An FPGA Based Low Power Multiplier for FFT in OFDM Systems Using Precomputations

Mokhtar Aboelaze
Dept. of Electrical Engineering and Computer Science
Lassonde School of Engineering
York University
Toronto ON CANADA

Abstract*— OFDM is considered to be the technology of choice for many wireless and wire-line transmission systems. OFDM is the standard for IEEE802.11, digital audio and video broadcast, and DSL broadband internet access. The generation of orthogonal frequencies in OFDM is done by using Inverse Fourier Transform and implemented as IFFT. IFFT (for transmission) and FFT (for receiving) are considered to be very computationally intensive applications. In this paper we introduce a new multiplier for performing FFT (since both FFT and IFFT require the same computation structure, we will use FFT to refer to both of them). First, we use fixed point FFT instead of floating point. The BER due to noise and multipath is studied as a function of the number of bits chosen to represent data in the system. Then, we introduce an FPGA based multiplier using precomputations to be used in FFT. Our proposed multiplier requires less energy and consumes less resources on the FPGA chip compared to a standard fixed point IP core by Xilinx. The proposed architecture is implemented on FPGA using Xilinx Spartan 6 architecture and compared to the traditional implementation of the FFT (IP multiplier by Xilinx). Our proposed multiplier shows a decrease of 50% energy consumption and uses less FPGA resources than a standard IP multiplier.

I. Introduction

OFDM is the dominant technology in many communication systems such as ADSL and Wireless communication. OFDM is used in IEEE802.11 and IEEE802.16m. It is also the main standard in digital audio broadcasting and digital terrestrial TV broadcasting in Europe (DVB-T and DVB-H). For example, IEEE802.16 require OFDM symbol rate of 1.75-20MHz and require performing up to 2048 point FFT per symbol.

There have been a lot of work in designing FFT processors and accelerators in order to speed the computation and to decrease energy consumption. Energy consumption is particularly important for hand held devices, since it determines the battery life. In this section, we briefly review some of the work in FFT processors.

A multipath delay commutator structure is introduce in [4] to increase the throughput of radix-2 and Radix-4 FFT computation. The author proved that their architecture can improve the throughput by a factor of 2-4 and decrease the latency by a factor of 2-3 for large N FFT computation.

[3] presents a novel VLSI architecture for pipeline FFT. The proposed architecture can produce the normal output order sequence of the FFT. The proposed design utilizes the decimated dual-path delay feedforward data commutator. The architecture can achieve full hardware efficiency. Also a new sequence converter is integrated into the last stage of the FFT computation.

In [17] the authors proposed two optimized implementation of pipelined FFT processors. They investigated different optimization techniques and different rounding schemes and their effect on the SNR. They implemented R2^2SDF and R4SDF using both Xilinx Spartan-3 and Virtex-4 FPGAs. The achieved a maximum clock rate of 219.2 MHz.

A reconfigurable systolic array for DSP functions is proposed in [7]. The proposed architecture utilizes coarse-grained processing elements that can be configured to implement a wide variety of signal processing algorithms (DFT, IDFT, polyphase FIR, phase shifter, ...). Their architecture is reconfigurable in real time and configuration data can be loaded without interrupting the circuit operation.

In [16] the authors proposed a new CORDIC algorithm and a new architecture for FFT. They mentioned that their design is suitable for variable length FFT. They synthesized their design using 0.18μ technology. Their design runs at 222MHz clock and consumes 26.75 mW.

In [8] the authors proposed an FFT processor for 4x4 MIMO-OFDM that is capable of performing 64 and 128 points FFT. They used a Radix 4 Multi-path Delay Commutator (R4MDC). They also used a mixed-radix FFT implementation. They synthesized their architecture using 0.18 μ technology using standard CMOS processing. Their results show improvement of 80% and 64% compared to R2^3SDF and R4MDC architectures respectively.

In this paper, we concentrate on efficient design of the multiplier used in calculating FFT for OFDM. Since multiplications in FFT is performed by multiplying the twiddle factors by the data sequence, we show that we can optimize our multiplier according to the number of bits required in representing the twiddle factor. Finally we compare our design from the chip area on the FPGA chip and energy consumption points of view to existing multipliers.

