http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 11 (1): 118-125, 2012
ISSN 1812-5638 / DOI: 10.3923/itj.2012.118.125

Efficient 16-points FFT/IFFT Architecture for OFDM Based Wireless Broadband Communication

[1]M. Arioua, [2]S. Belkouch and [1]M.M. Hassani
[1]Electronic and Instrumentation Laboratory, Faculty of Science Semlalia,
Cadi Ayyad University, Marrakech, Morocco
[2]Embedded Systems and Digital Controls Laboratory, Department of Electrical Engineering,
ENSAM, Cadi Ayyad University, Marrakech, Morocco

Abstract: The Fast Fourier Transform (FFT) and its inverse transform IFFT processors are a key component in OFDM based wireless broadband communication systems. Hence, it is important to develop a low-power and high-performance FFT/IFFT processor to meet the requirement of low cost and real time in such communication systems. In this study, an efficient 16-points FFT/IFFT module is developed to build a large FFT/IFFT processor for OFDM based communication system (IEEE 802.16). The 16-point FFT architecture consists of an optimized pipeline implementation based on Radix-2 butterfly Processor Element. This proposed architecture reduces the multiplicative complexity and power consumption compared to other efficient architectures. The FFT processor has been implemented in VHDL code. The simulation results show that this processor significantly achieves a better performance with less complex multiplications and lower resource usage.

Key words: FFT/IFFT, OFDM, R2MDC, FPGA, IEEE 802.16, wireless applications

INTRODUCTION

Fast Fourier Transform (FFT) and its inverse (IFFT) play a significant role in many digital signal processing applications. It has been applied in a large range of fields and applications such as Asymmetrical Digital Subscriber Line (ADSL), Digital Audio Broadcasting (DAB), Digital Video Broadcasting (DVB) and Orthogonal Frequency Division Multiplexing (OFDM) systems.

Recently, OFDM has become a key modulation technique for both broadband wireless and wire-line applications (Van Nee and Prasad, 2000; Keller and Hanzo, 2000). It has been advocated for Wireless Local Area (WLAN) and Metropolitan Area (WMAN) applications; it incorporates a large number of orthogonally selected carriers to transmit a high-data-rate stream in parallel form in the frequency domain. The problem of Inter-Symbol-Interference (ISI) introduced by a multi-path channel, in conventional single carrier system confines the data rate, however, it is importantly reduced in OFDM due to the parallel of data transmission through multiple carriers (Arioua *et al*., 2009). The sub carrier's orthogonality in OFDM allows the sub carrier spectra to be densely packed in the frequency domain resulting in a high spectral efficiency. Therefore, multipath immunity and spectral efficiency are two major features of OFDM systems (Verma *et al*., 2009).

The fast increasing demand of OFDM-based applications, including future UWB systems and wireless LAN and MAN, makes the processing speed a significant factor in Fast Fourier Transform (FFT) architecture design. Indeed, these applications need the FFT and IFFT processors to perform real-time operations for modulation and demodulation of signals. Hence, the study of high-performance VLSI FFT architecture becomes increasingly important.

The FFT/IFFT processor is one of the most complex and intensive computation module of various communication systems PHY layer (IEEE 802.11a, IEEE 802.11n and IEEE 802.16, etc.) (Li, 2003). However, the main constraints nowadays for such processors are execution time and lower power consumption (He and Torkelson, 1996). One of the most essential arithmetic operations used in FFTs is complex multiplication. It is often the most expensive arithmetic operation and one of the dominate factors in determining the performance in terms of speed and power consumption. As observed by Widhe (1997), the complex multiplier may consume more than 70% of the power in an FFT/IFFT processor. Therefore, an effective design of FFT processor is vital in low-power applications. The aim here is to reduce the multiplication complexity and develop architecture for use in OFDM based communication system (IEEE 802.16).

Corresponding Author: M. Arioua, Electronic and Instrumentation Laboratory, Faculty of Science Semlalia,
Cadi Ayyad University, Marrakech, Morocco

One method proposed in this study to reduce the complexity is to replace the complex multiplication with less expensive real and constant multiplications (Arioua *et al*., 2010; Jiang *et al*., 2004). We applied this method to an efficient FFT 16-point based on 4-point module. The 16-point FFT processor has to satisfy the timing constraint required in the communication specification system employed. This impose that one has to implement highly parallel or parallel-pipelined architecture (Kannan and Srivatsa, 2007) or to use a very high frequency operation. However it leads to high area usage and power consumption. Therefore, it is essential to build a simple and efficient architecture to keep the area and power consumption as low as possible while meeting the timing constraint. The performance analysis of the proposed architecture reveals its superiority compared to other efficient butterfly approaches (Li and Wanhammar, 2002; Li *et al*., 2001; Shanine, 2009).

