element $X_{[l]}$ of the v^{th} product $X_{(v)[l]}$ is labeled as

$$l_{(v)} = j_0\lambda\beta + v \tag{17}$$

for v=0,1,...,$\lambda\beta-1$. Special properties are shown in Appendix.

Based on our proposition in the previous section, Eq. (1) for the first factorization may be rewritten as

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{kn} = \hat{*}_{(r,r,N/r)} \left(\mathbf{T}_r, \begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{[rn]} w_N^{rnv_0} \\ \sum_{n=0}^{(N/r)-1} x_{[rn+1]} w_N^{(rn+1)v_0} \\ \vdots \\ \sum_{n=0}^{(N/r)-1} x_{[rn+(r-1)]} w_N^{(rn+(r-1))v_0} \end{bmatrix} \right) \tag{18}$$

for v_0=0,1,..., $(N/r) - 1$, and n=0,1,...,$N - 1$. Since

$$w_N^{rnk} = w_{N/r}^{nk} \tag{19}$$

then Eq. (18) becomes

$$\mathbf{X}_{[k]} = \hat{*}_{(r,r,N/r)} \left(\mathbf{T}_r, \begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{[rn]} \, w_{N/r}^{nv_0} \\ w_N^{v_0} \sum_{n=0}^{(N/r)-1} x_{[rn+1]} w_{N/r}^{nv_0} \\ \vdots \\ w_N^{(r-1)kv} \sum_{n=0}^{(N/r)-1} x_{[rn+r-1]} w_{N/r}^{nv_0} \end{bmatrix} \right) \tag{20}$$

which for simplicity may be expressed as

$$\mathbf{X}_{[k]} = \hat{*}_{(r,r,N/r)} \left(\mathbf{T}_r \left[\mathbf{W}_N^{j_0 v_1} \right], col \left(\sum_{n=0}^{(N/r)-1} x_{[rn+j_0]} w_{N/r}^{nv_0} \right) \right) \tag{21}$$

where for simplification in notation the column vector in (21) is set equal to:

$$\begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{[rn]} w_{N/r}^{nv_0} \\ w_N^{v_0} \sum_{n=0}^{(N/r)-1} x_{[rn+1]} w_{N/r}^{nv_0} \\ \vdots \\ w_N^{(r-1)v_0} \sum_{n=0}^{(N/r)-1} x_{[rn+r-1]} w_{N/r}^{nv_0} \end{bmatrix} = col \left(\sum_{n=0}^{(N/r)-1} x_{[rn+j_0]} w_{N/r}^{nv_0} \right) \tag{22}$$

for j_0=0,1,...,r – 1, v_0=0,1,...,(N/r) – 1 and $\left[\mathbf{W}_N^{j_0 v_0}\right] = diag\left(w_N^0, w_N^{v_0}, \cdots, w_N^{(r-1)v_0}\right)$.

For the second factorization, (22) is factored as follow:

$$X_{[k]} = \hat{*}_{\left(r,r,\frac{N}{r}\right)}\left(\mathbf{T}_r\left[\mathbf{W}_N^{j_0 v_0}\right], \begin{bmatrix} \hat{*}_{\left(r,r^2,\frac{N}{r^2}\right)}\left(\mathbf{T}_r, \begin{bmatrix} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[r(rn)]} W_{N/r^2}^{nv_1} \\ \vdots \\ w_N^{r(r-1)v_1} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[r(rn+(r-1))]} W_{N/r^2}^{nv_1} \end{bmatrix}\right) \\ \hat{*}_{\left(r,r^2,\frac{N}{r^2}\right)}\left(\mathbf{T}_r, \begin{bmatrix} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[r(rn)+1]} W_{N/r^2}^{nv_1} \\ \vdots \\ w_N^{r(r-1)v_1} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[(r(rn+(r-1))+1)]} W_{N/r^2}^{nv_1} \end{bmatrix}\right) \\ \vdots \\ \hat{*}_{\left(r,r^2,\frac{N}{r^2}\right)}\left(\mathbf{T}_r, \begin{bmatrix} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[r(rn)+(r-1)]} W_{N/r^2}^{nv_1} \\ \vdots \\ w_N^{r(r-1)v_1} \sum_{n=0}^{\left(\frac{N}{r^2}\right)-1} x_{[r(rn+r-1)+(r-1)]} W_{N/r^2}^{nk_1} \end{bmatrix}\right) \end{bmatrix}\right) \quad (23)$$

which could be simplified as:

$$\mathbf{X}_{[k]} = \hat{*}_{(r,r,N/r)}\left(\mathbf{T}_r\left[\mathbf{W}_N^{j_0 v_0}\right], col\left(\hat{*}_{\left(r,r^2,N/r^2\right)}\left(\mathbf{T}_r\left[\mathbf{W}_N^{rj_1 v_1}\right], col\left(\sum_{n=0}^{v_1-1} x_{[r^2 n + rj_1 + j_0]} W_{N/r^2}^{nv_1}\right)\right)\right)\right)$$

(24)

for j_0=0,...,r-1, j_1=0,...,r-1, v_1=0,1,...,N/r^2 – 1, and $\left[\mathbf{W}_N^{rj_1 v_1}\right] = diag\left(w_N^0, w_N^{rv_1}, \cdots, w_N^{r(r-1)v_1}\right)$. If the factorization process will continue till we get r^s transform of size r, then Eq. (1) will be expressed as:

$$\mathbf{X}_{[k]} = \hat{*}_{\left(r,r^s,k_s\right)_{s=0,1,\ldots,\log_r N-2}}\left(\mathbf{T}_r\left[\mathbf{W}_N^{r^s j_s v_s}\right], col\left(\sum_{n=0}^{v_s-1} x_{[r^{(s+1)}n + r^{(s)}j_s + \ldots + j_0]} W_{N/r^{s+1}}^{nv_s}\right)\Bigg|_{v_s=0,1,\ldots,N/r^{s+1}-1}\right). \quad (25)$$

where $\left[\mathbf{W}_N^{r^{(s)}j_s v_s}\right] = diag\left(w_N^0, w_N^{r^s v_s}, \cdots, w_N^{r^s(r-1)v_s}\right)$.

In DSP Layman language, the factorization of an FFT can be interpreted as dataflow diagram (or Signal Flow Graph), which depicts the arithmetic operations and their dependencies. To be noted that the dataflow diagram is read from left to right we will obtain the decimation in frequency algorithm and where λ in Eq. (17) is equal to $r^{(-1)}$, meanwhile if the dataflow diagram is read from right to left we will obtain the decimation in time algorithm and where λ in (17) is equal to r.

4. The One Stage FFT

Eq. (25) could be simplified as:

$$\mathbf{X}_v[l] = \sum_{j_0=0}^{r-1}\sum_{j_1=0}^{r-1}\cdots\sum_{j_S=0}^{r-1}\sum_{n=0}^{r-1} x_{[r^S n + r^{S-1} j_{S-1} + \cdots + j_0]}\, w_N^{\left\| \frac{JNl}{r} + \left(J + \frac{Nn}{r}\right)v \right\|_N} \tag{26}$$

where $J = r^{S-1}J_{S-1} + r^{S-2}j_{S-2} + \cdots + rj_1 + j_0$ and for $j_s = 0,1,\ldots,r-1, s \in [0, S-1]$, l=0,1,…,r – 1, v = 0,1,...,(N/r) – 1, $S = log_r N$. The l^{th} output of X is stored at the address memory location given by the writing address generator (WAG):

$$WAG = l(N/r) + v. \tag{27}$$

Finally, we can represent the execution of FFT in one stage (or phase), by adopting the following notations:

$$x_{(J_{(S-1)},\cdots,j_0)}[m] = x_{[r^S m + r^{S-1} j_{S-1} + \cdots + j_0]} \tag{28}$$

$$\left[\mathbf{B}_r\right]_{(l,m,j_S\ldots,j_0,v)} = w_N^{\left\| \frac{JNl}{r} + \left(J + \frac{Nm}{r}\right)v \right\|_N}. \tag{29}$$

Fig. 3 illustrates the radix-r one stage processing element (OSPE) in which r multipliers are implemented in parallel and executed in one cycle. To increase the data throughput, we can easily increase the degree of parallelism as shown in Fig. 4, where the results are obtained in one clock cycle to satisfy high sustained throughput applications.

The data, $\left[\mathbf{B}_r\right]_{(l,m,j_S\ldots,j_0,v)}$, is localized at each multiplier and use an address generator based on simple digital counters.

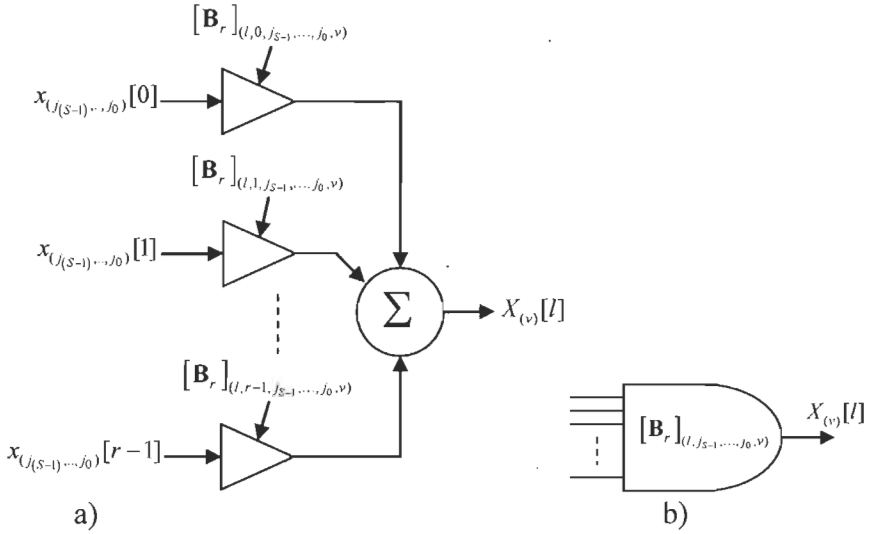


Figure 3: Radix-r OSPE (a) using (26) and the symbol (b).

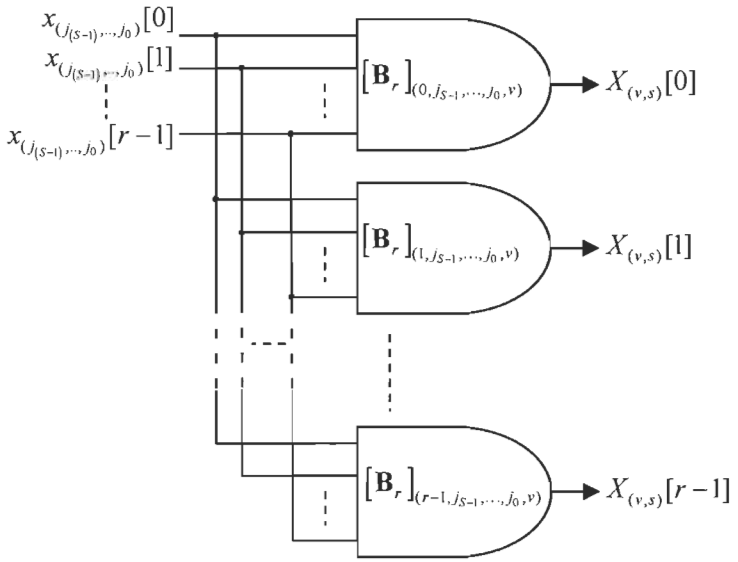


Figure 4: Maximize the data throughput using r OSPE in parallel.

The one stage FFT structure is suitable to customize the hardware implementation that take into account the VLSI constraints such as data throughput, area, and power consumption. The localized communication, regularity and recursiveness of the equations make it flexible to satisfy large application domains. The proposed structure gives us the ability to divide a process into serial and parallel portions (or pure parallel portions) where the parallel parts are executed concurrently.

Tables 1 to 3 show the performance comparison of the adder matrix versus the proposed iteration FFT (Fig. 2 and 4) [4]. Table 2 shows more clock cycles than Table 1 but with a lower number of hardware resources by applying a time multiplex implementation on the BPE (Fig. 2); however, we drastically reduce the clock cycles by using the r OSPE in parallel (Fig. 4) as shown in Table 3.

Table 1: Number of cycles need to execute a 4096-points FFT for different radices by factoring the adder matrix

Cycles Requirements	Radix – 2	Radix – 4	Radix – 8
Phases	2	4	7
Memory accesses	71680	17408	5632
Complex multiplication	24576	5120	3072
Complex addition	24576	12288	6144

Table 2: Number of cycles needs to execute a 4096-points FFT for different radices by implementing r BPEs in parallel (Fig. 2).

Cycles Requirements	Radix – 8	Radix – 16
Phases	4	3
Memory accesses	3072	1536
Complex multiplication	1536	512
Complex addition	2048	768

Table 3: Number of cycles needs to execute a 4096-points FFT for different radices by using r OSPE (Fig. 4).

Cycles Requirements	Radix – 8	Radix – 16
Phases	1	1
Memory accesses	1024	512
Complex multiplication	512	16
Complex addition	512	16

5. Conclusion

Finally this paper has presented an efficient way of implementing the FFT process by mean of the one iteration radix-r kernel where a serial parallel model and a pure parallel model have been represented. Also, it has been argued that a reduction in the chip size, a reduction of its power consumption and an increase of the performance of the system could be achieved

Appendix

Properties of special product $\hat{*}_{(r,r,\beta)}$

Lemma

$$\begin{aligned}\mathbf{X}_{[k]} &= \hat{*}_{(r,r,\beta)}\left(\mathbf{T}_r,\left(\mathbf{W}_r col\left[x_{[rn+j_0]}\right]\right)\right)\\ &= \hat{*}_{(r,r,\beta)}\left(\mathbf{T}_r\mathbf{W}_r,\left(col\left[x_{[rn+j_0]}\right]\right)\right)\end{aligned} \tag{30}$$

Proof:

$$\mathbf{X}_{[k]} = \hat{*}_{(r,r,\beta)}\left(\mathbf{T}_r,\left(\mathbf{W}_r col\left(x_{[rn+j_0]}\right)\right)\right) = \mathbf{T}_r\left(\mathbf{W}_r col\left[x_{(rn+j_0)}\right]\right)$$

$$\mathbf{X}_{[k]} = \left(\mathbf{T}_r\mathbf{W}_r\right) col\left[x_{(rn+j_0)}\right] = \hat{*}_{(r,r,\beta)}\left(\left(\mathbf{T}_r\mathbf{W}_r\right),\left(col\left[x_{(rn+j_0)}\right]\right)\right)$$

References

[1] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex fourier series", Math. Comput., 19, pp. 297-301, April 1965.

[2] T. Widhe, J. Melander, and L. Wanhammar, "Design of efficient radix-8 butterfly PEs for VLSI", IEEE Int. Symposium on Circuit and Systems, vol. 3, pp. 2084-2087, June 1997

[3] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6,751,643, 2004.

[4] Y. Jung, H. Yoon and J. Kim, "New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Applications", IEEE Trans. On Consumer Electronics, vol. 49, no. 1, pp. 14-20, Feb. 2003

Paper VI: M. Jaber and D. Massicotte, "Fast Method to Detect Specific Frequencies in Monitored Signal", *International Symposium on Communications, Control and Signal Processing* (ISCCSP), Cyprus, March 2010.

Fast Method to Detect Specific Frequencies in Monitored Signal

Marwan A. Jaber and Daniel Massicotte , *Senior Member, IEEE*
Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integrations
{marwan.jaber, daniel.massicotte}@uqtr.ca

Abstract – **The Discrete Fourier Transform (DFT) is a mathematical procedure that stands at the center of the processing that takes place inside a Digital Signal Processor. It has been known and argued through the literatures that the Fast Fourier Transform (FFT) is useless in detecting a specific frequency in a monitored signal because most of the computed results are ignored. In this paper we will present an efficient FFT based method to detect specific frequencies in a monitored signal which is compared to the most frequently used method "the Goertzel's Algorithm". Parallel implementation structure show a fast computation method compared to the Goertzel's algorithm. Computational speedup gains by a factor of *r* when using radix-*r* butterflies are also shown.**

1. Introduction

Digital Signal Processing (DSP) is the branch of engineering concerned with the representation and manipulation of signals in digital form. The discipline of signal processing, whether analog or digital, consists of a large number of specific techniques. One of the most important techniques is the Signal-analysis/feature-extraction techniques

Manuscript received November 14, 2009. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.
Marwan A. Jaber and Daniel Massicotte are with the Electrical and Computer Engineering Department, Laboratory of Signal and System Integrations, Université du Québec à Trois-Rivières, Quebec, G9A 5H7, Canada (D. Massicotte is the corresponding author: +1-819-376-5011 x3918; fax: +1-819-376-5219; e-mail: marwan.jaber@uqtr.ca and daniel.massicotte@uqtr.ca.

which aim to extract useful information from a given monitored signal. Signal monitoring is an expanding domain that deal in detecting any abrupt changes for a special known frequency such as fault detection machine or to scan a pre-selected set of frequencies, as in radio-frequency identification (RFID) tags [1], the recognition of the dual-tone multi-frequency (DTMF) [2] and in the orthogonal frequency division multiplex wireless communication (OFDM), wherein the Fast Fourier Transform (FFT) is a major key operator [15], particularly for cognitive radio. Since the scope of our work will be targeting the wireless communications and specifically the OFDM applications, reduction in terms of complexity and increasing speed would be essential.

It was clearly evident that computing a specific frequency $X_{(k)}$ for a complex sequence $\{x_{(n)}\}$ with $n = 0,1,\ldots,N-1$ and $k = 0,1,\ldots,N-1$ by mean of the radix-r FFT is not advisable Since, the required number of complex multiplications is $O(N \log N)$. In order to compute the same frequency $X_{(k)}$ by using the canonical DFT equation is N complex multiplications.

For instance N=8 points DFT will require 12 complex multiplications when employing a radix-2 FFT algorithm meanwhile the canonical DFT equation will require 8 complex multiplications. As a result it seemed that the FFT algorithms are very useful for a wide class of problem but they are not the most efficient choice in all situations.

There were two other techniques for computing the DFT such as the Goertzel's algorithm [3] and the chirp transform [4]. In this paper we will be limiting our comparison study to the Goertzel's algorithm since the second one is unrealizable; it is neither causal nor stable [4]. However, the data dependency in the recursive form of Goertzel' algorithm limits seriously the parallel implementation. In this paper, our main objective is to propose

another approach enabling parallel pipeline implementations for fast specific frequencies computation.

This paper is organized as follows: Section 2 describes the first and second order Goertzel's algorithm. Section 3 will detail the proposed method, meanwhile Section 4 will draw the performance results of the proposed method and Section 5 is devoted to the conclusion.

2. The Goertzel's Algorithm

As previously stated; there are various ways to detect the presence of a specific frequency in a monitored signal. The FFT algorithm which will require the highest amount of complex multiplications to compute a specific frequency $X_{(k)}$ plus an extra memory of size N which is used to store the intermediate result. A direct computation of the DFT that will require less complex multiplication than the FFT with no need of the extra memory of size N to store the intermediate result, is the Goertzel's algorithm which is an efficient method (in terms of multiplications and memories) for computing $X_{(k)}$.

The derivation of the first-order Goertzel algorithm, which is developed in [4], begins by noting that the DFT can be formulated in terms of a convolution. In fact the DFT of the signal $x_{(k)}$:

$$\begin{aligned} X_{(k)} &= \sum_{n=0}^{N-1} x_{(n)} w_N^{nk}, \quad w_N^k = e^{-j\frac{2\pi k}{N}} \\ &= \sum_{n=0}^{N-1} x_{(n)} w_N^{-k(N-n)}, \; w_N^{-kN} = 1, \\ &= x_{(n)} * h_{(n)}\Big|_{n=N}, \end{aligned} \tag{1}$$

where * represents the convolution product of the signal $x_{(n)}$ through a linear time invariant (LTI) filter with the impulse response $h_{(n)} = w^{-nk} u_{(n)}$ and evaluating the result, $y_{k(n)}$, at $n=N$ as illustrated in Fig. 1 [1].

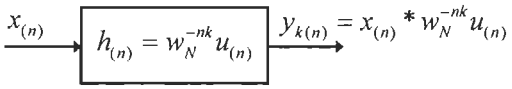


Figure 1: The LTI filter represented by Eq. (1).

According to the same reference [1], the representation of the LTI filter by its z transform will lead to the same filtering operation as illustrated in Fig. 2. The filtering operation of first-order Goertzel algorithm with the associated flow graph is depicted in Fig. 3. We can write the recurrent equations as;

$$y_{k(n)} = w_N^{-k} y_{k(n-1)} + x_{(n)}, \tag{2}$$

where $y_{k(-1)} = 0$, $n = 0,1,2,\ldots,N-1$, and $k = 0,1,2,\ldots,N-1$.

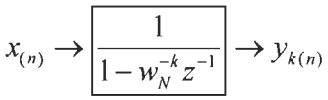


Figure 2: Representation of the LTI filter by its z transform.

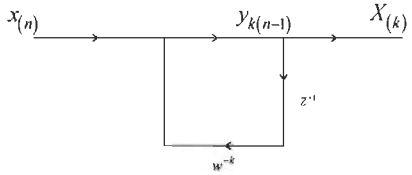


Fig. 3 The first-order Goertzel

After N iterations, the output of the filter for the k^{th} frequency is

$$X_{(k)} = y_{k(N-1)}. \tag{3}$$

The transfer function H(z) of the equivalent filter could be developed as [2]

$$H(z) = \frac{1}{1-w_N^{-k}z^{-1}} = \frac{1-w_N^{-k}z^{-1}}{1-w_N^{-k}z^{-1}} \times \frac{1}{1-w_N^{-k}z^{-1}} = \frac{1-w_N^{-k}z^{-1}}{1-\left(2\cos\frac{2\pi k}{n}\right)z^{-1}+z^{-2}} \tag{4}$$

where the second-order Goertzel algorithm is obtained and the filtering operation with the associated flow graph is depicted in Fig. 4. The recurrent equations are

$$y_{k(n)} = 2\cos(2\pi k / N) y_{k(n-1)} - y_{k(n-2)} + x_{(n)}, \tag{5}$$

and after N iterations, we have the k^{th} frequency output

$$X_{(k)} = \cos(2\pi k / N) y_{k(N-1)} - y_{k(N-2)} + j \sin(2\pi k / N) y_{k(N-1)}, \tag{6}$$

where $y_{k(-2)} = y_{k(-1)} = 0$.

For applications with a real-valued measurement stream, the C implementation of the Goertzel's algorithm is illustrated in Fig. 5 where the results are the real and imaginary parts of the DFT transform for specific frequency k.

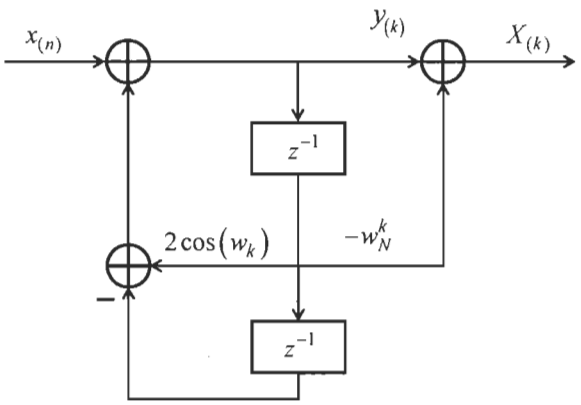


Fig. 4 The second-order Goertzel filter

```
realW = 2.0*cos(2.0*pi*k/N);
imagW = sin(2.0*pi*k/N);
d1 = 0.0;
d2 = 0.0;
for (n=0; n<N; ++n)
{
y  = x(n) + realW*d1 - d2;
d2 = d1;
d1 = y;
}
resultr = 0.5*realW*d1 - d2;
resulti = imagW*d1;
```

Figure 5: C Function of the Goertzel's algorithm [7].

The computational complexity of the first-order Goertzel's algorithm is:

4N real multiplications and 4N real additions,

and from Fig. 4, the computational cost of the second-order Goertzel's algorithm is thus [7]:

$$2N + 2 \text{ real multiplies and } 4N - 2 \text{ real adds,}$$

which gives a reduction by almost a factor of two in the number of real multiplications if compared to the DFT equation. This cost is halved again if the data are real-valued. Furthermore, the first-order filter needs more resources due to the complex multiplication by twiddle factors in the feedback loop, Eq.(2). Beraldin and al. [16] showed an interesting overflow analysis in fixed-point implementations for the first and second-order Goertzel's algorithm. In fixed-point implementation, it was concluded that the first-order filter achieves better accuracy than the second-order filter. Thus, the first-order filter is more interesting than the second-order filter in practical way.

3. The Proposed Method

The definition of the DFT is represented by the following equation

$$\mathbf{X}_{(k)} = \sum_{n=0}^{N-1} x_{(k)} w_N^{nk}, \quad k \in [0, N-1], \tag{7}$$

which could be factorized as follow [8], [9]:

$$\begin{aligned}\mathbf{X}_{(v+qN/r)} = {} & w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{nv} + w_N^v w_N^{qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{nv} + \\ & w_N^{2v} w_N^{2qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{nv} + \cdots + w_N^{(r-1)v} w_N^{q(r-1)N/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{nv}\end{aligned} \tag{8}$$

with $k = 0,1,\cdots,N-1$, $v = 0,1,....,(N/r)-1$ and $q = 0,1,....,r-1$.

Further decomposition of the DFT in terms of its partial DFT will yield to the one iteration FFT (one stage) expressed as [10][4]:

[4] Patented

$$X_{(qV+v)} = \sum_{a_0=0}^{r-1}\sum_{a_1=0}^{r-1}\cdots\sum_{a_{S-2}=0}^{r-1}\sum_{n=0}^{r-1} x_{(r^S n+A)} w_N^{[\![qAV+(A+nV)v]\!]_N} \tag{9}$$

where $[\![x]\!]_N$ represents the operation x modulo N, $A = r^{S-2}a_{S-2} + r^{S-3}a_{S-3} + \cdots + ra_1 + a_0$ and for $a_s = 0,1,\ldots,r-1$, $s = 0,1,\ldots,S-2$ with $S=log_r N$-1.

The input of $x_{(RDAG)}$ is read from the address memory location given by the reading data address generator (RDAG):

$$RDAG = r^S n + A, \tag{10}$$

and the output of $X_{(WDAG)}$ is stored by the writing data address generator (WDAG) [11]:

$$WDAG = qN / r + v. \tag{11}$$

The implementation of the radix-r one iteration FFT is illustrated in Fig. 6, where in this figure the coefficients are defined as:

$$[\mathbf{B}_r]_{(n,q,p,v)} = w_N^{[\![qAV+(A+nV)v]\!]_N}, \tag{12}$$

The coefficients in equation 12 are provided by the reading coefficients (twiddle factors) address generator (RCAG) expressed as:

$$RCAG = [\![qAV + (A+nV)v]\!]_N. \tag{13}$$

Fig. 6 represents the one stage FFT butterfly processing element (BPE) using r parallel complex value multipliers plus r input complex value accumulators (Fig. 6a) that could be simplified to one complex multiply-accumulator (C-MAC) as shown in Fig. 6b. Fig. 7 represents the implementation of the one iteration FFT for the specific radices 4 and 8. In order to perform a fair comparison with our proposed parallel structure, our performance study will take into consideration the parallel structure of the Goertzel's algorithm as illustrated in Fig. 8.

For such types of ordered input ordered output (OIOO) FFTs presented in [8], the k domain is subdivided into r equal sub-domain of size N/r-1 as illustrated in Eq. (11). In order to compute a specific frequency $X_{(k)}$ for a given k by mean of the one iteration radix-r FFT (the proposed method), we have to know the values of q and v which are computed according to:

$$\begin{cases} 0 \le k < \dfrac{N}{r} & q = 0 \text{ and } k = v \\ \dfrac{N}{r} \le k < \dfrac{2N}{r} & q = 1 \text{ and } k - \dfrac{N}{r} = v \\ \vdots & \\ \dfrac{(r-1)N}{r} \le k < N & q = (r-1) \text{ and } k - \dfrac{(r-1)N}{r} = v \end{cases} \quad (14)$$

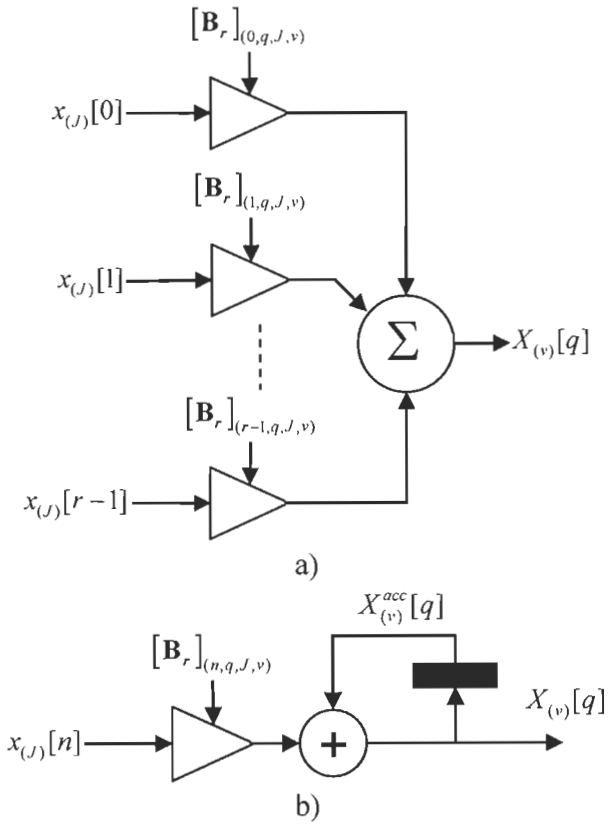


Figure 6: Radix-r one iteration FFT using r consecutive complex multipliers (a) and one complex multiplier (b).

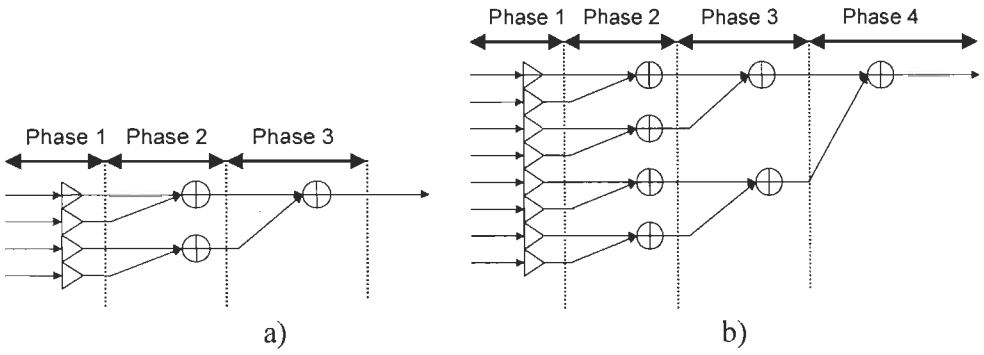


Figure 7: SFG of the radix-4 (a) and radix-8 (b) BPE from Fig. 5.