We summarize our contribution in this paper as follows

- We characterize the effect of the word size on BER for both white Gaussian noise and inter symbol interference.
- We propose a multiplier architecture based on precomputation that consumes less energy than a standard IP based multiplier by Xilinx.
- We implement our design on a Virtex-6 FPGA and calculate the power saving

The organization of the paper is as follows: Section II presents OFDM and explain why it is a popular choice for modern communication. Section III presents the effect of the word-length of fixed point calculation of FFT on the OFDM performance. Section IV discusses power saving techniques. Section V presents the precomputation based multiplier we used to implement FFT. Section VI provides an FPGA implementation of our proposed multiplier and compares it with the IP fixed point multiplier supplied by Xilinx. Section VII is a conclusion and future work.

II. OFDM

OFDM is not a new technology, it was proposed first in [2] as a transmission technique to reduce the required bandwidth. It was not practically implemented because of the complexity of the scheme. In 1971, Weinstein and Ebert [14] proposed the use of DFT to generate the orthogonal subcarrier. It was not until the huge advances in digital electronics that allowed the implementation of such a system.

The main idea of OFDM is to divide a high bit rate data stream into many parallel low bits data streams using multi carrier transmission. Figure 1 shows a regular FDM system where the data is divided into streams and each stream is modulated by a different carrier. The different carrier frequencies $f_0, f_1, ...f_{n-1}$ must be separated by some frequency band to minimize interference between different carriers. That is one of the main drawbacks of the regular FDM communication systems. In such a system, where we use N carriers, the symbol length in case of multi carriers could be N times the symbol system if we use a single carrier for modulation. It turns out that this is very important in wireless communication since it could be used to combat multipath fading.

One of the major impediments of wireless communication is multi-path delay. In wireless communication signal might reflects or refracts then it will be received by the receiver. More than one path can be taken by the signal to reach the receiver, each path has a different delay. The receiver receives multiple signals delayed by different values, that leads to intersymbol interference (ISI). if the data rate is high, which means the symbol period is short, the ISI might affect a big part of the previous symbol which leads to error. In case of FDM (or OFDM) the signal period is N times the signal period in single carrier systems, which means the ISI will be limited to a small part of the signal period resulting in less errors. Moreover, by introducing a *guard* time between the different symbols, that will further reduce the effect of ISI.

Another impediment is *frequency selective fading*. OFDM is known to be very effective in combating frequency selective fading. In a large bandwidth signal, By dividing the wideband signal into many narrowband signals, each narrowband signal is subjected to flat fading which could be dealt with through error coding and simple equalization.

For FDM transmission, we have to leave frequency bands between the different carriers in order to reduce inter carrier interference. That leads to wasting bandwidth by introducing unused frequency bands between the carriers. if the carriers are orthogonal to each other, there will be no inter carrier interference since the effect of any of the carriers on the rest is zero (orthogonal to each other). One way to achieve orthogonality if all the carrier frequencies are multiple of some basic frequency. However having N oscillators (N could be in the hundreds or even thousands) tuned to orthogonal frequencies is not an easy solution for the problem.

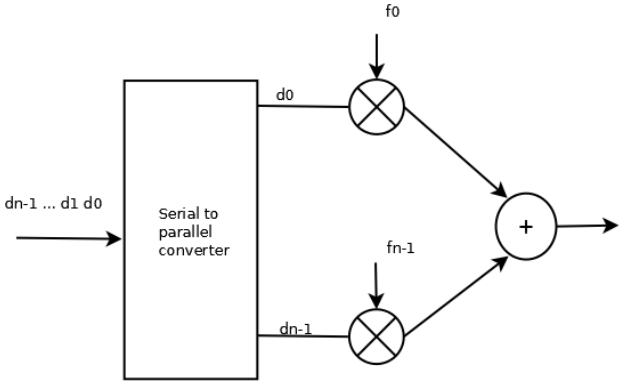

Fig. 1. FDM transmitter

OFDM solves this problem by replacing modulating the signal using N different frequencies by calculating the IFFT of the N signals to be transmitted. Thus, changing the signal from the time domain to the frequency domain. By changing the signal from the time domain to the frequency domain using IFFT, That guarantees orthogonality, and at the same time could be achieved easily by using DSP. Figure 2 shows a very simple OFDM transmitter. The actual transmitter is more complicated, we only show the relevant parts (IFFT) and the main idea of the OFDM transmission. The OFDM transmission could be done by

