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 for higher radix butterflies suitable for pipeline implementation. Based on the butterfly computation introduced by Cooley-Tukey [1], we introduce a novel approach for the factorization of the Discrete Fourier Transform (DFT), by redefining the butterfly computation, which is more suitable for efficient VLSI implementation. This proposed factorization motivated 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 value multiplier in the butterfly critical path for any given *r*. Algorithmic description and performance of low complexity FFT method are considered in this paper and parallel pipelined FFT in a companion paper [15], Part II Parallel Pipelined FFT 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. Frequency analysis provides spectral information for signals that examined or used in further processing such as speech compression, while frequency domain processing allows for efficient computation, of the convolution integrals (for linear filtering), of the correlations integral (for correlation analysis) and in the orthogonal frequency division multiplex wireless communication (OFDM), wherein the Fast Fourier Transform (FFT) is a major key operator [2], [3]. 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 for an equivalent implementation complexity.

The definition of the DFT is represented by the following equation

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

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 $X_{[k]}$ are complex valued 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, known as fast DFT computation algorithms which are based on the divide-and-conquer approach. The principle of these methods is that a large problem is divided into smaller sub-problems that are easier to solve. In the FFT case, the division into sub-problems means that the input data $x_{[n]}$ can be 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 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], [6].

The overall arithmetic operations used to compute an 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 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 that the number of multiplications and the number of stages to execute a FFT decreases [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 proposition is that our proposed BPE structure maintains lower arithmetic operations within the critical path (one complex value multiplier and certain adders). Consequently, we propose a solution for high 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 demystified the radix-r DFT factorization while Section 3 defines the prior art butterfly operation. Section 4 provides a detailed to 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

978-1-4244-3298-1/09/$25.00 ©2009 IEEE

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

for $k = 0,1,\cdots,N-1$. In the summations, the variables r and k are independents of n. We factorize w_N^{rk} in (2)

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

We can consider that $w_N^{rnk} = w_{N/r}^{nk}$ and rewrite (3) as follow

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

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 $q = 0,1,....,r-1$. Therefore, $X_{(k)}$ is replaced by using new indices p and q

$$X_{(k)} = X_{(p+q(N/r))}, \quad (5)$$

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

Considering that $w_{N/r}^{\alpha N/r} = \left(w_{N/r}^{N/r}\right)^{\alpha} = 1^{\alpha} = 1$, therefore Eq. (7) could be expressed as follow

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

Finally, from Eq. (8), the Eq. (1) could be formulated in a matrix-vector equation as follow

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

or

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

We recognize the first and the second matrix, the well known adder tree matrix $\mathbf{T}_r$ and the twiddle factor matrix $\mathbf{W}_N$,

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

$$X_{(p+qN/r)} = w_N^0 \sum_{n=0}^{\frac{N}{r}-1} x_{(rn)} w_{N/r}^{n(p+qN/r)} + w_N^{(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+1)} w_{N/r}^{n(p+qN/r)} + w_N^{2(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+2)} w_{N/r}^{n(p+qN/r)} + \cdots + w_N^{(r-1)(p+qN/r)} \sum_{n=0}^{\frac{N}{r}-1} x_{(rn+(r-1))} w_{N/r}^{n(p+qN/r)} \quad (7)$$

respectively. Equation (10) can be expressed in a compact form as:

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

for $k = 0,1,2,\cdots,N-1$, $p = 0,1,2,....,(N/r)-1$ and $q = 0,1,2,....,r-1$, with

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

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

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

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

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N. \qquad (13)$$

3. In place (or butterfly) computation

Considering the particular case of BPE, we have $N=r$ input-output data, therefore the column vector, $col(\bullet)$, in (11) become

$$col\left(\sum_{n=0}^{0} x_{(rn+q)} w_{N/r}^{np} \middle| q = 0,1,\cdots,r-1 \right) = [x_{(0)}, x_{(1)}, x_{(2)}, \cdots, x_{(r-1)}]^T \quad (14)$$

and the BPE output is expressed as

$$\mathbf{X} = \mathbf{B}_r \mathbf{x}, \qquad (15)$$

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 input and output BPE vectors. $\mathbf{B}_r$ is the butterfly matrix, dim $(\mathbf{B}_r) = r \times r$, which can be expressed as

$$\mathbf{B}_r = \mathbf{W}_N^r \mathbf{T}_r, \qquad (16)$$

for decimation in frequency (DIF) process, and

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N^r, \qquad (17)$$

for decimation in time (DIT) process corresponding to (13). In both cases, DIT and DIF, the twiddle factor matrix, $\mathbf{W}_N$, is a diagonal matrix which is defined by $\mathbf{W}_N = \mathrm{diag}\left(1, w_N^p, w_N^{2p}, \ldots, w_N^{(r-1)p}\right)$ 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).