FFT ALGORITHM

The FFT/IFFT is derived from the main function DFT (Discrete Fourier Transform). The computation for N-points of the DFT will be calculated one by one for each point while for FFT/IFFT, the computation is done simultaneously which saves a quite a lot of time.

The N-point discrete Fast Fourier Transform is defined as:

$$X[k]=\sum_{n=0}^{N} x[n].W_N^{nk} \quad (1)$$

Where:

$$W_N^{nk}=e^{-j\frac{2\pi nk}{N}} \quad 0\le k\le N-1$$

X (k) is the k-th harmonic and x (n) is the n-th input sample. Direct DFT calculation requires a computational complexity of O (N^2). By using The Cooley-Tukey FFT algorithm, the complexity can be reduced to O ($N.\log_r N$) (He and Torkelson, 1996; Li and Wanhammar, 2002).

Cooley-Tukey FFT algorithm: The Cooley-Tukey FFT is the most universal of all FFT algorithms, because of any factorization of N is possible (Cooley and Tukey, 1965; Meyer-Baese, 2007). The most popular Cooley-Tukey FFTs are those where the transform length is a power of a basis r, i.e., $N = r^S$. These algorithms are referred to as radix-r algorithms. The most commonly used are those of basis r = 2 (radix-2) and r = 4 (radix-4). Those algorithms and others such as radix-2^i and Split-Radix have been developed based on the basic Cooley-Tukey algorithm to further reduce the computational complexity (Chang and Nguyen, 2006).

For r = 2 and S stages, for instance, the following index mapping of Cooley-Tukey algorithm gives:

$$n = 2^{S-1}n_1+2^{S-2}n_2+...+2n_{S-1}+n_S$$
$$k = 2^{S-1}k_S+2^{S-2}k_{S-1}+...+2k_2+k_1$$

And:

$$n_1, n_2,..., n_{S-1}, n_S = 0, 1 \qquad K_S, k_{S-1},..., k_2, k_1 = 0, 1$$

The Cooley-Tukey algorithm is based on a divide-and-conquer approach in the frequency domain and is therefore referred to as Decimation-In-Frequency (DIF) FFT. The DFT formula is split into two summations:

$$X(k)=\sum_{n=0}^{\frac{N}{2}-1} x(n).W_N^{nk}+\sum_{n=\frac{N}{2}}^{N-1} x(n).W_N^{nk}=\sum_{n=0}^{\frac{N}{2}-1} x(n).W_N^{nk}+\sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right).W_N^{(n+\frac{N}{2})k}$$
$$=\sum_{n=0}^{\frac{N}{2}-1} x(n).W_N^{nk}+\sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right).W_N^{nk}.W_N^{\frac{N}{2}k} \quad \text{and} \quad W_N^{\frac{N}{2}k}=(-1)^k$$
$$=\sum_{n=0}^{\frac{N}{2}-1}\left(x(n)+(-1)^k.x\left(n+\frac{N}{2}\right)\right)W_N^{nk} \quad (2)$$

X (k) can be decimate into even-and odd-indexed frequency samples:

$$X(2k)=\sum_{n=0}^{\frac{N}{2}-1}\left(x(n)+x\left(n+\frac{N}{2}\right)\right)\cdot W_N^{2nk}=\sum_{n=0}^{\frac{N}{2}-1}\left(x(n)+x\left(n+\frac{N}{2}\right)\right)\cdot w_{\frac{N}{2}}^{nk} \quad (3)$$

$$X(2k+1)=\sum_{n=0}^{\frac{N}{2}-1}\left(x(n)-x\left(n+\frac{N}{2}\right)\right)\cdot W_N^{2nk}=\sum_{n=0}^{\frac{N}{2}-1}\left(x(n)-x\left(n+\frac{N}{2}\right)\right)\cdot w_{\frac{N}{2}}^{nk}\cdot w_N^{n} \quad (4)$$

The computational procedure can be repeated through decimation of the N/2-point DFTs X (2k) and DFTs X (2k+1). The entire algorithm involves $\log_2 N$ stages. Where each stage involves N/2 operations units (Butterflies). The computation of the N point DFT via the decimation-in-frequency FFT as in the decimation-in-time algorithm requires $(N/2)\cdot\log_2 N$ complex multiplications and $N.\log_2 N$ complex addition (Petrov and Glesner, 2005).