1) First the data is encoded using any encoding technique (usually QAM).
2) Encoded symbols are serial to parallel converted. The incoming data are split among N different streams.
3) Calculate the IFFT of N symbols from the N streams.
4) The output of the IFFT is sampled and transmitted one sample after the other.

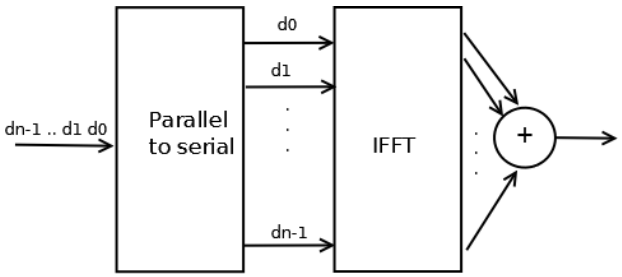

Fig. 2. OFDM transmitter

The output of the IFFT is the frequency domain representation of the N signals corresponding to the data symbols $d_0\ d_1\ ...d_{n-1}$. These signals are transmitted sequentially (mainly D/A conversion). The transmitted signal is a frame of length NT_0 where T_0 is the time it takes to send the original data without OFDM.

In order to avoid multi-path fading frames are separated by a time equal to the maximum difference between any two path delays. Peled and Ruiz in [11] proposed using *cyclic prefix* which maintains the orthogonality propoerty by filling the guard period with a copy of a portion from the beginning of the frame as shown in Figure 3. By using the cyclic prefix frames are separated by a distance equal to the maximum difference in propagation delay without sacrificing orthogonality.

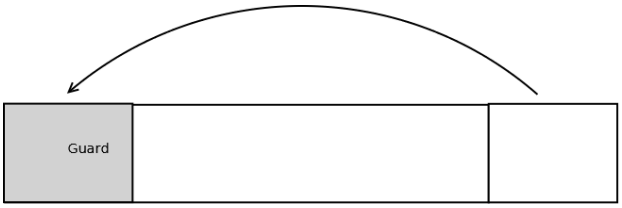

Fig. 3. OFDM frame

The receiver works in a similar but reversed way. The different symbols are received and A/D converted. Then FFT is performed on the N received signals, then a QAM decoder is used to generate the data.

III. Fixed Point FFT

With the proliferation of wireless hand held devices, Energy consumption has become a major design issue. While in desktop and non-wireless application energy concerns affect the energy cost, cooling, and reliability, in hand-held devices battery life is added to that mix. FFT calculation is one of the major energy consuming parts of the OFDM sender/receiver. The FFT (DFT in general) is represented as

$$X(k) = \sum_{i=0}^{N-1} x(i)\ e^{-j2\pi ik/N} \tag{1}$$

Generally, in embedded application and in energy-sensitive application the FFT calculation is implemented using fixed point instead of floating point representation. Fixed-point requires less energy (and less chip area) than floating point implementation, however it suffers from less accuracy and much less dynamic range. If the range of data is known beforehand, dynamic range is not a major problem. In OFDM transmitter, the input of the IFFT is the output of QAM (Quadrature Amplitude Modulation). If we know the scheme, the range of data is limited and is known beforehand.

usually floating point numbers are represented in **Qm.n** notations. The number will be represented by $m+n+1$ bits. m bits to represent the integer portion, n bits to represent the fraction part, and one bit for sign. For example Q3.4 number is represented by 3 bits for the integer part, 4 bits for the fraction part, and one bit for sign for a total of 8 bits. if the number has 0 bits for integer part (fraction number only) Q.m format is usually used. In case of OFDM transmitter, the input to the IFFT is usually the output of a QAM modulator. That means we know the range of inputs to the IFFT. Ideally the input of the FFT in the receiver is the output of the IFFT unit and the added noise. That makes the choice of the Qm.n easy since we know the range of values to represent.

