Reconfigurable Fast Fourier Transform Architecture for Orthogonal Frequency Division Multiplexing Systems

Konstantinos E. MANOLOPOULOS, Konstantinos G. NAKOS,
Dionysios I. REISIS and Nikolaos G. VLASSOPOULOS
Electronics Laboratory, Department of Physics
National and Capodistrian University of Athens
Physics Buildings IV & V, Panepistimiopolis, Athens, Greece – 15784
Email : dreisis@cc.uoa.gr

Abstract

The performance of the Fast Fourier Transform computation plays a crucial role in the overall operation of OFDM modems. A VLSI architecture realizing the FFT has to perform in real time, adapt to various data rates and accommodate support for power dissipation requirements. This paper presents a VLSI architecture for real time FFT processing. The design involves four (4) radix-4 and one (1) radix-2 processing elements with complex multipliers and can be configured in real time to accommodate FFT computations of length 16, 32, 64, 128 and 256. For further speeding up the entire algorithm, the design can include a technique for parallel accessing the memory banks of each processing element. The validity and the efficiency of the architecture have been shown by an example implementation on FPGAs with throughput: each 256-length FFT at 2.48 usec.

1. Introduction

The spectral analysis of discrete signals plays an important role both in Digital Signal Processing and in Telecommunication Applications. The Fast Fourier Transform (FFT) Algorithm, as introduced by Cooley and Tukey [1], is an important improvement on the Discrete Fourier Transform (DFT) Algorithm [2] due to the achievement of significantly lower computational complexity. Among other technological advancements, the FFT has enabled the development of real time, high-speed applications, such as the Orthogonal Frequency Division Multiplexing (OFDM) Modems [24], [25]. Lately, OFDM systems have been designed to sustain a transmitting rate of 50 Mwords/sec [13].

The OFDM modems exploit the perpendicularity of the sinus and co-sinus as means to generate and transmit a set of symbols through the channel. This procedure requires real time Inverse FFT and FFT processing during the modulation and demodulation of the signal respectively. The performance required by the FFT processing demands either a single processor driven to a very high clock frequency (O(logN) multiplied by the sampling frequency)[12], or alternatively the implementation of an Application Specific Integrated Circuit (ASIC) solution utilizing parallel processing and bit-pipelining techniques[14]. Both solutions have to meet requirements that are extremely demanding in terms of throughput and low power consumption, especially when the applications involve mobile communications and/or battery-powered systems. Moreover, the need for variable data rate in OFDM modems imposes the requirement for computations of various FFT sizes. To accomplish this task, researchers are concerned with the issue of reconfiguring the FFT architecture during subsequent frames (in real time)[13]. In most cases, the solution of an off-the-shelf high-speed processor is shown to be inadequate with respect to the variations of the processing speed requirements and power consumption ([15], [16], [17], [18], [19], [20], [21], [22], [23]).

This paper presents a FFT architecture with four (4) radix-4 processing elements and one (1) radix-2 processing element. Each processing element includes a complex multiplier and uses a word-serial access to each processing element's memory. Each processing element realizes an FFT stage. The radix-4 computation has been chosen because of the requirement of minimizing the round-trip delay of frames in the telecommunication systems using OFDM [13]. The proposed FFT architecture can be configured in real time, in order to perform computations of 16, 32, 64, 128 or 256 points. Also, this paper considers a technique for word parallel accesses to the memory banks of each processing element, which if it is applied, it will improve further the performance of each application.

The FFT architecture can sustain a throughput equal to the length of the FFT (N words) multiplied by the register access time provided by the VLSI technology. An example implementation on Xilinx or Altera Field Programmable Gate Arrays (FPGAs) results in a throughput of 256-points FFT at 2.48 us This result provides an enhanced throughput comparing to the results in the literature [9], [11], [12], [15], [16], [17], [18], [19], [20], [21], [22], [23]. This performance can be further improved by parallelizing the access to the memory, as mentioned above. Moreover, the design is power efficient by using the lowest possible frequency and the minimal subset of processing elements and memory blocks to accomplish the FFT computations.