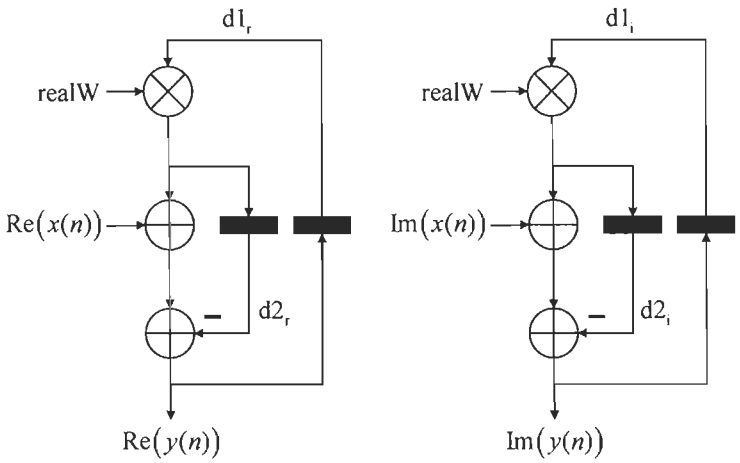


Figure 8: parallel Implementation of Goertzel' algorithm.

4. Performance Results

The computational complexity of our algorithm, Eq. (9), will be the same as the parallelized first-order Goertzel's algorithm with no feedback multiplication. This can reduce significantly the fixed-point arithmetic computation where both, first and second order versions of Goertzel filters, are very sensitive [16]. Furthermore, we can parallelize the implementation of Eq. (9) to show a fast computation method compared to the first and second-order Goertzel's algorithms.

To compare the computational time, t_c of our proposed approaches with the Goertzel's algorithm, we will define the number of clock cycle (T_{clk}) needed for both approaches.

Fig. 8 represents the hardware implementation of the Goertzel's algorithm Eq. (6) in which by adding a simple multiplexing control, we can reuse the hardware. The clock cycle in this architecture is defined by the time delay of one real multiplier (T_M) plus two adders (T_A), $T_{clk}=T_M+2T_A$. Based on this approach, the computational cost of the second-order Goertzel's algorithm for k^{th} frequency will be:

$$t_{c\ \text{Goertzel's}} = NT_{clk}. \tag{15}$$

Our approach will be based on the one iteration FFT where we will consider two different implementations as shown in Fig. 6. Unlike Goertzel's algorithm (Fig. 3 and 4), our proposed structures are based on complex multipliers. We can reduce the number of real value multiplications to compute the product of two complex numbers

$$(a_r + ja_i)(b_r + jb_i) = (a_r b_r - a_i b_i) + j(a_r b_i + a_i b_r) = p_r + jp_i \tag{16}$$

by mean of pipelined complex multipliers as shown in Fig. 11a.

However, the number of real multiplier can be reduced by using the following simplification [12]

$$\big(e - b_i(a_r + a_i)\big) + j\big(e + b_r(a_i - a_r)\big), \tag{17}$$

where $e = a_r(b_r + b_i)$. The pipelined structure of the complex multiplier which is based on three real multipliers is shown in Fig. 9b. Based on these pipeline structures, the number of clock cycle in the critical path is $T_{clk}= T_M +T_A+T_{lost}$ in which T_{lost} is the time lost in the control, communication and connection. We can assume that, $T_{lost}<T_A$ which will be equivalent to Goertzel's algorithm clock period, $T_{clk}=T_M+2T_A$, (Fig. 7).

Eq. (5), which is based on a simple nested summation, a straight feed-forward pipeline implementation can be applied where the computational cost in term of clock cycle of our proposed method is

$$t_{c\ \text{Fig.6a}} = \left(N/r + \log_2 r + S_p + 2\right) T_{clk}/S_p\ , \tag{18}$$

for the BPE of Fig. 6a, where the $\log_2 r + S_p + 2$ represents the clock cycles needed to drain the pipeline C-MAC and the adder tree.

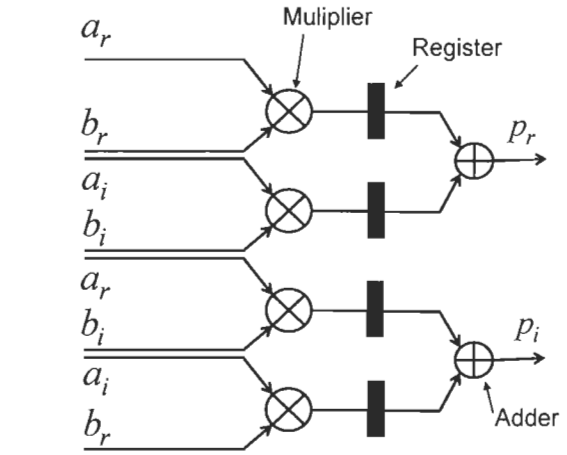
For BPE of Fig. 6b, we have

$$t_{c\ \text{Fig.6b}} = \left(N + S_p + 2\right) T_{clk}/S_p\ , \tag{19}$$

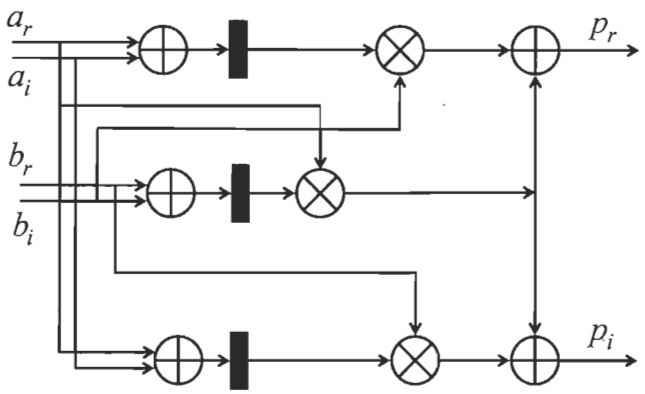
where S_p represents the number of pipelined stage applied to the real multiplier of the Fig. 8. As, shown in [13], we can easily, implement a pipelined complex multiply-accumulator (C-MAC) in our BPE, which is based on the wave pipelined approach [13], [14] in order to decrease T_{clk} up-to a factor of 5. In our evaluation study, we will consider S_p=2.

Fig. 6b needs fewer resources with more clock cycles to compute the k^{th} frequency signal's output. The adder, in Fig. 6a, where the r complex multipliers are combined can be realized by adopting both solutions as shown in Fig. 10 where Table 1 shows the performance of the r-input adder tree structure.

In our performance study, we will present the results in terms of time computation (t_c) between the proposed one iteration FFT, (Eq. (18) and (19) derived from (9)) which is implemented on the one iteration radix-r BPE as illustrated in Fig. 6. The gain G, compared to the Goertzel's algorithm, is shown in Fig. 11, for both proposed radix-r one iteration structures Fig. 6a and 6b where the gains are respectively:

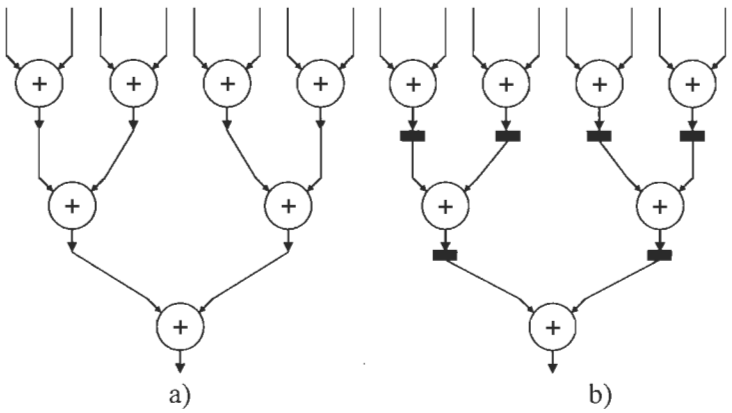


a)



b)

Figure 9: Pipelined complex multiplier using 4 real multipliers (a) and three real multipliers (b).



a) b)

Figure 10: Adder tree circuit structures *a*, and *b*.

$$G_{\text{Fig.6a}} = S_p N \Big/ \left({}^{N}\!/_{r} + \log_2 r + S_p + 2 \right), \tag{20}$$

and

$$G_{\text{Fig.6b}} = S_p N \Big/ \left(N + S_p + 2 \right). \tag{21}$$

Table 1: Performance of adder tree structures for r-input (Fig. 9)

	Structure *a*	Structure *b*
Critical path in number of delay of an adder	$\lceil \log_2(r) \rceil$	1
Number of adders	r-1	r-1
Number of registers	0	$\lceil \log_2(r) \rceil$

Fig. 11a and 11b reveal that the speedup gain is 2 for radix-2 and by using more resources (BPE Fig. 6a) the speedup gains for higher radices present significant speedup improvements (up to r for radix-r) in order to detect specific frequencies in monitored signal applications.

The hardware resources in term of real multipliers and adders depend of the radix-r BPE. Table 2 presents the resources needed to compute the DFT for specific frequencies signal's output. The implementation's comparison of the resources needed to execute the respective method are exhibited in terms of: i) number of real number multiplications and additions, ii) number of full adders (FA) needed to implement both fixed-point arithmetic operators in the VLSI. In this comparison we considered 8-bit and 16-bit global lengths for the real multiplication and addition, respectively. The resources increase with the increasing BPE's radix-r of Fig. 6a. In the case of BPE of Fig. 6b, we need 7 real adders and 3 real multipliers for a total of 304 FA. We need the same resources to implement the first-order Goertzel filter.

Using the proposed pipeline structure in Fig. 6b, we increase the implementation area by 58% to double the speed. We can conclude from the results shown in Fig. 11 and Table 2 compared to Goertzel's algorithm, the following

– we have a computation speedup of 2 by using low complexity hardware implementation which is based on one pipelined complex multiplier and tree adders.

– we have a computation speedup of r by using more resources in the radix-r butterfly (speedup of 16 for radice-16).

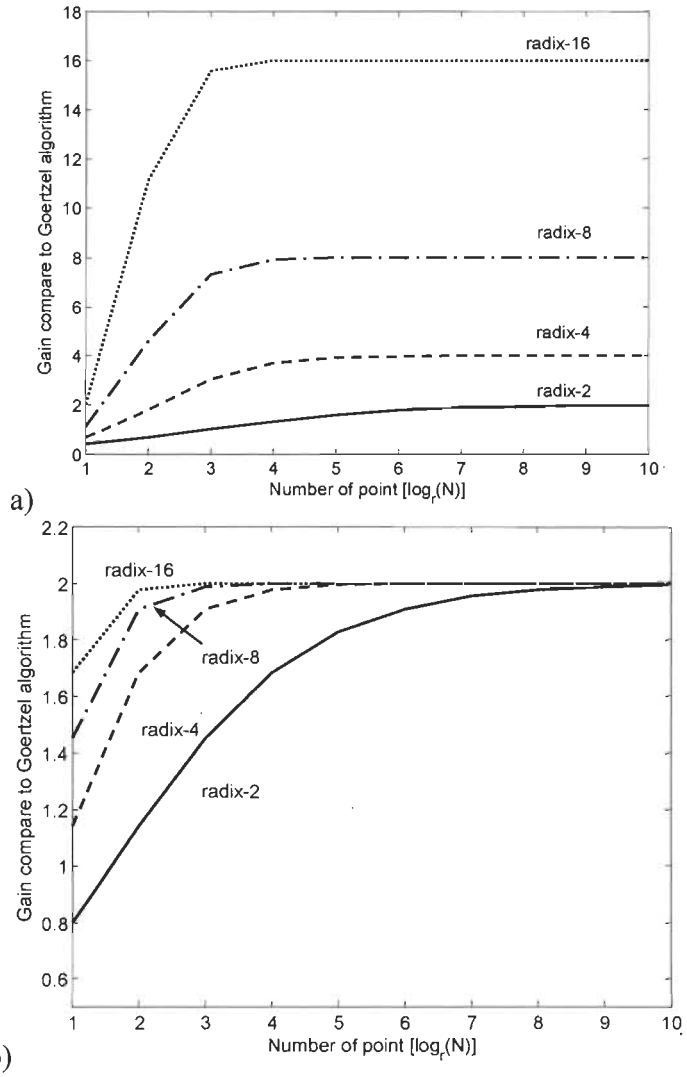


Fig. 11 Speed gain of proposed one iteration compared to Goertzel's algorithms using a) the BPE Fig. 6a Eq. (20) with Sp =1 and b) the two stages pipelined MAC BPE Fig. 6b Eq. (21) with Sp =2.

Table 2: Hardware resources in term of real adders (Adder) and multipliers (Mult) to implement the different BPEs in our study

Radices	Goertzel Algorithms (Fig. 8)			One iteration BPE Fig. (6a)			FA Ratio
	Adder	Mult.	FA	Adder	Mult.	FA	
Radix-2	4	2	192	12	6	576	3
Radix-4	-	-	-	26	12	1440	7.5
Radix-8	-	-	-	52	24	2368	12.3
Radix-16	-	-	-	110	32	3808	19.8

5. Conclusion

It is not unusual to find numerous algorithms to complete a given DFT task, so, finding the best trade off algorithm-architecture with the best performance is a crucial engineering problem for the real time signals' analysis. In this paper we have shown that the FFT algorithm known as the one iteration FFT algorithm could be the perfect choice for computing $X_{(k)}$ for a given k. Our performance comparison to Goertzel's algorithm had shown a significant speed gain as shown in Fig. 11. Furthermore, the feedback operation in the Goertzel's algorithm causes a high sensitivity to fixed-point operations [16] contrarily to our proposed method where we need only a complex multiplier accumulation operator in absence of any feedback. We should cumulate the input data multiplied with the corresponding twiddle factor to the specific detected frequency.

Future works will be done in fixed-point analysis and FPGA implementation for FFT pruning in real time applications such as in OFDM based cognitive radio.

Acknowledgment

The authors would like to thank the financial support from the Natural Sciences and Engineering Research Council of Canada.

References

[1] B Corbett and M. Woodman, "Radio frequency identification tags", US Patent 7338497, June 17, 2008.

[2] C J Chen, "Modified Goertzel Algorithm in DTMF Detection Using the TMS320C80", Texas Instuments SPRA006, June 1996.

[3] G Goertzel, "An Algorithm for the Evaluation of Finite Trigonomentric Series", American Mathematical Monthly, 1958.

[4] Oppenheim, Schafer, and Buck "Discrete-Time Signal Processing", 2nd Ed., Prentice-Hall, 1999.

[6] http://www.mstarlabs.com/dsp/goertzel/goertzel.html.

[7] Douglas Jones, "Goertzel's Algorithm," Connexions, September 12, 2006, http://cnx.org/content/m12024/1.5/.

[8] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009, pp.1-6.

[9] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing", DSP'09, Santorini, Greece, 5-7 July 2009, pp.1-5.

[10] M Jaber and D. Massicotte "The Radix-r One Stage FFT Kernel Computation" Int. Conf. Acoustic, Speech, and Signal Processing, April First, Las Vegas, Nevada USA, 2008, pp. 3585 - 3588.

[11] M. Jaber and D. Massicotte "A Novel Approach for FFT Data Reordering", accepted to IEEE International Symposium on Circuit and Systems (ISCAS), Paris, May 2010.

[12] "Complex Multiplier v2.1", Xilinx, Product specification, DS291, April 2005.

[13] P.A. Laporte, D. Massicotte, C. Wang, M. Ahmed-Ouameur, "A WCDMA Multipath Searcher Based on a Wave Pipeline Method", Conf. NEWCAS'2007, Montreal, 5-8 August, 2007, pp.11-53-1156.

[14] F. Morin, M. Vidal et D. Massicotte, "A High Throughput Architecture for Channel Equalization Based on Neural Network Using Wave Pipeline Method", IEEE-CCECE'99, Calgary, May 1999, pp. 560-564.

[15] Q. Zhang, A.B.J. Kokkeler, and G.J.M. Smit, "An Efficient FFT For OFDM Based Cognitive Radio On A Reconfigurable Architecture", IEEE International Conference on Communications (ICC), Glasgow, June 2007, pp. 6522-6526.

[16] J.A. Beraldin and W. Steenaart, "Overflow analysis of a fixed-point implementation of the Goertzel algorithm", IEEE Transactions on Circuits and Systems, vol. 36 , no 2, Feb. 1989, pp. 322-324.

Paper VII: M. Jaber and D. Massicotte, The JM-filter to Detect Specific Frequencies in Monitored Signal, *to be submitted to a Journal after the end of the confidentiality.*

The JM-Filter to Detect Specific Frequencies In Monitored Signal

Marwan A. Jaber and Daniel Massicotte, *Senior Member, IEEE*

Abstract – **The Discrete Fourier Transform (DFT) is a mathematical procedure that stands at the center of the processing that takes place inside a Digital Signal Processor. It has been known and argued through the literatures that the Fast Fourier Transform (FFT) is useless in detecting a specific frequency in a monitored signal because most of the computed results are ignored. In this paper we will present an efficient JM filter (Jaber-Massicotte Filter) to detect specific frequencies in a monitored signal which is compared to the most frequently used method "the Goertzel's Algorithm". Computational Speedup gains of *r* using radix-*r* JM filters are shown.**

1. Introduction

Digital Signal Processing (DSP) is the branch of engineering concerned with the representation and manipulation of signals in digital form. The discipline of signal processing, whether analog or digital, consists of a large number of specific techniques. One of the most important techniques is the Signal-analysis/feature-extraction techniques which aim to extract useful information from a given monitored signal. Signal monitoring is an expanding domain that deal in detecting any abrupt changes for a special known frequency such as fault detection machine or to scan a pre-selected set of frequencies, as in radio-frequency identification (RFID) tags [1], the recognition of the dual-tone multi-frequency (DTMF) signaling and a lot of none cited domains [2]. The most well-known techniques to compute the DFT are the Goertzel's algorithm [3] and the chirp transform [4],

where in this paper we will be limiting our comparison study to the Goertzel's algorithm since the second one is unrealizable due to the fact that it is neither causal nor stable [4].

This paper is organized as follows: Section 2 describes briefly the first and second order Goertzel's algorithm and section 3 will deeply elaborate the One Iteration Radix-*r* JMFFT meanwhile section 4 will detail the first and second order JM-filter. Section 5 will detail the reduced complexity of the proposed method. Section 6 will draw the performance results of the proposed method Signal to Quantization Noise Ratio (SQNR) meanwhile Section 7 is devoted to the conclusion.

2. The First and second order Goertzel's Algorithm

As previously stated; there are various ways to detect the presence of a specific frequency in a monitored signal. The FFT algorithm which will require the highest amount of complex multiplications to compute a specific frequency $X_{(k)}$ plus an extra memory of size N which is used to store the intermediate result. A direct computation of the DFT which will require less complex multiplication than the FFT with no need of the extra memory of size N to store the intermediate results, is the Goertzel's algorithm which is an efficient method (in terms of multiplications and memories) for computing $\mathbf{X}_{(k)}$.

The derivation of the first-order Goertzel algorithm, which is developed in [4], begins by noting that the DFT can be formulated in terms of a convolution. In fact the DFT of the signal $x_{(k)}$:

$$\begin{aligned} X_{(k)} &= \sum_{n=0}^{N-1} x_{(n)} w_N^{nk}, \quad w_N^k = e^{-j\frac{2\pi k}{N}} \\ &= \sum_{n=0}^{N-1} x_{(n)} w_N^{-k(N-n)}, \; w_N^{-kN} = 1, \\ &= x_{(n)} * h_{(n)}\Big|_{n=N}, \end{aligned} \tag{1}$$

where * represents the convolution product of the signal $x_{(n)}$ through a linear time invariant (LTI) filter with the impulse response $h_{(n)} = w^{-nk}u_{(n)}$ and evaluating the result, $y_{k(n)}$, at $n=N$ as illustrated in Fig. 1 [1].

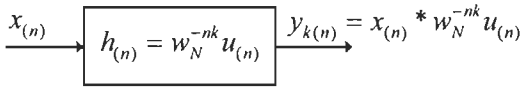


Figure 1: The LTI filter represented by Eq. (1).

According to the same reference [1], the representation of the LTI filter by its z transform will lead to the same filtering operation as illustrated in Fig. 2. The filtering operation of the first-order Goertzel algorithm with the associated flow graph is depicted in Fig. 3. We can write the recurrent equations as;

$$y_{k(n)} = w_N^{-k} y_{k(n-1)} + x_{(n)}, \tag{2}$$

where $y_{k(-1)} = 0$, $n = 0,1,2,\ldots,N-1$, and $k = 0,1,2,\ldots,N-1$.

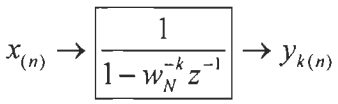


Figure 2: Representation of the LTI filter by its z transform.

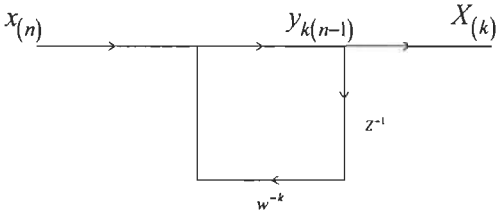


Fig. 3 The first-order Goertzel

After N iterations, the output of the filter for the k^{th} frequency is

$$X_{(k)} = y_{k(N-1)}. \tag{3}$$

The transfer function H(z) of the equivalent filter could be developed as [2]

$$H(z)=\frac{1}{1-w_N^{-k}z^{-1}}=\frac{1-w_N^{-k}z^{-1}}{1-w_N^{-k}z^{-1}}\times\frac{1}{1-w_N^{-k}z^{-1}}=\frac{1-w_N^{-k}z^{-1}}{1-\left(2\cos\frac{2\pi k}{n}\right)z^{-1}+z^{-2}} \tag{4}$$

where the second-order Goertzel algorithm is obtained and the filtering operation with the associated flow graph is depicted in Fig. 4. The recurrent equations are

$$y_{k(n)}=2\cos(2\pi k/N)y_{k(n-1)}-y_{k(n-2)}+x_{(n)}, \tag{5}$$

and after N iterations, we have the k^{th} frequency output

$$X_{(k)}=\cos(2\pi k/N)y_{k(N-1)}-y_{k(N-2)}+j\sin(2\pi k/N)y_{k(N-1)}, \tag{6}$$

where $y_{k(-2)}=y_{k(-1)}=0$.

For applications with a real-valued measurement stream, the C implementation of the Goertzel's algorithm is illustrated in Fig. 5 where the results are the real and imaginary parts of the DFT transform for a specific frequency k.

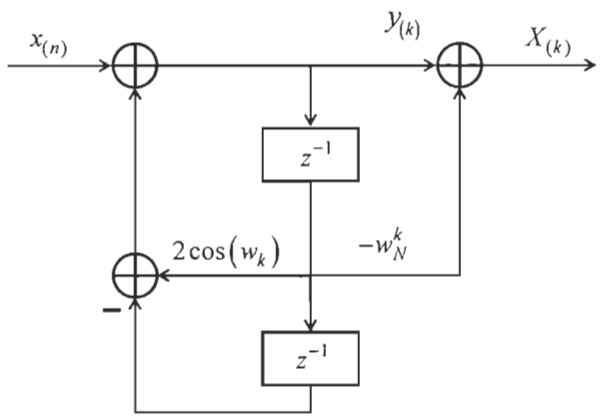


Fig. 4 The second-order Goertzel filter

In case of complex-valued input sequences, the computational complexity of the first-order Goertzel's algorithm is:

$4N$ real multiplications and $4N$ real additions, (7)

and from Fig. 4, the computational cost of the second-order Goertzel's algorithm is thus:

$2N+2$ real multiplications and $4N-2$ real additions, (8)

which gives a reduction by almost a factor of two in the number of real multiplications if compared to the DFT equation. This cost is halved again if the data are real-valued. Furthermore, Furthermore, the first-order filter needs more resources due to the complex multiplication by twiddle factors in the feedback loop, Eq.(2). Beraldin and al. [16] showed an interesting overflow analysis in fixed-point implementations for the first and second-order Goertzel's algorithm. In fixed-point implementation, it was concluded that the first-order filter achieves better accuracy than the second-order filter. Thus, the first-order version is more interesting than the second-order version in practical way.

```
realW = 2.0*cos(2.0*pi*k/N);
imagW = sin(2.0*pi*k/N);
d1 = 0.0;
d2 = 0.0;
for (n=0; n<N; ++n)
{
y  = x(n) + realW*d1 - d2;
d2 = d1;
d1 = y;
}
resultr = 0.5*realW*d1 - d2;
resulti = imagW*d1;
```

Figure 5: C Function of the second-order Goertzel's algorithm [7].

3. The One Iteration Radix-*r* JMFFT

The definition of the DFT is represented by

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(k)} w_N^{nk} \quad , \quad k \in [0, N-1], \tag{7}$$

which could be factorized as follow [9], [10]:

$$\begin{aligned} X_{(v+qN/r)} = {} & w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{nv} + w_N^v w_N^{qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{nv} + \\ & w_N^{2v} w_N^{2qN/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{nv} + \cdots + w_N^{(r-1)v} w_N^{q(r-1)N/r} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{nv} \end{aligned} \tag{8}$$

with $k = 0,1,\cdots,N-1$, $v = 0,1,....,V-1, q = 0,1,....,r-1$, and $V = N/r$.

Further decomposition of the DFT in terms of its partial DFT will yield to the one iteration FFT (one stage) expressed as [11] and [12]:

$$X_{(qV+v)} = \sum_{a_0=0}^{r-1}\sum_{a_1=0}^{r-1}\cdots\sum_{a_{S-2}=0}^{r-1}\sum_{m=0}^{r-1} x_{\left(r^S m+A\right)} W_N^{[\![qAV+(A+mV)v]\!]_N} \tag{9}$$

where $[\![x]\!]_N$ represents the operation x modulo N, $A = r^{S-2}a_{S-2} + r^{S-3}a_{S-3} + \cdots + ra_1 + a_0$ and for $a_0 = a_1 = \cdots = a_{(s-2)} = 0,1,\ldots,r-1$, $s = 0,1,\ldots,S-2$ with $S=log_r N - 1$.

Further simplification of Eq. (9) will yield

$$X_{(qV+v)} = \sum_{p=0}^{V-1}\sum_{m=0}^{r-1} x_{(Vm+p)} W_N^{[\![p(qV+v)+mvV]\!]_N}, \tag{10}$$

where the memory address location from which the data $x_{(k)}$ are collected is given by the reading data address generator (RDAG):

$$RDAG = mV + p, \tag{11}$$

and the processed data $X_{(k)}$ is stored by the writing data address generator (WDAG) [13]:

$$WDAG = qV + v. \tag{12}$$

For such types of FFTs, the k domain is subdivided into r equal sub-domain of size N/r-1 as presented in [9] and in order to compute a specific frequency $X_{(k)}$ for a given k the values of q and v should be known as shown in Eq. (13).

$$\begin{cases} 0 \le k < V & q = 0 \text{ and } v = k \\ V \le k < 2V & q = 1 \text{ and } v = k - V \\ \vdots & \\ (r-1)V \le k < N & q = (r-1) \text{ and } v = k - (r-1)V \end{cases}. \tag{13}$$

The implementation of the radix-r one iteration FFT is illustrated in Fig. 6 where in this Figure the coefficients are defined as

$$\left[\mathbf{B}_r\right]_{(m,q,p,v)} = w_N^{\left\lfloor p(qV+v)+mvV \right\rfloor_N}, \tag{14}$$

which are picked up by the reading coefficients (twiddle factors) address generator (RCAG) expressed as:

$$RCAG = \left[\!\left[p(qV+v)+mvV \right]\!\right]_N, \tag{15}$$

and the one iteration DIT FFT algorithm will be expressed as:

$$X_{(qV+v)} = \sum_{p=0}^{V-1} w_N^{\left\lfloor p(qV+v) \right\rfloor_N} \sum_{m=0}^{r-1} x_{(mV+p)} w_N^{\left\lfloor mvV \right\rfloor_N}. \tag{16}$$

Fig. 6 represents the two structures of the one iteration FFT butterfly processing element (BPE) in which r parallel complex multipliers are used (Fig. 6a) or one complex multiply-accumulator (C-MAC) is implemented as shown in (Fig. 6b). In order to perform a fair comparison for our proposed structure; our performance study will take into consideration the parallel structure of the Goertzel's algorithm as illustrated in Fig. 7.

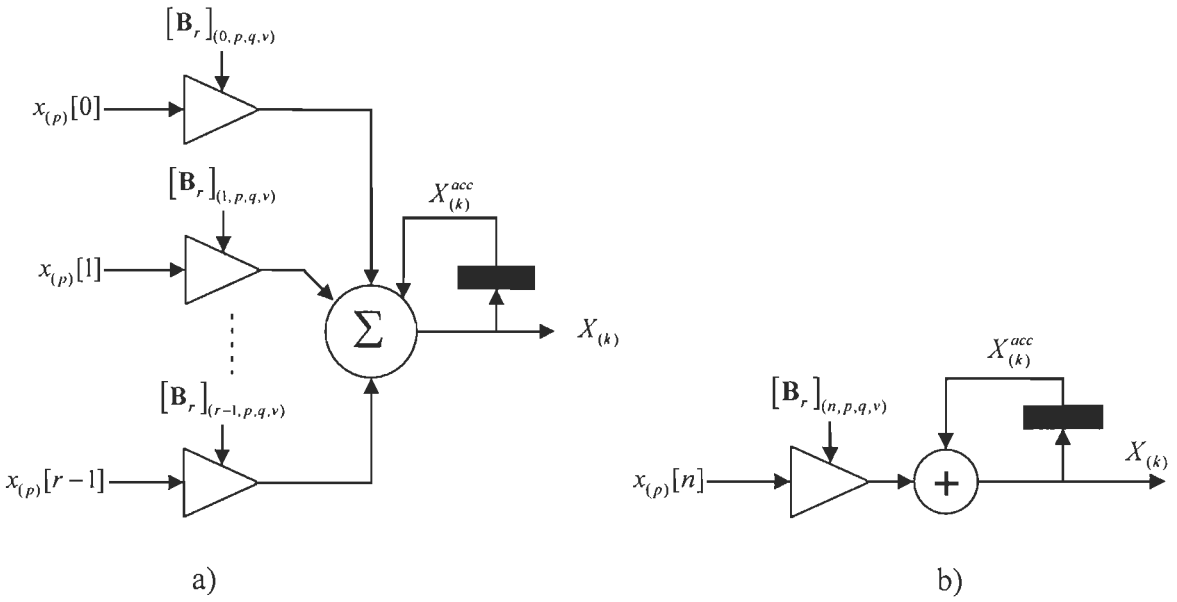


Figure 6: Radix-r one iteration JMFFT using r consecutive complex multipliers (a) and one C-MAC (b).