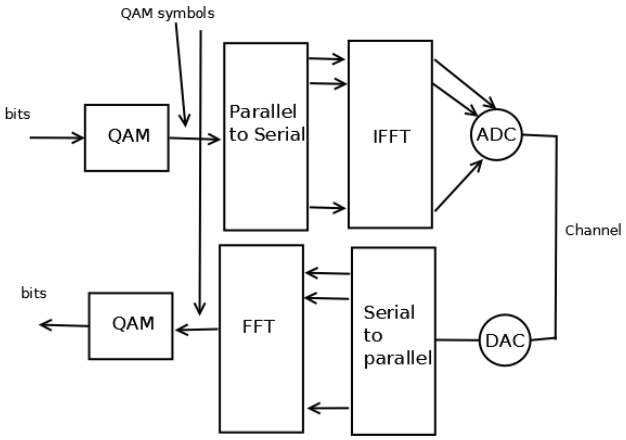

Fig. 4. Simulation block diagram

We simulate an OFDM transmitter/receiver to determine the loss of accuracy due to fixed-point representation. In our simulation we considered only the effect of fixed point representation on the accuracy. Our simulation ignored interleaving, coding, and channel estimation.

Figure 4 shows a block diagram of our simulator. The bit stream is modulated using 16-QAM. The output of the QAM decoder is converted to N parallel streams (In our simulation and implementation we used $N = 64$). The N parallel signals then changed from time domain to frequency domain using IFFT. The output of the IFFT is converted to analog and transmitted through the channel. The channel can introduce white Gaussian noise and multipath fading. Although multi path fading is dealt with using cyclic prefix, we simulated some multi-path fading since the cyclic prefix might not completely eliminate multipath fading. We simulated our system using MATLAB and using both floating point representation and fixed-point representation. The figure of merit is Bit Error Rate (BER).

In our simulation, we used 16–QAM modulation with FFT sizes of 64. Since we used 16-QAM, the range of the QAM modulator output values are limited to ± 3. That means the integer part of the fixed point representation should be 2 bits. The number of bits in the fraction part will determine the performance.

Figures 5,6,7 show the bit error rate (BER) vs. the signal to noise ratio (SNR) for single precision floating point numbers and fixed point representation. We considered only white Gaussian noise. Figure 5 shows the BER for Q2.9 representation (a total of 12 bits and 9 bits for the fraction part. Figures 6 and 7 show the BER for fraction bits of 8 and 7 respectively.

As shown in these figures, the BER is almost identical up to SNR of 16 dB. After 16 dB we notice a difference of about 0.5 dB for floating point representation over fixed point representation.

Another issue is multipath fading. In wireless communication the propagating signals might be reflected off buildings and propagates to the same receiver. The receiver will receive the direct signal and extra delayed copies from the reflected signals. The cyclic prefix is suppose to take care of this by