The paper is organized as follows: The following section presents the problem definition, including relevant work regarding FFT architectures. Section 3 describes the proposed FFT architecture, including the organization of the main blocks and the interconnection between them, the memory management and the overall control unit. Section 4 analyzes the power consumption of the FFT architecture and adapts techniques for power dissipation reduction. Section 5 describes an efficient technique for parallelizing the access to the memory banks of the FFT architecture to enhance the memory to each processing element throughput. Finally, Section 6 concludes the paper.

2. Problem Definition

This Section presents the FFT computation and its application in OFDM systems and related results with respect to the FFT performance.

***The FFT in OFDM systems*:** The OFDM requires a modulation of the 2^N points (the 2^N points form a *symbol*) using an IFFT calculation at the transmitter side (Tx). The corresponding demodulation at the receiver side (Rx) involves an FFT calculation of the 2^N words of the symbol.

As shown in [1] the Decimation In Time FFT (DIT FFT) X[k] of a sequence x[n] is computed by successively decomposing the input signal in odd and even samples and recursively applying the DFT algorithm to the resulting sequences. This can be illustrated as follows:

$$X[k] = \sum_{i=1}^{N} x[i] W_N^{ki} \Leftrightarrow X[k] = \sum_{i=1}^{N/2} x[2i] W_N^{k2i} + \sum_{i=1}^{N/2} x[2i+1] W_N^{k(2i+1)} \Leftrightarrow$$

$$X[k] = \sum_{i=1}^{N/2} x[2i] W_N^{k2i} + W_N^{k} \sum_{i=1}^{N/2} x[2i+1] W_N^{k2i}$$

where: $W_N = \exp(-j2\pi/N)$. Exploiting the inherent symmetry of the twiddle factors and recursively applying the above formula, we obtain the elementary “butterfly” operation used in the calculation of the FFT.

The FFT length and the throughput depend on the OFDM application. Another aspect related to OFDM systems is the variation of the size of the FFT due to variable bit rates. The modem changes the combination of these two parameters dynamically. The principles behind the decisions for varying the parameters are different among OFDM modems. For example, the bit rate should be lowered when the channel conditions deteriorate during a link-up, or increased when channel conditions are better. Also various modem applications impose limits on the power consumption. For instance, mobile transceivers require both variable bit rates and low power operation.

Therefore, the most efficient solution is to process the data at the transmitting (receiving) rate. This implies that the architecture should be organized in a parallel mode and it introduces latency in FFT lengths proportional to the number of processing elements. This latency constitutes an additional design problem that should be addressed. More specifically another issue regarding the FFT implementation in OFDM modems is the round-trip delay of symbols. This delay has to be kept minimal, a fact that leads to solutions of processing elements with radix computation higher than 2.

Related Work: The Fast Fourier Transform (FFT) has been presented in [1] and details of design and applications can be found in [2]. A variety of VLSI architectures is presented in [3] optimized with respect to the AT^2 bound ((Area complexity)*(Time complexity)^2). [4] Refers to the power dissipation and complexity of a pipelined parallel FFT architecture, showing that parallelizing the processing units results in a significant reduction of the power consumption. A different multiplication scheme is presented in [5], applied to a single multiplier CMOS based DSP processors, in order to implement a low-power FIR filter. The multiplier can be used with slight alterations to perform the multiplications in the FFT algorithm as well. [10] Presents the problem of power estimation in VLSI architectures. It also introduces probabilistic techniques to estimate power dissipation and presents a survey on several power estimation techniques. [6] Presents how the perfect shuffle interconnection pattern can be used in order to perform FFT transforms. Several methods for performing FFT computations are presented in [7]. A variety of algorithms for pipeline and parallel

Number of points in data frame	**Radix-4 Stages needed**
16	**2**
32	**2**
64	**3**
128	**3**
256	**4**

Table 1: PE utilization for different lengths of frames

pipeline processors are examined, with respect to VLSI implementation. [8] Presents among others a set of algebraic tools
that can be used to describe processor networks in terms of their patterns of connections. A radix-2^2 algorithm is presented in [9], which combines the radix-2 butterfly structure and the radix-4 multiplicative complexity. A low-power FFT architecture is presented in [11]. Asynchronous circuit design and multirate signal processing are combined in order to produce a globally shared result algorithm. [12] Presents a DSP architecture for high-speed FFT transforms proposing a different flow of the computations for the butterfly operations. [13] Presents an OFDM modem and the benefits of architecture flexibility, adaptability and reconfigurability.

