Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 792862, 7 pages
http://dx.doi.org/10.1155/2015/792862

Research Article

High Real-Time Design of Digital Pulse Compression Based on FPGA

Xiujie Qu,[1] Cuimei Ma,[2] Shixin Zhang,[1] and Sitong Lian[1]

[1]*School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, China*
[2]*Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China*

Correspondence should be addressed to Xiujie Qu; quxiujie@bit.edu.cn

Received 31 July 2014; Revised 26 September 2014; Accepted 8 October 2014

Academic Editor: Wuhong Wang

Because of the poor real-time performance of in-place fast Fourier transforms, a reconfigurable radix-4 FFT processor is studied and designed, which is based on decimation-in-time and single floating-point computation. The proposed method adopts "pipeline and parallel" structure for accessing multiple memories to improve the FFT processing speed, and then it is applied to digital pulse compression. The experimental result shows that the proposed FFT based on radix-4 computation can implement digital pulse compression rapidly under no adding hardware resources. The proposed method can be also applied to other radix FFTs.

1. Introduction

The concept of pulse compression begins from the Second World War. Because of the technological difficulty, pulse compression signal has applied to long-range surveillance and long-range tracking radars until the early 1960s. From the 1970s, with theoretic maturity and improving technology, pulse compression can be widely applied to radars of 3D, phased array, reconnaissance, fire control and so on. Therefore the performance of these radars is proved obviously. As many novel technologies and new devices are progressively used to radar systems, the property of radar systems is much improved. Specially, the emerging of the fast Fourier transform (FFT) lays a solid foundation. It is a hotspot for radar design to study high real-time pulse compression [1–3]. In order to obtain high range resolution, pulse compression has been used.

There are two methods for digital pulse compression (DPC), that is, convolution integral in time domain and matched filter in frequency [4]. In engineering design, DPC can be mainly implemented in frequency. So the matched filtering of LFM signal is realized and shown as Figure 1.

The procedure of matched filtering in frequency is (1) using FFT to make discrete time signal into discrete spectrum, (2) multiplying by the frequency response function of the filter, and (3) using IFFT to back into time series, namely, gaining the time domain single of DPC.

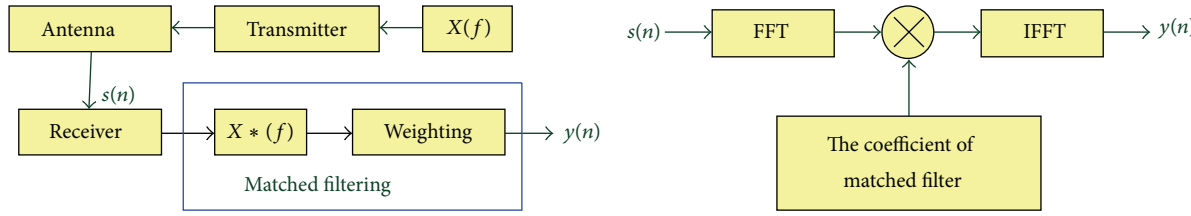
Suppose that the transmitter signal is $x(n)$, and the corresponding signal spectrum is $H(\omega)$; thus the function of the matched filter is $H(\omega) = X^*(\omega)$. Given that the receiver signal is $s(n)$, the output of pulse compression $y(n)$ is

$$y(n) = \mathrm{IFFT}\{\mathrm{FFT}\{s(n)\} \times H(\omega)\}. \quad (1)$$

It is seen that FFT/IFFT is still the concern in implementing DPC.

In the early days, general high-speed signal processor is mainly adopted to implement DPC, and this method is gradually eliminated as the new radar system with high-resolution real-time processing technology [5]. However, it becomes more and more mature for using hardware programmable logic, and FPGA (Field Programmable Gate Array) [6, 7] can implement DPC meeting precision requirements and increasing speed.

Usually, the structures of FFT processor have the constant geometry [8], the pipeline [9], and in place [10, 11]. The constant geometry FFT costs double memories. The pipeline architectures have high throughput, but they waste a large of resouces. Therefore, in-place architecture is employed to implement the FFT/IFFT, which are the main computations in DPC.



(a) Convolution integral in time domain

(b) Matched filtering in frequency domain

Figure 1: Two methods for implementing DPC of LFM.

A reconfigurable FFT processor based on "pipeline and parallel" structure is proposed, and it is applied to DPC.

