Design and Implementation of 256-Point Radix-4 100 Gbit/s FFT Algorithm into FPGA for High-Speed Applications

Gokhan Polat, Sitki Ozturk, and Mehmet Yakut

The third-party FFT IP cores available in today's markets do not provide the desired speed demands for optical communication. This study deals with the design and implementation of a 256-point Radix-4 100 Gbit/s FFT, where computational steps are reconsidered and optimized for high-speed applications, such as radar and fiber optics. Alternative methods for FFT implementation are investigated and Radix-4 is decided to be the optimal solution for our fully parallel FPGA application. The algorithms that we will implement during the development phase are to be tested on a Xilinx Virtex-6 FPGA platform. The proposed FFT core has a fully parallel architecture with a latency of nine clocks, and the target clock rate is 312.5 MHz.

Keywords: Chromatic dispersion, optical communication, Dragonfly (Radix-4), fully parallel architecture, FFT, FPGA.

Manuscript received June 8, 2014; revised Feb. 19, 2015; accepted Mar. 19, 2015.
This research was supported by C2TECH, partner of Celtic-EONET project CP07-006.
Gokhan Polat (gokhanpol@gmail.com), Sitki Ozturk (sozturk@kocaeli.edu.tr), and Mehmet Yakut (corresponding author, myakut@kocaeli.edu.tr) are with the Department of Electronics and Telecommunication Engineering, Faculty of Engineering, Kocaeli University, Kocaeli, Turkey.

I. Introduction

Increasing demands for long-distance and high-speed data communication bring about wide bandwidth and low attenuation necessities. So, copper wires are gradually replaced with optical wires due to their wide bandwidth and lower attenuation ability. However, signal transmission via light introduces some disadvantages, such as chromatic dispersion (CD) and polarization mode dispersion (PMD). Signal processing is at its best when it successfully combines the unique ability of mathematics to generalize with both the insight and prior information gained from the underlying physics of the problem at hand [1]. The CD effect on a fiber optic transmission line can be modeled and distortion effects can be compensated using digital signal processing (DSP) techniques instead of a fiber Bragg grating–based dispersion compensator.

Although there are time-domain solutions for CD compensation, frequency domain–based compensation is commonly preferred in DSP applications. A Fourier transformation is widely used to transfer an examined signal into the frequency domain. A fast Fourier transform (FFT), as an efficient algorithm to calculate a discrete Fourier transform (DFT), is one of the most significant operations in modern DSP systems [2]. An FFT strongly minimizes the cost of implementing a DFT on digital computing systems [3].

There is no way to buffer optical digital signals for a long time because data is continuously fed through an optical transmission line. So, a real-time solution is inevitable for CD compensation. Input values should be processed as fast as

possible to obtain outputs in real time [4]. To maintain a high-speed data flow, a solution has to be based on a parallel architecture. Field-programmable gate arrays (FPGAs) are very attractive platforms for high-speed signal processing due to their parallel processing abilities, such as those used in our FFT-based CD compensation.

The rest of this paper is organized as follows. Section II addresses CD compensation. Section III gives background information about FFT and related hardware solutions. Section IV presents the synthesis results. Finally, Section V summarizes the paper.

II. CD Compensation in Optical Communication

Fiber-optic lines are an ideal transmission environment for high-speed data communication; however, such an environment has its own disadvantages, such as CD and PMD. This work is focused on CD compensation.

Different wavelengths of light have different velocities, known as CD. In optical lines, every pulse that is sent by a transmitter reaches a given destination having a different time delay to the next. CD, or the aforementioned time delay between different colors, is defined as the group delay between different wavelengths of light, as shown in Fig. 1 [5].

The transfer function of a fiber with CD can be written as in (1) below [6]. The expression for group velocity is given in (2) [7]. The group-velocity dispersion parameter, D, is defined as the time delay between two different spectral components separated by a certain wavelength interval. The parameters given in (1) are as follows: c is the speed of light, L is the transmission line length, D is the CD factor, λ is the wavelength of color, β is the propagation constant, and v_g is the group velocity.

$$H(f) \cong e^{j\frac{\pi\lambda^2 DL}{c}f^2}, \tag{1}$$

$$D = -\frac{2\pi c}{\lambda^2}\frac{d^2\beta}{d\omega^2} = \frac{2\pi c}{(v_g)^2\lambda^2}\frac{dv_g}{d\omega}. \tag{2}$$

The CD effect can be corrected in either the electrical or optical domain. One adopted method for solving the problem of compensation involves the usage of dispersion compensating fiber (DCF), which can cause additional loss on a signal; thus, a system needs additional optical amplifiers that can increase the noise and cost of the system [8]. Therefore, a compensation method that utilizes the electrical domain with DSP is commonly preferred. Compensating with a DCF module on a fiber-optic line is shown in Fig. 2.

Electrical compensation methods can be performed both in time and frequency domains. Savory's method, adaptive filters,

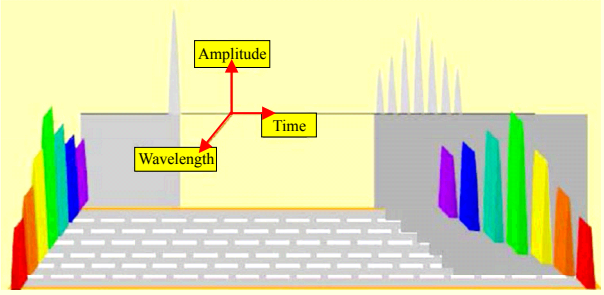

Fig. 1. Chromatic dispersion.

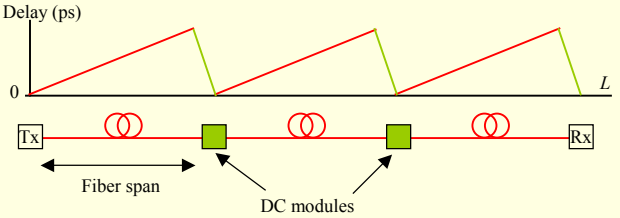

Fig. 2. Compensating with DCF.