We define $\left[\mathbf{T}_r\right]_{l,m}$ as the element at the l^{th} line and m^{th} column in the matrix $\mathbf{T}_r$. We rewrite (12) as

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

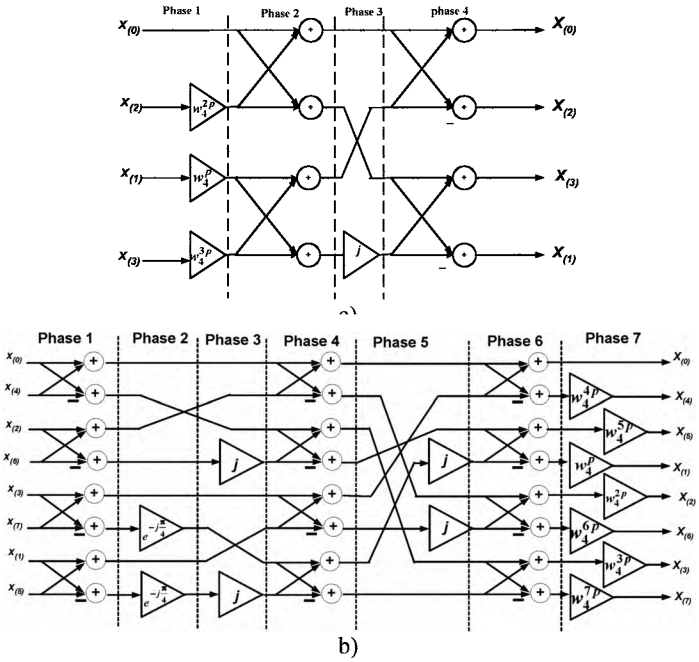


Fig.1 SFG of the a) radix-4 and b) radix-8 butterflies [2]

with l=0,1,...,r-1, m=0,1,...,r-1 and $\left[\!\left[x \right]\!\right]_N$ represents the operation x modulo N.

Knowing that the higher radix will decrease the number of complex value multiplications and the number of stages needed to execute a total N-point FFT, we researchers were consequently motivated to implement a higher radix butterfly. Since the higher radix automatically reduces the communication load, the only problem remaining was the complexity (computational load and phase number) needed to implement the butterfly. 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 radix-4 and 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 or BPEs implementation 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 [2].

4. 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 $\mathbf{W}_N^r$, 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 adder tree matrices into a single stage of calculation.

According to Eq. (16), $\mathbf{B}_r$ is the product of the twiddle factor matrix $\mathbf{W}_N$ and the adder-tree matrix $\mathbf{T}_r$. By defining $\mathbf{W}_{N\,(u,v,s)}$ the set of the twiddle factor matrix as

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

where the indices are $u = 0,1,\ldots,r-1$, $v = 0,1,\ldots,V-1$, and $s = 0,1,\ldots,S-1$ where r is the radix-r, V is the number of words ($V = N/r$), and S is the number of stages ($S = \log_r N$). From Eq. (19), we can define all matrix elements as (l^{th} line, m^{th} column)

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

l=0,1,…,r-1, m=0,1,…,r-1 and $\lfloor \mathrm{x} \rfloor$ represents the integer part operator of x. Therefore, the proposed modified radix-r butterfly computation $\mathbf{B}_r$ is expressed, for the DIF process, as:

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

which we rewrite as

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

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

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

where the l^{th} output is

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

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

5. 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.

From the three indices u, v, and s (the FFT element, butterfly, and stage), a simple mapping takes place to address, using an address generator based on three counters [10], the multiplier coefficients needed.

Fig. 2 shows the radix-r partial BPE for one output element l. Each input $x_{(v,s)}[m]$ is multiplied with the corresponding coefficient $\left[\mathbf{B}_r\right]_{l,u\,(v,s)}$, and thus for m=0,1,…,r-1 we use a complex value multiply accumulator (CMAC) to obtain the l^{th} BPE output. Using a time multiplexing implementation to reduce the single partial BPE as shown in Fig. 2a, we need r computation cycles to complete one BPE radix-r output. Using r multipliers in parallel with r-1 additions, we need one cycle for each l^{th} output.

It was shown in [11] that with further development of the proposed structure in Fig. 2 could yield 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.