This paper presents a parallel FFT architecture, which is reconfigurable with respect to the size of the FFT problem, the clock speed of execution and the number of memory modules used at each stage. The architecture is optimized for use in Orthogonal Frequency Division Multiplexing Modems (OFDM). It has incorporated several features of the above designs while it introduces reconfiguration of a high-speed architecture along with a variable number of memory modules. It also considers the power efficiency of the FFT architecture. The architecture’s description follows in the next Section.

3. Architecture

This Section describes the overall architecture and the details of the individual blocks, namely the processing elements, the interconnection and the control. The FFT-architecture performs FFT or IFFT computation of 2^n points, $4 \leq n \leq 8$. This is accomplished by implementing split radix decimation in frequency (DIF) algorithm. The collection of the 2^n data words are regarded as one frame of data which, for the specific application in OFDM modems, correspond to the length of one OFDM symbol.

The FFT architecture consists of five (5) Processing Elements (PEs): Four (4) radix-4 PEs and one (1) radix-2 PE, as shown in Figure 1. Each PE performs a single stage of the FFT computation within the time required to input one data frame. The FFT architecture can be configured in real time, in order to perform FFT computations of 16, 32, 64, 128 or 256 points. These computations require a subset of four (4) radix-4 PEs plus optionally one (1) radix-2 PE depending on the number of the FFT points, as shown in Table 1.

The FFT architecture has been designed to process consecutive frames of either the same or different lengths. An external arbiter must signal a change in the frame size to the architecture.

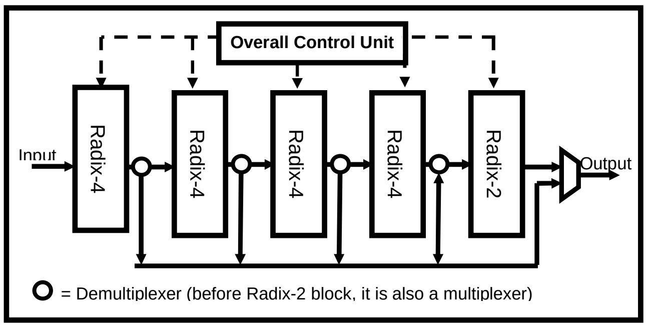
Overall Control Unit
Input
Radix-4
Radix-4
Radix-4
Radix-4
Radix-2
Output
○ = Demultiplexer (before Radix-2 block, it is also a multiplexer)

Figure 1: FFT organization

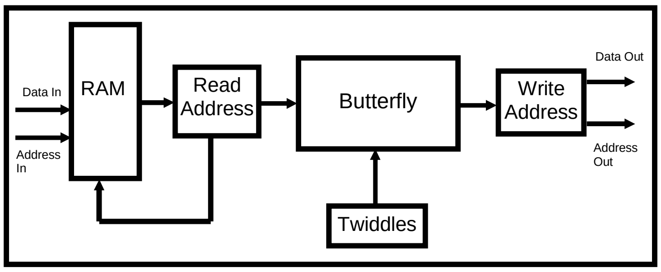

Figure 2: Radix-4 & Radix-2 processing element internal units

3.1 Processing Element Description

This section describes the functionality of the blocks that support each processing element (PE). The following paragraphs define the interfaces of the blocks within the PEs, as well as the interfaces among the PEs.

Radix-4

A radix-4 processing element performs one stage of radix-4 butterfly computations to the data. Each radix-4 PE consists of the following blocks (refer to Figure 2):

RAM: This block is used to store the input data to the processing element. The preceding butterfly stage supplies the write addresses to the radix-4 PE. The read addresses are generated within the radix-4 PE (as described below). The memory block is organized internally with two memory banks. Each bank can store one (1) frame of data. The first bank can be considered as the working bank for the FFT core. The second bank is used to store the incoming input data. The two memory banks switch roles at the beginning of each incoming frame.

Address Generators: There are two address generators. The first is used for supplying read addresses to the RAM block internal to the PE. The second is used for supplying write addresses for the data that exit the PE. The addressing scheme is the same for both Address Generators. Each radix-4 PE uses a distinct addressing scheme depending on the FFT stage realized by the PE.