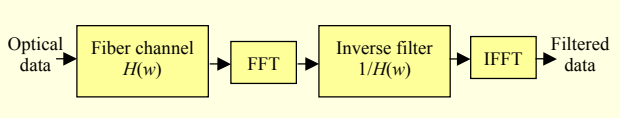

Fig. 3. FFT method for CD compensation.

and the constant modulus algorithm are examples of such methods performed in the time domain. The FFT method and its derivatives can be considered as examples of such methods performed in the frequency domain [8]. The CD effect causes a time-domain delay in different color groups. This effect brings phase distortion to the frequency domain. Fortunately, the CD effect is linear, which means it is stable, predictable, and controllable [9]. Both the multiplication of the frequency responses of the signals and the inverse of the CD channel transfer function compensate the CD effect. Finally, corrected signals should be transferred to the time domain using an IFFT algorithm, as shown in Fig. 3.

In this study, the CD effect is compensated by means of DSP algorithms running on FPGA. However, the acquisition of optical signals into an FPGA should be considered precisely. In this paper, a possible solution is designed for the 40 GSamples/s scenario. The details for this scenario are depicted in Fig. 4. 40 GSamples/s optical data is carried by two polarizations. A 40 GSamples/s signal is split by a polarization splitter into two 20 GSamples/s signals. Each 20 GSamples/s polarization carries its own I and Q components of the optical signals (see Fig. 4). Data from an optical channel is passed to an FPGA via four high-speed time-interleaved 5-bit ADCs

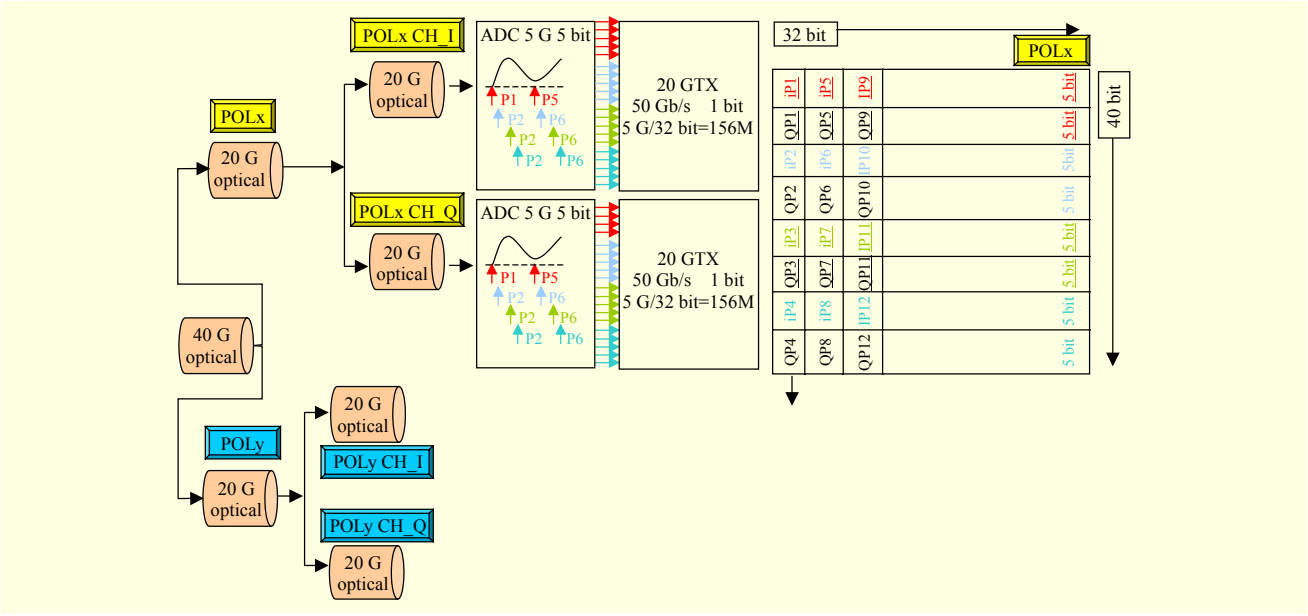

Fig. 4. Applying 40 GSample/s signal to FPGA.

sampled at 50 ps time shifts. The data obtained from the ADCs is transferred into the FPGA serially, bit by bit, using high-speed transmission channels (one GTX for each data bit). FPGAs can obtain high-speed serial data with high-speed transmission channels known as GTXs [10]. 20 GTX channels are used for each I and Q component per polarization. Sampled bits from time-interleaved ADC's are shown in Fig. 4. As can be seen from the bit numbers of the same column, there is one clock time shift between successive rows. This indicates that it is necessary to suitably order the data that is to be applied to the FFT core. Higher data communication speeds are possible by using faster ADCs and high-speed transmission channels such as GTHs and GTZs.

III. Fourier Transformation and Efficient FFT Implementation

1. FFT and Radix

A DFT of a signal provides a representation of the signal in the frequency domain. An FFT is a fast algorithm for calculating a DFT. An FFT provides a speed advantage by using the symmetrical and periodical values of a phase factor (twiddle factor) to calculate a DFT [4]. Equations for an N-point DFT are given in (3) and (4) below [11], where n represents the discrete time-domain index and k is the normalized frequency-domain index.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \tag{3}$$

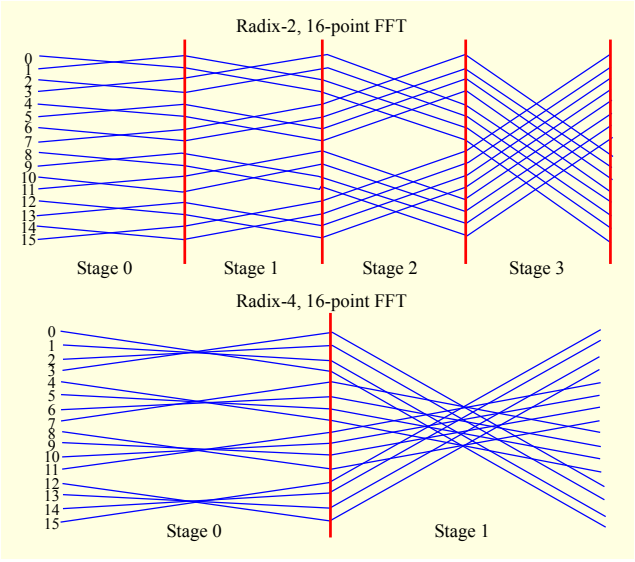

Fig. 5. Comparing number of stages in 16-point Radix-2 and 16-point Radix-4 FFTs [13].