The modified radix-r FFT butterfly based on the partial BPEs proves to be a useful aspect, by duplicating it l times as shown in Fig. 3. We increase the hardware resources, but we execute all partial BPE outputs in only one cycle. The structure in Fig. 3 is desirable for implementing single instruction multiple data (SIMD) on some of the latest DSP cards.

Based on Eq. (24), an alternative hardware implementation could be achieved as shown in Fig. 4 and 5 for radix-4 and 8, respectively. The hardware reductions in complex value multipliers and adders are obtained by increasing the complexity

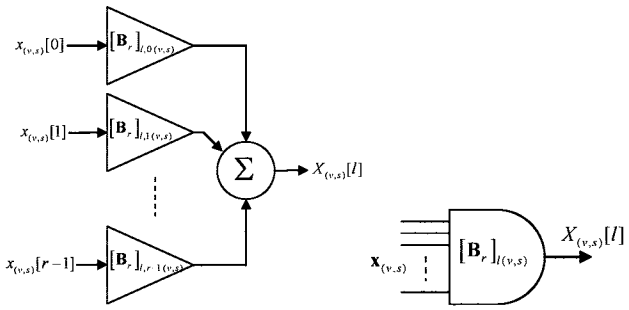


Fig. 2 Radix-r partial BPE (lth output) (a) using $\mathbf{B}_r$ in Eq. (10) and the symbol (b)

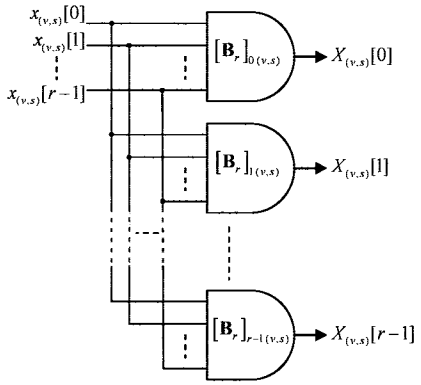


Fig. 3 Maximize the data throughput using r BPE in parallel.

of the butterfly structures.

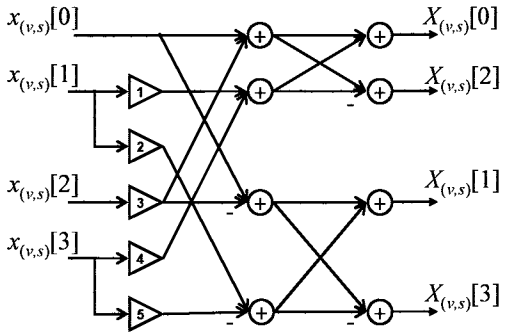
6. PERFORMANCE EVALUATION

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

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

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

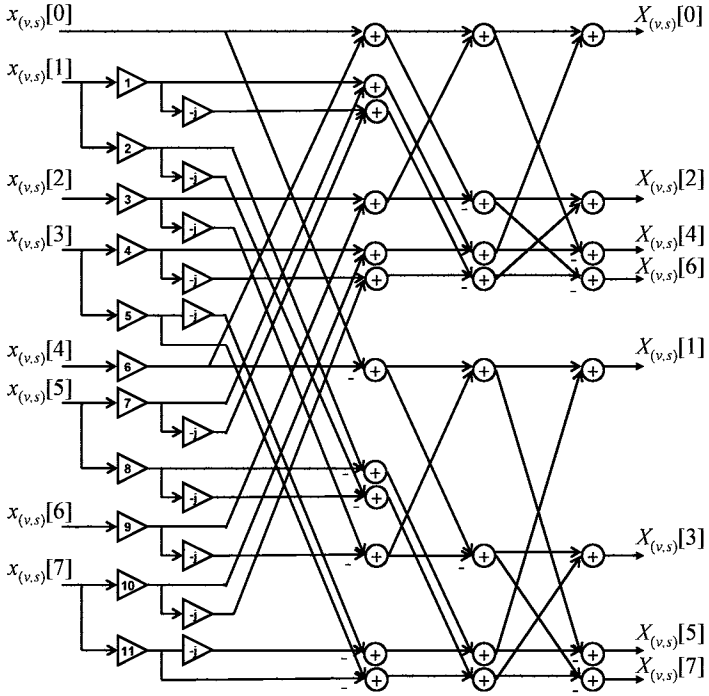
As for computation time, in order to compare our proposed BPE radix-r with the conventional radix-r structures, we assume that all data is present at the input, in a pre-ordered manner. We quantified this comparison based on the critical path delay for one