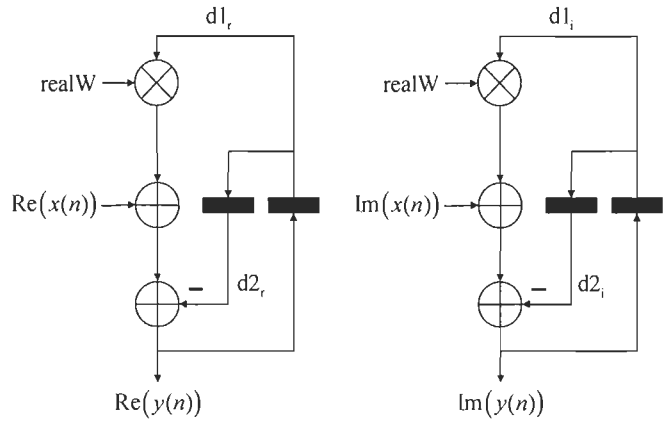


Figure 7: Parallel Implementation of Goertzel' algorithm.

4. The DFT as an Output Filter: The First and Second Order JM-filter

As stated earlier that in order to compute a specific frequency $X_{(k)}$ for a given k; the values of q and v are known in advance and by adopting the following notation

$$a_{(p)} = \sum_{m=0}^{r-1} x_{(Vm+p)} w_N^{|mvV|_N} = \sum_{m=0}^{r-1} x_{(Vm+p)} e^{-j\frac{2\pi\frac{N}{r}}{N}mv} = \sum_{m=0}^{r-1} x_{(Vm+p)} e^{-j\frac{2\pi}{r}mv}, \tag{17}$$

therefore, Eq. (16) can be expressed as:

$$X_{(qV+v)} = \sum_{p=0}^{V-1} a_{(p)} w_N^{p(qV+v)}, \tag{18}$$

where $k = qV + v$, and $w_N^k = e^{-j\frac{2\pi k}{N}} = e^{-j\frac{2\pi}{N}(qV+v)}$,

as a result the radix r-first order JM (Jaber Massicotte) filter could be derived as

$$y_{(p,(qV+v))} = w_N^{-(qV+v)} y_{(p-1,(qV+v))} + a_{(p)}, \tag{19}$$

where $y(-1,(qV+v)) = 0$ with $p=0,1,...,V-1$, and the k^{th} computed frequency is given by

$$X_{(qV+v)} = e^{-j\frac{2\pi}{r}(qV+v)} w_N^{-(qV+v)} y_{(V-1,(qV+v))}, \tag{20}$$

and the radix-r second-order JM-filter will be:

$$y_{(p,\langle qV+v\rangle)} = 2\cos(2\pi k/N)\, y_{(p-1,\langle qV+v\rangle)} - y_{(p-2,\langle qV+v\rangle)} + a_{(p)}, \tag{21}$$

from which the k^{th} computed frequency is

$$X_{(k)} = e^{-j\frac{2\pi}{r}(qV+v)}\left(0.5\cos(2\pi k/N)\, y_{(V-1,\langle qV+v\rangle)} + \sin(2\pi k/N) - y_{(V-2,\langle qV+v\rangle)}\right), \tag{22}$$

where $y_{(-2,\langle qV+v\rangle)} = y_{(-1,\langle qV+v\rangle)} = 0$.

The filtering operation for first and second-order JM-filter with the associated flow graph is depicted in Fig. 8 and 9, respectively.

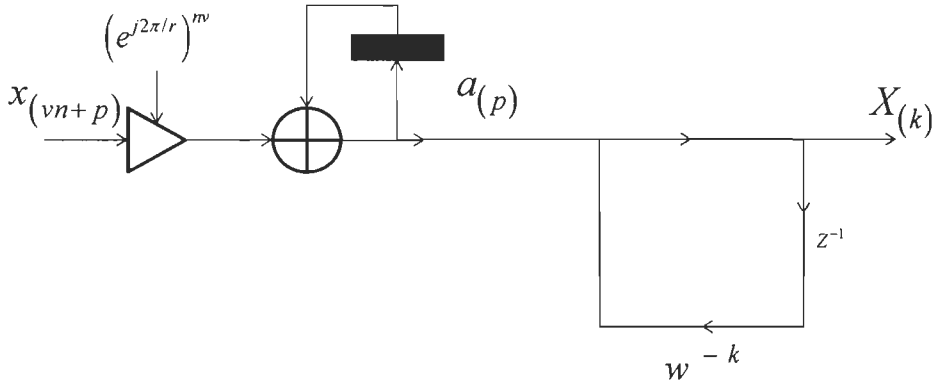


Figure 8: The radix-r first order JM-filter.

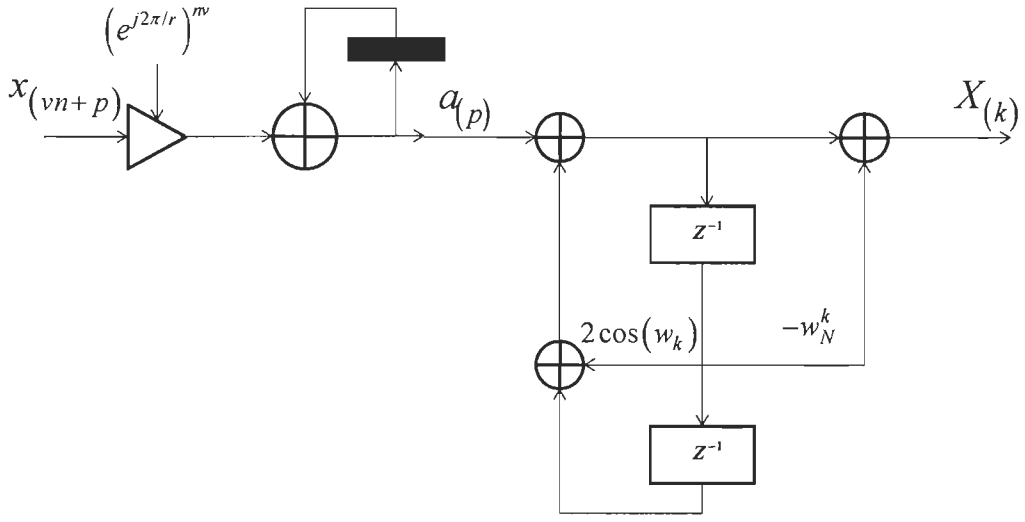


Figure 9: The radix-r second order JM-filter.

5. Complexity reduction

By examining Eq. (17) further reductions in terms of complexity could be achieved for the radix-2 case since:

$$e^{-j\frac{2\pi}{r}mv} = e^{-j\pi mv} = (-1)^{mv}, \tag{23}$$

therefore; based on equation (23), we can re-write Eq. (17) as:

$$a_{(p)} = \sum_{m=0}^{r-1} x_{(Vm+p)}(-1)^{mv} = x_{(p)} + (-1)^{v} x_{(Vm+p)}, \tag{24}$$

and the radix-2 JM first order filter would be:

$$y_{(p,(qV+v))} = w_N^{-(qV+v)} y_{(p-1,(qV+v))} + \left(x_{(p)} + (-1)^{v} x_{(Vm+p)} \right), \tag{25}$$

where the k^{th} computed frequency according to equation (20) is given by as shown in figure 10:

$$X_{(k)} = (-1)^{(qV+v)} w_N^{-(qV+v)} y_{(V-1,(qV+v))}. \tag{26}$$

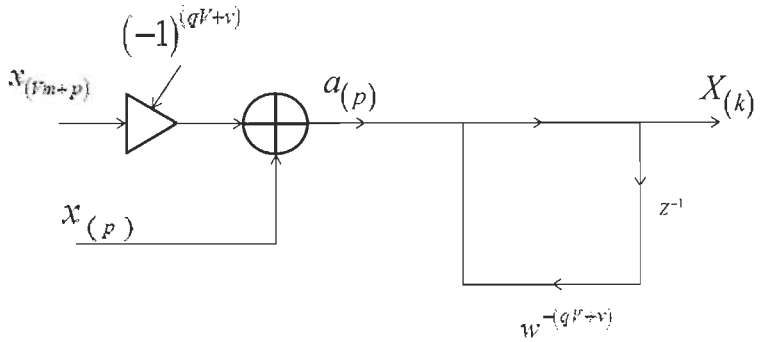


Figure 10: The radix-2 first order JM-filter.

The radix-2 second-order JM-filter will be (Figure 11):

$$y_{(p,(qV+v))} = 2\cos(2\pi k/N) y_{(p-1,(qV+v))} - y_{(p-2,(qV+v))} + \left(x_{(p)} + (-1)^{v} x_{(Vm+p)} \right), \tag{27}$$

from which the k^{th} computed frequency is

$$X_{(k)} = (-1)^{(qV+v)}\left(0.5\cos(2\pi k/N)y_{(V-1,(qV+v))} + \sin(2\pi k/N) - y_{(V-2,(qV+v))}\right), \qquad (28)$$

where $y_{(-2,(qV+v))} = y_{(-1,(qV+v))} = 0$.

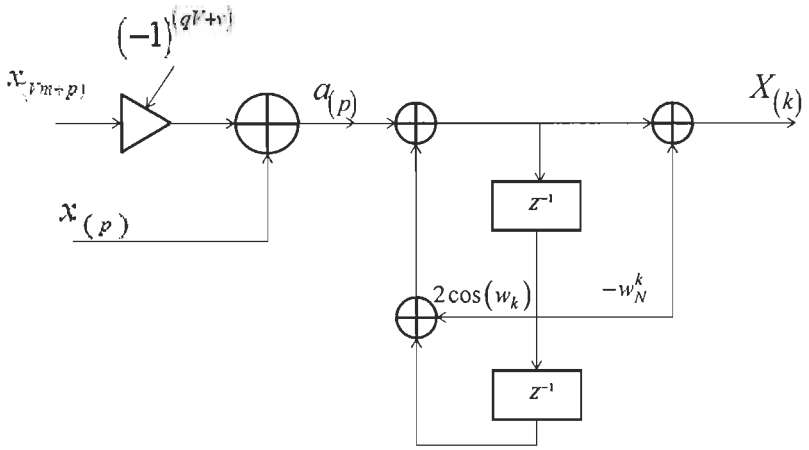


Figure 11: The radix-2 second order JM-filter.

With the same reasoning as above, further reductions in terms of complexity for the radix-4 could be achieved, in fact:

$$e^{-j\frac{2\pi}{4}mv} = e^{-j\frac{\pi}{2}mv} = (-j)^{mv}, \qquad (29)$$

therefore, based on equation (29)), we can re-write Eq. (17) as:

$$a_{(p)} = \sum_{m=0}^{r-1} x_{(Vm+p)}(-1)^{mv} = \left(x_{(p)} + (-j)^{v} x_{(V+p)} + (-j)^{2v} x_{(2V+p)} + (-j)^{3v} x_{(3V+p)}\right). \qquad (30)$$

and the radix-4 JM first order filter would be:

$$y_{(p,(qV+v))} = w_N^{-(qV+v)} y_{(p-1,(qV+v))} + \left(x_{(p)} + (-j)^{v} x_{(V+p)} + (-j)^{2v} x_{(2V+p)} + (-j)^{3v} x_{(3V+p)}\right), \qquad (31)$$

where the k^{th} computed frequency according to equation (20) is given by as shown in Figure 12

$$\mathbf{X}_{(k)} = (-j)^{(qV+v)} w_N^{-(qV+v)} y_{(V-1,(qV+v))}. \qquad (32)$$

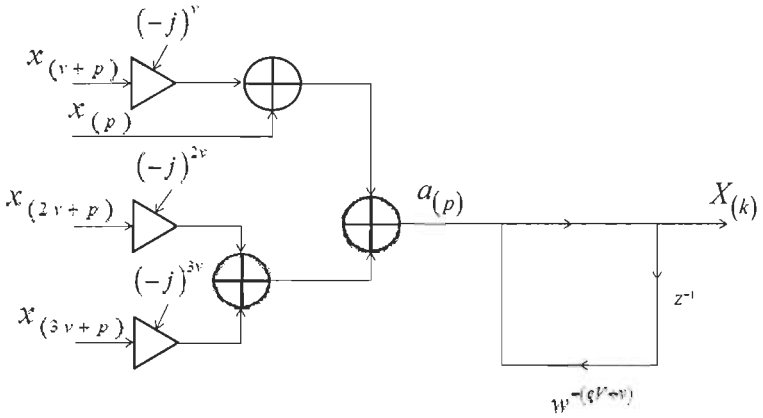


Figure 12: The radix-4 first order JM-filter.

The radix-4 second-order JM-filter will be (Figure 13):

$$y_{(p,\{qV+v\})} = 2\cos(2\pi k/N)\,y_{(p-1,\{qV+v\})} - y_{(p-2,\{qV+v\})} + \left(x_{(p)} + (-j)^{v} x_{(V+p)} + (-j)^{2v} x_{(2V+p)} + (-j)^{3v} x_{(3V+p)}\right), \quad (33)$$

from which the k^{th} computed frequency is

$$X_{(k)} = (-j)^{\{qV+v\}}\left(0.5\cos(2\pi k/N)\,y_{(V-1,\{qV+v\})} + \sin(2\pi k/N) - y_{(V-2,\{qV+v\})}\right), \quad (34)$$

where $y_{(-2,\{qV+v\})} = y_{(-1,\{qV+v\})} = 0$, with $k = qV + v$ and k=0,1,…,$N-1$.

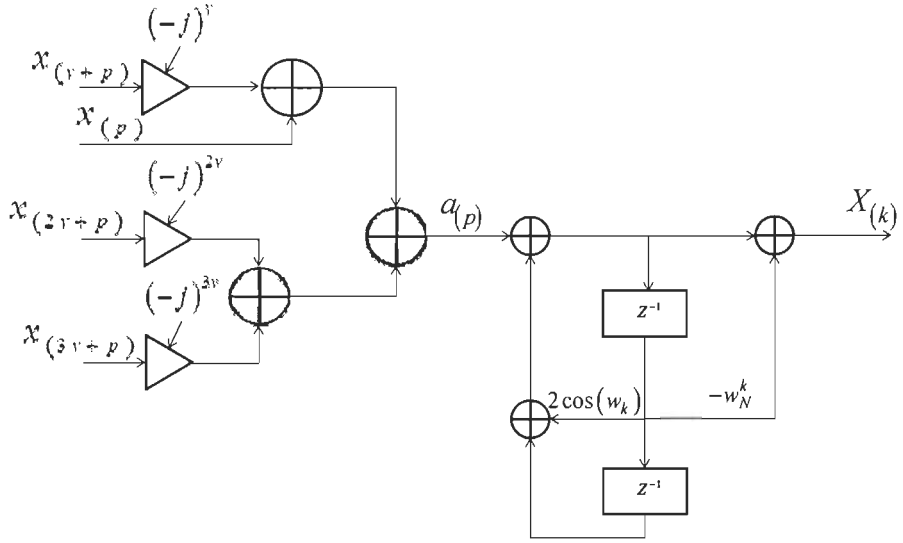


Figure 13: The radix-4 second order JM-filter.

6. Performance Results and Signal to Quantization Noise Ratio (SQNR)

6.1 Performance results

As detailed in Section 2, the computational complexity of the first order Goertzel algorithm in the case of complex-valued input sequences is:

$$4N \text{ real multiplications and } 4N \text{ real additions,} \tag{35}$$

and from Fig. 4, the computational cost of the second-order Goertzel's algorithm is thus [14]:

$$2N + 2 \text{ real multiplications and } 4N - 2 \text{ real additions,} \tag{36}$$

which gives a reduction of almost a factor of two in the number of real multiplications and if the data are real-valued, this cost is almost halved again.

In general for the radix-r case, the computational complexities of the first and second order radix-r JM-filters are respectively:

$$4N/r + N_{a_{(p)}}^{MULT} \text{ real multiplications and } 4N/r + N_{a_{(p)}}^{ADD} \text{ real additions,} \tag{37}$$

$$2N/r + 2 + N_{a_{(p)}}^{MULT} \text{ real multiplications and } 4N/r - 2 + N_{a_{(p)}}^{ADD} \text{ real additions,} \tag{38}$$

where $N_{a_{(p)}}^{ADD}$ and $N_{a_{(p)}}^{MULT}$ are the total number of the required operations required to compute $a_{(p)}$. As a result and according to figures 10 and 11, the computational complexity of the first and second order radix-2 JM-filter, including $N_{a_{(p)}}^{MULT}$ and $N_{a_{(p)}}^{ADD}$, is respectively:

$$2N \text{ real multiplications and } 3N \text{ real additions,} \tag{39}$$

$$N + 2 \text{ real multiplications and } 3N - 2 \text{ real additions,} \tag{40}$$

which give us a reduction in the multiplications' computational cost by a factor of 2 where $N_{a_{(p)}}^{MULT} = 0$ and we need $3N$ real additions compared to $4N$ real additions for Goertzel as shown in table 1.

According to figures 12 and 13, the computational complexity of the first and second order radix-4 JM-filter is respectively:

$$N \text{ real multiplications and } 5N/2 \text{ real additions,} \tag{41}$$

$$N/2 + 2 \text{ real multiplications and } 5N/2 - 2 \text{ real additions,} \tag{42}$$

which give us a reduction in the multiplications' computational cost by a factor of 4 where $N_{a_{(p)}}^{MULT} = 0$ and we need $5N/2$ real additions compared to $4N$-2 real additions for Goertzel. A summary for complexity analysis is shown in Table 1 and Table 2 for the first and second order, respectively.

Table 1: Computational complexity in terms of real arithmetic operation of the proposed FIRST order radix-2/4 JM-Filters and first order Goertzel filter for different sizes of complex input N.

Operation	Goertzel	Radix-2 JM filter	Radix-4 JM filter
Mult	$2N+2$	$2N$	N
Add	$4N$	$3N$	$5N/2$

Table 2: Computational complexity in terms of real arithmetic operation of the proposed SECOND order radix-2/4 JM-Filters and second order Goertzel filter for different sizes of complex input N.

Operation	Goertzel	Radix-2 JM filter	Radix-4 JM filter
Mult	$4N$	$N+2$	$N/2+2$
Add	$4N$-2	$3N$-2	$5N/2$-2

6.2 Signal to Quantization Noise Ratio (SQNR)

Goertzel's algorithm is the most powerful algorithm that is used in the detection of specific frequency in monitored signal's applications. Their implementation is very attractive in fixed point due to the reduction in cost compared to the floating point implementation. In digital processing, signal-to-quantization noise ratio, often written SQNR, is a measure of signal strength relative to background noise. The ratio is usually measured in decibels (dB).The higher the ratio, the less obtrusive the background noise is.

Two major concerns about the computation of the Goertzel's algorithm which are the speed and the high SQNR. The fixed point implementation generates noise sources due to the bit representation in hardware implementation that propagate through the system which will modify the overall system accuracy.

According to [16] in which it was cited that the first order Goertzel's algorithm performs better than the second order in fixed point implementation of real-valued input sequences by using the scaling factor $1/N$ for the first order and $1/N^2$ for the second order. Meanwhile, the proposed scaling factor in [17], based on complex-valued input sequences, for the proposed first order a scaling factor of π / ($4N$). Based on this proposed scaling factor, [17] assures no significant error for all frequency. This is not the case for the scaling factor proposed in [16] currently used in practice since long time where [17] had shown that one frequency output is affected by the factor 1/N. Therefore, our comparative study will be based on the cited reference [16] since it performs better than [17] as shown in Figure 14 where we will be considering complex valued input data that has been quantized to 16 and 24 bits width and coefficient multiplier that has been quantized to 16 bits width in order to reduce the implementation cost.

Figure 15 shows the SQNR comparison between the proposed radix-2 and radix-4 JM-filter and the first order Goertzel's algorithm that is based on the method cited in [16] which reveals a significant gain up to 4.5 dB as illustrated in Figure 16 for a complex input data of size 256.

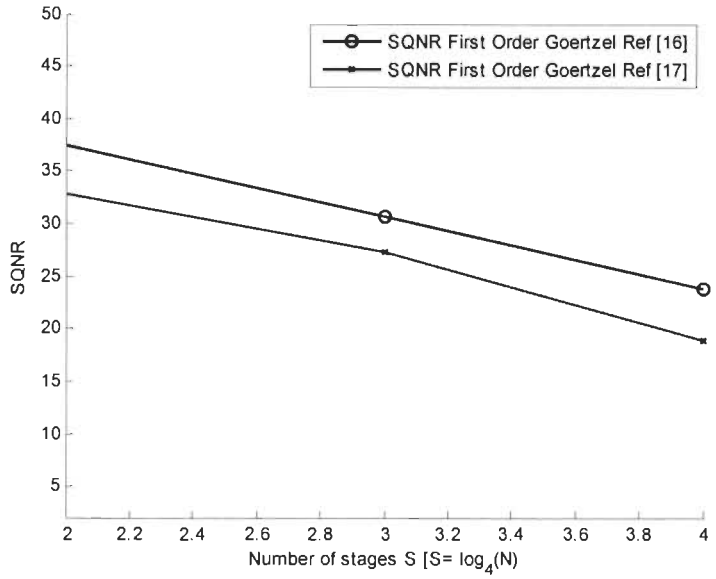


Figure 14: SQNR Comparison between the cited method in [16] with a scaling factor $1/N$ and the cited method in [17] with a scaling factor $\pi/(4N)$ where the data and twiddle factor are quantized to 16 bits width.

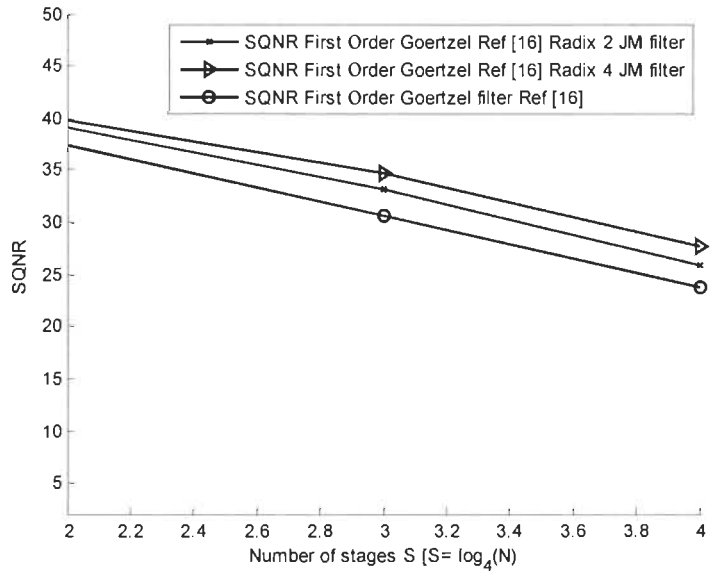


Figure 15: SQNR Comparison between the proposed first order radix-2/4 and the first order Goertzel's algorithm on a data and twiddle factor of 16 bits width where the Scaling factor for all method is $1/N$.

Figure 17 shows the SQNR comparison between the proposed radix-2/4 JM-filter and the first order Goertzel's algorithm on a data of 24 bits width and twiddle factor of 16 bits

which reveals a significant gain up to 6.5 dB as illustrated in Figure 18 for a complex input data of size 256.

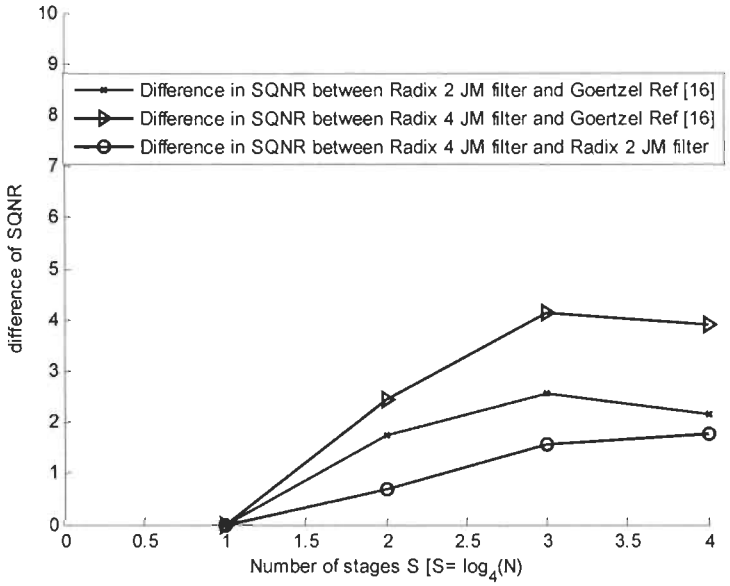


Figure 16: Difference in SQNR between the proposed first order radix-2/4 and the first order Goertzel's algorithm on a data and twiddle factor of 16 bits width the Scaling factor for all method is $1/N$.

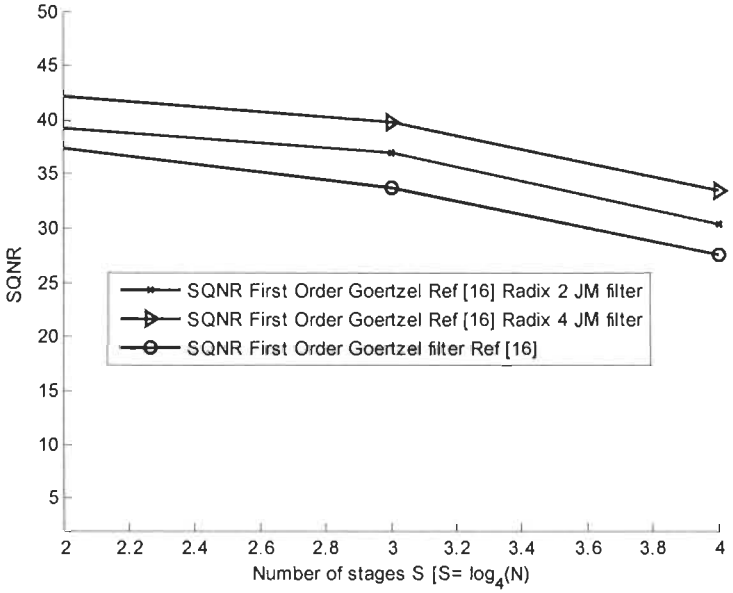


Figure 17: SQNR Comparison between the proposed first order radix-2/4 and the first order Goertzel's algorithm on a data of 24 bits width and twiddle factor of 16 bits width where the Scaling factor for all method is $1/N$.

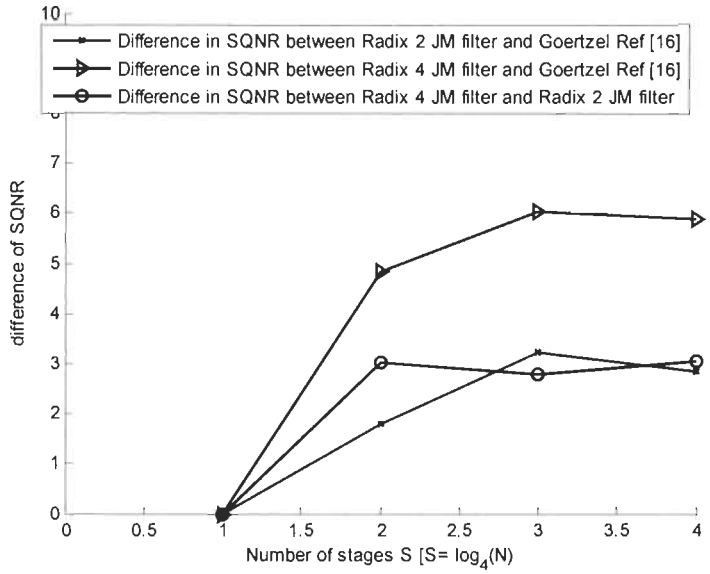


Figure 18: Difference in SQNR between the proposed first order radix-2/4 and the first order Goertzel's algorithm on a data of 24 bits width and twiddle factor of 16 bits width where the Scaling factor for all method is 1/N.

7. Conclusion

Finally, this paper has presented an efficient algorithm to compute a specific frequency compared to the well-known Goertzel's algorithm in which we have proven a reduction in the multiplication computational load by a factor of r and a significant gain in SQNR. The significant gain in the SQNR is due to the fact that the recursive equation of Goertzel's algorithm has been reduced from N to N/r where r is the radix of the introduced JM Filter.

References

[1] B Corbett and M. Woodman, "Radio frequency identification tags", US Patent 7338497, June 17, 2008.

[2] C J Chen, "Modified Goertzel Algorithm in DTMF Detection Using the TMS320C80", Texas Instruments SPRA006, June 1996.

[3] G Goertzel, "An Algorithm for the Evaluation of Finite Trigonometric Series", *American Mathematical Monthly*, pp. 34-35, 1958.

[4] A. Oppenheim, R. Schafer, and J. Buck "Discrete-Time Signal Processing", 2nd Edition

[6] http://www.mstarlabs.com/dsp/goertzel/goertzel.html.

[7] http://cnx.org/content/m17369/latest/

[8] D. Jones, "Goertzel's Algorithm," Connexions Website, September 12, 2006, http://cnx.org/content/m12024/1.5/.

[9] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", *16th International Conference on Digital Signal Processing* (DSP'09), Santorini, Greece, pp. 1-6, 5-7 July 2009.

[10] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing", *16th International Conference on Digital Signal Processing* (DSP'09), Santorini, Greece, pp. 1-5, 5-7 July 2009.

[11] M. Jaber and D. Massicotte, "The Radix-r One Stage FFT Kernel Computation" *Int. Conf. Acoustic, Speech, and Signal Processing*, April First, Las Vegas, Nevada USA, pp. 3585-3588, 2008.

[12] M. Jaber, D. Massicotte, "Fast Method to Detect Specific Frequencies in Monitored Signal", *International Symposium on Communications, Control and Signal Processing* (ISCCSP), Cyprus, pp. 1-5, March 2010.

[13] M. Jaber and D. Massicotte, "A Novel Approach for FFT Data Reordering", *International Symposium on Circuits and Systems* (ISCAS), Paris, pp. 1615-1618, May 2010.

[14] D. Jones, "Goertzel's Algorithm," Connexions Website, September 12, 2006, http://cnx.org/content/m12024/1.5/.

[15] W. Chang and T. Nguyen, "On The Fixed-Point Accuracy Analysis of FFT Algorithms", *IEEE Transactions On Signal Processing*, Vol. 56, No. 10, pp. 4673-4682, October 2008.

[16] J.-Angelo Beraldin and W. Steenaart, "Overflow Analysis of a Fixed-point Implementation of the Goertzel Algorithm", *IEEE Transactions On Circuits And Systems*, Vol 3.6 , No. 2, pp. 322-324, February 1989.

[17] Modesto Medina-Melendrez, Miguel Arias-Estrada and Albertina Castro, "Overflow Analysis in the Fixed-Point Implementation of the First-Order Goertzel Algorithm for Complex-Valued Input Sequences", *IEEE International Midwest Symposium on Circuits and Systems* (MWSCAS) , pp. 620-623, 2009.

Chapter 5 The JMFFT Core Kernel Computation

The JMFFT Core Kernel Computation

Paper VIII: M. Jaber and D. Massicotte, "The Self-Sorting JMFFT Algorithm Eliminating Trivial Multiplication and Suitable for Embedded DSP Processor", accepted in *NEWCAS*, Montreal Canada, June 2012.