$$W_N^{kn} = e^{-2\pi j\left(\frac{kn}{N}\right)} = \cos\left(2\pi\frac{kn}{N}\right) - \mathrm{j}\sin\left(2\pi\frac{kn}{N}\right). \tag{4}$$

Different FFT algorithms provide different benefits, but there is always a trade-off between computation speed and used FPGA area. Reducing an FFT's computation time reduces hardware complexity [12]. Different decomposition methods are available, such as Radix-2 (Radix-2 nodes are known as Butterfly nodes) and Radix-4 (Radix-4 nodes are known as Dragonfly nodes). The different stages of an N-point FFT can be calculated in log 4(N) operations for Radix-4 and in log 2(N)

operations for Radix-2, as shown in Fig. 5.

If the computational cost of multiplication is taken into consideration, then Radix-2 brings additional integer twiddle factors at angles of 0° and 180°, and Radix-4 brings additional integer twiddle factors at angles of 0°, 90°, 180°, and 270°. Sine and cosine equivalents of the aforementioned angles within a unit circle are simple integers and do not place any additional burden on the multiplication load, as shown in Fig. 6. Although a high Radix number reduces the number of computation stages, Radix-8 is not preferred, because it brings fractional twiddle factors ($\sqrt{2}$) in a unit circle at angles of 45°, 135°, 225°, and 315° [13]. If Radix-4 is compared to the Radix-2 algorithm, then Radix-4 has a higher complexity and less computational cost. A decimation-in-frequency (DIF) Radix-4 FFT approach is used in our implementation, as it is an approach that is frequently preferred to reduce computational complexity [14].

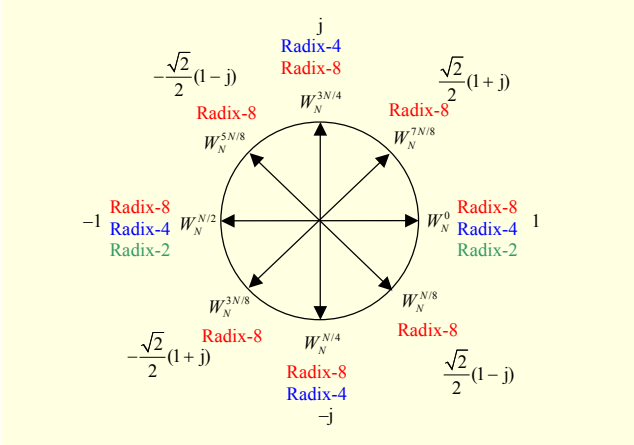

Fig. 6. Twiddle factors for Radix-2, Radix-4, and Radix-8.

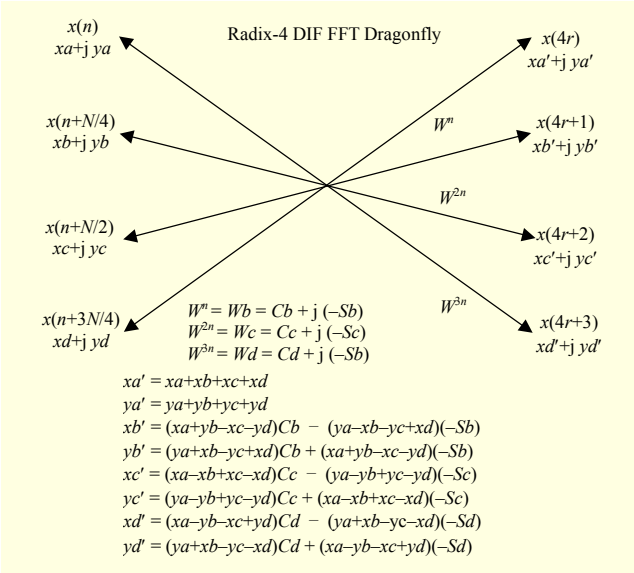

Fig. 7. Radix-4 DIF FFT Dragonfly [15].

The operations for every Dragonfly node are described in Fig. 7. When $\theta = 0$ in equations xa' and ya', then $\sin(0) = 0$ and $\cos(0) = 1$, which provides a simplification in calculation. As a result, only cosine-related calculations are enough for the first equation. This means that the first branch of each Dragonfly node can be calculated by using only addition operations.

$$xb' = (xa + yb - xc - yd)Cb - (ya - xb - yc + xd)(-Sb), \quad (5)$$

$$yb' = (ya - xb - yc + xd)Cb + (xa + yb - xc - yd)(-Sb), \quad (6)$$

$$xb' = P \times Cb - T \times (-Sb), \quad (7)$$

$$yb' = T \times Cb + P \times (-Sb). \quad (8)$$

If we analyze the equations in Fig. 7 over all values of θ, with the exception of $\theta = 0$, then it is clear (for example, in (5) and (6)) that twiddle factors are common to both equations; imaginary and real parts are consecutively used for both equations (see the P and T parts in (7) and (8)). Using the same hardware for common parts in the equations in Fig. 7 reduces the hardware complexity.

2. Serial vs. Parallel and IP Solutions on Market

Serial or parallel methods are chosen depending on the requirements of the application at hand. Serial approaches are usually suitable for general-purpose designs. On the other hand, in high-speed applications, parallel realization becomes inevitable, since it is the fastest method. However, its logic-area consumption is in the order of multiples of the serial realization and requires a longer implementation time. The advantage of a fully parallel design is shown in Fig. 8.