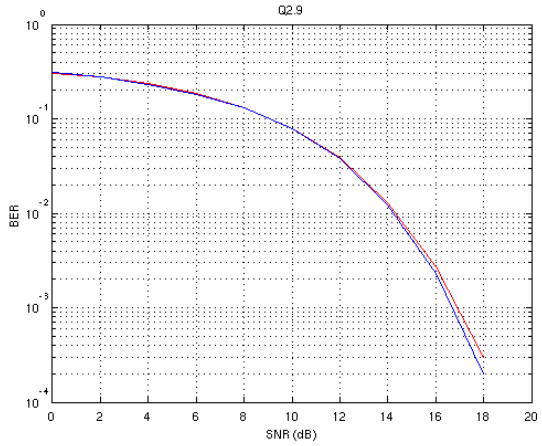

Fig. 5. BER vs. SNR for format Q2.9

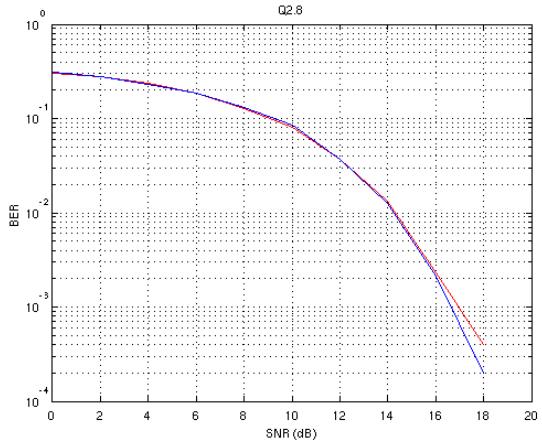

Fig. 6. BER vs. SNR for format Q2.8

introducing guard bands to eliminate the multipath fading. however if the cyclic prefix is not long enough, there might be some multipath interference.

It is difficult to quantize the effect of multi-path fading. Not only because cyclic prefix is suppose to take care of multipath fading, but also because there is an infinite number of possibilities. However just to conceptually illustrate the effect of fixed-point FFT we included a very small subset of BER as a function of multipath fading. In our simulation we sampled the signal at the rate of 40 samples/symbol time. For the multipath fading, we assumed an attenuation of 3dB and a path delay of multiple of 2.5% of the symbol time. Table I shows the bit error rate as a function of delayed signal (one delayed signal with 3dB attenuation). The overlap due to the multipath fading is between 2.5% and 12.5% in 2.5% increments.

As we can see from Table I for the values that we reported here the effect of fixed-point FFT on BER is minimal. Even sometimes it has a better performance than fixed point representation. Most probably that is because of the simulation limitation rather than any intrinsic added value because of the fixed point representation.

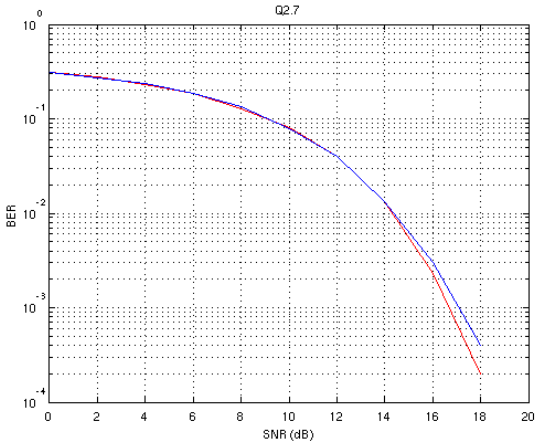

Fig. 7. BER vs. SNR for format Q2.7

TABLE I
THE EFFECT OF MULTIPATH FADING ON BER

Multipath fading	floating point FFT	Fixed point FFT Q2.9
2.5%	0	0
5%	0.0618	0.0620
7.5%	0.0786	0.0795
10%	0.1871	0.1862
12.5%	0.2426	0.2422

IV. POWER REDUCTION TECHNIQUES

Power reduction techniques are numerous, and there is no way that we can cover them all here. We only review the use of precompuation techniques to reduce power. We also mention multiplier sharing techniques. Multiplier sharing is different than precomptation, however this research started using multiplier sharing scheme and ended up using precomputation.

Precomputation [1] is a technique used to reduce dynamic power consumption. Their technique depends on calculating the output value of the function using only a subset of the inputs. if the output can be calculated using this subset of inputs, parts of the circuits are turned off (disabled). Disabling parts of the circuits will reduce the switching activities that could have happened in this part thus reducing the dynamic power. The authors in [1] proposed a method to automatically synthesize precomputation logic to reduce power consumption in the circuit.

In [13], the authors studied the effectiveness of using precomputaion to reduce dynamic power on commercial off the shelf FPGAs. They designed a 32-bit comparator using their technique and achieved a reduction of 43% in dynamic power. They also show the effect of the number of inputs used in precomputation on the power consumption (both logic and routing), and on resource utilization in FPGAs.

In [12] the authors proposed the use of precomputation in designing content addressable memory (CAM). They illustrated that their Block-XOR PB-CAM system can be designed without a special CAM cell design. Using their technique, they

show a 30% average power reduction using TSMC 0.35-μm CMOS technology.