Twiddle (W) Generator: A Look-Up-Table (LUT) contains the max(N) roots of unity, where max(N) is the maximum size of FFT that is supported by the architecture (256). The Twiddle Address Generator is a simple N-counter-based architecture. At each data cycle the Twiddle Generator fetches the appropriate twiddle factor.

Butterfly Core: This block performs the radix-4 FFT butterfly computation. The N input data are read sequentially from the RAM block. Each set of four consecutive input data forms the input to each radix-4 calculation. The corresponding twiddle factors are also fetched from the Twiddle Generator. Four (4) complex accumulators are used to process the input data in parallel. Each accumulator-process involves the add-subtract of the four data, as these operations are defined by the radix-4 data flow. A single complex multiplier unit operates on the four (4) accumulated results and the twiddle factors in a pipeline fashion. The resulting data are written sequentially to the RAM block of the following (FFT) PE.

Each radix-4 processing element in the reconfigurable FFT pipeline has to be provided the length of the data frame it is processing by the Overall Control Unit (OCU). With this information the addressing scheme used by the address generator block and the twiddle factor generator block is changed. This is accomplished by selecting the proper permutation of the binary counter that all three generator blocks realize.

Radix-2

The radix-2 PE applies one stage of radix-2 butterfly computations to its data. It is used when the size of the frame to be processed is 32 or 128 points. The radix-2 PE is realized as a simplified radix-4 PE (Figure 2). The Butterfly Core is replaced with the simpler radix-2 butterfly network, consisting of two (2) complex adders/subtractors and one (1) complex multiplier. This circuit though is optimized further. In the split radix 128 and 32 point FFT computation, the twiddle factors for all radix-2 butterflies have the constant value of (1+0j). Plugging into the radix-2 butterfly equations we obtain:

$$\begin{cases}\mathbf{A'} = \mathbf{A} + \mathbf{B}\cdot\mathbf{W} \\ \mathbf{B'} = \mathbf{A} - \mathbf{B}\cdot\mathbf{W}\end{cases} \overset{\mathrm{W=1}}{\Rightarrow} \begin{cases}\mathbf{A'} = \mathbf{A} + \mathbf{B} \\ \mathbf{B'} = \mathbf{A} - \mathbf{B}\end{cases}$$

Consequently, the complex multiplier (in the butterfly core) and the twiddle generator blocks are omitted.

3.2 Interconnection

The architecture performs FFT or IFFT computation by implementing split radix decimation in frequency (DIF) algorithm. The pipeline consists of four (4) consecutive radix-4 PEs and a single PE of radix-2. The input to the FFT is always directed to the first radix-4 PE of the pipeline. With the use of a broadcast bus the address and data output of each PE can be diverted to the bus or to the following PE using multiplexers/demultiplexers (In our implementation we have been using broadcast lines). The bus is used in two ways. First, for data that do not need to be processed by all PEs of the pipeline and the bus performs as a bypass to the exit of the FFT architecture. Second, it is used for processing the split-radix cases of frame lengths, namely the 32 and 128 point, in which case output from any previous PE (radix-4) in the pipeline can be diverted to the input of the radix-2 PE for processing. The output from the last PE (radix-2) is stored into a RAM block. It consists of two banks of memories and utilizes the bank switching mechanism described above. In this block, data are buffered in order to perform the bit-reversal permutation.

An additional external input signal specifies the beginning of a data frame. This signal is propagated to each successive PE of the pipeline, and allows proper bank switching functionality. The architecture provides this signal as an output to mark the start of a data frame at the output.

3.3 Overall Control Unit

The pipeline Overall Control Unit (OCU) has to provide signals that control: first each multiplexer's select port for output on the broadcast bus. Second, it supplies the clock for each PE. Third, OCU selects the proper permutation of the Twiddle and Address generators within each PE. Each input frame can consist of any number of points (16, 32, 64, 128 or 256).

The OCU maintains a table consisting of five (5) entries. Each entry specifies the number of points in the frame that each of the five PEs (4 radix-4 and 1 radix-2) is currently processing. All entries in the table are updated each time the architecture is reconfigured. Using this information the OCU can assess whether:

a) A PE is currently processing the last stage of the FFT for the respective frame. In this case it enables the corresponding multiplexer so that the output of the PE is directed to the bus, and out of the FFT architecture.
b) A PE is currently processing an intermediate FFT computation stage. Thus its output is directed, using the corresponding multiplexer, to the following block or the radix-2 PE using the broadcast bus.
c) A PE is not processing any valid data. This situation arises when all frames in the pipeline do not need to traverse all the PEs in order to complete the FFT computation. The PEs not performing butterfly operations can be disabled to conserve power.