Paper IX: M. Jaber and D. Massicotte, "A Novel Radix-2^3 JMFFT Suitable for Embedded DSP Processor", *to be submitted to a Journal after the end of the confidentiality.*

Résumé du Chapitre 5

Une des stratégies récentes visant à réduire la complexité de la transformée rapide de Fourier (TRF) est de cibler la multiplication triviale en regroupant les données avec ses coefficients multiplicateurs correspondants en se servant de nos générateurs d'adresses proposés pour la TRF. En agissant ainsi toutes les multiplications triviales par ± 1 ou $\pm j$ ont été exclus du processus et en ajoutant à cela les accès à ces coefficients multiplicateurs ont été également exclus. Notre méthode proposée dans ce chapitre, qui se repose sur la même stratégie, permettra de réduire davantage les accès mémoire en prédisant l'apparition des coefficients multiplicateurs $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$. En obtenant ceci, le nombre d'opérations arithmétiques pour la multiplication par $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ peut être réduite de 6 à 2 et les accès à ces coefficients multiplicateurs ont été également exclus. L'architecture de l'algorithme obtenue nous a permis de définir un noyau de calcul à base 2^3 (papillon) dont la structure est basée sur 4 noyaux de calcul à base 2. L'avantage de cette structure permet au noyau d'accéder 8 entrées simultanées avec un seul accès au coefficient multiplicateur ce qui est traduit par une énorme réduction d'accès mémoires d'une part et d'autre part la quantité mémoire requise pour stocker les coefficients multiplicateurs est réduite de $N/2 - 1$ jusqu'à $N/8 - 1$. Comme l'accès mémoires est très coûteux en cycle sur une carte DSP ce qui entraîne une réduction en cycle dans l'exécution de la TRF.

Paper VIII: M. Jaber and D. Massicotte, "The Self-Sorting JMFFT Algorithm Eliminating Trivial Multiplication and Suitable for Embedded DSP Processor", accepted in *NEWCAS*, Montreal Canada, June 2012.

The Self-Sorting JMFFT Algorithm Eliminating Trivial Multiplication and Suitable for Embedded DSP Processor

Marwan A. Jaber and Daniel Massicotte

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratory of Signal and System Integration
{marwan.jaber, daniel.massicotte}@uqtr.ca

Abstract: **The Discrete Fourier Transform (DFT) is a mathematical procedure that stands at the center of the processing that takes place inside a Digital Signal Processor. Speed and low complexity are crucial in the FFT process; they can be achieved by avoiding trivial multiplications through a proper handling of the input/output data and the twiddle factors. Accordingly, this paper presents an innovative approach for handling the input/output data efficiently by avoiding trivial multiplications. This approach consists of a simple mapping of the three indices (FFT stage, butterfly and element) to the addresses of the input/output data with their corresponding coefficient multiplier. A self-sorting algorithm that reduces the amount of memory accesses to the coefficient multipliers' memory can also reduce the computational load by avoiding all trivial multiplications. Compared with the most-recent work [5], performance evaluation in terms of the number of cycles on the general-purpose TMS320C6416 DSP shows a reduction of 29% (FFT of size 4096) and 87.5% memory reduction to stock the twiddle factors. The algorithm has also shown a speed gain of 24% on the FFTW platform for a FFT of size 4096.**

1. Introduction

DSPs are typically used to input large amounts of data; perform mathematical transformations on that data and output the results at very high rates. In a real time system,

data flow must be understood and controlled in order to achieve high performance. Such peripheral devices that control data transfers between I/O (Input Output) subsystems in the same manner as a processor can reduce CP (Computational Processor) interrupt latencies and leave valuable DSP cycles free for other tasks leading to increased performance. Handling input/output data in a FFT process is based on the bit reversing technique [1], and in order to reduce the computational load by avoiding trivial multiplication, several well-structured C library codelets known as planners (i.e. FFTW) adopt the algorithm to the available hardware by maximizing its performance [2]. This paper presents a data mapper based on the concept introduced in [3] - [4], which will be compared with the latest work done on the implementation of the FFT on embedded processors [5].

The paper is organized as follows: Section 2 describes the proposed method; Section 3 provides the performance results of that method on the TI's DSP TMS320C6416 in terms of clock cycles and on FFTW platform. Finally, Section 4 discusses the conclusion.

2. Proposed Method

The basis of the radix-r FFT is that a DFT can be divided into r smaller DFTs, each of which is divided into r smaller DFTs, in a continuing process that results in a combination of r point DFTs. The advantage of properly dividing the DFT into partial DFTs is to control the number of multiplications and stages. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation; a reduced number of stages is therefore beneficial. The conceptual key to the use of our proposed FFT Address Generator is the simple mapping of the three indices (FFT stage, butterfly, and element) to the addresses of the multiplier coefficients needed [4].

The definition of the DFT is represented by the following equation:

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \ , \quad k \in [0, N-1] \tag{1}$$

where $x_{[n]}$ is the input sequence, $X_{[k]}$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ called the twiddle factor in butterfly structure, and $j^2 = -1$. Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex number sequences.

Equation (1) can be expressed in compact form as [4]:

$$\mathbf{X} = \mathbf{T}_r \mathbf{W}_N col\left(\sum_{n=0}^{\frac{N}{r}-1} x_{(rn+q)} w_{N/r}^{np} \middle| q = 0,1,\cdots,r-1 \right), \tag{2}$$

for $k = 0,1,2,\cdots,N-1$, $p = 0,1,2,....,(N/r)-1$ and $q = 0,1,2,....,r-1$, with

$$\mathbf{X} = \left[X_{(p)},\ X_{(p+N/r)}, X_{(p+2N/r)},\ \ldots,\ X_{(p+(r-1)N/r)} \right]^T, \tag{3}$$

$$\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p} \right), \tag{4}$$

and

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix}. \tag{5}$$

Therefore, by defining $\left[\mathbf{T}_r\right]_{l,m}$ as the element of the l^{th} line and the m^{th} column in the matrix $\mathbf{T}_r$, equation (5) could be rewritten as:

$$\left[\mathbf{T}_r\right]_{l,m} = w_N^{\left[(lmN/r)\right]_N}, \tag{6}$$

where l=0,1,…,r-1, m=0,1,…,r-1 and $[\![\mathrm{x}]\!]_N$ represents the operation x modulo N.

The set of the twiddle factor matrix $\mathbf{W}_{N(m,v,s)}$ is defined as

$$[\mathbf{W}_N]_{l,m(v,s)} = \mathrm{diag}\left(w_{N(0,v,s)}, w_{N(1,v,s)}, \ldots, w_{N(r-1,v,s)}\right), \tag{7}$$

where the indices r is the FFT's radix, $v = 0,1,\ldots,V-1$ represents the number of words of size r ($V = N/r$) and $s = 0,1,\ldots,S$ is the number of stages (or iterations $S = \log_r N - 1$). Finally, Eq. (7) can be expressed for the different stages in an FFT process as [3] and [4]:

$$[\mathbf{W}_N]_{l,m(v,s)} = \begin{cases} w_N^{\left[\!\left[\lfloor v/r^s \rfloor\, l\, r^s\right]\!\right]_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}, \tag{8}$$

for the DIF process and Eq. 7 would be expressed as:

$$[\mathbf{W}_N]_{l,m(v,s)} = \begin{cases} w_N^{\left[\!\left[\lfloor v/r^{(S-s)} \rfloor\, l\, r^{(S-s)}\right]\!\right]_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}, \tag{9}$$

for the DIT Process, where l=0,1,…,r – 1 is the l^{th} butterfly's output, m=0,1,…,r-1 is the m^{th} butterfly's input and $\lfloor \mathrm{x} \rfloor$ represents the integer part operator of x.

As a result, the l^{th} transform output during each stage can be illustrated as:

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[\!\left[lmN/r + \lfloor v/r^s \rfloor\, l\, r^s\right]\!\right]_N}, \tag{10}$$

for the DIF process and

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[\!\left[lmN/r + \lfloor v/r^{(S-s)} \rfloor\, mr^{(S-s)}\right]\!\right]_N}, \tag{11}$$

for the DIT process.

In this article we will be adopting respectively the following abbreviation (RAG) for the reading address generator, (WAG) for the writing address generator and (CAG) for the coefficient address generator for DIF and DIT process.

The m^{th} butterfly's input of v^{th} word $x_{(m)}$ at the s^{th} stage (s^{th} iteration) is fed by equations (12) and 13 for the DIF process and by equation (13) for the DIT process illustrated as [4]:

$$RAG_{(m,v,0)} = m \times \frac{N}{r} + v \tag{12}$$

and for s>0

$$RAG_{(m,v,s)} = m \times \frac{N}{r^2} + \left[\left[\left\lfloor \frac{v}{r^{(s-1)}} \right\rfloor \times \frac{N}{r} \right]\right]_N + \left[\left[k \right]\right]_{r^{(s-1)}} + \left\lfloor \frac{v}{r^s} \right\rfloor \times r^{(s-1)} \tag{13}$$

for the DIF process and

$$\mathrm{RAG}_{(m,v,s)} = m \times \left(\frac{N}{r^{(s+1)}} \right) + \left[\left[v \right]\right]_{r^{(S-s)}} + \left\lfloor \frac{v}{r^{(S-s)}} \right\rfloor r^{(S+1-s)}, \tag{14}$$

for the DIT process where the butterfly's input $m = 0,1,\ldots,r-1$, $v = 0,1,\ldots,V-1$ and $s = 0,1,\ldots,S$, $S = \log_r N - 1$.

For both cases, the l^{th} processed butterfly's output $X_{(l,v,s)}$ for the v^{th} word at the s^{th} stage should be stored into the memory address location given by the WAG:

$$WAG_{(l,v,s)} = l\left(N / r\right) + v. \tag{15}$$

It should be noted that for both algorithms the input and output data are in natural order during each stage of the FFT process known as the Ordered Input Ordered Output (OIOO) algorithms.

The coefficients multipliers (Twiddle Factors) needed during each stage, which are fed to the m^{th} butterfly's input of v^{th} word $x_{(m)}$ at the s^{th} stage (s^{th} iteration), are provided by:

$$CAG_{(m,v,s)} = \left[\left[l \times \left(mV + \left\lfloor \frac{v}{r^s} \right\rfloor r^s \right) \right]\right]_N, \tag{16}$$

for the DIF process and

$$CAG_{(m,v,s)} = \left[\left[m \times \left(lV + \left\lfloor \frac{v}{r^{(S-s)}} \right\rfloor \times r^{(S-s)} \right) \right]\right]_N \qquad (17)$$

for the DIT process.

By examining equations (16) and (17) we can clearly see that the data in both algorithms were grouped with their corresponding coefficient multipliers at each stage because the m^{th} coefficient multiplier of the l^{th} butterfly's output shifts if and only if v ($v = 0,1,\ldots,V-1$) is equal to $r^{(S-s)}$ in the DIF process or $v = r^s$ in the DIT process. As a result and since $V=N/r = r^S$; the total number of shifts during each stage in the DIT process would be r^s and the total number of shifts during each stage in the DIF process is $r^{(S-s)}$. Therefore, by implementing the word counter $r^{(S-s)}$ (*wordcounter* = 0, 1, …, $r^{(S-s)}$-1) and the shifting counter r^s (*shiftcounter* = 0,1,…, $r^s - 1$) in the DIT process or the word counter r^s and the shifting counter $r^{(S-s)}$ in the DIF process, we obtain highly efficient self-sorting DIT/DIF radix-r algorithms in which the access to the coefficient multiplier's memory is reduced compared with the conventional radix-r DIT/DIF algorithms.

As well, the occurrence of the multiplication by one (i.e., the elements of the twiddle factor matrix illustrated in (4) are all equal to one) can be easily predicted when the shifting counter in both cases is equal to zero (i.e. $v < r^s$ or $v < r^{(S-s)}$). The trivial multiplication by one (w^0) during the entire FFT process is consequently avoided. Furthermore, by manipulating the exponent of the twiddle factor, the occurrence of the multiplication by $\pm j$ can be predicted where this multiplication can be incorporated into the additions by switching the real and imaginary parts of the data. The trivial multiplication $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ can also be predicted where the number of arithmetical operation of this multiplication can be reduced from 6 to 2.

Compared with the cited reference [5] (referred in this paper as REF) in which a combination of 2 butterflies that requires one memory access per 4 complex inputs (Fig. 1); the comparison was initially made in terms of memory accesses (the most costly in DSP implementation) to the coefficient multiplier as shown in Table I.

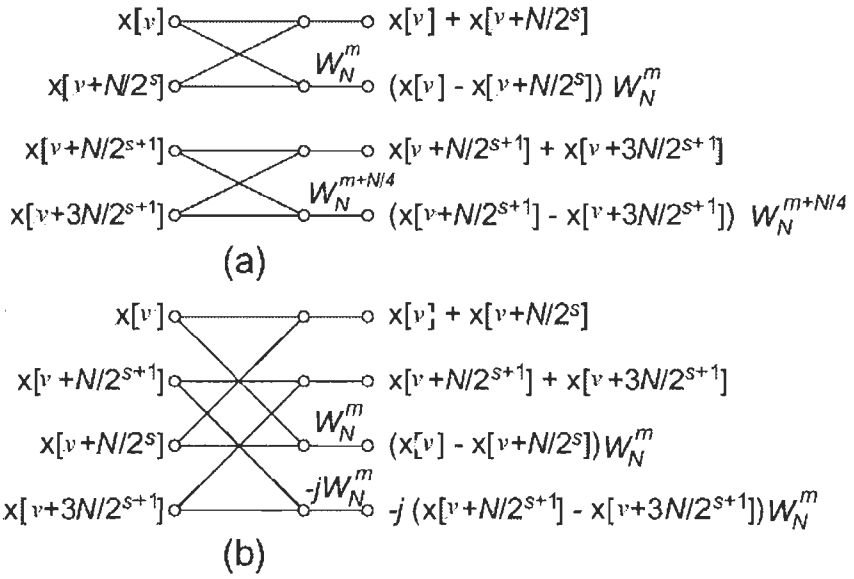


Fig. 1: Computing two butterflies together in one stage of the radix-2 DIF FFT diagram [5] where m is given in equation (16).

Table 1: Comparison in terms of memory accesses to the coefficient multiplier in [5] versus the proposed method where each complex access is counted as 1:

				Memory accesses reduction (%)	
N	TI [5]	Cited [5]	Proposed	TI	Cited
8	7	1	0	100	100
16	15	5	2	86.7	60
32	31	15	8	74.2	46.7
64	63	37	22	65.1	40.5
128	127	83	52	59.1	37.35
256	255	177	114	55.3	35.6
512	511	367	240	53.1	34.7
1024	1023	749	494	51.7	34.1
2048	2047	1515	1004	49.1	33.7

Table 2: Comparison in terms of real multiplication between the cited methods in [5] versus the proposed method

				Multiplication reduction (%)	
N	TI [5]	Cited [5]	Proposed	TI	Cited
8	48	8	4	91.7	50
16	128	40	28	78.1	30
32	320	136	108	66.25	20.5
64	768	392	332	56.77	15.3
128	1792	1032	908	49.33	12.01
256	4096	2568	2316	43.45	9.8
512	9216	6152	5644	38.75	8.25
1024	20480	14344	13324	34.94	7.11
2048	45056	32776	30732	31.79	6.23

Table 3: Comparison in terms of real addition between the cited methods in [5] versus the proposed method

				Addition reduction (%)	
N	TI [5]	Cited [5]	Proposed	TI	Cited
8	72	52	48	33.34	7.69
16	192	148	136	29.16	8.1
32	480	388	360	25	7.21
64	1152	964	904	21.52	6.22
128	2688	2308	2184	18.75	5.37
256	6144	5380	5128	16.53	4.68
512	13824	12292	11784	14.75	4.13
1024	30720	27652	26632	13.30	3.68
2048	67584	61444	59400	12.10	3.32

Furthermore, the prediction of the multiplication by $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ is also beneficial when the number of real value arithmetical operations can be reduced from 6 to 2 as shown in Tables II and III, in which the split radix algorithm was excluded from this comparison since it has the lowest number of real value arithmetical operations [6] and [7].

3. Performance Evaluation

In our comparative study we will be testing the proposed DIT FFT structure on a general purpose DSP and FFTW platforms.

A) DSP Platform

This comparison was conducted on the TMS320C6416 DSP platform using TI's Code Composer Studio (CCS version 4.2), where the beauty of this platform lies in its capacity for executing 8 instructions in parallel; this is highly desirable for our proposed structure, which maximizes the use of the platform's resources. We compared the performance of our proposed method with the TI's DIF radix-2 FFT referred as "TI" [8], and with the best DIT method (to our knowledge) referred as "REF" [5].

In our performance study, the simulation results of the referenced methods are obtained in bit reverse order, whereas our obtained results are in natural order where the bit reversing process was not taken into consideration in the simulation results.

The simulation environment for all methods is detailed as follows:

- Clock 1000 MHz.
- Memory clock 100 MHz.
- Mode Release
- C6416 Device Cycle Accurate Simulator, little Endian.

Table 4 and Fig. 2 reveal the simulation results of the TI and REF methods cited in [5] where the performance evaluation was based on the number of clock cycles. The simulation was tested on the general-purpose DSP that shows a significant reduction of 15% to 29% for 64 and 1024 points, respectively.

Table 5 represents the required size (in bytes) of the coefficient multiplier for all methods where our proposed method demonstrates further reduction of this memory size.

Table 4 Comparative results in term of clock cycle of the cited methods versus the proposed method for different FFT sizes.

				Cycle Reductions (%)	
Length	TI	REF	Proposed	TI	REF
64	5252	4210	3648	43,97	15,41
128	11363	9048	7612	49,28	18,86
256	24578	19246	15832	55,24	21,56
512	53025	40676	32852	61,41	23,82
1024	113984	85594	68048	67,51	25,78
2048	244063	179536	140748	73,40	27,56
4096	520574	375622	290760	79,04	29,19

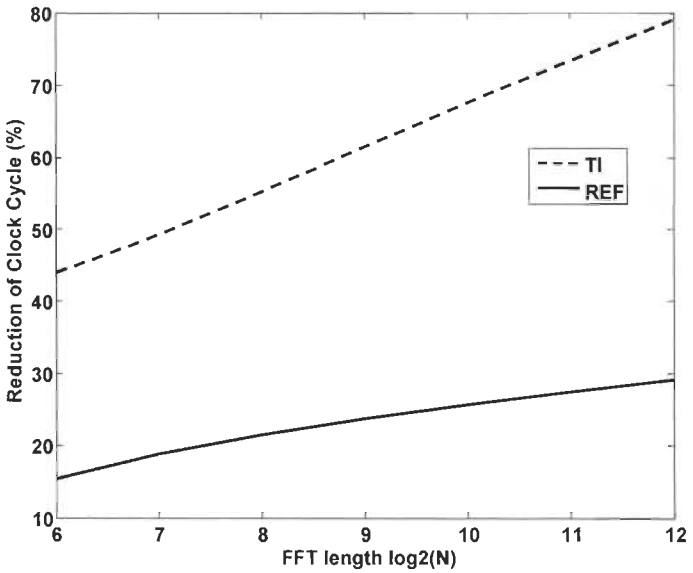


Fig. 2: Comparison of clock cycle reduction between our proposed method and the Referenced methods (TI and Cited).

Table 5: Comparison of the coefficients multiplier's memory requirement of the cited methods versus the proposed method where the size is computed in term of byte

FFT Length	TI [5]	REF [5]	Proposed
N	$2N$	$N/2-2$	$N/8-1$

B) FFTW Platform

Furthermore, the proposed DIT FFT method was benchmarked on the FFTW platform (version 2.0) as shown in Fig. 3 in which our proposed structure revealed that the execution of the FFT was greatly accelerated. This gain is occurred because the proposed structure first reduced the amount of the coefficient multiplier's memory accesses, (which is costly in DSP implementation) and then, reduced the computational load achieved by eliminating the multiplication by $\pm j$. The FFTW benchmark [9] is an FFT bench platform assembled by Matteo Frigo and Steven G. Johnson at MIT (Massachusetts Institute of Technology). This platform compares the performance of different complex FFT implementations (40 FFT methods) based on speed and accuracy where the performance is computed on a single processor environment even though the benchmark is run on multi-processor systems [9] and [10]. The FFTW platform includes an FFT method known as FFTW_ESTIMATE that outperform all other methods and is actually used in Matlab® software R2009a.

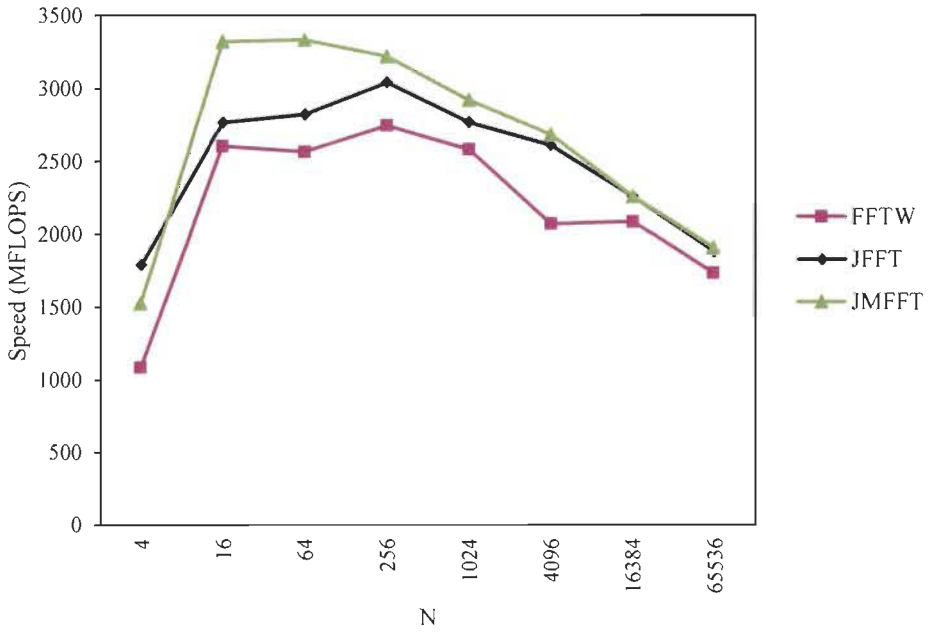


Figure3: FFTW benchmark results of the proposed method for the radix-4.

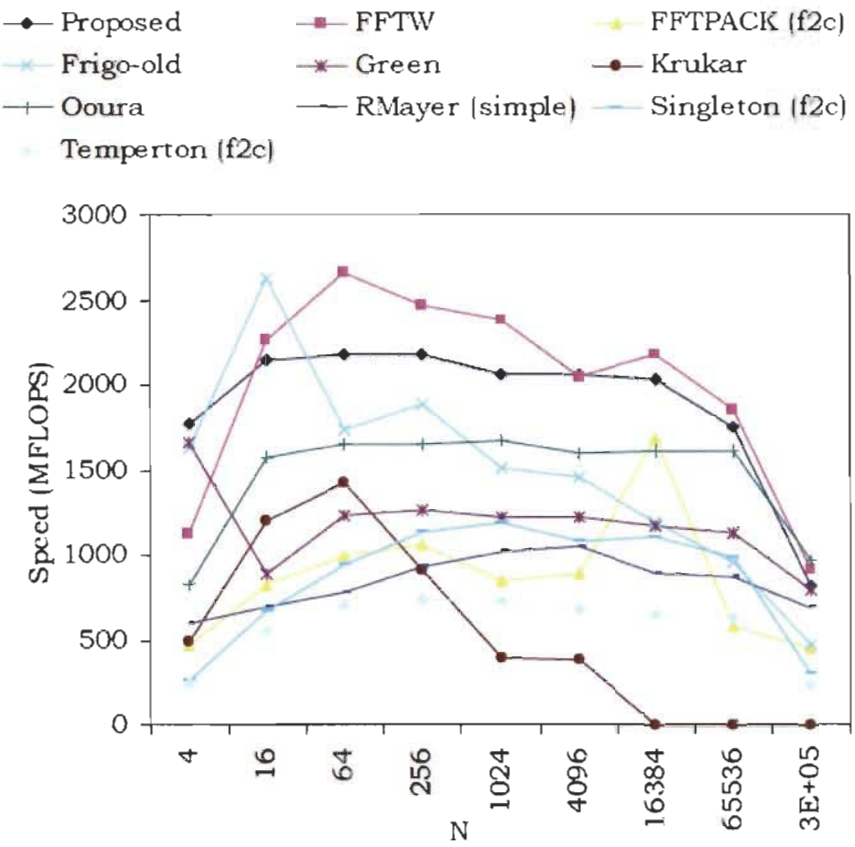


Figure 4: FFTW benchmark results of our previous work (JFFT) for the radix-4 [4].

This bench platform is internationally recognized; the complex FFT performance is plotted in terms of "MFLOPS" (efficiency axis), and the FFT size N, which is a scaled version of speed defined by:

$$\text{MFLOPS} = \left(5N \log_2 N\right)/t \ , \tag{18}$$

where t is the computational time in μs to execute the N-point FFT [10].

Furthermore, the FFTW benchmark of Fig. 3 shows that the FFT's execution time is significantly improved when our proposed method is implemented on the radix-4 butterfly which manifested a gain of 24% for an FFT size of 4096. The JFFT result in figure 3 is an optimized version of our previously proposed code cited in [4] as shown in Fig. 4. The greatest impact of our proposed method is the reduction in complexity compared to

FFTW_ESTIMATE that requires complex and lengthy codelets plus an exhaustive search which is costly in time in order to achieve the obtained results. For all results, the computational time of the FFTW planner is not included in MFLOPS' computation as shown in Fig. 3 and 4.

4. Conclusion

Finally, this paper has presented an efficient ordered input ordered output algorithm that reduces the complexity and the computational effort in comparison to the most recent proposed methods. Furthermore, the proposed method had showed a significant reduction in term of clock cycles compared to the cited methods in [5]. In addition to that and by predicting the 8th root of unity, the memory size needed to stock the coefficient multiplier is reduced to $N/8 - 1$.

Acknowledgment

The authors wish to thank the Natural Sciences and Engineering Research Council of Canada and Jabertech Canada Inc. for their financial and technical support.

References

[1] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematical Computer 19, pp. 297-301, April 1965.

[2] M. Frigo and S.G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT", IEEE International Conference on Speech, and Signal Processing, Seattle, pp. 1381-1384, 12-15 May 1998.

[3] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", International Conference on Digital Signal Processing, Santorini, Greece, pp. 1-6, 5-7 July 2009.

[4] M. Jaber and D. Massicotte, "A Novel Approach for FFT Data Reordering", International Symposium. on Circuits and Systems, Paris, pp. 1615-1618, May 2010.

[5] Y. Wang and al., "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors", IEEE Transactions on Signal Processing, vol. 55, no. 5, pp. 2338-2349, May 2007.

[6] L. Moreno and al, "Digital Signal Processors for a Signal Processing Laboratory", IEEE Transactions on Education, vol. 42, no. 3, pp. 192-199, August 1999.

[7] I. Uzun, A. Amira, and A. Bouridane, "FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing", IEE Proceeding on Vision, Image and Signal Processing, vol. 152, no. 3, pp. 283-296, June 2005.

[8] "TMS320C64x DSP Library Programmer's Reference", Literature Number: SPRU565B, Oct. 2003, (code DSP-radix-2, p. 4-9,4-10).

[9] M. Frigo and S.G. Johnson, "The Design and Implementation of FFTW3", Proceeding of IEEE, vol. 2, no. 2, Feb. 2005, pp. 216-231.

[10] FFTW, http://www.fftw.org, (visited in 2012).

Paper IX: M. Jaber and D. Massicotte, "A Novel Radix-2^3 JMFFT Suitable for Embedded DSP Processor", *to be submitted to a Journal after the end of the confidentiality.*

A Novel Radix-2^3 JMFFT Suitable for Embedded DSP Processor

Marwan A. Jaber and Daniel Massicotte

Abstract – **Digital Signal Processing is a technique by which analog, or continuously varying, signals are converted to digital form and mathematically manipulated by a specialized computer (a digital signal processor), to modify or improve the signal in some way. Discrete or digitized signals can be transformed into the frequency domain using the discrete Fourier transform where any signal can be analyzed into its frequency components to understand the characteristics of the signal's pattern information and to compare it to other types of signals. Memories' accesses are major concerns in implementation on DSP cards which on the most cases are costly in DSP cycles. Therefore, in a real time implementation, executing and controlling the data flow structure is important in order to achieve high performance that could be obtained by regrouping the data with its corresponding coefficient multiplier. This article will present a novel hardware oriented Radix 2^3 JMFFT (Jaber-Massicotte FFT) which is an alternative way of representing higher radices by mean of less complicated and simple butterflies in which we used the symmetry and periodicity of the root unity to further lower down the coefficient multiplier memories' accesses, since the proposed core requires one memory access per eight inputs compared to the conventional radix-2 FFT. Finally this article will present the performance evaluation of the proposed method on the TMS320C6416 DSP by using the TI's Code Composer Studio (CCS V 4.0) that will be compared to the most recent methods. Fixed-point arithmetic evaluations are done with respect to the two reference methods where we**

have showed an SQNR gain that is maximized for 10-bit twiddle factor and 16-bit for the other variable data in the FFT process.

Key-words: FFT, SQNR, FFT Address Generator, Access Memory Reduction, Trivial Multiplication, DSP).

1. Introduction

One "rediscovery" of the FFT, that of Danielson and Lanczos in 1942, provides one of the clearest derivations of the algorithm [1] as shown in Figure 1. Danielson and Lanczos showed that a discrete Fourier transform could be written as the sum of two discrete Fourier transforms each of length $N/2$. One of the two is formed from the even-numbered points of the original N, the other from the odd-numbered points. An important advance then changed the situation completely: the discovery by Cooley and Tukey of a numerical algorithm, which allows the DFT to be evaluated with a significant reduction in the amount of calculation required. This algorithm, called the Fast Fourier Transform, or FFT, allows the DFT of a sampled signal to be obtained rapidly and efficiently. In essence, Cooley and Tukey realized that the straightforward approach to the DFT had the computer doing the exact same multiplications over and over [2]. The idea behind the FFT is to breakdown the DFT problem into sequences and to organize the computation in a manner, which takes advantage of the algebraic properties of the Fourier matrix. By doing so, they found that they could eliminate almost all of these redundant calculations. The computational saving achieved by the Cooley and Tukey algorithm is staggering which reduces the computational load from N^2 to $N/r \log_r N$. Since the breakthrough by Cooley and Tukey, several other FFT algorithms have been devised such as Common Factor Algorithms (decimation-in-time (DIT) or Cooley-Tukey FFT algorithm [2] and

decimation-in-frequency (DIF) or Sande-Tukey FFT algorithm [3]), Prime Factor Algorithm (PFA) [4], Mixed Radix Algorithm (MRA) [4], Winograd Fourier Transform Algorithm (WFTA) [5] and Split-Radix Algorithm (SRA) [6].