In general, the other fast algorithms like radix-4, radix-8, radix-2^2 and split-radix (based on the same approach) recursively divide the FFT computation into odd- and even-half parts and then obtain as many common twiddle factors as possible. The number of needed real additions and multiplications is generally used to compare efficiency of different FFT algorithms. As indicated by multiplicative comparison by

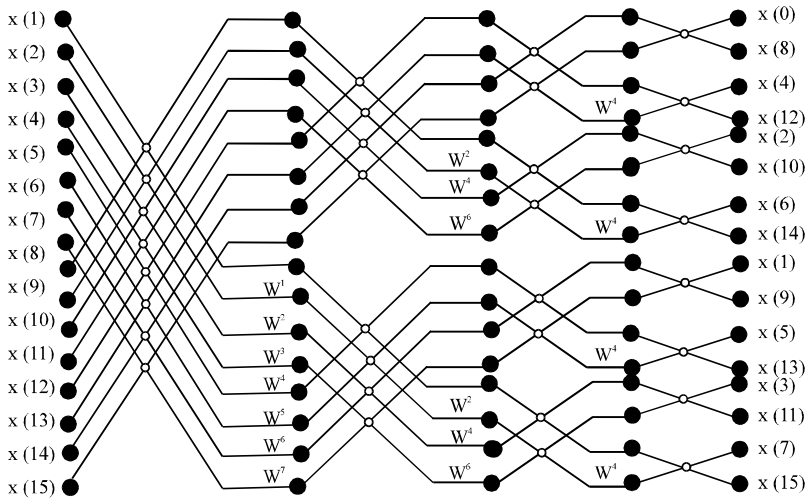

Fig. 1: 16-point DIF FFT algorithm based on radix-2

Shanine (2009), the split-radix is computationally better than others FFT algorithms, due to its most trivial multiplication with twiddle factors W^n, i.e., ±1 and ±j. Nevertheless, the split-radix is an irregular algorithm by its nature, because of the combination of two algorithms radix-2 and radix-4 stages used, respectively for the even-half operations and for the odd-half operations. Therefore, this architecture leads to an L-shaped butterfly units and affect the delay of the pipeline path and make it unbalanced (Chang and Nguyen, 2006).

16-points FFT module: The flow graph of complete decimation-in-frequency decomposition of 16-point DFT computation based on radix-2 is represented in Fig. 1. The basic operation in the signal flow graph is the butterfly operation; it's a 2-point DFT computation as shown in Fig. 2.

The FFT computation of 16-points radix-2 based architecture is achieved in four stages. The x (0) until x (15) variables are denoted as the input values for FFT computation and X (0) until X (15) are denoted as the output values. In the butterfly process as shown in Fig. 2, the upward arrow will execute addition operation, beside that; downward arrow will execute subtraction operation. The subtracted value is multiplied with twiddle factor value W_N before being processed into next stage; this computation accomplished concurrently. The complex multiplication with the twiddle factor requires four real multiplications and two add/subtract operations (Li, 2003; Li and Wanhammar, 2002; Petrov and Glesner, 2005).

Complex multiplication: Complex multiplication is an expensive operation, because its computation need large chip area and power consumption. We can reduce the multiplicative complexity of the twiddle factor inside the butterfly processor by calculating only three real

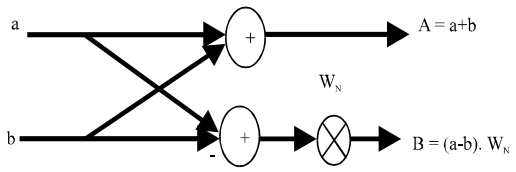

Fig. 2: Basic butterfly computation

multiplications and three add/subtract operations. The complex twiddle factor multiplication:

$$R+j1 = (X+jY).(C+jS) \quad (5)$$

However, the complex multiplication can be simplified:

$$R = (C-S).Y+Z \quad (6)$$

$$I = (C+S).X-Z \quad (7)$$

And:

$$Z = C.(X-Y) \quad (8)$$

The above implementation of reduced complex multiplication is useful for general complex multiplication. However, in our FFT implementation, the twiddle factors coefficients are known in advance. i.e., C and S in Equations 6 and 7 are pre-computed and stored in a memory table. Therefore, it is necessary to store the following three constants C, C+S and C-S. those constants can be saved as Canonical Signed Digits (CSD) to implement complex multiplication with carry and save tree (Jiang *et al*., 2004; Kannan and Srivasta, 2007). Thus, the area and power consumption of the complex multiplier can both be reduced.

The storage operation is used to simplify the complex multiplication, for instance, the complex multiplication with:

$$W^2_{16} = e^{\frac{-j\pi}{4}}$$

requires only two real multiplications rather than three multiplications. Moreover, the complex multiplication can be reduced further with an efficient number representation of fixed-point arithmetic (He and Torkelson, 1996).