The pipeline OCU also has to provide a clock signal to each PE. The time interval (T_i) required for processing one frame (F_i) by one PE is fixed, irrespective to the number of points it consists of (P_i). Furthermore, the frame F_i takes T_i time to input or output the FFT architecture. Thus the clock frequency (f_i) that is required to input, process or output the frame F_i is:

$$f_i = \frac{1}{T_i} \cdot P_i$$

The OCU, using its internal table, is able to compute the clock frequency needed by each PE of the pipeline and distributes the clock signals.

4. Power Consumption

This section presents the performance of the FFT architecture described, with respect to the power dissipation of the architecture. Power saving can be accomplished by considering the features of the architecture and taking advantage of those that can be modified to provide lower power consumption features. The following paragraphs describe the techniques used in the architecture leading to power reduction.

The first technique that can be incorporated is the neutralization of the non-processing PEs of the FFT architecture. Each time a new frame enters the architecture, an external signal informs the FFT control of the frame size. The number of points within each frame is 2^n, $4 \le n \le 8$. Table 1 shows that depending on the frame size there are PEs (at worst case one) that remain unused while still consuming power. The idle PEs can be deactivated resulting into lowering the power consumption.

A second technique to improve on power consumption is to parallelize the functionality of multiplications and additions of the FFT calculations. It has been shown [4] that power dissipation is higher for architectures designed with low degrees of parallelism. The degree of spatial parallelism is defined as the number of data samples consumed and produced by a butterfly stage in one (1) execution cycle. The frequency is proportionally reduced by the number of the butterfly-processing PEs incorporated by the architecture. The proposed architecture uses five (5) butterfly PEs and performs in the test implementation at a frequency of 100MHz .A single PE processing the same amount of data must perform at 5*100MHz. Applying the analysis presented in [4], it follows that the power saving is 40%.

In addition to the described techniques, there are several other ways of achieving power reduction. A different multiplication scheme can be applied to the complex multipliers of the FFT architecture, leading to further power consumption. It has been proven in [5] that instead of entering new data into the multiplier, for each multiplication, a transpose direct form structure can be utilized. In this manner each input data sample does not change value until it is multiplied by all coefficients. Since the switching activity at the multipliers inputs decreases significantly, it will follow a proportionally lower switching activity within the multiplier. Therefore, a considerable reduction in power dissipation is achieved.

5. Parallelizing the memory access

This section presents a memory/processor configuration that can reduce the number of clock cycles required to retrieve and store the FFT data from and to the memories. In the following we use a single processing element with two memory banks. The results can be extended to the case of *k* processors and *2k* memory banks.

In the following we will use the algebra developed by Parker in [8] and extended by Wold and Despain [7]. As shown in [8], the FFT network can be decomposed in a series of operators that describe how the interconnections should be designed. We will use a notation similar to that of [7]. We will prove that the proposed addressing produces a correct FFT algorithm and will conclude by describing how this transform can be implemented.

Let $N=2^n$ be the number of points for a radix-2, DIT-FFT. Let the index of a data be defined as its coordinates on the input stream, $[x, y] = [[x_u \cdots x_1],[y_v \cdots y_1]]$, with x_i and y_i the digits of x and y in binary notation. The one-dimensional input data stream is an array with indices from [0, 0] to [N-1, 0], where [0, 0] = x(0) and [N-1, 0] = x(N-1). We use an operator that can be described by its effect on the indices. This operator divides the input stream in blocks of 2^k data and distinguishes the data within each block into 2^j rows, so that each resulting column contains 2^{j-1} butterfly transformation *pairs*. In binary notation:

$$\begin{cases} \mu_{(j,k)}[x,y] = \mu_{(j,k)}[[x_u \cdots x_1],[y_v \cdots y_1]] = \\ = [[x_u \cdots x_{k+1} x_{k-j} \cdots x_1],[y_v \cdots y_1 x_k \cdots x_{k-j+1}]] \\ \mu_{(k)} = \mu_{(j,k)} \end{cases} \qquad (1)$$