FFT IP Core solutions available on the market for FPGAs,

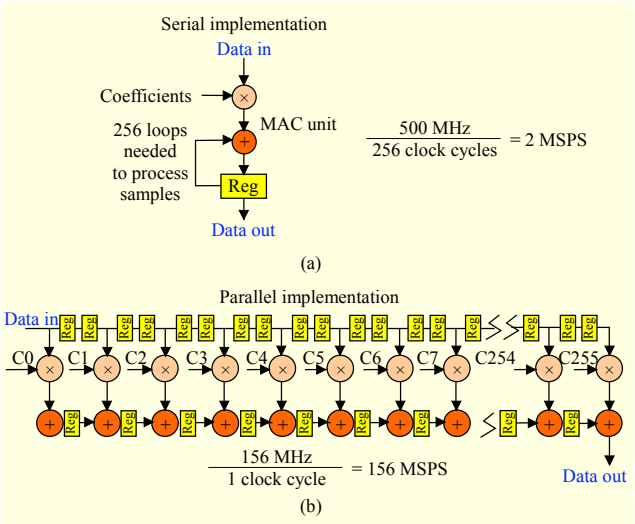

Fig. 8. Serial vs. parallel [16].

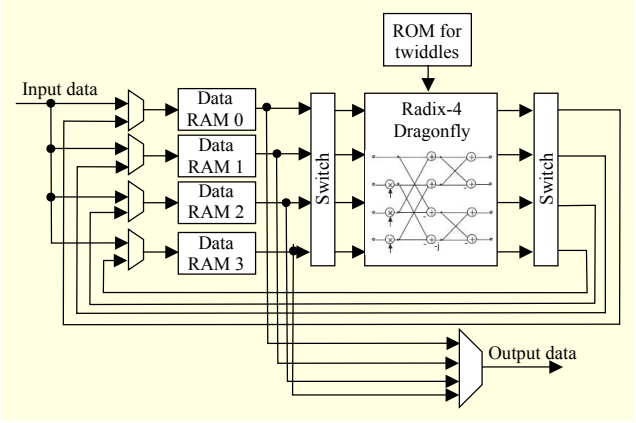

Fig. 9. Serial-processing FFT IP Core (Xilinx) [17].

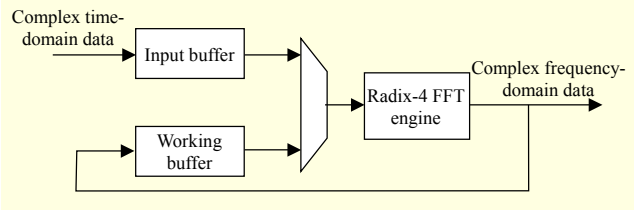

Fig. 10. Serial-processing FFT IP Core (Commsonic) [18].

provided by IP Core suppliers or IDEs of the popular FPGA producers, are based on a serial processing technique [17]–[18]. FFT IP Cores, based on serial processing, generally have only one Radix-4 (or Radix-2) node, and such a node is used over and over for every twiddle factor to overcome the disadvantage of logic area consumption; examples of such FFT IP Cores, available in today's market, are shown in Figs. 9 and 10. Although a serial processing approach requires less logic area, the speed of its data processing performance is far below that desired for optical data transmissions.

As an example, a serial processing–based 256-point FFT IP Core needs 256 clock pulses to perform transformation operations. Two hundred and fifty six samples of I and Q input data are needed to be applied to the core, one by one in every clock (see Fig. 4). In this case, 255 extra clock cycles are needed to process the data, taken in one clock cycle from the optical line. In our application, 256 samples of I and Q input data are collected from the optical system in one clock cycle. The data have to be buffered when serial architecture is preferred, but in this case, the buffer is overloaded in a short time or else an unlimited buffer size becomes necessary. Therefore, a fully parallel architecture is unavoidable for an optical communication system, which means every branch of such an architecture needs to be taken into consideration to obtain an optimal solution.

3. Multiplication

Multiplication is an intensively used operation in FFT computations and one that is performed by FPGAs. In FPGAs, intensive use of multiplication in FFT computations, itself a challenging issue, is generally handled by dedicated multipliers (DSP48 blocks for Xilinx) [19]. These specialized and dedicated blocks are limited, so they should be used effectively. Therefore, FFT IP Cores that are based on serial architecture use a limited number of dedicated multipliers over and over for different FFT nodes. Dedicated blocks can also be used for parallel FFT multiplication operations; however, the total number of multipliers available in the latest state-of-the-art FPGA is not enough for a 256-point Radix-4 FFT calculation. In our application, these prominent resources are reserved for other data processing operations due to their high-frequency clock rates. Details of FFT equations are carefully investigated for minimum size usage in FPGA and maximum computation speed.

There are two multiplication operations for every equation in Fig. 7. For example, the real part of a twiddle factor is denoted by Cb and the imaginary part by $-Sb$, in (5) and (6). If the sizes of an FFT and Radix are determined, then it means that the numerical values of the twiddle factors on every node are constant and known. Hence, the operation is turned into multiplication by a constant.

One method of fast calculation for multiplication by a constant involves the use of a memory-based system. In such an approach, precalculated constant twiddle factors are stored in a RAM block as a lookup table. Unfortunately, there is not enough available RAM block capacity to perform simultaneous access in a fully parallel design. Even state-of-the-art FPGAs in the market do not have enough RAM block capacity to access twiddle factors at the same time. Using common addresses for common values may reduce the size of a RAM block, but this will then cause memory conflicts.

Another solution for multiplication by a constant is one that is known as the shift-add method. Fortunately, both the shift operation and the add operation can easily be handled on FPGAs. When using this method, only configurable logic blocks are to be used as opposed to dedicated multipliers. In the shift-add method, the constant value is converted into a binary representation. If the digit under operation is "1," then the multiplicand is shifted by one bit and the shifted value is added to the result; otherwise, an addition operation is not performed. After this shifting process is completed for all bits, the multiplication operation ends. Details of the operation are shown in Fig. 11.