Multiplier sharing was introduced in [10] and it was shown to improve speed in FIR filter design. However it was shown to slightly increase power delay product. However, they implemented their design as ASIC using 0.35-μm technology. In the case of FFT and IFFT the data is multiplied by twiddle factors raised to some power. Knowing the twiddle factor can lead to a more efficient design than the one proposed in [10].

Most of the work in precomputation depends on turning off or disabling parts of the circuits that we do not need to get the results, thus saving dynamic power. Using this technique, power is usually reduced, however the hardware or computational resources in general are increased. We still have to implement the same hardware without precomputation, then we have to add routing and probably small amount of logic to control enabling or disabling parts of the circuit.

In this paper, we use the knowledge from the problem domain to perform precomputation and use Look Up Tables (LUTs) instead of multipliers. Our design uses only adders and LUT's. Our technique resulted in not only low power, but also using much less resources on the FPGA to design the multiplier. A complete description of the multiplier and its performance is presented in the next section

V. Using Precomputation in Multiplier Design

In this section, we introduce our multiplier-less technique for FFT/IFFT in an OFDM system. in OFDM, the transmitter performs IFFT on the output of a QAM encoder, while the receiver performs FFT on the received signal (the D/A conversion of the IFFT together with the added noise). We use the fact that the multiplier multiplies the input by one of the twiddle factors. Although there are $N/2$ twiddle factors for $N-$point FFT/IFFT, there are $N/4$ distinct values for the real and imaginary parts of the Twiddle factors. That means we have only $N/4$ or 16 different values for the Twiddle factors for 64-point FFT.

The second observation is that for the transmitter the input to the IFFT unit is the QAM output. For 16-QAM like the value we assume in this paper, the output (both real and imaginary parts) are between $+3$ and -3. By proper shifting (divide by 2) in the different stages of the IFFT we can guarantee no overflow. That makes the design of the multiplier simpler.

To illustrate our technique, assume we want to multiply a 12-bit number A by a 12-bit Twiddle W. If we divide the 12-bit A into four parts $A_3A_2A_1A_0$ where each A_i is a 3-bit number. The multiplication of AW can be represented as

$$AW = 2^9A_3W + 2^6A_2W + 2^3A_1W + A_0W \qquad (2)$$

Performing the above computations is *completely* equivalent to multiplying A by W using 12×12 multiplier. However using equation 2 we can use one multiplier sequentially to save chip area or use pipelining to increase the speed. Figure 8 shows the multiplication of AW according to the above mentioned scheme.

Since we are multiplying by a Twiddle factor (real or imaginary part of a twiddle factor), the number of twiddle

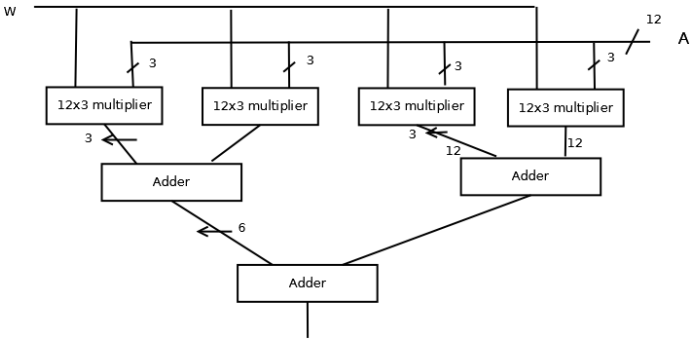

Fig. 8. Multiplication of 2 12-bit numbers using 12x3-bit multipliers

factors in a stage is limited. For 64-point IFFT/FFT there are 16 in the last stage, 8 in the next to the last stage, then 4 and 2 different values for the Twiddle factors . By precomputing the multiplication of all 3-bit numbers by a twiddle factor and storing them in a LUT, we can eliminate a multiplier. The lookup table is accessed according to A_i (3 bits) and the twiddle factor to be multiplied with log_2m where m is the number of different Twiddle factors in a stage. Since the maximum value of m for a 64-point FFT is 16, we need a maximum of 4 bits to access the appropriate Twiddle factor. In the last stage, we need 7-bit LUT (3 for the three bit numbers and 4 for the 16 different twiddle factors). Every stage after the last one we need one less bit (since the number of Twiddle factor is divided by 2).