Where μ is defined if $j \le k \le u$. The operator $\mu_{(k)}$ rearranges the input stream into two rows, according to the kth bit of the x index of each data on the stream. As a consequence, the $\mu_{(j,k)}$ operator separates the input stream into 2^j memory banks. The $\mu_{(k)}$ operator rearranges the input data in the correct order, as to perform the butterfly operations in the column pairs. The form of $\mu_{(k)}^{-1}$ (reverse transform) can be deduced from the above definition. The butterfly calculations can be defined as an operator (B) that reads a two-dimensional array in columns and performs a DFT on the data pairs. The accurate definition of B is not essential, provided that the operator does not shuffle the resulting pairs.

The structure of an FFT transform using the $\mu_{(k)}$ and B operators is ([8], [7])

$$FFT = \mu_{(n)} B \mu_{(n)}^{-1} \mu_{(n-1)} B \mu_{(n-1)}^{-1} \cdots \mu_{(1)} B \mu_{(1)}^{-1} \qquad (2)$$

where the composition of the operators takes place as $f_1 f_2 x = f_2(f_1(x))$. From Eq. (2) we note that the $\left(\mu_{(k)}^{-1}\mu_{(k-1)}\right)$ operator forms the transformation that rearranges the data during two consecutive stages of the algorithm.

If we map the two rows of this array onto two memory banks, then the indices of a transformation pair [x_a, y_a] and [x_b, y_b] are:

$$[x_a, y_a] = \left[\left[x_{a(u)} \cdots x_{a(k+1)} x_{a(k-1)} \cdots x_{a(1)}\right], \left[x_{a(k)}\right]\right]$$
$$[x_b, y_b] = \left[\left[x_{b(u)} \cdots x_{b(k+1)} x_{b(k-1)} \cdots x_{b(1)}\right], \left[x_{b(k)}\right]\right] =$$
$$= \left[\left[x_{a(u)} \cdots x_{a(k+1)} x_{a(k-1)} \cdots x_{a(1)}\right], \left[\overline{x_{a(k)}}\right]\right]$$

since, the two data are on the same column (x) and on a different row (y). The read operation can be performed concurrently. The write back operation, on the other hand cannot be performed in a single cycle, because both data will be competing for the same memory bank, due to the y index, which is equal to $x_{a(k-1)}$ for both a, b.

Now, let $\lambda_{(k)}$ be the following operator:

$$\lambda_{(k)}[x] = [x_u \cdots x_{k+1}(x_{k+1} \oplus x_k)x_{k-1} \cdots x_1]$$

where $\lambda_{(k)}$ is defined for $1 \le k \le u-1$ and the symbol $\oplus$ denotes the XOR operation. The operator is a permutation such that $\lambda_{(k)} = \lambda_{(k)}^{-1}$ and $\lambda_{(k)}\lambda_{(k)}^{-1} = I$, I being the unity operator. Using the identity $\mu_{(k)}\mu_{(k)}^{-1} = I$ Eq. (2) can be rewritten as:

$$FFT = \mu_{(n)} B \mu_{(n)}^{-1} \lambda_{(n-1)} \lambda_{(n-1)} \mu_{(n-1)} \cdots \lambda_{(1)} \mu_{(1)} B \mu_{(1)}^{-1} = \qquad (1^{st}\ \text{Form})$$
$$\mu_{(n)} B \underbrace{\left(\mu_{(n)}^{-1} \lambda_{(n-1)}\right)}_{\substack{\text{Write}\\ \text{Operation}}} \underbrace{\left(\lambda_{(n-1)} \mu_{(n-1)}\right)}_{\substack{\text{Read}\\ \text{Operation}}} \cdots \left(\lambda_{(1)} \mu_{(1)}\right) B \mu_{(1)}^{-1} = \qquad (2^{nd}\ \text{Form})$$
$$\mu_{(n)} B \underbrace{\left(\mu_{(n)}^{-1} \lambda_{(n-1)} \mu_{(n-1)}\right)}_{\substack{\text{Write}\\ \text{Operation}}} \underbrace{\left({}^{-1}_{(n-1)}\ {}_{(n-1)}\ {}_{'(n-1)}\right)}_{\substack{\text{Read}\\ \text{Operation}}} \cdots \left(\mu_{(1)}^{-1} \lambda_{(1)} \mu_{(1)}\right) B \mu_{(1)}^{-1} \qquad (3^{rd}\ \text{Form})$$