2. Basic Theory

Given a length-N complex sequence $x(n)$, $\{n = 0, 1, \ldots, N-1\}$, its DFT $X(k)$ is also a length-N complex sequence defined by

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0, 1, \ldots, N-1, \tag{2}$$

where $W_N = e^{-j2\pi/N}$ and $j = \sqrt{-1}$. In this proposed algorithm, the sequence length N is a composite number and it is equal to ML; therefore, n can be expressed as

$$n = Mn_1 + n_0, \tag{3}$$

where $n_1 = 0, 1, \ldots, L-1$, $n_0 = 0, 1, \ldots, M-1$. n_0, n_1 are the numbers of columns and rows, respectively. M and L are the amounts of columns and rows, respectively.

In a similar way, the frequency index k for the output sequence is expressed as

$$k = Lk_1 + k_0, \tag{4}$$

where $k_1 = 0, 1, \ldots, M-1$, $k_0 = 0, 1, \ldots, L-1$. k_1 is a column vector, and k_0 is a row vector.

Equation (2) can be rewritten as

$$\begin{aligned} X(k) = (Lk_1 + k_0) = X(k_1, k_0) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n_0=0}^{M-1} \sum_{n_1=0}^{L-1} x(Mn_1 + n_0) W_N^{(Mn_1+n_0)(Lk_1+k_0)} \\ &= \sum_{n_0=0}^{M-1} \sum_{n_1=0}^{L-1} x(n_1, n_0) W_N^{Mn_1Lk_1} W_N^{Mn_1k_0} W_N^{n_0Lk_1} W_N^{n_0k_0} \\ &= \sum_{n_0=0}^{M-1} \sum_{n_1=0}^{L-1} x(n_1, n_0) W_N^{Mn_1k_0} W_N^{n_0k_0} W_N^{n_0Lk_1}. \end{aligned} \tag{5}$$

Then, we have

$$\begin{aligned} X(k_1, k_0) &= \sum_{n_0=0}^{M-1} \left[\sum_{n_1=0}^{L-1} x(n_1, n_0) W_N^{Mn_1k_0} \right] \cdot W_N^{n_0k_0} \cdot W_N^{n_0Lk_1} \\ &= \sum_{n_0=0}^{M-1} \left[\sum_{n_1=0}^{L-1} x(n_1, n_0) W_L^{n_1k_0} \right] \cdot W_N^{n_0k_0} \cdot W_M^{n_0k_1} \\ &= \sum_{n_0=0}^{M-1} \left\{ X_1(k_0, n_0) \cdot W_N^{n_0k_0} \right\} \cdot W_M^{n_0k_1} \\ &= \sum_{n_0=0}^{M-1} X_1'(k_0, n_0) \cdot W_M^{n_0k_1} \\ &= X_2(k_0, k_1), \end{aligned} \tag{6}$$

where

$$\begin{aligned} X_1(k_0, n_0) &= \sum_{n_1=0}^{L-1} x(n_1, n_0) W_L^{n_1k_0}, \\ X_1'(k_0, n_0) &= X_1(k_0, n_0) \cdot W_N^{n_0k_0}, \\ X_2(k_0, k_1) &= \sum_{n_0=0}^{M-1} X_1'(k_0, n_0) \cdot W_M^{n_0k_1}. \end{aligned} \tag{7}$$

From above, $x(n)$ can be mapped to $x(n_1, n_0)$; that is, N can be decomposed into M sets data with L points. The data of the M sets are independent of each other.

3. Proposed FFT Design

3.1. Structure. 4096-point radix-4 FFT is mainly discussed. The design is based on a "pipeline and parallel" structure. In the proposed design, only single radix-4 butterfly unit is needed; meanwhile, four dual-port memories are used for reducing processing time.

Before discussing the structure, a counter should be designed. It keeps consistent with the input data. When one datum is input, the counter adds "1". So, the range of the counter is from 0 to 4095.

First, the 4096 input data should be distributed into the four memories. According to the results of the modulo 4 of the designed counter, 4096 input data can be assigned to the four memories with depth being 1024, that is, RAM0~RAM3. Then each set of 1024 data can be computed by radix-4 FFT. Because the four sets of data are independent of each other according to (6), they can compute in parallel. At the same time, the four sets are the same in computing; therefore,

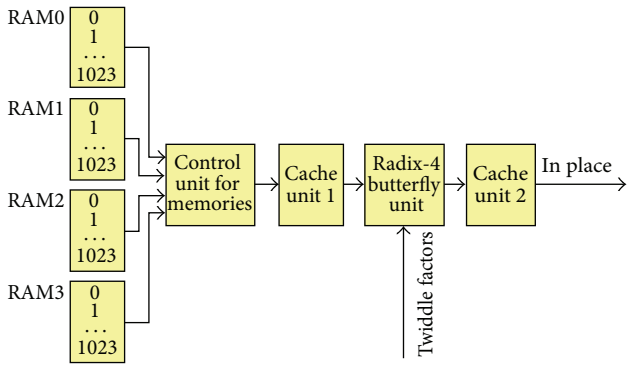


Figure 2: Pipeline structure for 4 sets of 1024-point FFT.