The implemented algorithm of complex multiplication used in this work uses three multiplications, one addition and two subtractions as shown in Fig. 3. This is done at the cost of an additional memory Table. In the Hardware Description Language (VHDL) program, the twiddle factor multiplier was implemented using component instantiations of three lpm-mult and three lpm-add-sub modules from Altera library. Worth to note that lpm modules are supported by most of EDA vendors and LPM provides an architecture-independent library of logic functions or modules that are parameterized to achieve scalability and adaptability (Qi, 2010).

In the 16-point FFT processor using radix-2 algorithm, multiplications with $W_{16}^4 = -j$ and W_{16}^0 factors are trivial, multiplication with W_{16}^4 simply can be done by swapping from real to imaginary part and vice versa, followed by changing the sign (Li *et al*., 2001; Petrov and Glesner, 2005). The rest of complex multiplications in this FFT scheme are non-trivial, W_{16}^2 was implemented with two multiplications and three multiplications for the other non-trivial coefficients. Therefore, the total number of real multiplications is 24 (10 complex multiplications).

The radix-2 algorithm is attractive for its simplicity but has the disadvantage for being not adapted for large point FFT calculation, due to the high multiplier requirement. However, to reduce the power consumption in the 16-points FFT, the number of multiplications must be reduced. For this reason, we propose a design methodology for simple and efficient 16-point FFT architecture to keep the area and power consumption as low as possible.

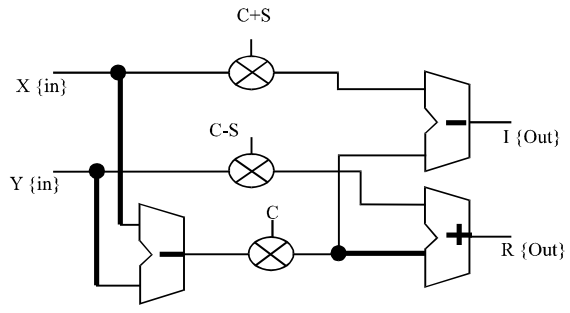

Fig. 3: Implementation of complex computation

16-POINT FFT/IFFT ARCHITECTURAL DESIGN

Mathematical formulation: The 16-point FFT/IFFT proposed architecture internally uses one 4-points module for computation. The radix-16 representation of the FFT/IFFT can be formulated in the following way:

$$FFT(x) = X[k] = \sum_{n=0}^{N} x[n].W_N^{nk} \quad (9)$$

We suppose: N = 4T, k = s+Tt and n = l+4m where, s, $l \in$ {0, 1, 2, 3} and m, t $\in$ {0, 1,...,T-1}.

We apply this values in Eq. 9, we obtain:

$$X[s+Tt] = \sum_{l=0}^{3} W_4^{lt}[W_{4T}^{sl} \sum_{m=0}^{T-l} x[l+4m].W_T^{sm}] \quad (10)$$

For T = 4, N = 16. The 16-point FFT can be expressed as:

$$X[s+4t] = \sum_{l=0}^{3} [W_{16}^{sl} \sum_{m=0}^{3} x[l+4m].W_4^{sm}]W_4^{lt} \quad (11)$$

Multiplier unit — Second FFT — First FFT

Equation 11 shows that the application of the FFT algorithm for computation of the 16-point FFT requires calculation of 4-point FFT two times. The first calculation of the appropriate data slot of 4-point FFT takes place as described in Eq.10 and then multiplies the output with 4*4 inter-dimensional constants coefficients and once again computing the 4-point FFT of the resultant data with the fitting data reordering.

The 4-point FFT module uses radix-2 algorithms for computation. The multiplication with the twiddle factor $W_4^1 = -j$ can be done by swapping and inversion (Li *et al*., 2001; Maharatna *et al*., 2000). Therefore, the proposed processor computes complex multiplications exclusively with inter-dimensional constants coefficients, this computation processed by a Multiplier Unit as shown in Fig. 4.

Architectural design: Many communication systems require high throughput and continuous input/output data. The pipeline architecture is considerably appropriate to attain these ends (Umar *et al*., 2004); also it is an ideal choice to implement high-speed long-size FFT due to its regular structure and simple control (Wang *et al*., 2010). The efficiency of a pipeline FFT processor can be improved by optimizing the structure and saving hardware resources (Zhi *et al*., 2009). The proposed architecture of 16-point FFT/IFFT module is illustrated in Fig. 4. Our design consists of an essential unit, the butterfly unit which is the kernel of the 16-FFT processor as interpreted in Eq.11. It has two-stages pipelined structure carrying out trivial complex multiplication and eventually