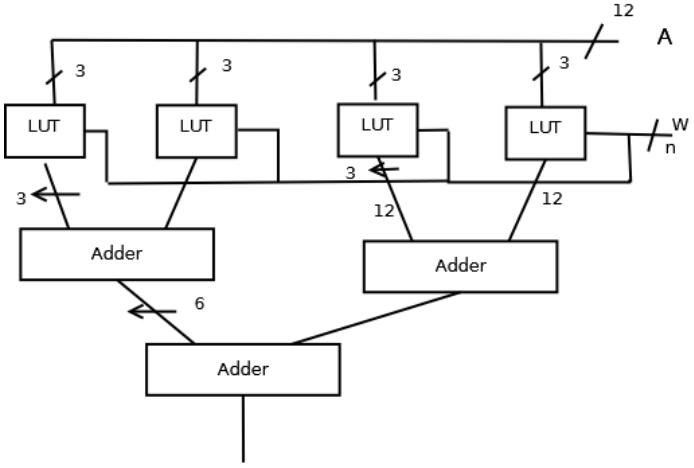

Fig. 9. Our proposed architecture

Figure 9 shows how to implement the multiplier using LUTs and adders only. LUTs are ideal for FPGA implementation since they are the basic building blocks for FPGA chips. They require less area and consume less power than multipliers or adders. Every LUT is accessed using $log_2m + 3$ bits, where m is the number of different twiddle factors in a stage (the size of the LUT is 2^{m+3}. For example in the last stage of a DIT FFT the multipliers needed are W^i where $1 \leq i \leq 15$, 3 bits are used to choose the value of A_i and 4 bits are used to multiply it by a specific twiddle value. Table II shows the contents of one of the LUTs. For example if we consider the middle numerical entry in Table II, address 010_0101 means

TABLE II
LUT ADDRESSES AND CONTENTS

LUT Address	Contents
000_0001	$0 \times W_{64}^{1} = 0$
⋮	⋮
010_0101	$2W_{64}^{5}$
⋮	⋮
101_1000	$5W_{64}^{8}$
⋮	⋮

TABLE III
COMPARISON BETWEEN OUR PROPOSED ARCHITECTURE AND STANDARD ARCHITECTURE

Metric	Our architecture	Standard
Slice registers	179	60
Slice LUTs	244	375
Occupied slices	70	108
MUXCYs	124	432
Power (mW)	29	71

that location contains the multiplication of the number 2 (the first three bits of the address 010) by W_{64}^{5} where 5 is the last 4 bits of the address (0101). By accessing this particular location in the LUT we get the result $2 \times W_{64}^{5}$.

The contents of the LUT is precomputed and a LUT construct is used to implement the look up table in Verilog. In the next section, we show the results of implementing our method in FPGA.

VI. IMPLEMENTATION AND RESULTS

The proposed multiplier is implemented in FPGA using Xilinx Spartan-6 FPGA (XC6SLX16) [15] and implemented on Nexys-3 board [5]. In our implementation we implemented only the multipliers. Our objective is to compare between a standard floating point implementation using Xilinx IP and our proposed implementation.

Since the multiplier size (and consequently power consumption) depends on the number of Twiddle factors, the multiplier size and power will be different from a stage to stage. Assuming a pipelined FFT, there is a multiplier for each stage. The number of Twiddle factors to be multiplied varies from 2 to 16. In our implementation we used the total area of the multipliers for the 6 stages. Table III shows the resource usage and the power consumption for the 5 multipliers used in the 5 stages of a 64-point FFT (one stage is multiplied by 1, and does not need any actual multiplier.

WE can see in Table III that our proposed multiplier reduces the power consumption by almost 60%, and at the same time uses less area on the chip than using a standard multiplier. Although our proposed multiplier uses more slice registers, but that is more than being offset by the saving in the number of slice LUT's, occupied slices, and MUXCy's used.