It is hard to make a fair and general comparison between the different algorithms because the importance of different properties of the algorithms is depending on the implementation. In the case of hardware implementation of FFT processors there are number of other algorithm's properties that should be dealt with such as: Regularity, Modularity, Parallelism and simplicity which is mostly offered by the common factor and prime factor algorithms.

One of the recent strategies to reduce the computational load is to target the trivial multiplication that could be achieved by grouping the data with its corresponding coefficients multipliers [7] - [9]. By doing so in our cited references 8, all trivial multiplications by ± 1 or $\pm j$ have been excluded from the process and adding to that the accesses to the coefficient multipliers have been also reduced.

Our proposed method which is based on the same concept, will further reduce the memory accesses and adding to that the multiplication by $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ can be also predicted where the number of arithmetical operation required for the complex multiplication can be reduced from 6 to 2.

The paper is organized as follows; Section 2 provides a brief description on the background of the DIT/DIF FFT algorithms while Section 3 will deeply elaborate the proposed method. Section 4 provides a performance evaluation of the proposed method on the TMS320C6416 DSP by using the TI's Code Compose Studio (CCS V 4.0) meanwhile section 5 will report the conclusions.

2. Background of the DIT/DIF FFT

Fourier analysis is named after Jean Baptiste Joseph Fourier (1768-1830) who was interested in heat propagation, and presented a paper in 1807 to the Institut de France on the use of sinusoids to represent temperature distributions. The paper contained the controversial claim that any continuous periodic signal could be represented as the sum of properly chosen sinusoidal waves. Among the reviewers were two of the history's most famous mathematicians, Joseph Louis Lagrange (1763-1813), and Pierre Simon de Laplace (1749-827). While Laplace and the other reviewers voted to publish the paper, Lagrange adamantly protested. For nearly 50 years, Lagrange has insisted that such an approach could not be used to represent signals with corners, i. e., discontinuous slopes, such as in square waves. The Institut de France bowed to the prestige of Lagrange, and rejected Fourier's work. It was only after Lagrange died that the paper was finally published, some 15 years later [10].

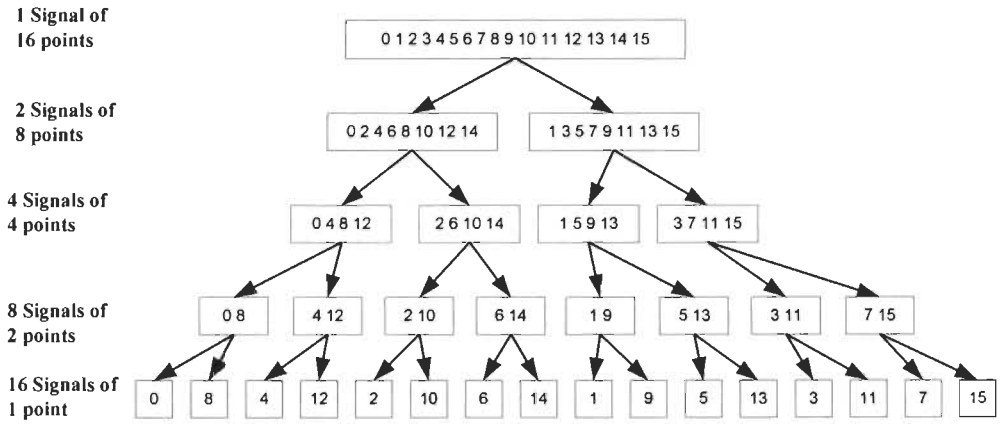


Figure 1: The FFT decomposition. An N point signal is decomposed into N signals each containing a single point. Each stage uses an interlace decomposition, separating the even and odd numbered samples.

The definition of the DFT is represented by the following equation

$$\mathbf{X}_{[k]} = \sum_{n=0}^{N-1} x_{[n]} w_N^{nk} \quad , \quad k \in [0, N-1], \tag{1}$$

where $x_{[n]}$ is the input sequence, $\mathbf{X}_{[k]}$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ is called the twiddle factor in butterfly structure, and $j^2 = -1$. Both $x_{[n]}$ and $\mathbf{X}_{[k]}$ are complex number sequences.

In mid-1960s, J.W. Cooley and J.W. Tukey proposed their first algorithm known as decimation-in-time (DIT) or Cooley-Tukey FFT algorithm, which first rearranges the input elements into bit-reverse order, then builds up the output transform in log2 N iterations [2] and [4]. This process of splitting the 'time domain' sequence into even and odd samples is what gives the algorithm its name, 'Decimation In Time' (DIT). Based on the divide and conquer approach, the input data is subdivided into two sets of even-numbered and odd numbered data. If $N/2$ is even, as it is when N is equal to power of 2, then we can consider computing each of the $N/2$ points DFTs by breaking each of the sums into two $N/4$ points DFTs, which could be then combined to yield the $N/2$ points DFTs. If we proceed by decomposing $N/4$ into $N/8$ points transforms and continue until left with only 2 points transforms, this requires m stages where $m = log_2 N$ Figure 2. Then, the butterfly computation (in place computation Figure 3 or pre-multiplication technique) is used by obtaining a pair of values in one stage from a pair of values in the previous stage Figure 4.

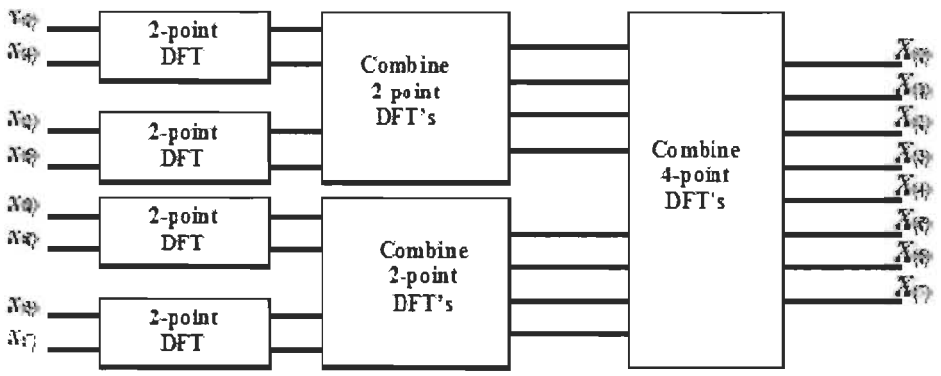


Figure 2: Three stages in the computation of an N = 8-point DIT DFT [16].

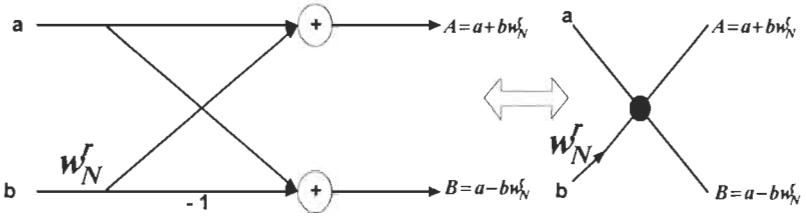


Figure 3: Basic butterfly computation for the DIT FFT algorithm [16].

It is also possible to derive FFT algorithms that first go through a set of $log_2 N$ iterations on the input data, and rearrange the output values into bit-reverse order. These are called decimation-in-frequency (DIF) or Sande-Tukey FFT algorithm [3] and [4]. The process of splitting the 'frequency domain' sequence into even and odd samples is what gives the algorithm its name, 'Decimation In Frequency' (DIF Figure 5). In fact, the output sequences $X_{(k)}$ is decimated (split) into the even- and odd-numbered samples, then, the DIF is obtained by performing the butterfly computation (in place computation or post multiplication technique) of the type shown in Figure 6. In this case the input data is sorted in normal order to provide an output in bit-reversed order Figure 7.

Briefly, the basic operation of a radix-r butterfly in which r inputs are combined to give the r outputs via the operation:

$$\mathbf{X} = \mathbf{B}_r \mathbf{x}, \tag{2}$$

where $x = [x_{(0)}, x_{(1)}, \ldots, x_{(r-1)}]^T$ is the input vector and $\mathbf{X} = [X_{(0)}, X_{(1)}, \ldots, X_{(r-1)}]^T$ is the output vector and T denotes the transpose of the vector.

$\mathbf{B}_r$ is the $r \times r$ butterfly matrix, which can be expressed as:

$$\mathbf{B}_r = \mathbf{W}_N \mathbf{T}_r, \tag{3}$$

for the decimation in frequency (DIF) process, and

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N \tag{4}$$

for the decimation in time (DIT) process where for both cases $\mathbf{W}_N$ is defined as:

$$\mathbf{W}_N = diag\left(w_N^0, w_N^p, w_N^{2p}, \cdots, w_N^{(r-1)p}\right), \tag{5}$$

and

$$\mathbf{T}_r = \begin{bmatrix} w_N^0 & w_N^0 & w_N^0 & \cdots & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & w_N^{2N/r} & \cdots & \cdots & w_N^{(r-1)N/r} \\ w_N^0 & w_N^{2N/r} & w_N^{4N/r} & \vdots & \vdots & w_N^{2(r-1)N/r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_N^0 & w_N^{(r-1)N/r} & w_N^{2(r-1)N/r} & \cdots & \cdots & w_N^{(r-1)^2 N/r} \end{bmatrix}. \tag{6}$$

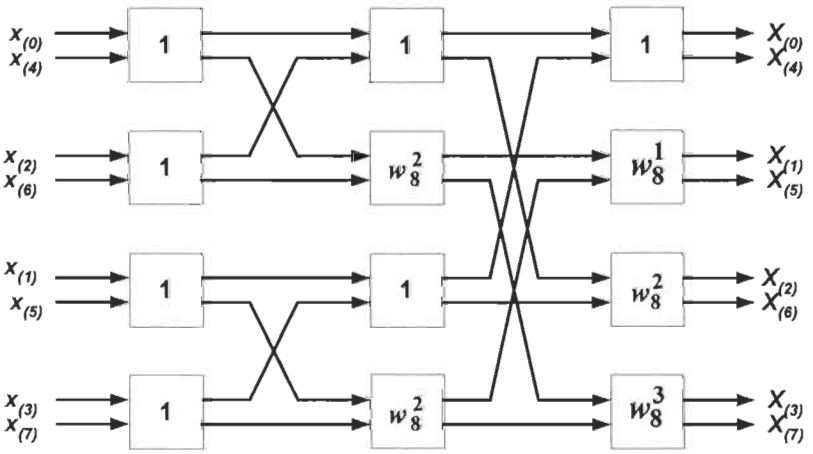


Figure 4: Eight-point DIT FFT Signal Flow Graph (SFG) [16].

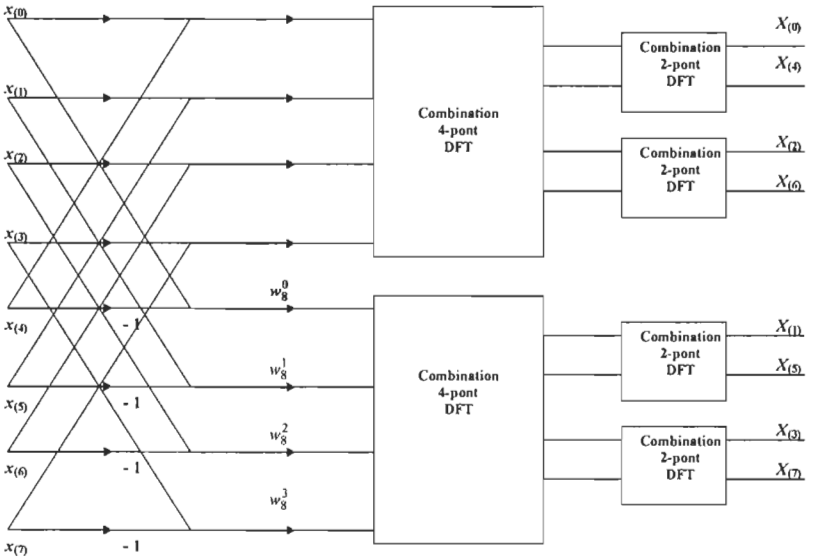


Figure 5: Three stages eight-point DIF FFT algorithm [16].

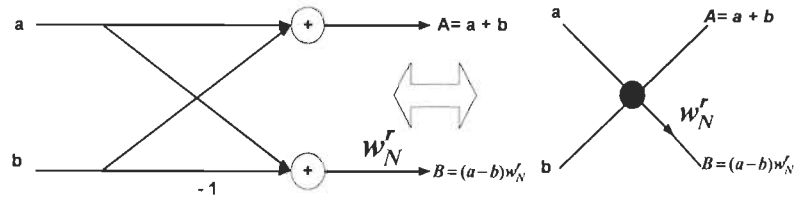


Figure 6: Basic butterfly computation for the DIF FFT algorithm [16].

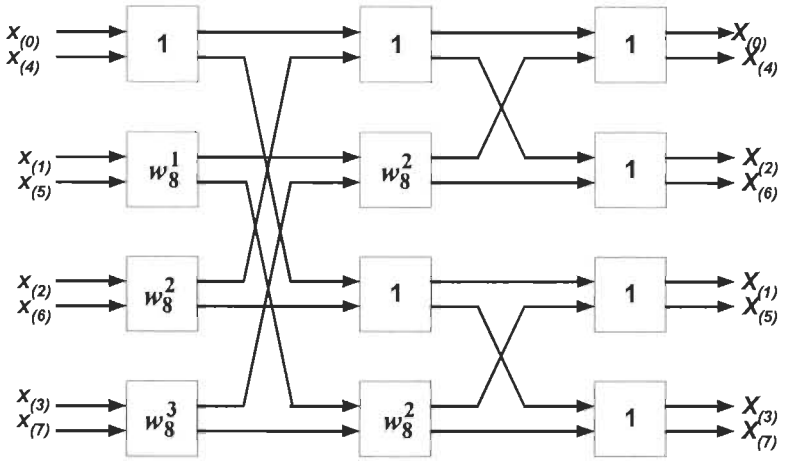


Figure 7: Eight-point DIF FFT Signal Flow Graph (SFG) [16].

3. The Proposed Method JMFFT

One of the bottlenecks in most applications, where high performance is required, is the FFT/IFFT processor. Given that higher radix implementation is attractive for reduction in computation, researchers have sought a higher radix butterfly implementation. Since the higher radix will reduce automatically the communication load, the only problem left, will be the computational load. The most well-known attempt to reduce the computational load is by factoring the adder matrix (or adder tree simplification) which did not provide a complete solution for the FFT problem due to the increasing complexity of the butterflies for higher radices due to the added multipliers in the butterfly's critical path [11] and [12](Figure 8). Thus, if we pay attention to the elements of the adder tree matrix $\mathbf{T}_r$ and to the elements of the twiddle factor matrix, we notice that both of them contain twiddle

factors. So, by controlling the variation of the twiddle factor during the calculation of a complete FFT, we can incorporate the twiddle factors and the adder tree matrices into a single stage of calculation.

Therefore, by defining $[\mathbf{T}_r]_{l,m}$ as the element at the l^{th} line and m^{th} column in the matrix $\mathbf{T}_r$ as a result equation (6) could be rewritten as:

$$[\mathbf{T}_r]_{l,m} = w_N^{[\![(lmN/r)]\!]_N}, \tag{7}$$

with l=0,1,…,r-1, m=0,1,…,r – 1 and $[\![\mathrm{x}]\!]_N$ represents the operation x modulo N and by defining $\mathbf{W}_{N(m,v,s)}$ the set of the twiddle factor matrix as

$$[\mathbf{W}_N]_{l,m(v,s)} = \mathrm{diag}\left(w_{N(0,v,s)}, w_{N(1,v,s)}, \ldots, w_{N(r-1,v,s)}\right), \tag{8}$$

where the indices r is the FFT's radix, $v = 0,1,\ldots,V-1$ represents the number of words of size r ($V = N/r$) and $s = 0,1,\ldots,S$ is the number of stages (or iterations $S = \log_r N - 1$).

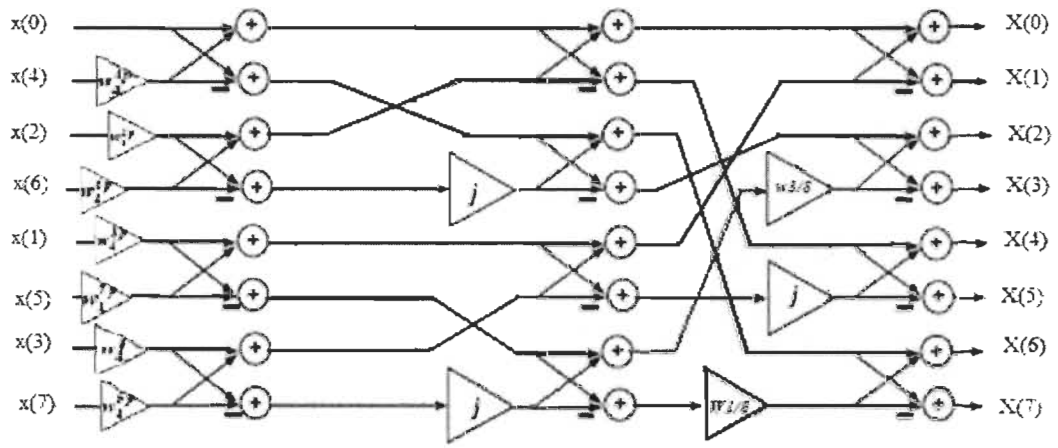


Figure 8: Radix-8 DIT butterflies where the highlighted red portion represents the butterfly critical path

Finally Eq. (8) could be expressed for the different stages in an FFT process as [11] and [12]:

$$[\mathbf{W}_N]_{l,m(v,s)} = \begin{cases} w_N^{\left[\left\lfloor v/r^s \right\rfloor l\, r^s\right]_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}, \tag{9}$$

for the DIF process and Eq. 8 would be expressed as:

$$[\mathbf{W}_N]_{l,m(v,s)} = \begin{cases} w_N^{\left[\left\lfloor v/r^{(S-s)} \right\rfloor l\, r^{(S-s)}\right]_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}, \tag{10}$$

for the DIT Process, where l=0,1,…,r-1 is the l^{th} butterfly's output, m=0,1,…,r-1 is the m^{th} butterfly's input and $\lfloor x \rfloor$ represents the integer part operator of x.

Consequently the l^{th} transform output during each stage could be illustrated as:

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[lmN/r + \left\lfloor v/r^s \right\rfloor l\, r^s\right]_N}, \tag{11}$$

for the DIF process and

$$\mathbf{X}_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[lmN/r + \left\lfloor v/r^{(S-s)} \right\rfloor mr^{(S-s)}\right]_N}. \tag{12}$$

In this article we will be adopting respectively the following abbreviation (RAG) for the reading address generator, (WAG) for the writing address generator and (CAG) for the coefficient address generator for DIF and DIT process.

The m^{th} butterfly's input of v^{th} word $x_{(m)}$ at the s^{th} stage (s^{th} iteration) is fed by [71]:

$$\mathrm{RAG}_{(m,v,0)} = m \times \frac{N}{r} + v \tag{13}$$

And for s> 0

$$\mathrm{RAG}_{(m,v,s)} = m \times \frac{N}{r^2} + \left[\left\lfloor \frac{v}{r^{(s-1)}} \right\rfloor \times \frac{N}{r}\right]_N + [\![k]\!]_{r^{(s-1)}} + \left\lfloor \frac{v}{r^s} \right\rfloor \times r^{(s-1)} \tag{14}$$

for the DIF process and

$$\mathrm{RAG}_{(m,v,s)} = m \times \left(\frac{N}{r^{(s+1)}} \right) + [\![v]\!]_{r^{(S-s)}} + \left\lfloor \frac{v}{r^{(S-s)}} \right\rfloor r^{(S+1-s)}, \tag{15}$$

for the DIT process where $m = 0,1,\ldots,r-1$, $v = 0,1,\ldots,V-1$ and $s = 0,1,\ldots,S$, $S = \log_r N - 1$ in which $[\![x]\!]_N$ represents the operation x modulo N and $\lfloor x \rfloor$ represents the integer part operator of x.

For both cases, the l^{th} processed butterfly's output $X_{(l,v,s)}$ for the v^{th} word at the s^{th} stage should be stored into the memory address location given by the WAG:

$$WAG_{(l,v,s)} = l(N/r) + v, \tag{16}$$

and to be noted that for both algorithms; the input data and the output data are in natural order during each stage of the FFT process known as Ordered Input Ordered Output (OIOO) algorithms.

The coefficients multipliers (Twiddle Factors) needed during each stage, which are fed to the m^{th} butterfly's input of v^{th} word $x_{(m)}$ at the s^{th} stage (s^{th} iteration), are provided by:

$$CAG_{(m,v,s)} = \left[\!\!\left[l \times \left(mV + \left\lfloor \frac{v}{r^s} \right\rfloor r^s \right) \right]\!\!\right]_N, \tag{17}$$

for the DIF process and

$$CAG_{(m,v,s)} = \left[\!\!\left[m \times \left(lV + \left\lfloor \frac{v}{r^{(S-s)}} \right\rfloor \times r^{(S-s)} \right) \right]\!\!\right]_N \tag{18}$$

for the DIT process.

By examining equations (16) and (17) we could clearly notice that in both algorithms the data are grouped with its corresponding coefficients multipliers during each stage due to the fact that the m^{th} coefficient multiplier of the l^{th} butterfly's output will shift if and only if v ($v = 0,1,\ldots,V-1$) will be equal to $r^{(S-s)}$ in the DIF process or $v = r^s$ in the DIT process. As a result and since $V = N/r = r^S$; the total number of shifting during each

stage in the DIT process would be r^s and the total number of shifting during each stage in the DIF process is $r^{(S-s)}$. Therefore, by implementing the word counter $r^{(S-s)}$ (*wordcounter* = 0, 1, …,$r^{(S-s)}$ – 1) and the shifting counter r^s (*shiftcounter* = 0,1,…, r^s – 1) in the DIT process or the word counter r^s and the shifting counter $r^{(S-s)}$ in the DIF process, we will obtain high efficient DIT/DIF radix-*r* algorithms in which the access to the coefficient multiplier's memory is reduced compared to the conventional radix-*r* DIT/DIF algorithms. In addition to that, the occurrence of the multiplication by one (i.e. the elements of the twiddle factor matrix illustrated in equation (8) are all equal to one) could be easily predicted when the shifting counter in both cases is equal to zero (i.e. $v < r^s$ or $v < r^{(S-s)}$). By doing so, the trivial multiplication by one (w^0) during the entire FFT process is avoided.

With the same reasoning as above the complexity of the DIT/DIF reading generators could be obtained and will be replaced with simple counters.

In this paragraph we will be deeply elaborating the radix-2 DIT FFT since it has higher Signal to Quantization Noise Ratio (SQNR) compared to the DIF technique [14].

Further reduction in computation and further reduction in the coefficient multiplier's memory access could be materialized and which will be elaborated in the next paragraph.

For simplicity and in order to reduce the complexity of the equations that will follow we will be defining the following terms:

$$\begin{aligned} &\alpha = r^{(S-s)} = 2^{(S-s)} \quad && \alpha_\lambda = \begin{cases} \alpha & for\,\lambda = 0 \\ \lambda\alpha & for\,\lambda \geq 1 \end{cases} \\ &\chi = \alpha && \chi_\lambda = \begin{cases} 0 & for\,\lambda = 0 \\ \lambda\alpha & for\,\lambda \geq 1 \end{cases} \\ &\beta = r \times r^{(S-s)} = 2 \times 2^{(S-s)} && \beta_\lambda = \lambda\beta \end{aligned} \quad . \tag{19}$$

For the radix 2 case equation (12) at the s^{th} stage could be rewritten as:

$$\begin{bmatrix} \mathbf{X}_{(k+\chi_\lambda)} \\ \mathbf{X}_{(k+\chi_\lambda+V)} \end{bmatrix} = \begin{bmatrix} x_{(n+\beta_\lambda)} + x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\left\| \lfloor v/2^{(S-s)} \rfloor 2^{(S-s)} \right\|_N} \\ x_{(n+\beta_\lambda)} + x^{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\left\| N/2+\lfloor v/2^{(S-s)} \rfloor 2^{(S-s)} \right\|_N} \end{bmatrix}, \tag{20}$$

that could be simplified as:

$$\begin{bmatrix} \mathbf{X}_{(k+\chi_\lambda)} \\ \mathbf{X}_{(k+\chi_\lambda+V)} \end{bmatrix} = \begin{bmatrix} x_{(n+\beta_\lambda)} + x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\left\| \lfloor v/2^{(S-s)} \rfloor 2^{(S-s)} \right\|_N} \\ x_{(n+\beta_\lambda)} - x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\left\| \lfloor v/2^{(S-s)} \rfloor 2^{(S-s)} \right\|_N} \end{bmatrix}, \tag{21}$$

where x denotes the input from the previous stage and $\mathbf{X}$ represents the transform output.

By replacing the term $\lfloor v/2^{(S-s)} \rfloor$ with λ which is the value of the shifting counter that cannot exceed $2^s - 1$ therefore, equation (21) that represents the proposed radix-2 algorithm, will have the final form as:

$$\begin{bmatrix} \mathbf{X}_{(k+\chi_\lambda)} \\ \mathbf{X}_{(k+\chi_\lambda+V)} \end{bmatrix} = \begin{bmatrix} x_{(n+\beta_\lambda)} + x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \\ x_{(n+\beta_\lambda)} - x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \end{bmatrix}. \tag{22}$$

For the first iteration ($s = 0$) the maximum value that v can attain is $V - 1$ as a result the term $\lfloor v/V \rfloor = \lambda$ is always zero therefore equation (22) for the first iteration will become:

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \end{bmatrix} = \begin{bmatrix} x_{(n)} + x_{(n+\alpha)} \\ x_{(n)} - x_{(n+\alpha)} \end{bmatrix}, \tag{23}$$

During the second iteration ($s = 1$) λ is either zero or one as a result equation (22) will be expressed as:

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \\ \mathbf{X}_{(k+\alpha)} \\ \mathbf{X}_{(k+\alpha+V)} \end{bmatrix} = \begin{bmatrix} x_{(n)} + x_{(n+\alpha)} \\ x_{(n)} - x_{(n+\alpha)} \\ x_{(n+\beta)} + x_{(n+\beta+\alpha)} w_N^{\alpha} \\ x_{(n+\beta)} - x_{(n+\beta+\alpha)} w_N^{\alpha} \end{bmatrix}, \tag{24}$$

that could be simplified as:

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \\ \mathbf{X}_{(k+\alpha)} \\ \mathbf{X}_{(k+\alpha+V)} \end{bmatrix} = \begin{bmatrix} x_{(n)} + x_{(n+\alpha)} \\ x_{(n)} - x_{(n+\alpha)} \\ x_{(n+\beta)} + (-j)\, x_{(n+\beta+\alpha)} \\ x_{(n+\beta)} - (-j)\, x_{(n+\beta+\alpha)} \end{bmatrix}. \tag{25}$$

Finally for the third iteration ($s = 2$) λ could have the following values 0, 1, 2 and 3 as a result equation (22) will be illustrated as:

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \\ \mathbf{X}_{(k+\alpha)} \\ \mathbf{X}_{(k+\alpha+V)} \\ \mathbf{X}_{(k+2\alpha)} \\ \mathbf{X}_{(k+2\alpha+V)} \\ \mathbf{X}_{(k+3\alpha)} \\ \mathbf{X}_{(k+3\alpha+V)} \end{bmatrix} = \begin{bmatrix} x_{(n)} + x_{(n+\alpha)} \\ x_{(n)} - x_{(n+\alpha)} \\ x_{(n+\beta)} + x_{(n+\beta+\alpha)} w_N^{\alpha} \\ x_{(n+\beta)} - x_{(n+\beta+\alpha)} w_N^{\alpha} \\ x_{(n+2\beta)} + x_{(n+2\beta+2\alpha)} w_N^{2\alpha} \\ x_{(n+2\beta)} - x_{(n+2\beta+2\alpha)} w_N^{2\alpha} \\ x_{(n+3\beta)} + x_{(n+3\beta+3\alpha)} w_N^{3\alpha} \\ x_{(n+3\beta)} - x_{(n+3\beta+3\alpha)} w_N^{3\alpha} \end{bmatrix}, \tag{26}$$

that could be simplified as:

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \\ \mathbf{X}_{(k+\alpha)} \\ \mathbf{X}_{(k+\alpha+V)} \\ \mathbf{X}_{(k+2\alpha)} \\ \mathbf{X}_{(k+2\alpha+V)} \\ \mathbf{X}_{(k+3\alpha)} \\ \mathbf{X}_{(k+3\alpha+V)} \end{bmatrix} = \begin{bmatrix} x_{(n)} + x_{(n+\alpha)} \\ x_{(n)} - x_{(n+\alpha)} \\ x_{(n+\beta)} + \left(\frac{\sqrt{2}}{2}(1-j) \right) \times x_{(n+\beta+\alpha)} \\ x_{(n+\beta)} - \left(\frac{\sqrt{2}}{2}(1-j) \right) \times x_{(n+\beta+\alpha)} \\ x_{(n+2\beta)} + (-j) \times x_{(n+2\beta+2\alpha)} \\ x_{(n+2\beta)} - (-j) \times x_{(n+2\beta+2\alpha)} \\ x_{(n+3\beta)} + \left(\frac{-\sqrt{2}}{2}(1+j) \right) \times x_{(n+3\beta+3\alpha)} \\ x_{(n+3\beta)} - \left(\frac{-\sqrt{2}}{2}(1+j) \right) \times x_{(n+3\beta+3\alpha)} \end{bmatrix}, \tag{27}$$

and the signal flow graph of an 8 point DIT FFT according to equation (27) is illustrated in Figure 9.

The multiplication by $-j$ in Figure 9 can be easily incorporated in the additions by switching the real and imaginary parts of the data and the multiplication cost of the input data by $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ is 2 real multiplication; as a result the total cost of real multiplication of the proposed structure will be 4 real multiplication compared to the structure of Figure 4 that will cost 20 real multiplication (5 complex multiplication).