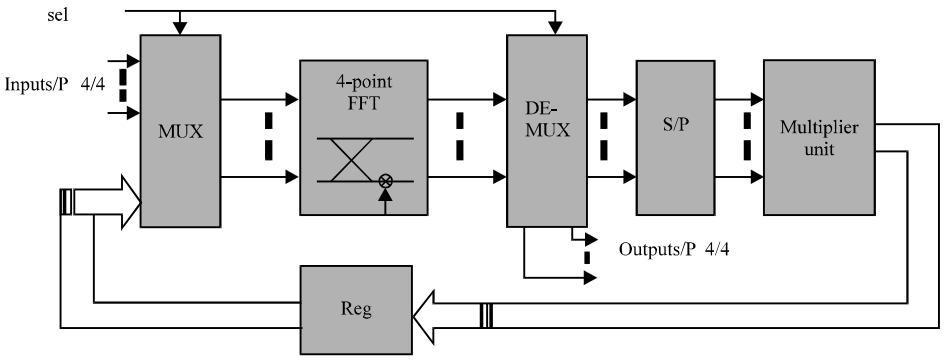

Fig. 4: The 16-point FFT proposed design

processes 4-point FFT. This block requires a data memory (input buffer) of size N/2 (N = 4) to store the input serial data in four parallel vectors, in order to be arranged for computation by FFT processor according to Eq. 11, Since the input data takes place in natural order.

After the computation of the initial input data sequences in the 4-point FFT, the demultiplexes holds the data to a serial parallel bloc in the first FFT computation or to 16-FFT output in the second computation. In the first case the data are arranged for multiplication in the multiplier unit according to Eq. 11.

The arranged data undergoes an operation of interdimensional complex multiplication. The multiplier unit should normally perform complex multiplications with 16 elements. However, in practice the reduction approach mentioned above leads to a significant reduction of complex multiplication at the cost of far less expensive additions:

$$\begin{pmatrix} W_{16}^{0} & W_{16}^{0} & W_{16}^{0} & W_{16}^{0} \\ W_{16}^{3} & W_{16}^{2} & W_{16}^{1} & W_{16}^{0} \\ W_{16}^{6} & W_{16}^{4} & W_{16}^{2} & W_{16}^{0} \\ W_{16}^{9} & W_{16}^{6} & W_{16}^{3} & W_{16}^{0} \end{pmatrix}$$

Indeed, adders requires less hardware in FPGA plateform and consume much less power than that of multipliers with the same word length. Moreover, they have fewer glitches.

At circuitry level, the multiplication with the elements W^0 can be replaced by a simple connexion and the multiplication by W_{16}^4 can be done by swapping and sign inversion. The non trivial multiplication with W_{16}^6 and W_{16}^2 can be implemented with two real multiplications, however, in order to reduce more multiplication complexity, the multiplication with W_{16}^6 and W_{16}^2 coefficients ($1/\sqrt{2}$) was done by add and shift operations (Jung *et al*., 2003) as shown in Fig. 5 and 6. Applying shift-and-add operations with twiddle factors inside the multiplier unit instead complex multiplications reduces

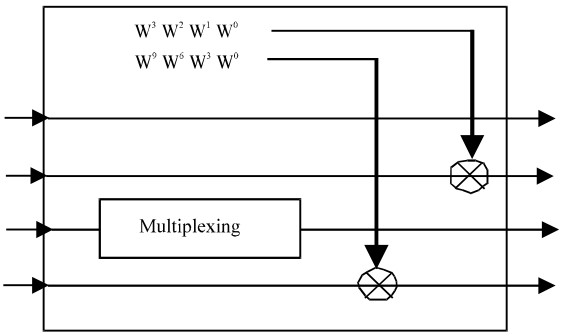

Fig. 5: Intern operations inside multiplier unit

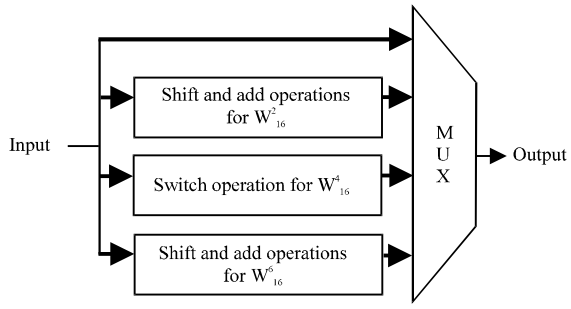

Fig. 6: Switch, shift-and-Add operations inside multiplier unit

area-usage as well as power consumption. The complex multiplication with other non-trivial twiddle factors W_{16}^1 W_{16}^3 W_{16}^9 was implemented with three real multiplications.