The number of logic units to be used for multiplication depends on the number of 1s in the binary representation of the constant. First of all, twiddle factors are calculated and then used as constant values. The number of 1s in the constant values is determined to figure out the required number of

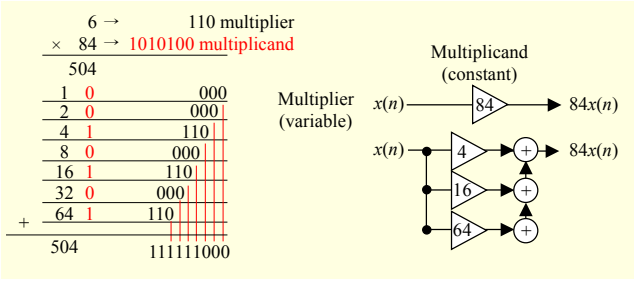

Fig. 11. Shift-add method for multiplication.

Table 1. Different expanded values for first eleven twiddle factors on unit circle with $N = 256$ points.

		Expand with 16	Expand with 64	Expand with 256	Expand with 1024
$n = 0$	1,000 + 0,000i	16 + 0i	64 + 0i	256 + 0i	1,024 + 0i
$n = 1$	0,999 – 0,024i	16 – 0i	64 – 2i	256 – 6i	1,024 – 25i
$n = 2$	0,998 – 0,049i	16 – 1i	64 – 3i	256 – 13i	1,023 – 50i
$n = 3$	0,997 – 0,073i	16 – 1i	64 – 5i	255 – 19i	1,021 – 75i
$n = 4$	0,995 – 0,998i	16 – 2i	64 – 6i	255 – 25i	1,019 – 100i
$n = 5$	0,992 – 0,122i	16 – 2i	64 – 8i	254 – 31i	1,016 – 125i
$n = 6$	0,989 – 0,146i	16 – 2i	63 – 9i	253 – 38i	1,013 – 150i
$n = 7$	0,985 – 0,170i	16 – 3i	63 – 11i	252 – 44i	1,009 – 175i
$n = 8$	0,980 – 0,195i	16 – 3i	63 – 12i	251 – 50i	1,004 – 200i
$n = 9$	0,975 – 0,219i	16 – 4i	62 – 14i	250 – 56i	999 – 2,24i
$n = 10$	0,970 – 0,242i	16 – 4i	62 – 16i	248 – 62i	993 – 2,49i

Table 2. MAE as result of using different expanding factors and inputs with different numbers of bits.

# input bits / Expanding factor	5 bit		6 bit		7 bit		8 bit	
	Real	Imag	Real	Imag	Real	Imag	Real	Imag
1,024	1.111	1.175	1.142	1.213	1.193	1.202	1.198	1.280
512	1.099	1.179	1.160	1.210	1.183	1.376	1.281	1.405
256	1.164	1.179	1.316	1.325	1.501	1.542	2.140	1.955
128	1.223	1.308	1.734	1.695	2.423	2.365	4.319	4.185
64	1.597	1.678	2.727	2.747	5.066	5.022	9.797	9.064
32	2.635	2.631	5.155	4.325	10.196	8.867	18.516	18.131
16	3.446	3.944	7.138	6.804	14.800	14.065	28.260	29.093
8	8.387	8.382	16.461	16.912	36.352	35.018	63.428	64.237

adders. The equation used to calculate twiddle factors is $W_N^n = e^{(-j2\pi n)/N}$. In this study, N is 256; therefore, the equation becomes $W_{256}^n = e^{(-j2\pi n)/256}$, where N represents the number points that are to be equally spaced on the circumference of a unit circle. The twiddle factors on the unit circle in Fig. 6 are fractional numbers.

Performing real-time operations on high-speed channels with fractional numbers is a challenging task. Using fixed-point arithmetic instead of floating-point arithmetic helps to reduce hardware requirements and processing times [20]. If fixed-point arithmetic is chosen, then twiddle factors will have to be normalized; however, since any required normalization factor depends on an individual user's needs, we can state that the normalized values of any two neighboring points on a unit circle must be distinguishable; namely, enough numbers of bits have to be used in the resulting fixed-point representations. As an example, let us consider the eleven twiddle factors (W_{256}^0 to W_{256}^{10}), which are normalized by different expanding factors, shown in Table 1.

Upon evaluating the values in Table 1, we can see that expanding the twiddle factors by 10 bits (2^{10} = 1,024) seems appropriate for reaching distinguishable successive values on a unit circle.

In this study, inputs taken from an optical line are converted by five-bit ADC's, and the selected appropriate normalization factor is 1,024 (10 bits). These parameters are application dependent and can be changed by a user when needed. Using different numbers of bits for normalization will cause different rounding errors. Preferable mean absolute error (MAE) values with respect to different normalization factors are shown in Table 2. The numbers of bits used for expanding twiddle factors can be increased until a desired MAE value is reached for an application.

4. Design and Implementation

The architecture of the upper-most level of a Radix-4 256-point FFT has four stages. Every stage except the last one consists of two subparts — adder blocks and multiplier blocks. A substructure of the upper-most level architecture has four stages, as shown in Fig. 12. Every adder block has 64 sub-blocks for Dragonfly-node addition operations. A Dragonfly node has eight input values (four real and four imaginary). Adder sub-blocks perform addition and subtraction operations for Dragonfly nodes in (5)–(6), as depicted in Fig. 13.

The inner architecture of the multiplier blocks of stages 1–3 is more complex (see Fig. 14). In such multiplier blocks, multiplicands are converted to unsigned values while keeping the sign values in mind to perform a proper multiplication (performed in an "ABS" block). After multiplications are completed, the results are converted to signed values to get the correct xb' and yb' (in a "Return ABS" block).