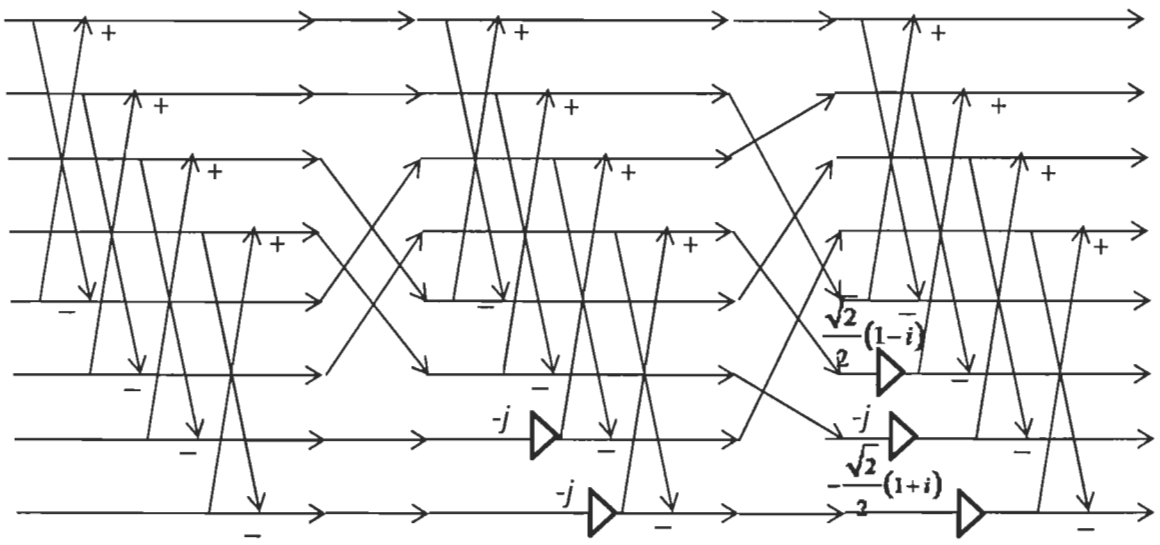


Figure 9: Signal flow graph (SFG) of 8 point DIT FFT on the proposed structure.

From the above (equations (23), (25) and (27)) we can conclude that the first, second and the third iterations of the DIT FFT process will contain only trivial multiplication.

In order to predict the occurrence of the trivial multiplication in the rest of the iterations (i.e. $s \geq 3$) which is a multiple of w_8 as shown in Figure 10; we will be introducing the term $2^{(s-2)}$ (will be referred in this article as separator) that will subdivide

2^s into 4 sub regions. The choice of the separator's value will be based on the following lemma:

Lemma 1:

For the all stages OIOOO FFT algorithm the product of $2^{(s-2)}$ and $2^{(S-s)}$ is always = $N/8 \; \forall s$.

Proof:

$$2^{(S-s)} \times 2^{(s-2)} = 2^{(S-2)} = \frac{N}{2^3} = \frac{N}{8}. \tag{28}$$

In this sub-section we will be computing equation (22) for the different values of λ:

i. $\lambda = 0$

ii. $\lambda_0 \in \left[1 \quad \cdots \quad 2^{(s-2)}\right[$

iii. $\lambda = 2^{(s-2)}$

iv. $\lambda_1 \in \left[2^{(s-2)}+1 \quad \cdots \quad 2\times 2^{(s-2)}\right[$

v. $\lambda = 2\times 2^{(s-2)}$

vi. $\lambda_2 \in \left[2\times 2^{(s-2)}+1 \quad \cdots \quad 3\times 2^{(s-2)}\right[$

vii. $\lambda = 3\times 2^{(s-2)}$

viii. $\lambda_3 \in \left[3\times 2^{(s-2)}+1 \quad \cdots \quad 2^s\right[$

For the i[th] case at the s^{th} iteration (stage) equation (22) will be expressed as:

$$\begin{bmatrix} X(k) \\ X(k+V) \end{bmatrix} = \begin{bmatrix} x(n)+x(n+\alpha) \\ x(n)-x(n+\alpha) \end{bmatrix}, \tag{29}$$

and for the iii[th] case equation (22) would be illustrated as:

$$\begin{bmatrix} X\left(k+2^{(S-2)}\right) \\ X\left(k+2^{(S-2)}+V\right) \end{bmatrix} = \begin{bmatrix} x\left(n+2^{(S-1)}\right)+x\left(n+2^{(S-1)}+2^{(S-2)}\right)\tau \\ x\left(n+2^{(S-1)}\right)-x\left(n+2^{(S-1)}+2^{(S-2)}\right)\tau \end{bmatrix}, \tag{30}$$

where $\tau = \frac{\sqrt{2}}{2}(1-j)$

For v[th] and vii[th] cases, equation (22) would be respectively:

$$\begin{bmatrix} \mathbf{X}_{\left(k+2^{(S-1)}\right)} \\ \mathbf{X}_{\left(k+2^{(S-1)}+V\right)} \end{bmatrix} = \begin{bmatrix} x_{\left(n+2^{S}\right)}+x_{\left(n+2^{S}+2^{(S-1)}\right)}(-j) \\ x_{\left(n+2^{S}\right)}-x_{\left(n+2^{S}+2^{(S-1)}\right)}(-j) \end{bmatrix}. \tag{31}$$

$$\begin{bmatrix} \mathbf{X}_{\left(k+3\times 2^{(S-2)}\right)} \\ \mathbf{X}_{\left(k+3\times 2^{(S-2)}+V\right)} \end{bmatrix} = \begin{bmatrix} x_{\left(n+3\times 2^{(S-1)}\right)}+x_{\left(n+3\times 2^{(S-1)}+3\times 2^{(S-2)}\right)}\sigma \\ x_{\left(n+3\times 2^{(S-1)}\right)}-x_{\left(n+3\times 2^{(S-1)}+3\times 2^{(S-2)}\right)}\sigma \end{bmatrix}, \tag{32}$$

where $\sigma = \frac{-\sqrt{2}}{2}(1+j)$.

Therefore; for $s \geq 3$ there is four sets of size $r^{(S-s)}$ words that have $\frac{\pm\sqrt{2}}{2}(1\pm j)$, 1 and $-j$ as trivial multiplications that should be grouped yielding to the following expression.

$$\begin{bmatrix} \mathbf{X}_{(k)} \\ \mathbf{X}_{(k+V)} \\ \mathbf{X}_{\left(k+2^{(S-2)}\right)} \\ \mathbf{X}_{\left(k+2^{(S-2)}+V\right)} \\ \mathbf{X}_{\left(k+2\times 2^{(S-2)}\right)} \\ \mathbf{X}_{\left(k+2\times 2^{(S-2)}+V\right)} \\ \mathbf{X}_{\left(k+3\times 2^{(S-2)}\right)} \\ \mathbf{X}_{\left(k+3\times 2^{(S-2)}+V\right)} \end{bmatrix} = \begin{bmatrix} x_{(n)}+x_{(n+\alpha)} \\ x_{(n)}-x_{(n+\alpha)} \\ x_{\left(n+2^{(S-1)}\right)}+x_{\left(n+2^{(S-1)}+2^{(S-2)}\right)}\tau \\ x_{\left(n+2^{(S-1)}\right)}-x_{\left(n+2^{(S-1)}+2^{(S-2)}\right)}\tau \\ x_{\left(n+2^{S}\right)}+x_{\left(n+2^{S}+2\times 2^{(S-2)}\right)}(-j) \\ x_{\left(n+2^{S}\right)}-x_{\left(n+2^{S}+2\times 2^{(S-2)}\right)}(-j) \\ x_{\left(n+3\times 2^{(S-1)}\right)}+x_{\left(n+3\times 2^{(S-1)}+3\times 2^{(S-2)}\right)}\sigma \\ x_{\left(n+3\times 2^{(S-1)}\right)}+x_{\left(n+3\times 2^{(S-1)}-3\times 2^{(S-2)}\right)}\sigma \end{bmatrix}, \tag{33}$$

and the resulting structure for this particular case is sketched in Figure 11.

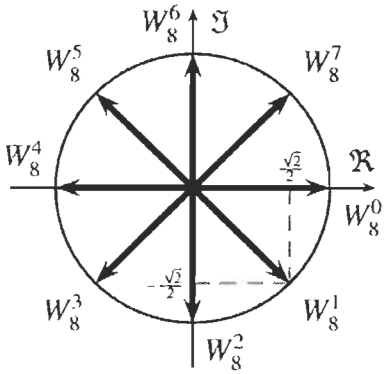


Figure 10: 8th root of unity

For the other cases and by comparing λ's domain; we could clearly notice that each domain of λ can be represented as:

$$\lambda \in \left[\xi r^{(s-2)}+1 \quad \xi r^{(s-2)}+2 \quad \cdots \quad \xi r^{(s-2)}+r^{(s-2)} \right[, \tag{34}$$

where $\xi = 0, 1, 2$ and 3, where the other cases will be expressed as:

$$\begin{bmatrix} \mathbf{X}_{\left(k+\alpha_{\lambda_\xi}\right)} \\ \mathbf{X}_{\left(k+\alpha_{\lambda_\xi}+V\right)} \end{bmatrix} = \begin{bmatrix} x_{\left(n+\beta_{\lambda_\xi}\right)} + x_{\left(n+\beta_{\lambda_\xi}+\alpha_{\lambda_\xi}\right)} w_N^{\alpha_{\lambda_\xi}} \\ x_{\left(n+\beta_{\lambda_\xi}\right)} - x_{\left(n+\beta_{\lambda_\xi}+\alpha_{\lambda}\right)} w_N^{\alpha_{\lambda_\xi}} \end{bmatrix}. \tag{35}$$

By regrouping these four cases where each of which will share the same coefficient multiplier; this will yield to the following expression:

$$\begin{bmatrix} \mathbf{X}_{\left(k+\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+\alpha_\lambda+V\right)} \\ \mathbf{X}_{\left(k+r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+r^{(s-2)}\alpha_\lambda+V\right)} \\ \mathbf{X}_{\left(k+2r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+2r^{(s-2)}\alpha_\lambda+V\right)} \\ \mathbf{X}_{\left(k+3r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+3r^{(s-2)}\alpha_\lambda+V\right)} \end{bmatrix} = \begin{bmatrix} x_{(n+\beta_\lambda)} + x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \\ x_{(n+\beta_\lambda)} - x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \\ x_{\left(n+r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+r^{(s-2)}\beta_\lambda+r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(r^{(s-2)}+\lambda\right)\alpha} \\ x_{\left(n+r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+r^{(s-2)}\beta_\lambda+r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(r^{(s-2)}+\lambda\right)\alpha} \\ x_{\left(n+2r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+2r^{(s-2)}\beta_\lambda+2r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(2r^{(s-2)}+\lambda\right)\alpha} \\ x_{\left(n+2r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+2r^{(s-2)}\beta_\lambda+2r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(2r^{(s-2)}+\lambda\right)\alpha} \\ x_{\left(n+3r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+3r^{(s-2)}\beta_\lambda+3r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(3r^{(s-2)}+\lambda\right)\alpha} \\ x_{\left(n+3r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+3r^{(s-2)}\beta_\lambda+3r^{(s-2)}\alpha_\lambda\right)} w_N^{\left(3r^{(s-2)}+\lambda\right)\alpha} \end{bmatrix}, \tag{36}$$

where $\lambda \in \left[1 \quad \cdots \quad 2^{(s-2)}\right[$.

The entity $w_N^{\left(2r^{(s-2)}+\lambda\right)\alpha}$ in the fifth and the sixth terms of equation (36) could be simplified as:

$$w_N^{\left(2r^{(s-2)}+\lambda\right)\alpha} = w_N^{\lambda\alpha} w_N^{2r^{(s-2)}\alpha} = w_N^{\alpha_\lambda} w_N^{\frac{N}{4}} = -jw_N^{\alpha_\lambda} \tag{37}.$$

By defining λ's domain for the entities $w_N^{\left(r^{(s-2)}+\lambda\right)\alpha}$ and $w_N^{\left(3r^{(s-2)}+\lambda\right)\alpha}$ as:

$$\lambda \in \left[2^{(s-2)} \quad \cdots \quad 1\right[, \tag{38}$$

therefore, these entities could be expressed respectively as:

$$w_N^{\left(r^{(s-2)}+\left(r^{(s-2)}-\lambda\right)\right)\alpha} = w_N^{2r^{(s-2)}\alpha-\lambda\alpha} = w_N^{-\left(\alpha_\lambda-\frac{N}{4}\right)} = conj\left(jw_N^{\alpha_\lambda}\right), \tag{39}$$

$$w_N^{\left(3r^{(s-2)}+\left(r^{(s-2)}-\lambda\right)\right)\alpha} = w_N^{4r^{(s-2)}\alpha-\lambda\alpha} = w_N^{-\alpha_\lambda} w_N^{\frac{N}{2}} = -conj\left(jw_N^{\alpha_\lambda}\right), \tag{40}$$

where *conj* in equations (39) and (40) refers to the complex conjugate process; as a result we can rewrite equation (36) as:

$$\begin{bmatrix} \mathbf{X}_{(k+\alpha_\lambda)} \\ \mathbf{X}_{(k+\alpha_\lambda+V)} \\ \mathbf{X}_{\left(k+r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+r^{(s-2)}\alpha_\lambda+V\right)} \\ \mathbf{X}_{\left(k+2r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+2r^{(s-2)}\alpha_\lambda+V\right)} \\ \mathbf{X}_{\left(k+3r^{(s-2)}\alpha_\lambda\right)} \\ \mathbf{X}_{\left(k+3r^{(s-2)}\alpha_\lambda+V\right)} \end{bmatrix} = \begin{bmatrix} x_{(n+\beta_\lambda)} + x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \\ x_{(n+\beta_\lambda)} - x_{(n+\beta_\lambda+\alpha_\lambda)} w_N^{\alpha_\lambda} \\ x_{\left(n+r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+2r^{(s-2)}\beta_\lambda-\left(2r^{(s-2)}-1\right)\alpha_\lambda\right)} \times conj\left(jw_N^{\alpha_\lambda}\right) \\ x_{\left(n+r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+2r^{(s-2)}\beta_\lambda-\left(2r^{(s-2)}-1\right)\alpha_\lambda\right)} \times conj\left(jw_N^{\alpha_\lambda}\right) \\ x_{\left(n+2r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+2r^{(s-2)}\beta_\lambda+2r^{(s-2)}\alpha_\lambda\right)} (-j) w_N^{\alpha_\lambda} \\ x_{\left(n+2r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+2r^{(s-2)}\beta_\lambda+2r^{(s-2)}\alpha_\lambda\right)} (-j) w_N^{\alpha_\lambda} \\ x_{\left(n+3r^{(s-2)}\beta_\lambda\right)} + x_{\left(n+3r^{(s-2)}\beta_\lambda-\left(3r^{(s-2)}-1\right)\alpha_\lambda\right)} \times -conj\left(w_N^{\alpha_\lambda}\right) \\ x_{\left(n+3r^{(s-2)}\beta_\lambda\right)} - x_{\left(n+3r^{(s-2)}\beta_\lambda-\left(3r^{(s-2)}-1\right)\alpha_\lambda\right)} \times -conj\left(w_N^{\alpha_\lambda}\right) \end{bmatrix}. \tag{41}$$

From equation (41) the JMFFT radix 2^3 butterfly can be derived (as shown in Figure 12 in which one complex coefficient multiplier (or twiddle factor) is needed for each

8 complex input. In addition to that, the coefficient multiplier memory is accessed once each 4×2^{s} word (a set of two inputs) for the DIT process and $2^{(S-s)}$ in the DIF process where s is the actual stage (iteration) of the FFT process and S is the total number of stages of the FFT process ($S = log_2 (N) - 1$).

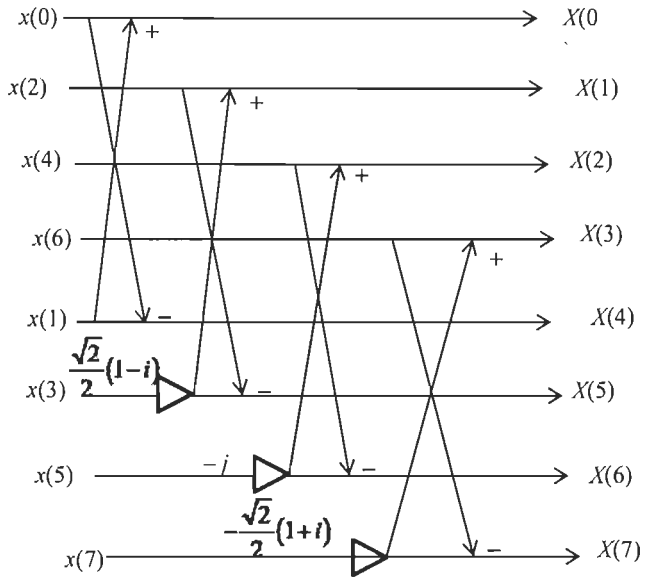


Figure 11: The proposed JMFFT 2^3 butterfly structure for trivial computation where the inputs/outputs are provided by equation (32).

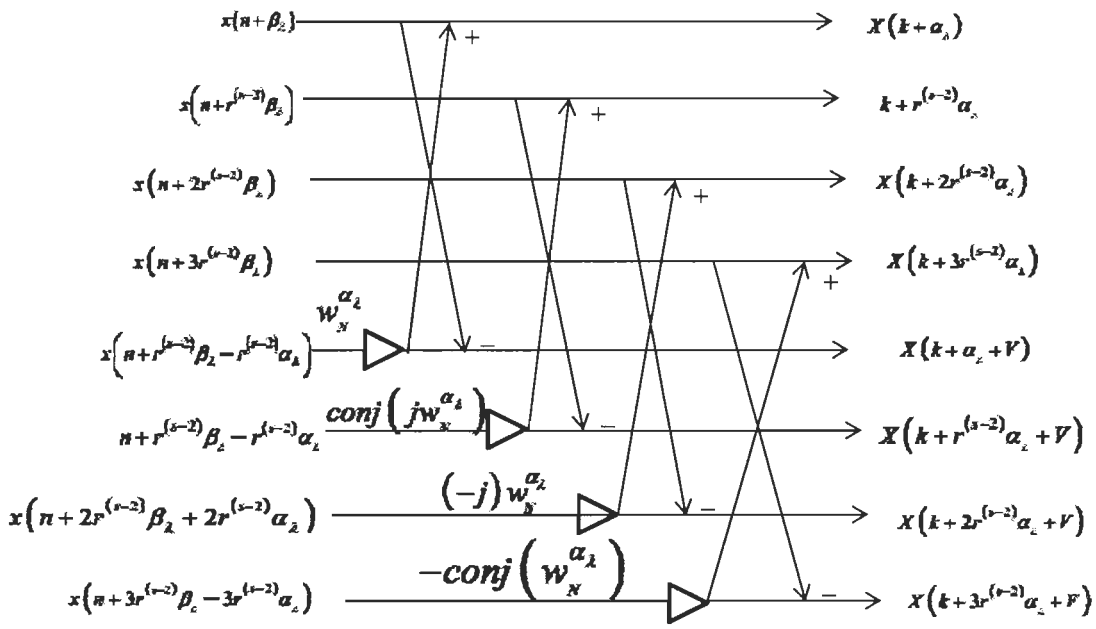


Figure 12: The proposed JMFFT 2^3 butterfly structure for non-trivial computation.

Compared to the cited methods in [8] and [9] that requires respectively 2 memory accesses per 4 inputs and 1 memory access per 2 inputs, our proposed method will require one memory access per 8 inputs; furthermore, the multiplication by $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ can be also predicted where the number of arithmetical operation required for the complex multiplication can be reduced from 6 to 2 as shown in Tables 1 and 2 meanwhile the reduction in memory access to the coefficient multiplier's memory is illustrated in table 3 for different FFT sizes.

Table 1: Comparison in terms of real multiplication between the cited methods in [8] versus the proposed method

				Multiplication reduction (%)	
N	TI [8]	Cited [8]	Proposed	TI	Cited
8	48	8	4	91.7	50
16	128	40	24	81.25	40
32	320	136	88	72.5	35.29
64	768	392	264	65.62	32.65
128	1792	1032	712	60.26	31.1
256	4096	2568	1800	56.05	29.90
512	9216	6152	4360	52.69	29.12
1024	20480	14344	10248	49.96	28.55
2048	45056	32776	23560	47.70	28.11

Table 2: Comparison in terms of real addition between the cited methods in [8] versus the proposed method

				Addition reduction (%)	
N	TI [8]	Cited [8]	Proposed	TI	Cited
8	72	52	48	33.34	7.6
16	192	148	140	27.08	5.40
32	480	392	380	20.83	3.06
64	1152	988	972	15.62	1.6
128	2688	2400	2380	11.45	0.83
256	6144	5668	5644	8.13	0.77
512	13824	13096	13068	5.46	0.21
1024	30720	28740	28708	6.54	0.11
2048	67584	66612	66572	1.49	0.06

4. Performance Evaluation

The proposed structure has been tested on the TMS320C6416 DSP platform by mean of TI's Code Composer Studio (CCS version 4.2). The beauty of this platform is its capability to execute 8 instructions in parallel that is highly desirable for our proposed structure which will maximize the use of the platform's resources. We compared the performance of our proposed method with the TI's DIF radix-2 FFT, referred as "TI" [15], and with the best DIT method referred as "Cited "[8].

Table 3: Comparison in terms of memory accesses to the coefficient multiplier in [8] versus the proposed method where each complex access is counted as 1:

				Memory accesses reduction (%)	
N	TI [8]	Cited [8]	Proposed	TI	Cited
8	7	1	0	100	100
16	15	5	2	86.7	60
32	31	15	8	74.2	46.7
64	63	37	22	65.1	40.5
128	127	83	52	59.1	37.35
256	255	177	114	55.3	35.6
512	511	367	240	53.1	34.7
1024	1023	749	494	51.7	34.1
2048	2047	1515	1004	49.1	33.7

In our performance study, the simulation results of the cited methods are obtained in bit reverse order meanwhile our obtained results are in natural order where the bit reversing process was not taken into consideration in the simulation results.

The simulation environment for all methods is detailed as follow:

- Clock 1000 MHz.
- Memory clock 100 MHz.
- Mode Release

- C6416 Device Cycle Accurate Simulator, little Endian.

Table 4 reveals the simulation results of the cited methods in [8] versus the proposed one where the term Loss is defined as the ratio of the cited method over the proposed method where Figure 13 illustrate the values of this ratio.

Table 4: Comparative results in term of clock cycle of the cited methods versus the proposed method for different FFT sizes

				Cycle Reductions (%)	
Length	TI	REF	Proposed	TI	REF
64	5252	4210	3648	43,97	15,41
128	11363	9048	7612	49,28	18,86
256	24578	19246	15832	55,24	21,56
512	53025	40676	32852	61,41	23,82
1024	113984	85594	68048	67,51	25,78
2048	244063	179536	140748	73,40	27,56
4096	520574	375622	290760	79,04	29,19

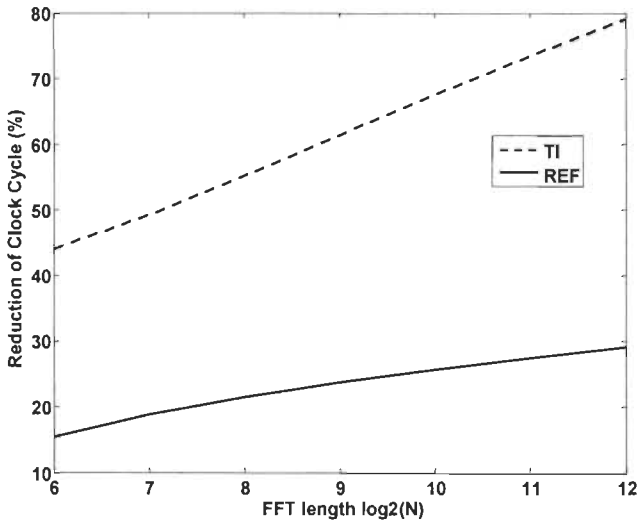


Figure 13: Reference methods, TI and REF, clock cycle reduction compared with our proposed method.

Table 5: Comparison of the coefficients multiplier's memory requirement of the cited methods versus the proposed method where the size is computed in term of byte

FFT Length	TI [8]	REF [8]	Proposed
N	$2N$	$N/2-2$	$N/8-1$

5. Conclusion

Finally, this paper has presented an efficient ordered input ordered output radix 2^3 algorithm that reduces the complexity and the computational effort in comparison to the most recent proposed methods. Furthermore, the proposed method had showed a significant execution time in term of clock cycles compared to the cited methods and by predicting the 8^{th} root of unity and the memory size needed to stock the coefficient multiplier is reduced to $N/8$.

References

[1] W. H. Press, S. A. Teuklosky, W. T. Vetterling and B. P. Flannery, "Numerical Recipes in C" second edition, Cambridge University Press, 1996.

[2] N. W. Cooley, J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. Comput., 19, pp. 297-301, April 1965.

[3] W. M. Gentleman and G. Sande, "Fast Fourier Transform-For fun and profit", Proceedings-Fall Joint Computer Conference, AFIPS Proc., Vol. 29, Washington D.C.: Spartan, pp. 563-578, 1966.

[4] T. Widhe, "Efficient Implementation of FFT Processing Elements" Linköping studies in Science and Technology, Thesis No. 619, Linköping University, Sweden, June 1997.

[5] S. Winograd, "On Computing The Discrete Fourier Transform", Proc. Nat. Acad. Sci. USA, Vol. 37, pp 1005-1006, April 1976.

[6] P. Duhamel, and H. Hollman, "Split Radix FFT Algorithm, Electronics Letters, Vol. 20, No. 1, pp. 14-16, Jan. 1984.

[7] M Jaber "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 B2 and European patent application Serial no: PCT/US01/07602, 2006

[8] Y. Wang and al, "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors", IEEE Transactions on signal processing, Vol. 55, No. 5, May 2007

[9] T. Sun, Y. Yu, "Memory Usage Reduction Method for FFT Implementations on DSP Based Embedded System", the 13th IEE international symposium on consumer electronics (ISCE2009).

[10] S. W. Smith "The Scientist and Engineer's Guide to Digital Signal Processing" second edition, California Technical Publishing, San Diego California, 1999.

[11] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6,751,643, 2004.

[12] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009.

[13] M. Jaber and D. Massicotte, "A Novel Approach for FFT Data Reordering", International Symposium On Circuit and System (ISCAS), Paris, May 2010.

[14] W. Chang and Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms", IEEE Transactions On Signal Processing, vol. 56, no. 10, October 2008.

[15] "TMS320C64x DSP Library Programmer's Reference", Literature Number: SPRU565B, Oct. 2003, (code DSP-radix-2, p. 4-9, 4-10).

[16] A. W. M Vanden Enden and N.A.M Verhoeckx " Traitement Numérique du signal", Masson 1992.

Chapter 6 _Low Complexity Input/Output Pruning JMFFT Kernel Core

Paper X: M. Jaber and D. Massicotte, "Low Complexity Input/output Pruning JMFFTs Suitable for the OFDMA's 3GPP LTE Implementation", *to be submitted to a Journal after the end of the confidentiality.*

Résumé du Chapitre 6

Dans de nombreuses applications telles que la radio cognitive, qui est basée sur l'OFDM, sera principalement basée sur la taille de la TRF où on applique de manière efficace des zéros pour les entrées avec des distributions arbitraires. Ceci sera la cible de notre travail futur du fait que les algorithmes existants sont principalement concentrés sur les séquences où on applique de manière efficace des zéros à la fin des entrées. Les aspects théoriques d'élagage de la TRF (PFFT – Pruning FFT) ont été soigneusement élaborés au cours des trois décennies et ont été principalement concentrés sur les séquences où on applique de manière efficace des zéros consécutifs à l'entrée Dans de nombreuses applications telles que la radio cognitive, qui est basée sur l'OFDM, on aura besoin de calculer un certain nombre de sorties de la TRF où on a appliqué de manière efficace des zéros sur les entrées. Notre contribution dans ce sujet est basée sur l'introduction d'un algorithme qui peut réduire la complexité du calcul en se comparant avec les méthodes les plus récemment proposées.

Les résultats expérimentaux montrent que l'utilisation de notre méthode d'élagage de la TRF peut en effet accélérer le processus de calcul jusqu'à 30% lorsque la taille de la TRF et le modèle des entrées /sorties non utilisées sont connus à l'avance.

Paper X: M. Jaber and D. Massicotte, "Lowest Complexity Input/output Pruning FFT", *to be submitted to a Journal after the end of the confidentiality.*

Lowest Complexity Input/Output Pruning FFT

Marwan A. Jaber and Daniel Massicotte, *Senior Member, IEEE*

***Abstract* – FFTs algorithms are used in digital signal processing which break down complex signals into elementary components and where the transform length N, is decomposed into arbitrary factors ($N = r_1, r_2, \ldots, r_k$). Input Pruning FFT's are efficient Fast Fourier Transform (FFT), where the efficiency can be increased by removing operations on input values which are zero. Furthermore, Output pruning FFT is a method used to compute a discrete Fourier transform (DFT) where only a subset of the outputs are needed. In this paper, we will propose a generalized radix-r input-output pruning FFT, which will compute efficiently the selected spectrum's bin of a sequence of size N that contains M consecutive non-zero input points from which only L_o outputs are desired.**

Index terms-Discrete Fourier Transform, Input/output pruning FFT, cognitive radio, VLIW DSP.

1. Introduction

Input pruning FFTs are commonly used in the padded FFT process which is known as the up-sampling process in digital signal processing that consists of extending a signal (or spectrum) with zeros. By doing so, this can increase the time sampling which is known as the time domain interpolation that people commonly use and which is translated into forcing the FFT algorithm to sample the spectrum at smaller frequency intervals. Theoretical aspects of the Pruning FFT (PFFT) have been thoroughly elaborated in past three decades and which was mainly concentrated on sequences that have L_i consecutive non-zero input points at the beginning. In many applications such as the OFDM based

Cognitive Radio will be mainly based on FFT pruning which applies efficiently zeros to the inputs with arbitrary distributions and this will be the target of our future work. In many applications the percentage of required input/output bins is very small and that is why our performance study will be targeting these applications. For instance, in the 3GPP LTE (Long Term Evolution) where the OFMDA's symbol size is 1024 in which 12 users equally share the available 600 sub-carriers, as a result only 50 of the 1024 FFT output bins (4.88%) are required for each mobile terminal [1]. These partial output/input cases are extraordinarily important for the future wireless systems and due to the fact that the PFFT can potentially achieve a significant speedup which is made it as a target by many applications such as: OFDMA (Orthogonal Frequency Division Multiplexing Access) cognitive radio [2], VLIW DSP for mobile Application [3], in multi-channel OFDM system [4] - [5], the latest MIMO-OFDM (Multiple Input Multiple Output – Orthogonal Frequency Division Multiplexing) systems and a lot of none cited applications. The paper is organized as follows; Section 2 will elaborate the proposed method of input Pruning FFT while Section 3 provides a detailed description of the proposed methods for input/output pruning FFT. Section 4 provides a performance evaluation while Section 5 reports the conclusions.