The employed multiplier unit allows us to cut down the number of complex multiplications. Therefore, the total number was reduced to 6 multiplications. Consequently, the total number of real multiplication in our design is 16 multiplications.

An important result within the proposed 16-point architecture is the fewer multiplications used when compared to the reduced 16-point FFT Radix-2 (24 real multiplications), to the reduced 16-points FFT with

Radix-4 (20 real multiplications) and to the attractive 16-point Split-Radix architecture (Li and Wanhammar, 2002; Li *et al*., 2001; Wang *et al*., 2010; Takahashi, 2001).

PERFORMANCE OF THE PROPOSED ARCHITECTURE

16-points FFT implementation: The 16-point FFT computation with the proposed architecture was first coded in VHDL using Quartus software tool from Altera and then simulated and synthesized on the low cost Altera Cyclone 2 EP2C35F672C6 device. The purpose is to determine the resource usage of the proposed design. The functional and timing simulation were successfully made. The main hardware resources of this design are given as follows. The total logic elements used are 1146 and the total embedded multiplier 9-bit elements are 6. Meanwhile, the maximum frequency is 75.93 MHZ.

The proposed structure combines three resource reductions, the complex multiplication reduction inside the multiplier unit, combined with the fact using pipeline architecture. In addition to that, this structure has eliminated the complex multiplication for some coefficients inside multiplier unit.

From the algorithmic perspective, the proposed architecture requires less number of arithmetic operations compared to the conventional Cooley-Tukey algorithm and to efficient ones implemented in pipeline structure for radix-2 and for radix-4 and split-radix. This comparison is shown in Table 1.

It shows that the simple architecture of 16-point FFT proposed in this work requires 13, 66, 80, 80% of real multiplications, used, respectively in Cooley-Tukey, 16-point Radix2 (3M3A: three Multiplications and three additions), 16-point Radix4 architectures and 16-point split-radix.

In order to verify the accuracy of computation of the FFT core, we had simulated the calculation of 16-point FFT in functional simulation mode. The output of the implemented FFT bloc approximately matches the output of an FFT function written in Matlab representing a theoretical example of FFT calculation. Two same input signals (square signals) were given to Matlab and Quartus tool for simulation. The inputs and outputs are plotted and are shown in Fig. 7. Another random signal was given to Matlab and Quartus tool for simulation to valide if the FFT results are correct as shown in Fig. 7 and 8.