A multiplication block has shift/add-based multipliers for coefficients (here, the real and imaginary parts of the twiddle

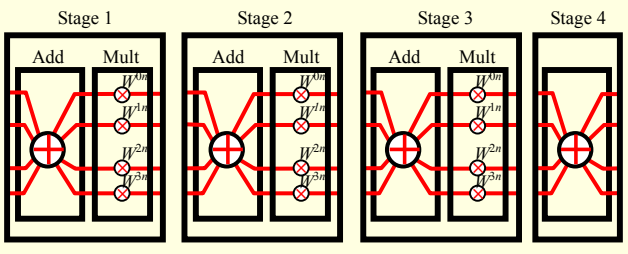

Fig. 12. Architecture of upper-most level of Radix-4 256-point FFT.

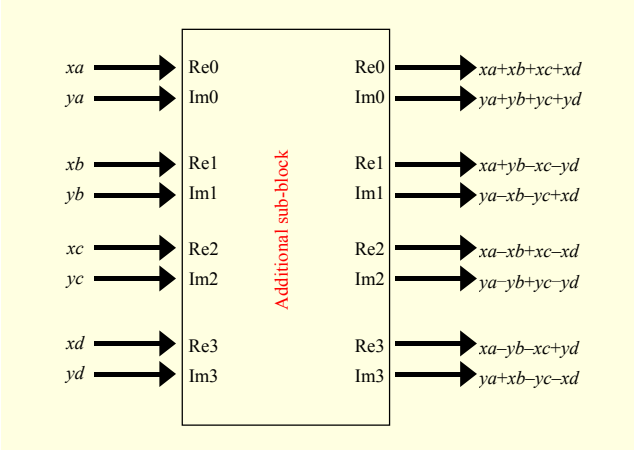

Fig. 13. Adder sub-block structure.

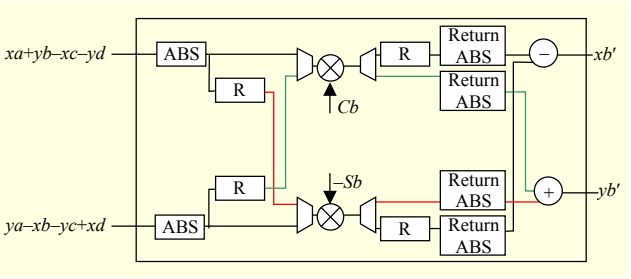

Fig. 14. Multiplication block structure.

Table 3. FFT synthesis results.

Device utilization summary (estimated values)			(·)
Logic utilization	Used	Available	Utilization
Number of slice registers	123,406	708,480	17%
Number of slice LUTs	213,700	354,240	60%
Number of fully used LUT·FF pairs	82,598	254,508	32%
Timing summary: --- Speed grade: –2 Minimum period: 2.977 ns (maximum frequency: 335.914 MHz) Minimum input arrival time before clock: 7.304 ns Minimum output required time after clock: 0.659 ns Minimum combinational path delay: no path found			

factors). In (7) and (8), the T and P parts are common to both equations; thus, the same logic block can be used for computing them consecutively. Similarly, the same shift/add block is used twice for computing xb' and yb' so as to reduce logic area consumption in an FPGA. In this case, data collision has to be avoided and any introduced time delay has to be kept in mind for the performance of the whole FFT core.

In the implementation phase of a multiplication block, at the first clock signal, parts of (7) are calculated and partial results are copied to output registers. Parts P and T are also copied to input registers for calculation of (8). At the second clock signal, the P and T values from the input registers are multiplied by their respective coefficients, Cb and $-Sb$, and the results of the multiplications are summed to form yb' at the output. Simultaneously, the partial values from the output registers are subtracted to form xb'. As a result, xb' and yb' are synchronously obtained at the output.

A block diagram of the aforementioned multiplication block is depicted in Fig. 14. The multiplication block in Fig. 14 is used in stages 1, 2, and 3 in Fig. 12 and is run at double clock speed to catch up on the speed of the remaining blocks in the FFT core. When a twiddle factor has an angle of $\theta = 0$, there is no need for multiplication. Then, the values multiplied by "1" are held in a register to synchronize with other blocks, which reduces logical area consumption in FPGAs.

IV. Synthesis and Simulation Results

In this section, the logic resources used in an FPGA for the implementation of an FFT core, and simulation and test results for the proposed high-speed FFT core are presented. In our implementation, VHDL language is preferred as HDL. This high-speed FFT core is especially designed for optical data transmission; therefore, a fully parallel architecture is used. The Virtex-6 565T is preferred as an FPGA chip, since huge numbers of logic units are necessary. In this study, FFT computation results are obtained only with nine-clock latency. The synthesis results are shown in Table 3.

In our test scenario, I and Q input data (each has 256 samples) are copied into our FFT processor in one clock. The input data and results are shown in Figs. 15, 16, and 17 partially, because it is not possible to display all 512 (256 I + 256 Q) values on one page. Our simulation results are verified using MATLAB. During simulations, the first FFT's of the test vectors are calculated using MATLAB. Then, the same test vectors are copied into an FPGA test bench. After a simulation ends, the FFT results of the test vectors are observed in Questa (as in Fig. 15).

The test bench results obtained from the Questa environment are compared the to MATLAB results, as shown in Fig. 16.