The write operation described by the product $\mu_{(k+1)}^{-1}\lambda_{(k)}$ (2nd Form) can be shown to resolve the memory congestion described above. Recall the form (1,2) of the addresses of an arbitrary pair during the kth step of the algorithm. Applying the above operators, we find that

$$[x_a]' = \left[x_{a(u)} \cdots x_{a(k+1)} x_{a(k)} \left(x_{a(k)} \oplus x_{a(k-1)}\right) x_{a(k-2)} \cdots x_{a(1)}\right]$$
$$[x_b]' = \left[x_{a(u)} \cdots x_{a(k+1)} x_{a(k)} \left(\overline{x_{a(k)}} \oplus x_{a(k-1)}\right) x_{a(k-2)} \cdots x_{a(1)}\right]$$

and the resulting addresses will always differ at bit k-1, which is the bank where the data will be written on the next stage of the algorithm. This ensures that the data will always reside on different rows during memory write operation.

The 3d form describes the implementation of the above scheme. After the completion of the write back operation on the kth stage of the FFT the indices of an element are

$$[x, y] = \left[\left[x_u \cdots x_k x_{k-2} \cdots x_1\right], \left[\left(x_k \oplus x_{k-1}\right)\right]\right]$$

The indices of a transformation pair differ only with respect to the y coordinate. Applying the inverse transformations $\left({}^{-1}_{(k-1)}\ {}_{(k-1)}\mu_{(k-1)}\right)$ does not affect the x coordinate of a datum. Therefore the transformation pairs will reside *on the same* x coordinates and their y coordinates will be inverted *if* the x_k bit of their x coordinate is equal to "1". This is equivalent to exchanging the butterfly inputs for those data pairs whose x-coordinate has the x_k bit set ("1").

The permutation and addressing scheme described above is depicted in Figure 3, where $W^{-1} = \left({}^{-1}_{(k-1)}\ {}_{(k-1)}\mu_{(k-1)}\right)$ and $W = \left(\mu_{(n)}^{-1}\lambda_{(n-1)}\mu_{(n-1)}\right)$. The W operator can be implemented using a simple combinational circuit that calculates $\left(x_k \oplus x_{k-1}\right)$ on the write address of the (k-1)th stage of the FFT. Finally, the W^{-1} operator can be implemented using two multiplexers on the inputs of the butterfly processor that invert the inputs from the two banks with respect to the x_k bit of the address.

6. Implementation on FPGAs and Concluding Remarks

The validity and the efficiency of the design have been shown by mapping the architecture on the Xilinx and Altera Field Programmable Gate Arrays (FPGAs: Xilinx:Virtex II - XC2V2000-5BF957 and Altera: EP20K600EBC33-1X) and for demonstration purposes it has been integrated in a OFDM modem [13]. The resulting maximal frequency has been shown to be 103MHz and the FFT computations can have throughput as shown in Table 2:

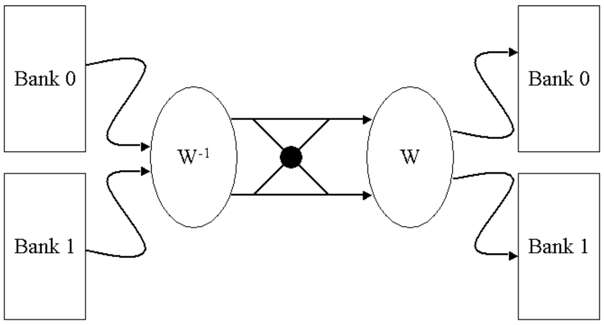

Figure 3: Parallel memory accesses

FFT Length (Words)	Sustained Throughput
16	0.15 us
32	0.31 us
64	0.62 us
128	1.24 us
256	2.48 us

Table 2: Sustained throughput for the supported FFT length

Comparing to related results in the literature ([9], [11], [12], [15], [16], [17], [18], [19], [20], [21], [22], [23]) the proposed architecture achieves an enhanced throughput in FFT processing. Furthermore, it is reconfigurable at real time to accommodate lower transmitting rates if needed. The architecture has been designed to minimize the round-trip delay of the network and to include the power saving techniques of Section 4. As mentioned in section 3 each processing element uses word-serial access to its memory bank. If the technique in Section 5 is applied the throughput will increase by the number of parallel memory accesses.