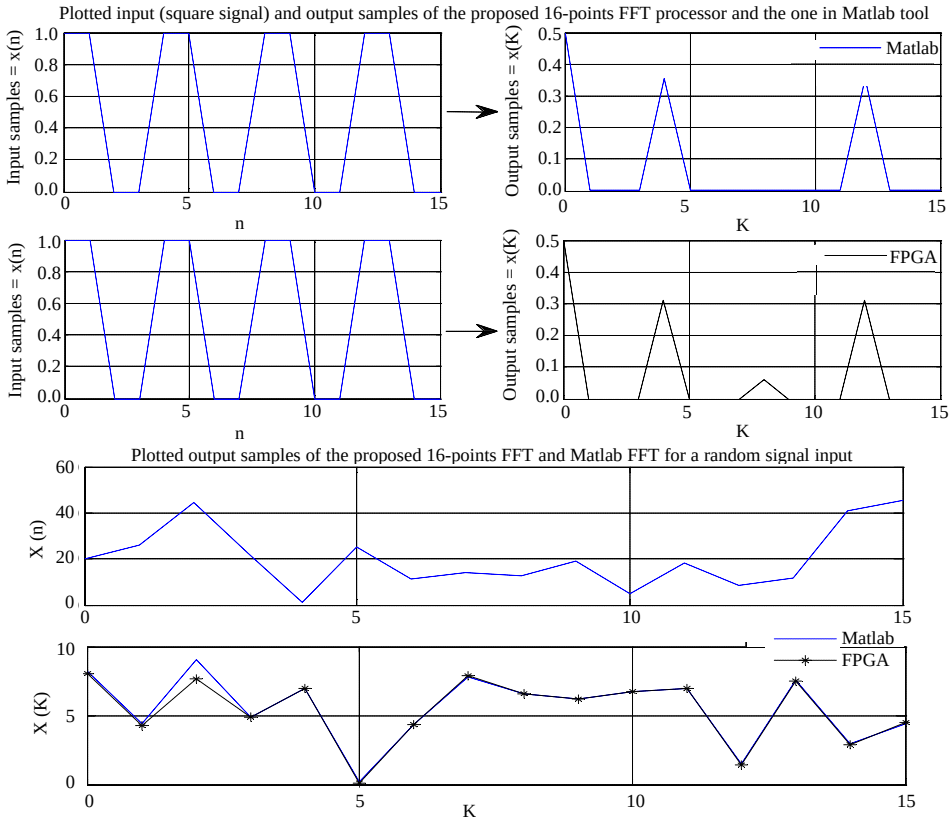

Fig. 7: 16-point FFT simulation

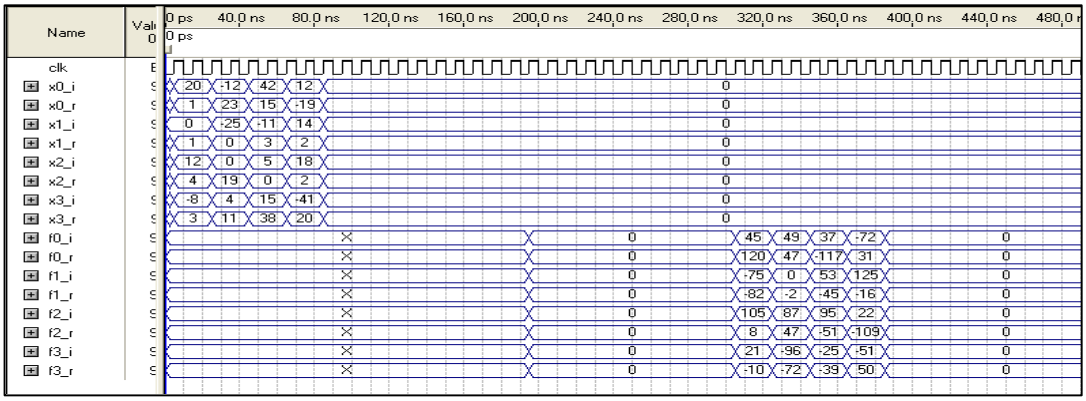

Fig. 8: Simulation results of the FFT block's response in Quartus tool

Table 1: Comparison of the proposed architecture with different pipelines architectures

	Complexity			
Architectures	Complex multiplications	Real multiplications	Real add/sub	Embedded multiplier 9-bit
Cooley-Tukey (4M2A) (trivial and non-trivial)	32	128	192	128
Radix-2MDC (4M4A) (Shanine, 2009)	10	28	148	8
Radix-2MDC (Li and Wanhammar, 2002)	10	24	152	6
Radix-4MDC (Li and Wanhammar, 2002)	08	20	148	9
Split-Radix-MDC (Li *et al.*, 2001)	08	20	148	9
Radix-4SDC (Shanine, 2009)	10	24	148	3
Radix-2SDF (Shanine, 2009)	10	24	152	6
Radix-4SDF	08	20	148	3
Radix-2^2SDF	10	24	152	3
Split-Radix-SDF	08	20	148	3
Proposed processor	06	16	154	6

The 16-point FFT processor was described with hardware VHDL using fixed-point arithmetic while Matlab simulation uses floating point based calculations. Consequently, the resulting VHDL simulation can not completely match the one with floating point exactly at the maximum value of the lobes due to the lost of point precision during the computation. In these simulation, we used only 8 bits to represent a FFT point. Using 16 bits; we would have an output signal close to the real one in Matlab shown in Fig. 7. In addition, published results (Umar *et al.*, 2004) of 16-bit fixed-point and the 16-bit fating point FFTs both give highly accurate and comparable results. Therefore, the 16-bit fixed-point is more suitable to implement the proposed FFT architecture.

The proposed architecture is more useful when a computation of a large FFT/IFFT is needed. 16-point processor elements can be then used as building blocks for these FFTs/IFFTs; like the 256-Point FFT for OFDM based communication system (IEEE 802.16).

CONCLUSION

The number of multiplications has been used in this study as a key metrics for comparing FFT algorithms since it has a large impact on the resource usage and power consumption. The efficient 16-point FFT/IFFT architecture proposed in this study gives an advantage in terms of resource usage and power dissipation using a complex multiplication reduction approach and pipelined method. The simulation results show that proposed architecture significantly reduces the number of operations inside the processor compared to others efficient processor. The proposed processor can be integrated with other components to be used as stand-alone processor (128, 256, 1024-point FFT) applied for OFDM based Wireless Broadband Communication (WiMAX).

REFERENCES

Arioua, M., S. Belkouch and M.M. Hassani, 2009. Simulation and performances comparison of the OFDM transmission chain. Proceedings of the IEEE International Conference on Multimedia Computing and Systems, April 2-4, Ouarzazate, Morocco.

Arioua, M., S. Belkouch, M.M. Hassani and M. Agdad, 2010. Complex multiplication reduction in pipeline FFT architecture. Proceedings of 20th International Conference on Computer Theory and Applications, Oct. 23-25, Alexandria, Egypt.