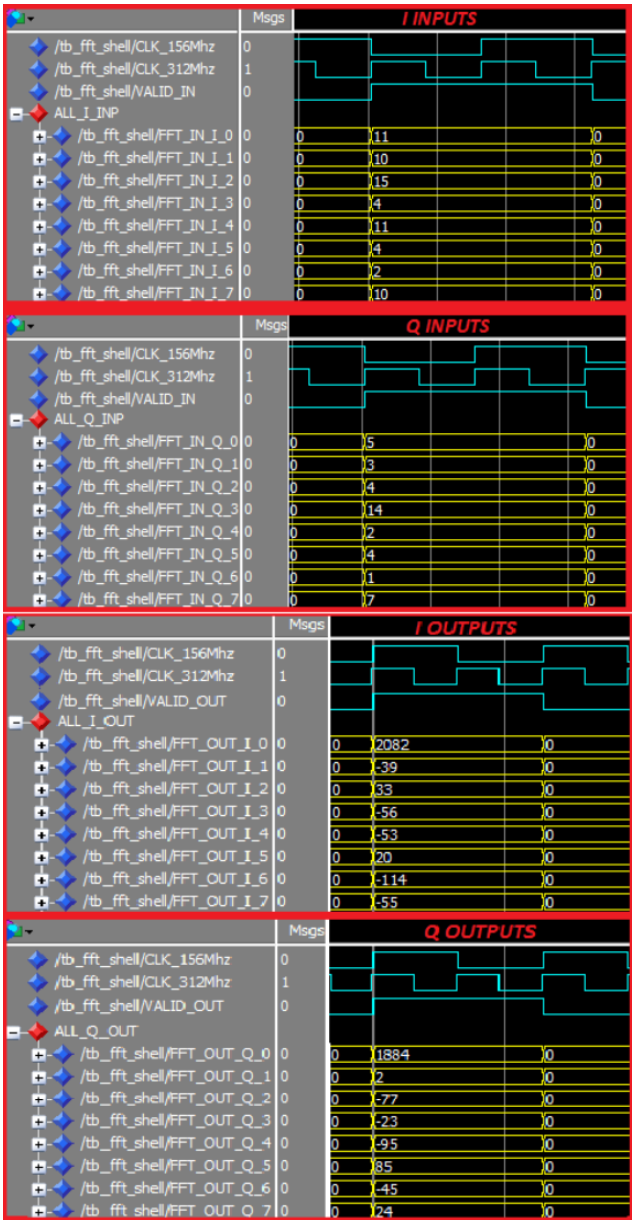

Fig. 15. Questa test bench results.

	MATLAB Results		Questa Results	
1	2082	1884	2082	1884
2	-39,9843	1,122015	-39	2
3	30,81792	-75,7036	33	-77
4	-57,2995	-22,5662	-56	-23
5	-53,0205	-94,1363	-53	-95
6	22,77702	85,59045	20	85
7	-114,459	-45,7134	-114	-45
8	-53,938	25,03714	-55	24
9	2,099795	1,237205	2	0
10	-36,7657	-58,0585	-37	-59

Fig. 16. Simulation comparison of Questa results.

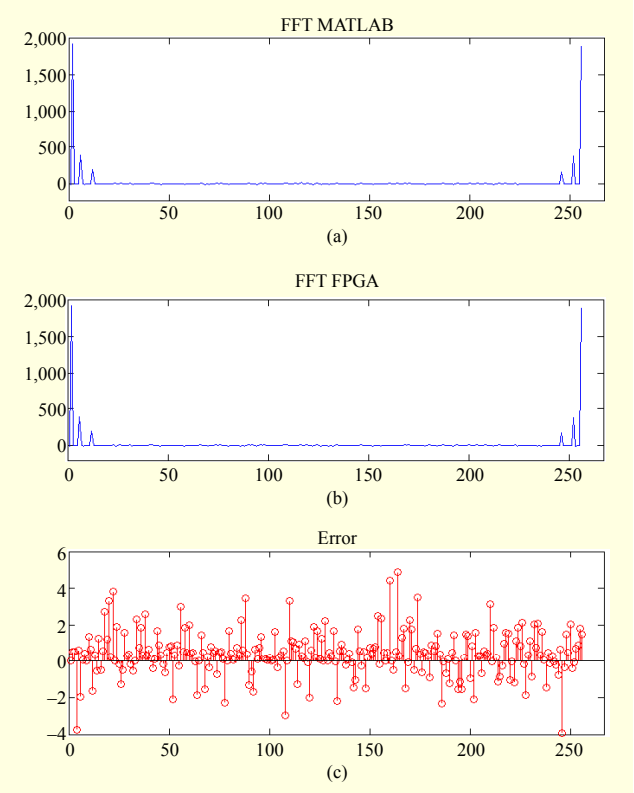

Fig. 17. Results comparison of FFT by MATLAB to FFT by FPGA.

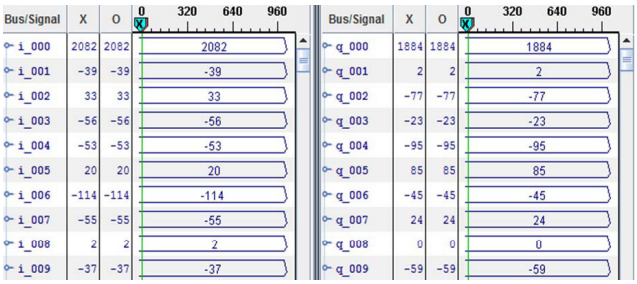

Fig. 18. Chip Scope logic analyzer results.

The results have immaterially small errors because of the rounding effect on the twiddle factors, which are represented by 10 bits. The different test vectors are applied to the test bench, and tests are repeated for validation purposes. Simulation results are captured and compared to MATLAB results; the resulting rounding errors are shown in Fig. 17. Simulation results without errors do not necessarily indicate that an FFT core will definitely run on an FPGA. An FFT core implementation is needed to be uploaded into FPGA. An FFT core implementation is validated once it has been run on a real FPGA environment.

The same test vectors simulating the optical data are also loaded into RAM blocks in the FPGA and applied to the proposed FFT core as the input variables to validate that our implementation runs on a real hardware platform. The outputs

of the FFT core are investigated via Chip Scope (a virtual logic analyzer). It is observed that the implemented FFT core results are exactly the same with Questa, as shown in Fig. 18.

V. Conclusion

In this study, an optimized Radix-4 fully parallel FFT processor on a Virtex-6 565T FPGA platform is designed and implemented. The 256-point Radix-4 FFT has a maximum clock frequency of 312 MHz. The utilization summary includes 123,406 slices and 213,700 LUTs on the Virtex-6 565T FPGA platform. In this work, different rounding schemes are analyzed and compared. In our application, 5-bit inputs and an expanding factor of 1,024 are used; the MAE of the FFT result is about 1.1. Moreover, to address different user needs, different numbers of bits and expanding factors were provided in Table 2. At first glance, the logic area consumption is excessive when compared to pipelined architectures on the market such as [12], [16], [17], and [18]. On the other hand, in the case of optical communication, a fully parallel architecture is inevitable to reach excessive speeds at the cost of higher logic area consumption.