2. The Proposed Input Pruning FFT

The basis of the radix-2 FFT is that a DFT can be divided into two smaller DFTs, each of which is divided into two smaller DFTs, and so on, resulting in a combination of two points DFTs [6]. Several methods are used repeatedly to split the DFTs into smaller (two or four-point) core calculations. The advantage of appropriately breaking the DFT in terms of its partial DFTs is that the number of multiplications and the number of stages may be controlled. The number of stages often corresponds to the amount of global

communication and/or memory accesses in implementation, and thus, reduction in the number of stages is beneficial. The research of the PFFT can be traced back to 1971 where the theoretical aspects of the PFFT have been thoroughly studied in the last three decades. The most important drawback is in the data-flow control which was the obstacle for efficient implementations on parallel architectures. In this paper we will derive a recursive general radix-*r* pruned FFT that is suitable for the input pruned FFTs where the proposed pruning FFT algorithm has fewer complex multiplications than the most recent other pruning FFT algorithms. The proposed algorithm shows substantial gain in the computational load and a significant speed up in the pruning FFT algorithm due to the data transfers and address computations [7] and [8]. The conceptual key to the use of the modified radix–*r* FFT butterfly is the formulation of the radix–***r*** as composed engines with identical structures and a systematic means of accessing the corresponding multiplier coefficients [9] and [10]. This enables the design of an engine with the lowest rate of complex multipliers and adders, which utilizes r or $r-1$ complex multipliers in parallel to implement each of the butterfly computations. There is a simple mapping from the three indices (FFT stage, butterfly, and element) to the addresses of the multiplier coefficients needed.

The DFT computation expressed as:

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} w_N^{nk} \quad \text{for } k = 0,1,\cdots,N-1, \tag{1}$$

where $w_N^k = e^{-j\frac{2\pi k}{N}}$.

Let us consider that the number of consecutive input elements that can be different of zero is $L_i \leq \mathrm{M} = N/D_{ip}$, equation (1) could be factorized into:

$$\mathbf{X}_{(k)} = \sum_{n=0}^{N-1} x_{(n)} w_N^{nk} = \sum_{n_2=0}^{D_{ip}-1} \sum_{n_1=0}^{M-1} x_{(n_1+Mn_2)} w_N^{(n_1+Mn_2)k} = \sum_{n_2=0}^{D_{ip}-1} w_N^{Mn_2 k} \sum_{n_1=0}^{M-1} x_{(n_1+Mn_2)} w_N^{n_1 k} \quad , \tag{2}$$

where we have adopted

$$\begin{aligned} n &= n_1 + Mn_2 & k &= k_1 + D_{ip}k_2 \\ n_1 &= 0,1,\cdots,M-1 & k_1 &= 0,1,\cdots,D_{ip}-1. \\ n_2 &= 0,1,\cdots,D_{ip}-1 & k_2 &= 0,1,\cdots,M-1 \end{aligned} \tag{3}$$

The indices n_2 will determine the position of the nonzero consecutive inputs into the sequence where we have applied efficiently zeros to the inputs with arbitrary distributions. Since we will be only interested in this article on sequences that have L_i consecutive non-zero input points at the beginning therefore, n_2 will be set to zero and as a result, equation (3) could be rewritten as:

$$\mathbf{X}_{(k)} = \sum_{n=0}^{M-1} x_{(n)} w_N^{n(k_1+Dipk_2)} = \sum_{n=0}^{M-1} x_{(n)} w_N^{nk_1} w_M^{k_2} \, . \tag{4}$$

The computational complexity of equation (4) could be performed in two ways where this subsection will deeply elaborate the comparison between these proposed methods.

The first method is known as the direct way where equation (3) can be expressed in:

$$\mathbf{X}_{(k)} = \sum_{n=0}^{M-1} x_{(n)} w_N^{nk_1} w_M^{k_2} = \sum_{n=0}^{M-1} y_{(n)} w_M^{k_2} \, , \tag{5}$$

Where y_n is illustrated as:

$$y_{(n)} = x_{(n)} w_N^{nk_1} \, . \tag{6}$$

The computational complexity of equation (4) would be:

$$t_{c-input} = D_{ip} M \left(t_{cFFT_M} + t_{cm} \right) = N \left(t_{cFFT_M} + t_{cm} \right), \tag{7}$$

where t_{cFFT} is the complexity of the FFT algorithm of size (M) and t_{cm} is the complexity of the complex multiplier.

Logically, the optimal solution of equation (5) is obtained by optimizing the complexity of the FFT algorithm where through the literature researchers were oriented in the optimization of the FFT algorithm. In this paper we are going to show another method of optimizing equation (5) which could be achieved by incorporating the twiddle factors $w_N^{nk_1}$ and the adder tree matrices into a single stage of calculation. Therefore, since we are targeting industrial applications this comparison will be devoted to the radix-2 algorithm in which the split radix algorithm has been excluded due to its L shaped butterfly that made it hard in hardware implementation.

The complexity of the Cooley-Tukey algorithm in term of complex multiplication is:

$$t_{cm-cooley-tukey} = \frac{M}{2}\left(log_2 M\right) = \frac{M}{2} S_M , \tag{8}$$

where $S_M = log_2 M$.

On the other hand, each radix-2 butterfly will require 2 complex additions/subtractions; as a result the total number of complex of additions/subtractions in $t_{ca/s}$ DIT process will require:

$$t_{ca/s-cooley-tukey} = \left(\frac{M}{2}\right) log_2 M . \tag{9}$$

Knowing that each complex multiplication will require 6 arithmetic operations and each addition/subtraction will require 2 arithmetic operations therefore, the total number t_o of the arithmetic operations in the DIT Cooley-Tukey algorithm will be estimated as:

$$t_{o-cooley-tukey} = 6\left(\frac{M}{2}\right) S_M + 2\left(\frac{M}{2}\right)\left(S_M\right) = 4MS_M = 4M\, log_2 M , \tag{10}$$

as a result the total amount required to compute the input pruning FFT is [13]:

$$t_{c-input-Puning\_Medina} = D_{ip} M\left(4MS_M + M + 6\right) = N\left(4M\, log_2 M - 3M + 6\right), \tag{11}$$

Our proposed method is based on the same reasoning of the radix-r DFT factorization introduced in [9] and [10]; equation (5) could be re-written as:

$$\mathbf{X}_{(k_1,k_2)} = \sum_{n=0}^{\frac{M}{r}-1} x_{(m)} w_N^{mk_1} w_M^{mk_2} + \ldots + \sum_{n=0}^{\frac{M}{r}-1} x_{(m_1+(r-1))} w_N^{(m+(r-1))k_1} w_M^{(m+(r-1))k_2} . \quad (12)$$

In the summations, the variables r, k_1 and k_2 are independents of n_2. We factorize $w_M^{rk_2}$ in (12) and considering that $w_M^{mk_2} = w_{M/r}^{nk_2}$ and rewrite (12) as follow:

$$\begin{aligned}\mathbf{X}_{(k_1,k_2)} = & \sum_{n=0}^{\frac{M}{r}-1} x_{(m)} w_{\frac{M}{r}}^{nk_2} w_{\frac{N}{r}}^{nk_1} + w_M^{k_2} w_N^{k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(m+1)} w_{\frac{M}{r}}^{nk_2} w_{\frac{N}{r}}^{nk_1} \\ & + \ldots + w_M^{(r-1)k_2} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(m+(r-1))} w_{\frac{M}{r}}^{nk_2} w_{\frac{N}{r}}^{nk_1}\end{aligned} . \quad (13)$$

To subdivide the axis k_2 in Eq. (13) in 2 new axis v and l, we pose $k_2 = v + lV$ with $v = 0,1,\ldots,V-1$ and $l = 0,1,\ldots,r-1$ where $V=M/r$. Therefore, $\mathbf{X}_{(k_1+D_{ip}k_2)}$ is replaced by using new indices v and l with $k_1 = 0,1,\cdots,D_{ip}-1$. As a result we can rewrite (13) in r equations as shown in Eq. (14), Eq. (15) and Eq. (16).

$$\mathbf{X}_{(k_1,v)} = \begin{bmatrix} w_M^0 \sum_{n=0}^{\frac{M}{r}-1} x_{(m)} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} + \\ \cdots + w_M^{(r-1)v} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(m+(r-1))} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} \end{bmatrix}$$

$$\mathbf{X}_{(k_1,2v)} = \begin{bmatrix} w_M^0 \sum_{n=0}^{\frac{M}{r}-1} x_{(m)} w_{M/r}^{2nv} w_{\frac{N}{r}}^{nk_1} + \\ \cdots + w_M^{(r-1)v} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(m+(r-1))} w_{M/r}^{2nv} w_{\frac{N}{r}}^{nk_1} \end{bmatrix}$$

$$\vdots$$

$$\mathbf{X}_{(k_1,(v+(r-1)V))} = \begin{bmatrix} w_M^0 \sum_{n=0}^{\frac{M}{r}-1} x_{(m)} w_{M/r}^{n(v+(r-1)V)} w_{\frac{N}{r}}^{nk_1} + \\ \cdots + w_M^{(r-1)(v+(r-1)V)} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(m+(r-1))} w_{M/r}^{n(v+(r-1)V)} w_{\frac{N}{r}}^{nk_1} \end{bmatrix} . \quad (14)$$

Considering that $w_V^{\alpha V} = \left(w_V^V\right)^\alpha = 1^\alpha = 1$ and $V=M/r$, therefore Eq. (14) could be expressed as follow

$$\mathbf{X}_{\left(k_1,(v+V)\right)} = \begin{bmatrix} w_M^0 \sum_{n=0}^{\frac{M}{r}-1} x_{(rn)} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} + \cdots \\ + w_M^{(r-1)\frac{M}{r}} w_M^{(r-1)v} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(rn+(r-1))} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} \end{bmatrix}$$

$$\vdots \qquad , \qquad (15)$$

$$\mathbf{X}_{\left(k_1,(v+(r-1)V)\right)} = \begin{bmatrix} w_M^0 \sum_{n=0}^{\frac{M}{r}-1} x_{(rn)} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} + \\ \cdots + w_M^{(r-1)^2\frac{M}{r}} w_M^{(r-1)v} w_N^{(r-1)k_1} \sum_{n=0}^{\frac{M}{r}-1} x_{(rn+(r-1))} w_{M/r}^{nv} w_{\frac{N}{r}}^{nk_1} \end{bmatrix}$$

$$\mathbf{X}_{\left(k_1,v\right)} = \begin{pmatrix} X_{(v)} \\ X_{(v+V)} \\ \vdots \\ X_{(v+(r-1)V)} \end{pmatrix} = \begin{pmatrix} w_N^0 & w_N^0 & \cdots & w_N^0 \\ w_N^0 & w_N^{N/r} & \cdots & w_N^{(r-1)(N/r)} \\ \vdots & \vdots & \ddots & \vdots \\ w_N^0 & w_N^{(r-1)(N/r)} & \cdots & w_N^{(r-1)^2(N/r)} \end{pmatrix}$$

$$\times \begin{pmatrix} w_N^0 & 0 & \cdots & 0 \\ 0 & w_N^{D_{ip}v} w_N^{k_1} & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & w_N^{D_{ip}(r-1)v} w_N^{(r-1)k_1} \end{pmatrix} \times \begin{bmatrix} \sum_{n=0}^{\frac{M}{r}-1} x_{(rn)} w_{\frac{M}{r}}^{nv} w_{\frac{N}{r}}^{nk_1} \\ \sum_{n=0}^{\frac{M}{r}-1} x_{(rn+1)} w_{\frac{M}{r}}^{nv} w_{\frac{N}{r}}^{nk_1} \\ \vdots \\ \sum_{n=0}^{\frac{M}{r}-1} x_{(rn+(r-1))} w_{\frac{M}{r}}^{nv} w_{\frac{N}{r}}^{nk_1} \end{bmatrix}. \qquad (16)$$

We recognize the first matrix, the well-known adder tree matrix $\mathbf{T}_r$ and the second matrix will be known as the IPJMFFT (Input Pruning JMFFT) twiddle factor matrix $\mathbf{W}_N$, respectively. Equation (16) can be expressed in a compact form as:

$$\mathbf{X}_{\left(k_1+D_{ip}(v+lV)\right)} = \mathbf{X}_{(k_1,k_2)} = \mathbf{T}_M \mathbf{W}_N col\left(\sum_{n=0}^{V-1} x_{(rn+l)} w_V^{nv} w_{\frac{N}{r}}^{nlk_1} \middle| l = 0,1,\cdots,r-1 \right), \qquad (17)$$

for $k_1 = 0,1,\cdots,D_{ip}-1$ and $v = 0,1,..,V-1$ with

$$\mathbf{X}_{(k_1,k_2)} = \left[\sum_{n=0}^{V-1} x_{(rn+l)} w_V^{nv} w_{\frac{N}{r}}^{nlk_1} \,\middle|\, l = 0,1,\cdots,r-1 \right]^T, \tag{18}$$

$$\mathbf{W}_N = diag\left(w_N^{D_{ip} lv} w_N^{lk_1} \,\middle|\, l = 0,1,\cdots,r-1 \right) \tag{19}$$

and

$$\mathbf{T}_M = \begin{pmatrix} w_M^0 & w_M^0 & w_M^0 & \cdots & \cdots & w_M^0 \\ w_M^0 & w_M^{M/r} & w_M^{2M/r} & \cdots & \cdots & w_M^{(r-1)M/r} \\ w_M^0 & w_M^{2M/r} & w_M^{4M/r} & \cdots & \cdots & w_M^{2(r-1)M/r} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ w_M^0 & w_M^{(r-1)M/r} & w_M^{2(r-1)M/r} & \cdots & \cdots & w_M^{(r-1)^2 M/r} \end{pmatrix}. \tag{20}$$

In DSP Layman language, the factorization of an FFT can be interpreted as dataflow which depicts the arithmetic operations and their dependencies. When equation (16) is read from left to right we will obtain the decimation in frequency algorithm, meanwhile if the equation is read from right to left we will obtain the decimation in time algorithm. We note that the DIF algorithm requires one shuffling stage in order to obtain an ordered output data.

In this section we will be limited to the DIT IPJMFFT where the DIF IPJMFFT could be easily derived. We can write the read address generator (RAG), write address generator (WAG) and coefficient address generator (CAG) for the DIT process, respectively [11] and [12]

$$WAG = lV + v, \tag{21}$$

and

$$RAD = n_1\left(\frac{M}{r^{(s+1)}}\right)+[\![v]\!]_r\left(S_M - s\right)+\left\lfloor\frac{v}{r^{(S_M-s)}}\right\rfloor r^{(S_M+1-s)}\ , \tag{22}$$

with a slight modification in the DIT Coefficient address generator the DIT IPJMFFT could be obtained as:

$$CAG = \left[\!\!\left[n_1\left(D_{ip}\left(lV + \left\lfloor\frac{v}{r^{(S_M-s)}}\right\rfloor r^{(S_M-s)}\right)+\left(r^{(S_M-s)}k_1\right)\right)\right]\!\!\right]_N\ , \tag{23}$$

where $[\![x]\!]_N$ represents the operation x modulo N, $\lfloor x \rfloor$ represents the integer part operator of x, the indices are $v = 0,1,\ldots,V-1$, and $s = 0,1,\ldots,S_M$ where r is the radix-r, V is the number of words ($V = M/r$), and S_M is the number of stages ($S_M = \log_r M - 1$). By the using the JFFT introduced in [9], the JMPFFT would be:

$$\mathbf{X}_{\left(k_1+D_{ip}\times k_2\right)} = \mathbf{X}_{\left(k_1+D_{ip}\times k_2,s+1,v,Wag\right)} = \sum_{n=0}^{r-1}\left[\mathbf{X}_{\left(k_1+D_{ip}\times k_2,s,v,Rad\right)} W_M^{CAG}\right]. \tag{24}$$

The computational complexity of the second proposed algorithm is similar to the complexity of the DIT Cooley-Tukey algorithm therefore; the complexity of equation (17) in term of arithmetic operation for the IPJMFFT is expressed as follow:

$$t_{c-input-IPJMFFT} = D_{ip}M\left(4MS_M\right) = N\left(4MS_M\right) = N\left(4M\,log_2\,M\right). \tag{25}$$

For real value arithmetic operations, the complexity ratio between our proposed method, IP-JMFFT, and cited input pruning FFT is [13]:

$$G_{IPJMFFT/input-Puning\_Medina} = \frac{\left(4MS_M\right)}{\left(4MS_M+8\right)} = \frac{\left(4M\,log_2\,M\right)}{\left(4M\,log_2\,M+8\right)}, \tag{26}$$

This ratio is sketched in Figure 1 for N=8192 and the input pruning, M, changes from 2 to N. We observe a complexity reduction for $M<2^6$.

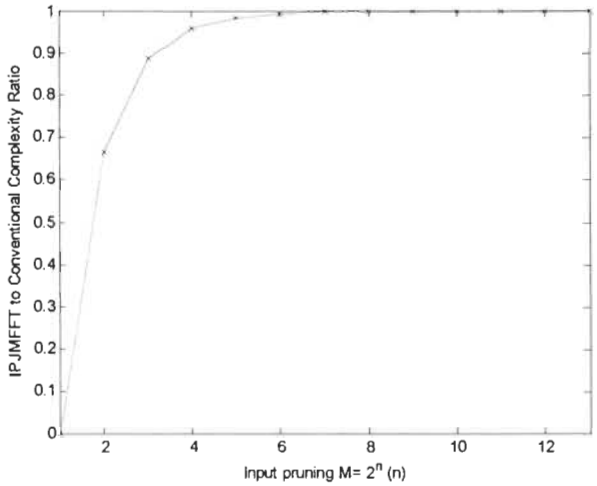


Figure 1 Comparison of real operations reduction between our proposed method and [13].

3. The Proposed Input/Output Pruned JMFFT Algorithm

Output pruning FFT is a method used to compute a discrete Fourier transform (DFT) where only a subset of the outputs are needed or if you have a transform of size M which has been zero padded to a size N and where only $L_o \leq P$ consecutive outputs of the sized N transform are desired. Suppose that P is the required consecutive outputs which are computed from the L_i non-zero consecutive inputs and where it is assumed that $L_o/D_{ip} \approx P=M/D_{op}$ therefore, equation (5) could be expressed as according to [13]:

$$\begin{aligned}\mathbf{X}_{(k)} &= \mathbf{X}_{(k_1+D_{ip}k_2)} = \mathbf{X}_{(k_1+D_{ip}(k_3+D_{op}k_4))} \\ &= \sum_{n_1=0}^{D_{op}-1}\left\{ w_M^{n_1k_3}\left[\sum_{n_2=0}^{D_{ip}-1} w_N^{(n_1+D_{op}n_2)} x_{(n_1+D_{op}n_2)}\right] w_P^{n_2k_3}\right\} w_{D_{op}}^{n_1k_3},\end{aligned} \tag{27}$$

which could be simplified as:

$$\begin{aligned}\mathbf{X}_{(k)} = \mathbf{X}_{(k_1+D_{ip}k_2)} = \mathbf{X}_{(k_1+D_{ip}(k_3+D_{op}k_4))} &= \sum_{n_1=0}^{D_{op}-1}\left[w_M^{n_1k_2} z(k_2,n_1,k_1)\right] \\ = \sum_{n_1=0}^{D_{op}-1}\left[w_M^{n_1k_2}\sum_{n_2=0}^{D_{ip}-1} y(n_2,n_1,k_1)\, w_N^{n_2[\![k_2]\!]_P}\right]&\end{aligned}, \tag{28}$$

where

$$y(n_2,n_1,k_1) = w_N^{(n_1+D_{op}n_2)} x_{(n_1+D_{op}n_2)}. \tag{29}$$

The direct implementation of equation (29) would have a complexity $t_{c\text{-}total}$ expressed as [13]:

$$t_{c-total} = t_{c-input-stage} + t_{c-intermediate-stage} + t_{c-output-stage}\ , \tag{30}$$

The complexity of the input stage in term of complex multiplication t_{cm} is approximated by [13]:

$$t_{c-input-stage} = \left(L_t - 1\right)\left(D_{ip} - 1\right)t_{cm}, \tag{31}$$

The complexity of the input stage in term of arithmetic operation t_o in which the DIT Cooley-Tukey algorithm is implemented is [13]:

$$t_{o-intermediate-stage} = D_{ip}D_{op}\left(4PS_P + P + 6\right) = \frac{N}{P}\left(4PS_P + P + 6\right), \tag{32}$$

and finally the complexity of the output stage is as follow:

$$t_{c-output-stage} = \begin{cases} L_o\left(D_{op} - 1\right)\times t_{a/s} & for\ L_o \le D_{op} \\ \left(L_o - D_{ip}\right)\left(D_{op} - 1\right)\times t_{cm} & otherwise \end{cases}\ . \tag{33}$$

where $S_P = \log_2 P - 1$ and ta/s is the complexity of the complex addition/subtraction.

Similarly to the same concept introduced in the input pruning and based on the radix *r* factorization introduced in [9] and [10], we can rewrite equation (5) as:

$$\begin{aligned} \mathbf{X}_{(k)} &= \sum_{n=0}^{N-1} x_{(n)} W_N^{nk} = \mathbf{X}_{\left(k_1 + D_{ip}k_2\right)} \\ &= \sum_{n_1=0}^{D_{op}-1}\sum_{n_2=0}^{P-1} x_{\left(n_1 + D_{op}n_2\right)} W_N^{\left(n_1 + D_{op}n_2\right)k_1} W_M^{\left(n_1 + D_{op}n_2\right)k_2}\ , \\ &= \sum_{n_1=0}^{D_{op}-1} W_N^{n_1\left(k_1 + D_{ip}k_2\right)} \sum_{n_2=0}^{P-1} x_{\left(n_1 + D_{op}n_2\right)} W_N^{D_{op}n_2k_1} W_P^{n_2k_2} \end{aligned} \tag{34}$$

where we have assumed:

$$
\begin{aligned}
n_1 &= n_3 + D_{op} n_4 \\
n_3 &= 0,1,\cdots, D_{op} - 1. \\
n_4 &= 0,1,\cdots, P-1
\end{aligned}
\tag{35}
$$

By adopting

$$
\begin{aligned}
k_2 &= k_3 + P k_4 \\
k_3 &= 0,1,\cdots, P-1 \quad , \\
k_4 &= 0,1,\cdots, D_{op} - 1
\end{aligned}
\tag{36}
$$

we can rewrite equation (34) as:

$$
\mathbf{X}_{(k)} = \mathbf{X}_{\left(k_1 + D_{ip}(k_3 + Pk_4)\right)} = \sum_{n_1=0}^{D_{op}-1} w_N^{n_1(k_1 + Dipk_2)} \sum_{n_2=0}^{P-1} x_{\left(n_1 + D_{op} n_2\right)} w_N^{D_{op} n_2 k_1} w_P^{n_2 [\![k_2]\!]_P} \tag{37}
$$

Equation (37) could be broken into two equations where we have assumed that

$$
k_3 = [\![k_2]\!]_P , \tag{38}
$$

and as a result equation (37) would be:

$$
\mathbf{X}_{(n_1,k_1,k_2)} = \sum_{n_2=0}^{P-1} x_{\left(n_1 + D_{op} n_2\right)} w_N^{D_{op} n_2 k_1} w_P^{n_2 [\![k_2]\!]_P} \quad . \tag{39}
$$

where

$$
\mathbf{X}_{(k)} = \mathbf{X}_{\left(k_1 + D_{ip} k_2\right)} = \sum_{n_1=0}^{D_{op}-1} X_{(n_1,k_1,k_2)} w_N^{n_1\left(k_1 + D_{ip} k_2\right)} . \tag{40}
$$

With the same reasoning as above by incorporating the twiddle factors $w_N^{D_{op} n_2 k_1}$ and the adder tree matrices into a single stage of calculation and by incorporating $w_N^{n_1 k_1}$ and $w_N^{D_{ip} k_2}$ into a single stage of calculation, equation (39) could be expressed as:

$$
\begin{aligned}
&\mathbf{X}_{(n_1,k_1,k_3)} = \mathbf{T}_r \mathbf{W}_N \mathbf{X} \\
&= \mathbf{T}_r \mathbf{W}_N col\left(\sum_{n_2=0}^{V-1} x_{\left(n_1 + D_{op}(rn_2 + l)\right)} w_V^{nlv} w_{\frac{N}{r}}^{n_2 l k_1 D_{op}} \,\middle|\, l = 0,1,\cdots, r-1 \right)
\end{aligned}
\tag{41}
$$

for $v = 0,1,..,V-1$ and with

$$\mathbf{X} = \left(\sum_{n_2=0}^{V-1} x_{\left(n_1 + D_{op}(rn_2+l)\right)} w_V^{nlv} w_{\frac{N}{r}}^{n_2 l k_1 D_{op}} \middle| l = 0,1,\cdots,r-1 \right)^T , \tag{42}$$

$$\mathbf{W}_N = diag\left(w_N^{D_{ip} lv} w_N^{n_2 l k_1 D_{op}} \middle| l = 0,1,\cdots,r-1 \right). \tag{43}$$

Finally, the input/output pruning FFT algorithm which computes any of the P consecutive outputs of the FFT spectrum could be expressed as [7]-[10]:

$$\begin{aligned} \mathbf{X}_{\left(k_1 + D_{ip} \times k_2\right)} &= \mathbf{X}_{\left(k_1 + D_{ip}[\![k_2]\!]_P\right)} \\ &= \sum_{n_1}^{D_{op}-1} w_N^{n_1\left(k_1 + D_{ip}[\![k_2]\!]_P\right)} X_{\left(k_1 + D_{ip} \times [\![k_2]\!]_P, s+1, v, WAG\right)} \\ &= \sum_{n_1=0}^{D_{op}-1} w_N^{n_1\left(k_1 + D_{ip}[\![k_2]\!]_P\right)} \sum_{n_2=0}^{r-1} X_{\left(k_1 + D_{ip} \times [\![k_2]\!]_P, s, v, RAD\right)} w_M^{CAG} \end{aligned} , \tag{44}$$

where the *WAG* is expressed in equation (21) and the *RAD* is expressed in equation (22) and the DIT Coefficient address generator of the DIT JMIOPFFT (Jaber Massicotte Input Output Pruning FFT) could be obtained as:

$$CAG = \left[\!\!\left[nD_{op}\left(D_{ip}\left(lV + \left\lfloor \frac{v}{r^{(S_P - s)}} \right\rfloor r^{(S_P - s)} \right) + \left(r^{(S_P - s)} k_1 \right) \right) \right]\!\!\right]_N , \tag{45}$$

where $[\![x]\!]_N$ represents the operation x modulo N, $\lfloor x \rfloor$ represents the integer part operator of x, the indices are $v = 0,1,\ldots,V-1$, and $s = 0,1,\ldots,S_P$ where r is the radix-r, V is the number of words ($V = P/r$), and S_P is the number of stages ($S_P = \log_r P - 1$).

4. Performance Evaluation

The complexity of the proposed algorithm t_c is computed as follow:

$$t_{c-Iimput/Output-JMIOPFFT} = t_{cm}\left(L_o D_{op}\right) + D_{ip} D_{op} FFT_P + t_{ca/s}\left(D_{ip} D_{op} + 2L_o D_{op}\right), \qquad (46)$$

therefore; the complexity of the proposed IOPJMFFT which is the same as the DIT Cooley-Tukey algorithm would be exactly:

$$\begin{aligned} t_{c-Iimput/Output-JMIOPFFT} &= 6N + \frac{N}{P}\left(4PS_P + P + 6\right) + 2\left(\frac{N}{P} + 2N\right) \\ &= 10N + \frac{N}{P}\left(4PS_P + P + 8\right) \\ &= N\left(4PS_P + 11P + 8\right) \end{aligned} \qquad (47)$$

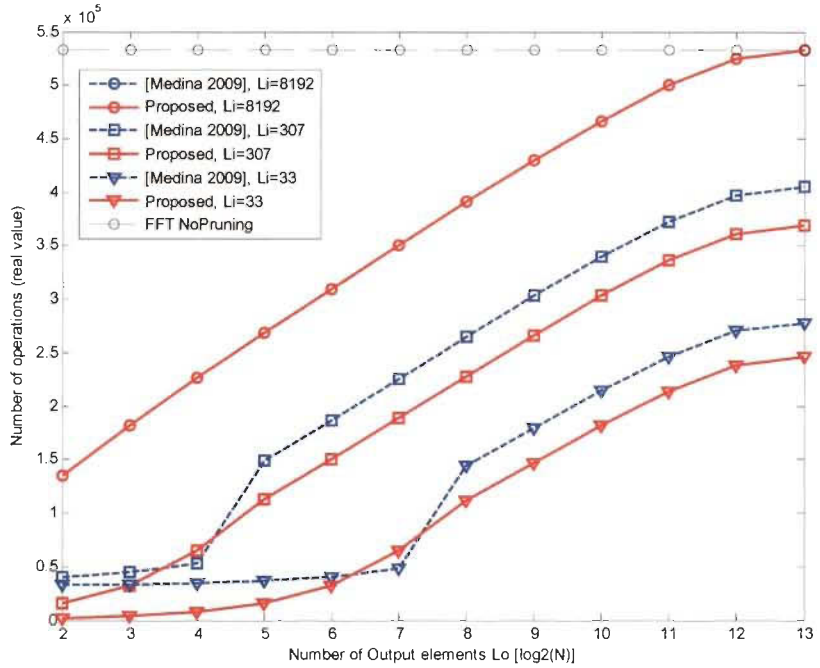
The complexity comparison between our proposed method and the one cited in reference [13] that uses the radix-2 DIT Cooley-Tukey is illustrated in Figure 2. Figure 2 reveals that both methods have the same complexity for L_i = 8192 and that our method for L_i = 307 is approximately equivalent to the cited method for L_i = 33 for the number of outputs is greater than 2^8. The operation's reduction ratio between the proposed method and the cited method is presented in Figure 3 where our proposed method manifested a gain that ranges between 1.4 and 1.2 for L_i = 307 and L_o>2^8.

According to equation (33), it seems that the output stage would be costly in implementation for Lo > *Dip* therefore, the author in [13] has combined the direct method and the 2 BF filtering method proposed by Sorensen et al [14] for 1< *Dop* ≤ 4 in order to achieve a gain estimated at 30% for large N that should be weighed against the loss in precision as shown in Figures 4-a and 4-b in which the SQNR was computed as:

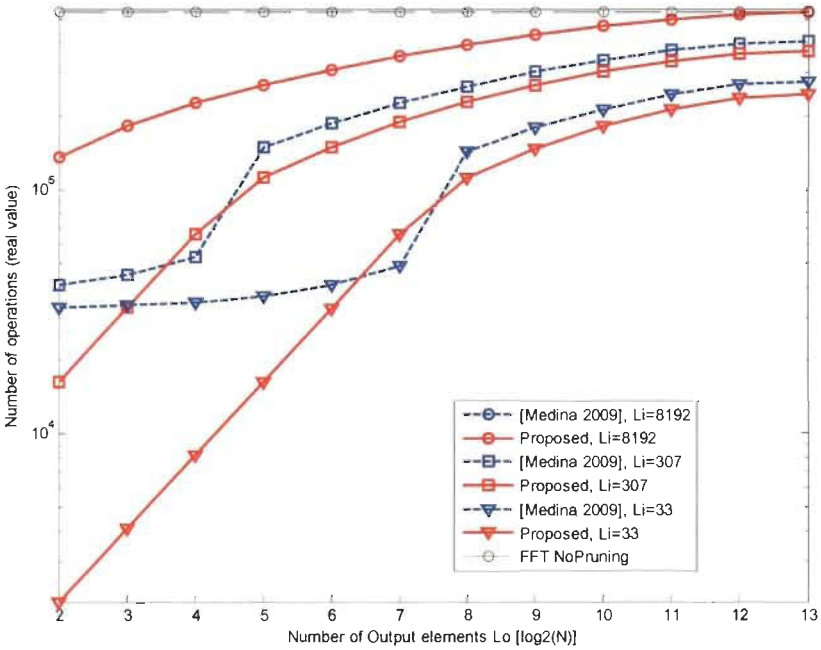
$$SQNR = 10\log_{10}\left(\frac{\|X_{Matlab}\|_2}{\|X_{Matlab} - X_{Method}\|_2}\right) \qquad (48)$$

where $\|X\|_2$ is the norm function of X, X_{Matlab} is the results obtained with fft.m Matlab® function meanwhile X_{Method} is the results obtained by either our proposed method,

IOPJMFFT, or the cited method [13] where all results were obtained in floating point using Matlab®.