$$M_1 = w_N^{\left\lfloor (v/4^s)\right\rfloor 4^s}, \quad M_2 = w_N^{\left\lfloor N/4+\left\lfloor (v/4^s)\right\rfloor 4^s\right\rfloor_N}, M_3 = w_N^{\left\lfloor 2\left\lfloor (v/4^s)\right\rfloor 4^s\right\rfloor_N},$$

$$M_4 = w_N^{\left\lfloor 3\left\lfloor (v/4^s)\right\rfloor 4^s\right\rfloor_N}, M_5 = w_N^{\left\lfloor N/4+3\left\lfloor (v/4^s)\right\rfloor 4^s\right\rfloor_N}$$

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



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

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

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

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

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

BPE in order to obtain the first FFT r outputs for the radix-2, 4 and 8 for an FFT of length N points. The conventional radix-r structures shown in Fig. 1, where the BPE critical paths are defined as following: i) radix-4 needs 2 complex value multiplications and 2 complex value additions, and ii) radix-8 needs 4 complex value multiplications and 3 complex value 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), compared to the conventional radix-8 and 16, our proposed FFT BPE have a critical paths 2 and 3 times lower, respectively. Table 1 summarizes the critical path delay based on T_M delays for each BPE.

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

By also assuming that the time delay T_M=4T_A, Table 2 shows the performance result in terms of time computation, 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 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

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 $(n\text{-}1)^{\text{th}}$ to the S^{th} stage for the s^{th} iteration ($s = 0,1,...,S-1$). We considered the switching concept was proposed in [14]. Since r data paths are used, the BPE pipeline achieves a data rate of S times the inter-module clock rate.

Compared to [12] and [13], where a method of deriving very

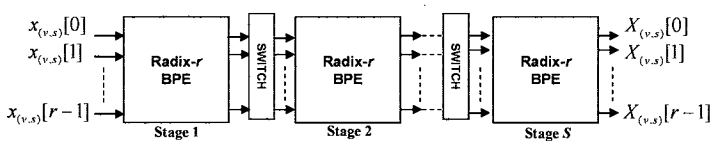


Fig. 6 S Stages Radix-r Pipelined FFT.

fast Fourier transforms (VFFT) was described with radix-16 butterfly was represented as cascaded radix-2 butterfly. By doing so, the 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 FFT for different FFT-length (N). From this we observe that: i) For the

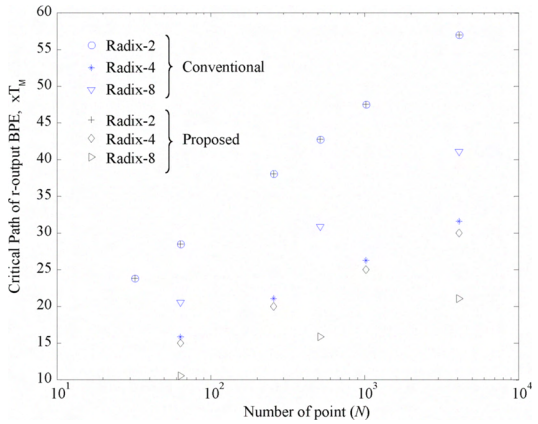


Fig. 7 Critical path delay for *r*-output pipeline FFT conventional and proposed FFT Radix-2, 4 and 8.

conventional case the critical path increases when the $r>4$; ii) The critical path of the proposed FFT decreases 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) need to implement both fixed-point arithmetic operators in the VLSI. We assumed that a complex multiplication can be implemented using 3 real value multiplications and 5 real value additions. We considered 16-bit and 32-bit global lengths for the real multiplication and addition, respectively. Given that we have only one multiplier in the critical path of our proposed model therefore, we can implement the real multiplication with less overall bit word length than the conventional BPE, in order to obtain the equivalent result.

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

7. 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. In radix-8 and 16, we have shown an improvement of critical path delay for 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 – "Part II – Parallel pipelined processing" – [15].

Acknowledgment

The authors would like to thank the financial and technical supports from the Natural Sciences and Engineering Research Council of Canada and the Jabertech Canada inc..

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

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

[4] P. Duhamel, and H. Hollman, "Split Radix FFT Algorithm", Electronics Letters, Vol. 20, No. 1, 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. 1, pp. 111-119, May 2007.

[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 B2 and European patent application Serial no: PCT/US01/07602

[11] M. Jaber and D. Massicotte "The Radix-r One Stage FFT Kernel Computation" ICASSP 2008, 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", J. 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.