Chang, W.H. and T. Nguyen, 2006. An OFDM-specified lossless FFT architecture. IEEE Trans. Circuits Syst. Regular Pap., 53: 1235-1243.

Cooley, J.W. and J.W. Tukey, 1965. An algorithm for the machine calculation of complex Fourier series. Math. Comput., 19: 297-301.

He, S. and M. Torkelson, 1996. A new approach to pipeline FFT processor. Proceedings of the 10th International Parallel Processing Symposium, April 15, Honolulu, HI., USA., pp: 766-770.

Jiang, M., B. Yang, Y. Fu, A. Jiang and X. Wang, 2004. Design of FFT processor with low power complex multiplier for OFDM-based high-speed wireless applications. Proceedings of IEEE International Symposium on Communications and Information Technologies, Oct. 26-29, Sapporo, Japan, pp: 639-641.

Jung, Y., H. Yoon and J. Kim, 2003. New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications. IEEE Trans. Consum. Electron., 49: 14-20.

Kannan, M. and S.K. Srivatsa, 2007. Low power hardware implementation of high speed FFT core. J. Comput. Sci., 3: 376-382.

Keller, T. and L. Hanzo, 2000. Adaptive multicarrier modulation: A convenient framework for time-frequency processing in wireless communications. Proc. IEEE, 88: 611-640.

Li, W., M. Vesterbacka and L. Wanhammar, 2001. An FFT processor based on 16-point module. Proceedings of the IEEE Nordic Event in ASIC Design Conference, Nov. 12-13, Stockholm, Sweden, pp: 125-130.

Li, W. and L. Wanhammar, 2002. Complex multiplication reduction in FFT processor. Proceeding of Swedish system-on-ship Conference, Mar. 18-19, Falkenbery, Sweden, pp:1-4.

Li, W., 2003. Studies on implementation of lower power FFT processor. Ph.D. Thesis, Linkoping University, Sweden.

Maharatna, K., E. Grass and U. Jagdhold, 2000. A lower-power 64-point FFT/IFFT architecture for wireless broadband communication. Proceedings of the 7th International Conference on Mobile Multimedia Communication, Aug. 7-10, Tokyo, Japan, pp: 1-5.

Meyer-Baese, U., 2007. Digital Signal Processing with Field Programmable Gate Arrays. 3rd Edn., Springer, ISBN-13: 978-8184890808, Pages: 774.

Petrov, M. and M. Glesner, 2005. Optimal FFT architecture selection for OFDM receivers on FPGA. Proceedings of the IEEE International Conference on Field Programmable Technology, Dec. 11-14, Singapore, pp: 313-314.

Qi, W., 2010. FPGA-based digital signal processing algorithm research. Proceeding of the 3rd IEEE International Conference on Computer Science and Information Technology, July 9-11, Chengdu, China, pp: 690-692.

Shanine, C., 2009. Architecture of reconfigurable Integrated Circuit, very high speed and low consumption for digital processing of the advanced OFDM. Polytechnic Institute of Grenoble, France.

Takahashi, D., 2001. An extended split-radix FFT algorithm. IEEE Signal Process. Lett., 8: 145-147.

Umar, A., M.M. Al-Akaidi, S.A. Khan, S. Khattak and M. Assadullah, 2004. Performance evaluation of a hiperlan type 2 standard based on arithmetic formats. Inform. Technol. J., 3: 1-5.

Van Nee, R.D.J. and R. Prasad, 2000. OFDM for Wireless Multimedia Communications. Artech House, Norwell, MA., USA., ISBN-13: 978-0890065303, Pages: 284.

Verma, P., H. Kaur, M. Singh and B. Singh, 2009. VHDL implementation of FFT/IFFT blocks for OFDM. Proceedings of International Conference on Advances in Recent Technologies in Communication and Computing, Oct. 27-28, Kottayam, Kerala, India, pp: 186-188.

Wang, B., Q. Zhang, T. Ao and M. Huang, 2010. Design of pipelined FFT processor based on FPGA. Proceedings of the 2nd International Conference on Computer Modeling and Simulation, Jan. 22-24, Hainan, China, pp: 432-435.

Widhe, T., 1997. Efficient implementation of FFT processing elements, linkoping studies in science and technology. Ph.D. Thesis, Linkoping University, Sweden.

Zhi, D., Z. Yimeng, H. Zhiping, T. Guilin and L. Chunwu, 2009. Simulation and application of FFT based pipelined stream. Proceeding of International Conference on Information Engineering and Computer Science, Dec. 19-20, Wuhan, China, pp: 1-4.