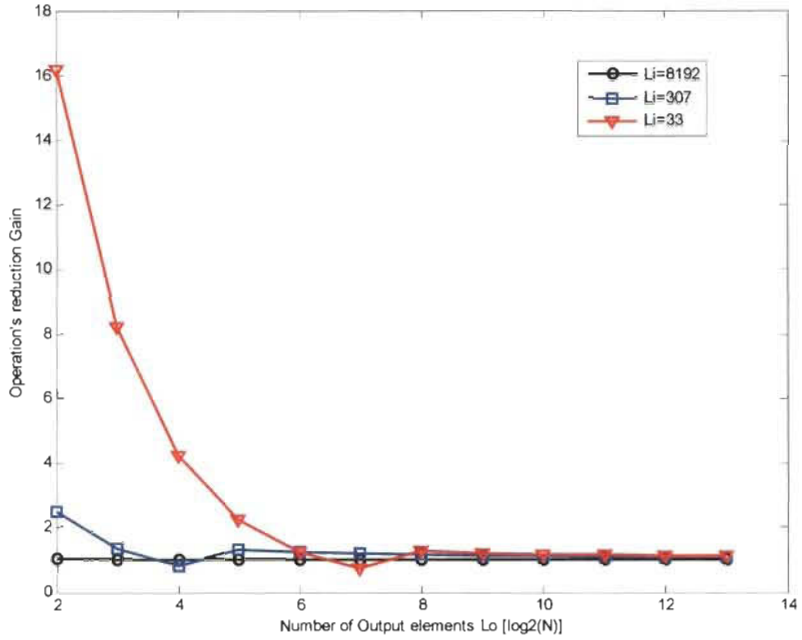


a)

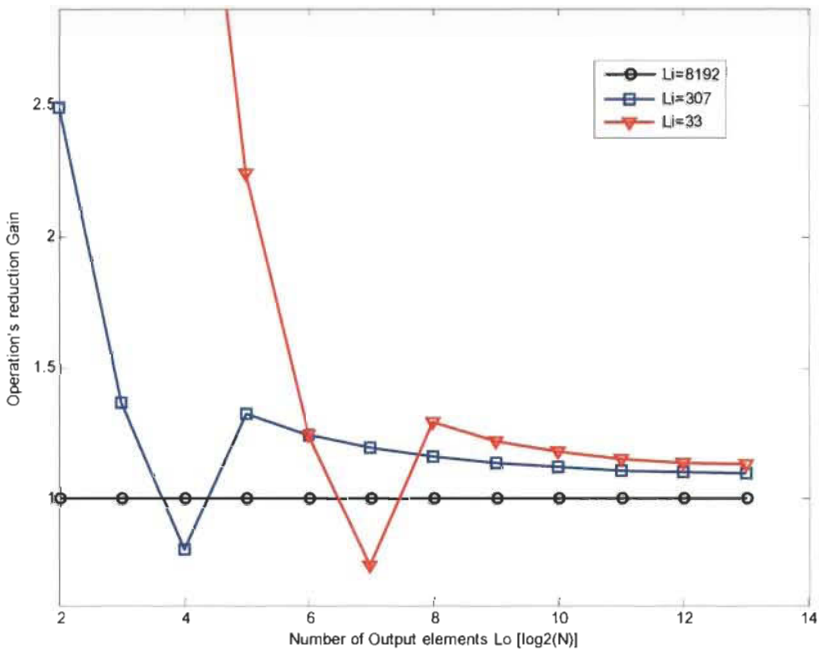


b)

Figure 2: Number of operations required by the one cited in reference [13] in direct implementation and our proposed method for N=8192 and Li = 8192, 307 and 33 for linear (a) scale and logarithmic scale (b).

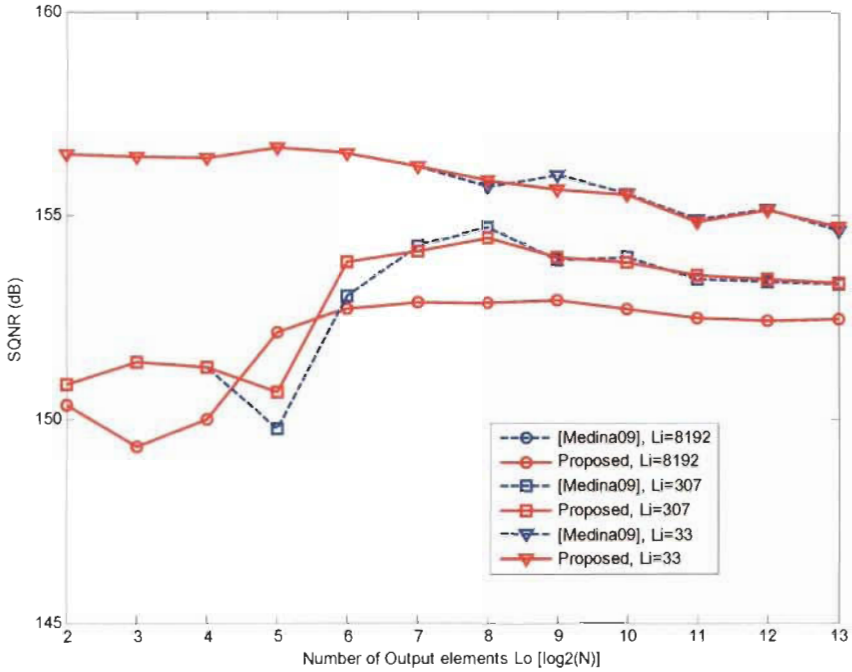


a)

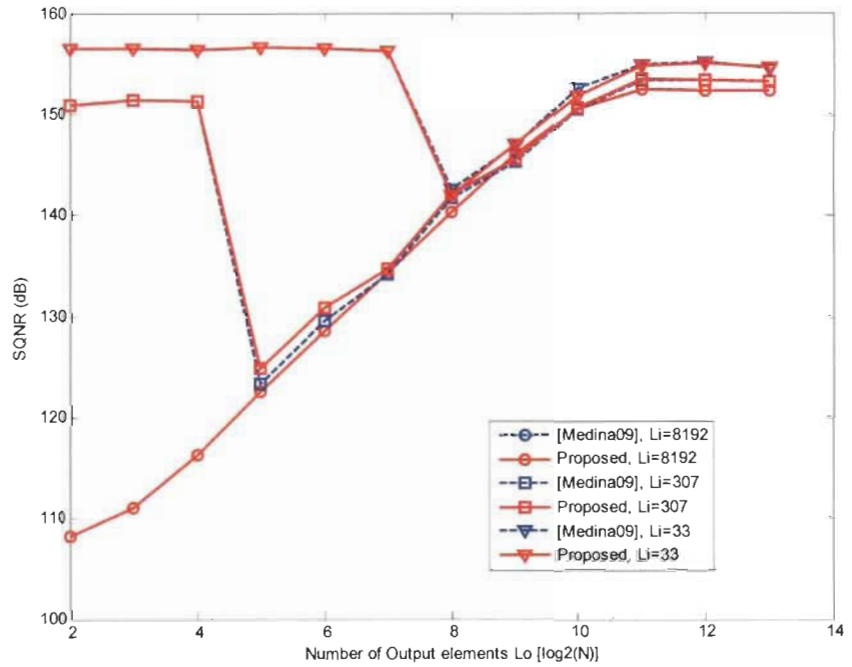


b)

Figure 3: Operation's reduction ratio between the proposed and the cited method in a) and b) is the zoomed version of figure 4a.



a)



b)

Figure 4: Precision based on SQNR, a) without 2BF filtering and b) by using the 2 BF filtering.

Figure 5 represents the complexity comparison between our proposed method and the cited one for N=1024 and $L_i = Lo$ by using the 2BF filter method meanwhile the operation's reduction ratio is revealed in Figure 6.

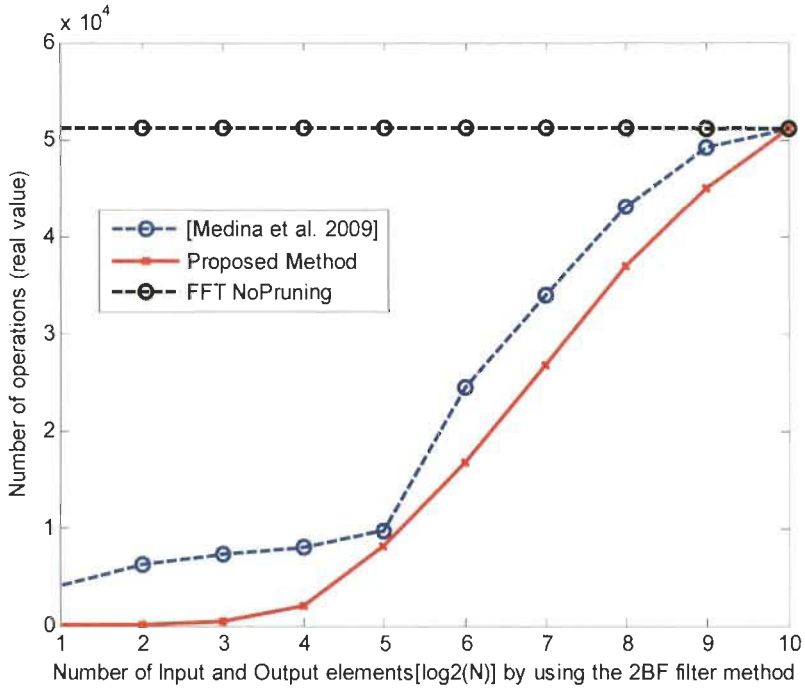


Figure 5: Comparison between the proposed and the cited pruned FFTs for N =1024 and $L_i = Lo$ by using the 2BF filter method.

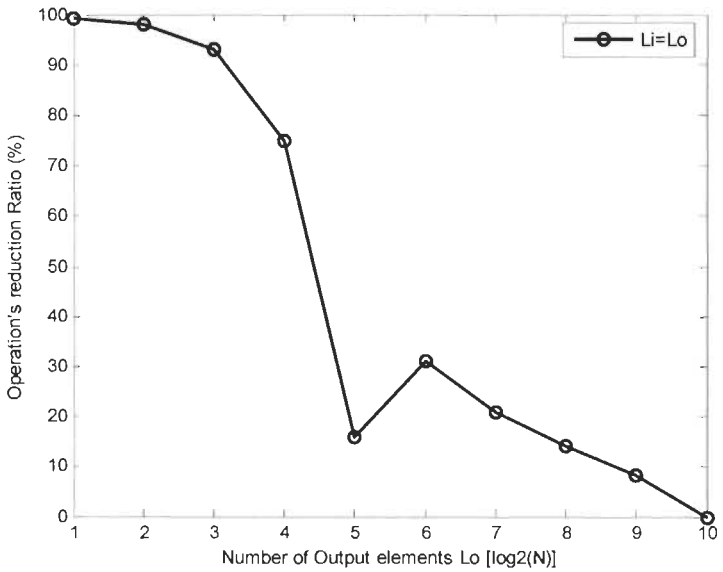


Figure 6: Operation's reduction ratio between the proposed and the cited method by using 2 BF method when $L_i = L_o$ and N = 1024.

5. Conclusion

Finally, this paper has presented an efficient input/output pruning FFT algorithm that reduces the complexity and the computational effort in comparison to the most recent proposed methods. The work put forth by this article on improvement of the input/output pruning FFT is a key contribution to advances in wireless communications. Reduction in computational time offered by the proposed method finds applications in the most recent wireless industry such as OFDMA and LTE technologies.

References

[1] Z.Hu and H. Wan, "A novel generic fast Fourier transform pruning technique and complexity analysis," IEEE Trans. Signal Process., Jan.2005 Volume: 53, page(s): 274- 282.

[2] R. Rajbanshi, A. M. Wyglinski, and G. J. Minden, "An Efficient Implementation of NCOFDM Transceivers for Cognitive Radios", IEEE CrownCom 2006

[3] Ta. Kumura, M. Ikakawa et al, "VLIW DSP for mobile Applications", IEEE Signal processing magazine July 2002

[4] Shousheng He, Torkelson, M. VLSI computation of the partial DFT for (de)modulation in multi-channel OFDM system, IEEE PIMRC 1995 Sep 1995

[5] Murphy, C. D. "Low-complexity FFT structures for OFDM transceivers", IEEE Trans. on Signal Process., Volume: 50, Issue: 12 On page(s): 1878-1881, Dec 2002.

[6] A. W. M Vanden Enden and N.A.M Verhoeckx " Traitement Numérique du signal", Masson 1992.

[7] M. Jaber and D. Massicotte "A Novel Approach for the FFT Data Reordering", ISCAS10, Paris, France 2010

[8] M Jaber "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 B2 and European patent application Serial no: PCT/US01/07602

[9] M. Jaber and D. Massicotte "A New FFT Concept for Efficient VLSI Implementation Part I: Butterfly Processing Element", DSP'2009, July 2009 Santorini Greece.

[10] M. Jaber, "Butterfly Processing Element for Efficient Fast Fourier Transform Method and Apparatus", US Patent No. 6,751,643, 2004.

[11] M Jaber "Parallel Multiprocessing for the Fast Fourier transform with Pipeline Architecture" US Patent No. 6, 792, 441.

[12] M. Jaber and D. Massicotte "A New FFT Concept for Efficient VLSI Implementation Part II: Parallel pipelined Processing", DSP'2009, July 2009 Santorini Greece.

[13] M. Medina-Melendrez, M. Arias-Estrada and A. Castro, "Input and/or Output Pruning of Composite Length FFTs Using a DIF-DIT Transform Decomposition", IEEE Transactions on Signal Processing, Vol. 57, No.10, pp. 4124-4128, October 2009.

[14] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input or output points," IEEE Transactions on Signal Processing, vol. 41, no. 3, pp. 1184–1199, Mar. 1993.

Chapter 7 Conclusion and Future Works

In this Thesis, we gave a solution for the FFT's parallel multiprocessing problem, where two mathematical models have revealed the global philosophy and the detailed strategy, in which the method was elaborated upon in a chronological order.

For the first Model, this thesis has developed and presented a radix-*r* fast Fourier butterfly that reduces the computational effort (as measured by the number of multiplications) by a factor of *r* in comparison to the most proposed radix-*r* FFT butterflies. The conceptual key to the modified radix–*r* FFT butterfly is the formulation of the radix–*r* as composed engines with identical structures and a systematic means of accessing the corresponding multiplier coefficients. This enables the design of an engine with the lowest rate of complex multipliers and adders which utilizes *r* or *r*-1 complex multipliers in parallel to implement each of the butterfly computations. There is a simple mapping from the three indices (FFT stage, butterfly, and element) to the addresses of the multiplier coefficients needed. For a single processor environment, this type of PE with *r* parallel multipliers would result in decrease in time delay for the complete FFT by a factor of O(*r*).

Furthermore, this thesis has proven that the higher radix FFT algorithms are advantageous for the hardware implementation, due to the reduced quantity of complex multiplications and memory access rate requirements. In this thesis, we showed that the implementation of a radix-*r* PE for the FFT is feasible by maintaining one complex multiplier in its critical path compared to radix-*r* butterfly as it was elaborated in paper I presented in chapter 2.

High performance parallel computing is essential for solving very large and complex scientific and engineering problems within a reasonable amount of computation time. These two mains tasks must be carried out in order to deliver a proper parallel computing

solution to a specific problem, and they involve choosing the appropriate parallel VLSI implementation. In this thesis we proposed a solution to the FFT's parallel multiprocessing problem, wherein the mathematical model described the global philosophy and the detailed strategy, and its resolution method was presented in chronological order. We have clearly shown that our proposed butterfly processing element structure in is an effective solution for the pipelined FFT implementation by means of higher radix butterflies. This objective was achieved by reducing the complexity of the critical path contrarily the conventional radix-r method. On the other hand, we clearly showed that the number of S stages in a pipelined architecture could be reduced through implementing our parallel method that boosts the FFT's execution time as it was demonstrated in paper II of chapter 2.

In addition to that this thesis has presented has presented an efficient way of implementing the FFT process by mean of the one iteration radix-r kernel where a serial parallel model, and a pure parallel model have been represented from which we have derived the JM-filter that is used to detect specific frequencies in monitored signals. The key contribution to our proposed JM-filter elaborated in chapter 4 (Papers V, VI, and VII) is that we have reduced the multiplication complexity by a factor of r compared to Goertzel's filter which is mostly used to detect specific frequencies in monitored signals and DNA analysis.

As we have seen that the FFT algorithm is especially memory access and storage intensive where the most studious task in this process is the data flow control. As we know that the butterfly, main function is to multiply the input data with its corresponding coefficient multipliers in order to compute the transform. As a result, a tool that could control efficiently the data flow would increase the overall system's performance. The FFT

Address Generator presented in this thesis, has detailed an embodied address generator for use with a variety of FFT algorithms in which the address generator is typically used to compute the addresses (locations in memory) where input data, output data and twiddle coefficients will be stored and retrieved from memory and based on this scheme, the design of the address generators is greatly simplified. In addition to the simplicity of structure, the speed of the address generators is by a considerable amount increased as it was proven in paper IV of chapter 3. Therefore, the present thesis has presented a novel approach for the FFT data reordering algorithms that boosted the FFT execution compared to recent bit reversing techniques where the implementation of this method would be recommended on low power DSP processors and this is achieved by reducing the memory usage by $N/2$ which is used as storage table of $N/2$ index numbers. By doing so the size and the power consumption of such a processor will be reduced which are desirable for portable devices?

One of the most relevant contributions of this thesis is the development of an efficient ordered input ordered output radix-2^3 algorithm that reduces the complexity and the computational effort in comparison to the most recent proposed methods. Furthermore, the proposed method had shown an execution time reduction in term of clock cycles compared to the cited methods and by predicting the 8^{th} root of unity in which one memory access to the coefficient multiplier is needed per 8 inputs and the memory size needed to stock the coefficient multiplier is reduced to $N/8$ as it was proven in papers VIII and IX of chapter 5.

Input pruning FFTs are commonly used in the padded FFT process which is known as the up-sampling process in digital signal processing, which consists of extending a signal (or spectrum) with zeros. By doing so, this can increase the time sampling which is translated into forcing the FFT algorithm to sample the spectrum at smaller frequency

intervals. Output pruning FFT is a method used to compute a discrete Fourier transform where only a subset of the outputs is needed. Our second significant contribution on this level is by developing an input/output pruning FFT that offers a reduction in the computational load compared to the most-recent recent method as it was revealed in paper X of chapter 6.

Our future work will be devoted to compare the JM-filter to the input/output pruning FFT and to integrate the JMFFT's 2^3 kernel core into the input/output pruning FFT which is mostly used in OFDM wireless communication system. In many applications such as the OFDM based Cognitive Radio will be principally relying on FFT pruning, which applies efficiently zeros to the L_i consecutive non-zero inputs with arbitrary distributions, and this will be the target of our future work due to the fact that the existing algorithms are mainly concentrated on sequences that have L_i consecutive non-zero input points at the beginning. Future work will also include the implementation of our proposed JMFFT on FPGA and ASIC by using our FFT Address Generator.

References

[1] R. Kumar and al, "Speech compression using different transform techniques", 2nd International Conference on Computer and Communication Technology (ICCCT) 2011, 15-17 Sept. 2011, pp. 146 – 151.

[2] D. Guel, J. Palicot, "FFT/IFFT Pair Based Digital Filtering for the Transformation of Adding Signal PAPR Reduction Techniques in Tone Reservation Techniques", Fifth International Conference on Wireless and Mobile Communications, 2009. ICWMC '09, Cannes, La Bocca, Aug. 2009, pp. 200 -204.

[3] E. Angelini and al "FFT-based imaging processing for cultural heritage monitoring", Instrumentation and Measurement Technology Conference (I2MTC), 2010 Austin Texas pp. 1106 - 1111

[4] P. Bishop, "Radar Measurements Utilizing FFT Spectrum Approach", IEEE Systems, Applications and Technology Conference, LISAT 2006, Long Island, May 2006 pp. 1

[5] T. Sansaloni, A. Pe´rez-Pascual, V. Torres and J. Valls, "Efficient pipeline FFT processors for WLAN MIMO-OFDM systems", Electronics Letters 15th September 2005 Vol. 41 No. 19 pp. 1043 - 1044.

[6] J. Yeol and M. Lim, "New Radix-2 to the 4Th Power Pipeline FFT Processor", IEICE Trans. Electron. Vol. E88-C, NO.8, August 2005.

[7] J. Kuo, C Wen, C Lin and A Wu, "VLSI Design of a Variable-Length FFT/IFFT Processor for OFDM-Based Communication Systems, EURASIP Journal on Applied Signal Processing 2003:13, 1306–1316.

[8] G. Klang, "A Study of OFDM for Cellular Radio Systems, M.Sc. Thesis, Linköping University, Sweden 1994.

[9] W. Han, T. Arslan, A. T. Erdogan and M. Hasan, "Multiplier-Less Based Parallel-Pipelined Fft Architectures For Wireless Communication Applications", ICASSAP 2005 ,Vol. 5, pp. v/45 - v/48.

[10] W. Li, J. Carlsson, J. Claeson, and L. Wanhammar, "A GALS Based 16-Point Pipeline FFT Core", Electronics Systems, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden

[11] V. Szwarc and L. Desormeaux, "A Chip Set for Pipeline and Parallel Pipeline FFT Architectures, Journal of VLSI Signal Processing, 1994 pp. 253-265

[12] W. H. Press, S. A. Teuklosky, W. T. Vetterling and B. P. Flannery, "Numerical Recipes in C" second edition, Cambridge University Press, 1996.

[13] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. Comput., April 1965, pp. 297-301.

[14] W. M. Gentleman and G. Sande, "Fast Fourier Transform-For fun and profit", Proceedings-Fall Joint Computer Conference, AFIPS Proc., Vol. 29, Washington D.C. 1966: Spartan, pp. 563-578

[15] T. Widhe, "Efficient Implementation of FFT Processing Elements" Linköping studies in Science and Technology, Thesis No. 619, Linköping University, Sweden, June 1997.

[16] S. Winograd, "On Computing The Discrete Fourier Transform", Proc. Nat. Acad. Sci. USA, April 1976, Vol. 37, pp. 1005-1006.

[17] P. Duhamel, and H. Hollman, "Split Radix FFT Algorithm", Electronics Letters, Vol. 20, Jan. 1984 No. 1, pp. 14 – 16.

[18] R. C. Singleton, "A short bibliography on the fast Fourier transform", IEEE Trans. Audio Electroacoust., Vol. AU-17, pp. 166 - 169 June 1969.

[19] W. M. Gentleman and G. Sande, "Fast Fourier Transform-For fun and profit", Proceedings-Fall Joint Computer Conference, AFIPS Proc., Vol. 29, Washington D.C.: Spartan, pp. 563-578, 1966.

[20] T. Widhe and al, "A 1-GS/s FFT/IFFT Processor for UWB Applications" 1999 IEEE International Symposium on Circuits and Systems, June 9-12, 1997. Hong Kong, pp. 2084 – 2087.

[21] Y. Lin and al, "Design of Efficient Radix-8 Butterfly PEs for VLSI" IEEE journal of solid-state circuits, Vol. 40, No. 8, Agust 2005, pp. 1726 – 1735.

[22] R. C. Singleton, "An Algorithm for Computing the Mixed radix Fast Fourier Transform", IEEE Transactions on Audio and Electro-acoustics, Vol. AU-17, No. 2, pp. 93-103, June 1969.

[23] R. C. Agarwal, F. G. Gustavson, and M. Zubair, IBM J. Res. 38, 563, 1994.

[24] A. V. Oppenheim and R. Schafer, "Discrete-Time Signal Processing", Prentice-Hall, Englewood Cliffs, New Jersey, 1989

[25] H. F. Silverman, "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-25, No. 2, pp. 152-165, April 1977.

[26] JR. Yavne, "An economical method for calculating the discrete Fourier transform," in Proc. AFIPS Fall Joint Computer Conf. 33, 115–125 (1968).

[27] C. M. Rader and N. M. Brenner. "A new principle for fast Fourier transformation," IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-24, pp. 264-265 1976.

[28] K. M. Cho and G. C. Temes, "Real-factor FFT algorithms," in Proc. IEEE ICASSP, 1978 pp. 634-637.

[29] R. D. Preuss, "Very fast computation of the radix-2, discrete Fourier transform," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-30, pp. 595-607 1982.

[30] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," Signal Processing, vol. 6, pp. 267-278, July 1984

[31] Z. Wang, "Fast algorithms for the discrete Wtransforms and the discrete Fourier transform," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-32, pp. 803-816, Aug. 1984.

[32] J. B. Martens, "Recursive cyclotomic factorization-A new algorithm for calculating the discrete Fourier transform," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-32, pp. 750-761, Apr. 1984.

[33] H. V. Sorenson, M. T. Heideman, and C. S. Burrus, "On Computing the Split Radix FFT, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 1, pp. 152-156, Feb. 1986.

[34] M. Richards, "On Hardware Implementation of the Split-Radix FFT, IEEE trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-36, No. 10, pp. 1575-1581, Oct. 1988.

[35] P. Duhamel, and M. Vetterli, "Fast Fourier Transform, IEEE Trans. on Computers, Vol. C-19, No. 11, N0v. 1970.

[36] M. Torkelson, "A New Approach to Pipeline FFT Processor", Proc. IPPS 1996, p 766

[37] E. Carr, L. G. Cuthbert and A radix 4 delay Commutator for fast Fourier transform processor implementation". IEEE J. Solid-State Circuits, SC-19(5), Oct. 1984 pp. 702-709

[38] R. Storn. "Radix-2 FFT pipeline architecture with reduced noise-to-signal ratio", IEE Proc. Vis. Image Signal Process., Apr. 1994 pp. 81-86.

[39] W. Chang and Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms", IEEE Transactions on Signal Processing, vol. 56, no. 10, October 2008, pp. 4673-4682.

[40] L. A. Bowman, N. B. MacDonald and M. P. Taylor, ``The Exploitation of Parallel Computing in Major UK Companies" in G. R. Joubert, D. Trystram, F. J. Peters and D. J. Evans (editors), Parallel Computing: Trends and Applications, Elsevier Science B. V., 1994 pp. 57-63.

[41] M. Allen, "Parallel Mesh Decomposition and Generation", EPCC-SS93-1, 1993.

[42] M. Flynn, Very High-Speed Computing Systems, Proceeding of the IEEE, 54(12), December 1966, p1901-1909

[43] M. Flynn, "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Sept. 1972, Vol. 24, No. 9, pp. 948-960

[44] M. Flynn, "Parallel Processors Were the Future ... and May Yet Be", IEEE Computer, Jan.1992, Vol. 29, No.1, pp. 82-94

[45] M. Flynn and K. Rudd, "Parallel Architectures", ACM Computing Surveys, Vol. 28, No. 2, March 1996, pp. 67-70.

[46] J. Prince and A. B. Rosen's 70th Birthday. Edited by Panos M. Pardalos. Published by Amsterdam; New York: North-Holland, 1992.

[47] Y. Wang and al, "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors", IEEE Transactions on signal processing, Vol. 55, No. 5, May 2007 pp. 2338 – 2349

[48] M. Medina-Melendrez, M. Arias-Estrada and A. Castro, "Input and/or Output Pruning of Composite Length FFTs Using a DIF-DIT Transform Decomposition", IEEE Tans. On Signal Processing, Vol. 57, No.10, October 2009 pp. 4124 - 4128.

[49] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input or output points," IEEE Trans. Signal Process., Mar. 1993, vol. 41, no. 3, pp. 1184–1199,

[50] R. Heaton and B. Hodson, "A Fine Frequency and Fine Sample Clock Estimation Technique for OFDM Systems", IEEE VTS 53rd Vehicular Technology Conference, 2001. VTC 2001 Spring vol.1, pp. 678 - 682

[51] Q. Chen and al, "Progressive Automatic Detection of OFDM System Parameters for Universal Mobile DTV Receiver", IEEE 72nd Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 pp. 1-5.

[52] P. M. Frigo "A Modified Split-Radix FFT With Fewer Arithmetic Operations", IEEE Transactions On Signal Processing VOL. 55, No. 1, January 2007, pp. 111 - 119.

[53] E. Kim and M. Sunwoo, "High speed eight-parallel mixed-radix FFT Processor for OFDM systems" IEEE International Symposium on Circuits and Systems (ISCAS), 2011, pp. 1684 – 1687.

[54] M. Shin and H. Lee, "A High-Speed Four-Parallel Radix-2^4 FFT/IFFT Processor for UWB Applications" 2008 IEEE International Symposium on Circuits and Systems, ISCAS 2008, pp. 960 – 963.

[55] M Jaber "Parallel Multiprocessing for the Fast Fourier transform with Pipeline Architecture" US Patent No. 6, 792, 441.

[56] M Jaber "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 B2 and European patent application Serial no: PCT/US01/07602

[57] FFTW, http://www.fftw.org, (visited in 2009).

[58] S. Bouguezel, M. Omair Ahmad and M. N. S. Swamy, "A General Class of Split-Radix FFT Algorithms for the Computation of the DFT of Length-2^m", IEEE Trans. on Signal Processing, Vol. 55, No. 8, pp. 4127 - 4138 AUGUST 2007

[59] Z.Hu and H. Wan, "A novel generic fast Fourier transform pruning technique and complexity analysis," IEEE Trans. Signal Process., Jan.2005 Volume: 53, page(s): 274- 282.

[60] S. Chen, and al, "Cognitive radio-enabled distributed cross-layer optimization via genetic algorithms", IEEE CrownCom 2009 pp. 1-6

[61] Ta. Kumura, M. Ikakawa et al, "VLIW DSP for mobile Applications", IEEE Signal processing magazine July 2002 pp. 10-21

[62] Shousheng He, Torkelson, M. "VLSI computation of the partial DFT for (de)modulation in multi-channel OFDM system", IEEE PIMRC 1995 Sep 1995 pp. 1257-1261

[63] Murphy, C. D. "Low-complexity FFT structures for OFDM transceivers", IEEE Trans. on Signal Process., Volume: 50, Issue: 12 pp. 1878-1881, Dec 2002.

[64] E. Wold, A. Despain, "Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementations", IEEE Trans. On Computers, Vol. C-33, No.5, pp. 414-426 May 1984.