References

[1] S. Haykin, "Signal Processing: Where Physics and Mathematics Meet," *IEEE Signal Process. Mag.*, vol. 18, no. 4, July 2001, pp. 6–7.

[2] B. Zhou, Y. Peng, and D. Hwang, "Pipeline FFT Architectures Optimized for FPGAs," *Int. J. Reconfigurable Comput.*, vol. 2009, Jan. 2009, pp. 1–9.

[3] J.M. Palmer, "*The Hybrid Architecture Parallel Fast Fourier Transform (HAPFFT)*," M.S. thesis, Brigham Young University, Provo, UT, USA, 2005.

[4] H. Kaptan, A. Tangel, S. Sahin, "FPGA Implementation of FFT Algorithms Using Floating Point Numbers," *Int. Conf. Electr. Electron. Eng.*, Bursa, Turkey, Dec. 2007, pp. 114–117.

[5] J. Gajewski and M. Przywecki, *Deliverable D.4.4: Events Proceedings*, Porta Optica Study, June 26, 2007, p. 61. Accessed June 15, 2013. http://www.porta-optica.org/publications/POS-D4.4_Events_proceedings.pdf

[6] K. Ishihara, "Frequency-Domain Equalization for High-Speed Fiber-Optic Transmission Systems," *Wireless Signal Process. Netw. Workshop*, NTT Network Innovation Laboratories NTT Corporation, Tohoku University, Japan, 2010.

[7] *Chromatic Dispersion (Optics)*, Knowledgebase at FiberOptic.com, 2012. Accessed June 15, 2013. http://www.fiberoptic.com/fiber_characterization/pdf/chromatic_dispersion.pdf

[8] M. Mussolin, "*Digital Signal Processing Algorithms for High-Speed Coherent Transmission in Optical Fibers*," M.S. thesis, Università deglistudi di Padova Facoltà di Ingegneria, Padova, Italy, 2010.

[9] G. Chauvel, *Dispersion in Optical Fibers*, Anritsu Corporation, 2012. Accessed June 15, 2013. http://www.ausoptic.com/Alltopic/Download/Disp_in_Opt_Fibers_PMD_CD.pdf.http://www.mericipristroje.eu/download/files/White-Paper_Dispersion-in-Optical-Fibers_PMD_CD_Ltr.pdf

[10] F. Dinechin, H. Takeugming, and J.-M. Tanguy, "A 128-Tap Complex FIR Filter Processing 20 Giga-Samples/s in a Single FPGA," *Asilomar Conf. Signals, Syst. Comput.*, Pacific Groove, CA, USA, Nov. 7–10, 2010, pp. 841–844.

[11] A.V. Oppenheim and R.W. Shafer, "*Discrete-Time Signal Processing*," 2nd ed., NJ, USA: Prentice Hall, 1998, p. 657.

[12] V.F.J. Alberto, R.T.R. Jesus, and O.M. Alejandro, "VHDL Core for 1024-Point Radix-4 FFT Computation," *Int. Conf. Reconfigurable Comput. FPGA's*, Puebla City, Mexico, Sept. 28–30, 2005, pp. 4–24.

[13] B. Baas, *Handout FFT2.pdf*, UC Davis, 2012. Accessed June 15, 2013. http://web.ece.ucdavis.edu/~bbaas/281/slides/Handout-.fft2.pdf

[14] A. Bonilla et al., "Design and Implementation of Fast Fourier Transform Algorithm in FPGA," *Simpósio Brasileiro de Telecomunicações*, Brasilia, Brazil, Sept. 13–16, 2012.

[15] C. Wu, "Implementing the Radix-4 Decimation in Frequency (DIF) Fast Fourier Transform (FFT) Algorithm Using a TMS320C80 D0 DSP," Texas Instruments, Taiwan, Application Report: SPRA152, 1998.

[16] S. Gallagher, *Mapping DSP Algorithms into FPGAs*, Xilinx Inc., 2012. Accessed June 15, 2013. http://www.ieee.li/pdf/viewgraphs/mapping_dsp_algorithms_into_ fpgas.pdf

[17] Xilinx, *DS260 LogiCORE IP Fast Fourier Transform v7.1*, Xilinx Inc., 2012. Accessed June 15, 2013. http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf

[18] Commsonic, *General-Purpose FFT Core*, *CMS0001*. Accessed June 15, 2013. http://www.commsonic.com/downloads/sd0001.pdf

[19] Xilinx, *DS260 LogiCORE IP DSP48 Macro v2.0.* Accessed June 15, 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_3/sysgen_ref.pdf, 2012

[20] A. Ferizi et al., "Design and Implementation of a Fixed-Point Radix-4 FFT Optimized for Local Positioning in Wireless Sensor Networks," *Int. Multi-conf. Syst., Signals Devices*, Chemnitz, Germany, Mar. 20–23, 2012, pp. 1–4.

Gokhan Polat received his BS and MS degrees in electronics and communication engineering from the University of Kocaeli, Turkey, in 2010 and 2014, respectively. His research interests include embedded systems and signal processing.

Sitki Ozturk received his BS, MS, and PhD degrees in electronics and communication engineering from the University of Yildiz, Istanbul, Turkey, in 1982, 1987, and 1995, respectively. He is currently working as an assistant professor with the Department of Electronics and Telecommunication, University of Kocaeli, Turkey. His research interests include switching networks, industrial networks, and communication.

Mehmet Yakut received his BS and MS degrees in electronics and communication engineering from the University of Yildiz, Istanbul, Turkey, in 1987 and 1991, respectively, and his PhD degree in electronics and communication engineering from the University of Kocaeli, Turkey, in 2002. He is currently working as an assistant professor with the Department of Electronics and Telecommunication, University of Kocaeli. His research interests include wireless sensor networks, embedded systems, and image processing.