References

[1] J.W Cooley and J.W Tukey ‘An algorithm for the machine calculation of complex Fourier series’

[2] A.V.Oppenheim and R.W Schafer. ‘**Digital Signal Processing**’, Prentice Hall 1975

[3] Clark D. Thompson ‘Fourier Transform in VLSI’, **IEEE Transactions on Computers**, 1973.

[4] Hong, Kim, Papaefthymiou, Stark ‘Power Complexity Analysis of Pipelined VLSI FFT Architectures for Low Energy Wireless Communication Applications’,

[5] A.T. Erdogan and T. Arslan ‘Low power multiplication scheme for FIR fliter implementation in single multiplier CMOS DSP processors’, **Electronic Letters**, vol. 32, 1996.

[6] H. D. Stone ‘Parallel Processing with the Perfect Shuffle.’, **IEEE Transactions on Computers**, vol. C-20, no.2, 1971.

[7] E.H. Wold and A.M. Despain. ‘Pipeline and Parallel FFT Processors for VLSI Implementations’, **IEEE Transactions on Computers**, vol. C-33, 1984.

[8] D. Stott Parker ‘Notes on Shuffle/Exchange-Type Switching Networks.’**, IEEE Transactions on Computers**, 1980

[9] S. He and M. Torkelson ‘A New Approach to Pipeline FFT Processor.’**, Proceedings of the IPPS**, 1996.

[10] Farid. N. Najm ‘A Survey of Power Estimation in VLSI Circuits’, **IEEE Transaction on VLSI**, 1994.

[11] Bruce W. Suter ‘A Low Power, High Performance Approach for Time-Frequency/Time-Scale Computations’

[12] J. Lee, J. Lee, M. H.Sunwoo, S. Moh and S. Oh ‘A DSP Architecture for High-Speed FFT in OFDM Systems’, **ETRI Journal**, 2002

[13] I. Saarinen, G. Coppola, A. Polydoros, J.L. Garcia, M. Lobeira, P. Dallas, M. Gertou, R. Cusani and G. Razzano. ‘High Bit Rate Adaptive WIND-FLEX Modem Architectures for Wireless Ad-Hoc Networking in Indoor Environments’

[14] S. He and M. Torkelson ‘Design and Implementation of a 1024-point Pipeline FFT Processor’, **IEEE 1998 Custom Integrated Circuits**.

[15] AMPHION Data Sheet ‘High Performance 256 Point FFT Core’ http://www.quicklogic.com

[16] AMPHION Data Sheet ‘High Performance 64-Point FFT/IFFT FFT64HPS’ http://www.quicklogic.com

[17] Xilinx Reference Design ‘High Performance 16-Point Complex FFT’ http://www.xilinx.com/ipcenter

[18] NewLogic Technologies AG Data Sheet ‘Fast 64-points FFT / IFFT (IP core)’ http://www.newlogic.com

[19] Sacet Data Sheet ‘64/256-Point Complex FFT/IFFT’ http://www.sacet.com

[20] *TMS320C62xx User’s Manual*, Texas Instruments Inc., Dallas, TX, 1997

[21] *SC140 DSP Core Reference Manual*, Motorola Semiconductors Inc., Denver, CO, 2000

[22] *DSP16210 Digital Signal Processor Data Sheet*, Lucent Technologies Inc., Alletown, PA, 2000

[23] Philips Semiconductors Inc. “Philips Semiconductors’ R.E.A.L DSP Core for Low-Cost Low-Power Telecommunication and Consumer Application,” **Technical Backgrounder From Philips Semiconductors**, Sept. 1998 http://www.us-3.semiconductors.com

[24] K. Sam Shanmugam, ‘**Digital and Analog Communication Systems**’, John Wiley & Sons Inc. 1979

[25] John A. C. Bingham ‘Multicarrier Modulation for Data Transmission: An Idea Whose Time Has Come’, **IEEE Communications Magazine**, 1990.