VII. CONCLUSION AND FUTURE WORK

In this paper we presented an implementation for a multiplier for a 64-point FFT used in an OFDM transmitter/receiver using precomputation. Our multiplier uses less chip resources than standard (IP) multiplier and at the same time reduces the energy consumption by 60%.

We plan to extend our work to design a pipelined FFT processor using the multiplier we proposed in this paper.

REFERENCES

[1] M. Alidina, J. Monteiro, S. Devads, A. Gosh, and M. Papaefthymiou. "Precomputation-based sequential logic optimization for low power". *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*. Vol. 2 no. 4 pp. 426-436. Dec. 1994

[2] R. W. Chang "Synthesis of band-limited orthogonal signals for multichannel data transmission". Bell System Tech. Journal. Vol. 45 Dec. 1966.

[3] Y.-N. Chang "An efficient VLSI architecture for normal I/O order pipeline FFT design" IEEE Trans. on Circuits and Systems II: Express Brief. Vol. 55. Issue 12 pp. 1234-1238. Dec. 2008.

[4] C. Cheng, K. Parhi "High-throughput VLSI architecture for FFT computation" IEEE Trans. on Circuits and Systems II: Express Brief. Vol. 54; Issue 10. pp 863:867 Oct. 2007.

[5] Digilent Inc. Nexys-3 Spartan-6 Board http://www.digilentinc.com/Products/Detail.cfm?NavPath= 2,400,897&Prod=NEXYS3 Checked July 2013.

[6] S. Hassoun, and C. Ebeling "Using precomputation in architecture and logic resynthesis". *IEEE/ACM International Conference on Computer-Aided Design (ICCAD98*. pp 316-323 1998.

[7] H. Ho, V. Szwarc, T. Kwasniewski "A reconfigurable systolic array architecture for multicarrier wireless and multirate applications. *International Journal of Reconfigurable Computing*. Vol. 2009 Article ID 529512 2009.

[8] B. Kang and J. Kim "Low complexity multi-point 4-channel FFT processor for IEEE802.11n MIMO-OFDM WLAN system" *Proc. of the International Conference on Green and Ubiquitous Technology* pp 94-97 2012.

[9] R. Koutsoyannis, P. Milder, C. Berger,M. Glick, J. Hoe, M. Puschel "Improving fixed-point accuracy of FFT in O-OFDM systems". *Proc. of the Intl. Conf. on Acoustics, Speech, and Signal Processing*. ICASSP 2012.

[10] J. Park, K. Muhammad, K. Roy "High-performance FIR filter design based on sharing multiplication" IEEE Trans. on Very Large Scale Integration (VLSI) Vol. 11, No. 2. pp 244-253. April 2003

[11] A. Peled and A. Ruiz "Frequency domain data transmission using reduced computational complexity algorithm" *Proc. of the IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing* pp 964-967. Denver, CO 1980.

[12] S.-J. Ruan, and C.-Y. Wu "Low power design of precomputation-based content-addressable memory" *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*. Vol. 16, issue 3. pp 331-335 2008.

[13] C. C. Tsang, H. K.-H. So "Reducing dynamic power consumption in FPGAs using precomputation". *Proc. of International Conference on Field Programmable Technology* (FPT 2009). Dec. 2009.

[14] S. Weinstein, P. Ebert "Data transmission by frequency division multiplexing using the discrete Fourier transform" IEEE Trans. on communication. Vol. 19 Issue 5 pp 628-634, Oct. 1971.

[15] Xilinx Spartan-6 FPGA Family http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm Checked July 2013.

[16] C.Y Yu S.-G. Chen "Efficient CORDIC designs for multi-mode OFDM FFT" *Proc. of the Intl. Conference on Acoustic, Speech, and Signal Processing* ICASSP pp III-1036 - III-1039. 2006.

[17] B. Zhou, Y. Peng, D. Hwang "Pipeline FFT architecture optimized for FPGAs". *International Journal of Reconfigurable Computing*. Vol. 2009 Article ID 219140. 2009.