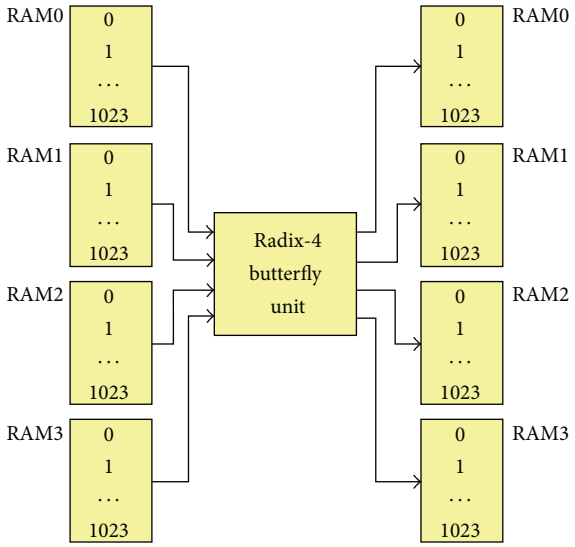


Figure 3: Parallel structure for 4 sets of 1024-point FFT.

the four FFTs can be implemented in pipeline structure. The pipeline structure is shown in Figure 2, and only one radix-4 butterfly is needed to implement four 1024-point FFTs.

Detailed explanation of each unit is as follows.

(i) RAM0~RAM3: four memories are used for storing the 4096 data. With high precision, the data are formatted in single-precision floating point. Therefore, the storage size of each memory is 64 K bits.

(ii) Control unit for memories: it is used to control the orders of memories accessed.

(iii) Cache unit 1: its function is to cache the pipeline data loaded from each memory. If the data are computed, the data should be in parallel.

(iv) Radix-4 butterfly unit: it is the basic radix-4 unit [12] and is computed in decimation-in-time. Considering hardware implementation, the butterfly unit mainly consists of floating-point adders and multipliers [13]. For the four operands, the first one needs not a multiplier, so there are three complex multipliers.

(v) Cache unit 2: its function is to cache the parallel results from radix-4 butterfly unit. Because the outputs are stored in pipeline, there is one cache unit to store them.

After the computation above, there is another stage to compute. In this stage, parallel accessing for the four memories is applied and the same butterfly unit is used. The structure is shown in Figure 3. After running 1024 radix-4 computations, the data are the results of 4096-point FFT.

Detailed explanation of each unit is as follows.

(i) The depth of each memory at the left part is 1024. The data are the results of 1024-point FFT.

(ii) One datum can be accessed from one memory and four data are obtained to input the radix-4 butterfly unit. There are 1024 times to process.

(iii) The parallel outputs are stored in parallel.

3.2. Pipeline Accessing for Memories. For the radix-4 FFT computation, there are two parts: (1) multiplication with coefficients; (2) 4-point DFT computation. The four operands of one 4-point butterfly are input in parallel, multiple by twiddle factors and run 4-point DFT.

Before being input into the butterfly unit, four data are accessed from one memory. Then, the four pipeline data are cached and output in parallel. Last, the four parallel data are input into the butterfly unit. The whole computing process of the butterfly unit is just as follows.

(1) According to the accessing addresses of the operands and the twiddle factors [14], the operands and the twiddle factors can be obtained. The four operands can be represented by Op(0)~Op(3).

(2) The operands and the twiddle factors in pipeline are input into cache unit 1 and the outputs are in parallel.

(3) The four parallel operands multiply by the four parallel twiddle factors and then addition and subtraction are computed. Finally, the results from multiplication with twiddle factors are $Op'(0)$~$Op'(3)$.

(4) After butterfly computation, the parallel outputs are $Op''(0)$~$Op''(3)$. Then, the outputs are input into cache unit 2 and pipeline outputs are obtained and then stored in place.

The timing diagram between the input and the output of the radix-4 butterfly unit is shown as in Figure 4.

From the timing diagram above, three cycles are idle in one radix-4 butterfly computation.

In the proposed method, the idle cycles can be used. So one butterfly unit is used for four 1024-point FFTs. It is important to arrange the order of loading data from the four sets, and the corresponding timing diagram is shown as in Figure 5.

Detailed explanations are as follows.

(i) The four operands from RAM0 are Op0(0)~Op0(3), and the complex multiplications with twiddle factors are $Op0'(0)$~$Op0'(3)$. The parallel outputs of butterfly unit are $Op0''(0)$~$Op0''(3)$.

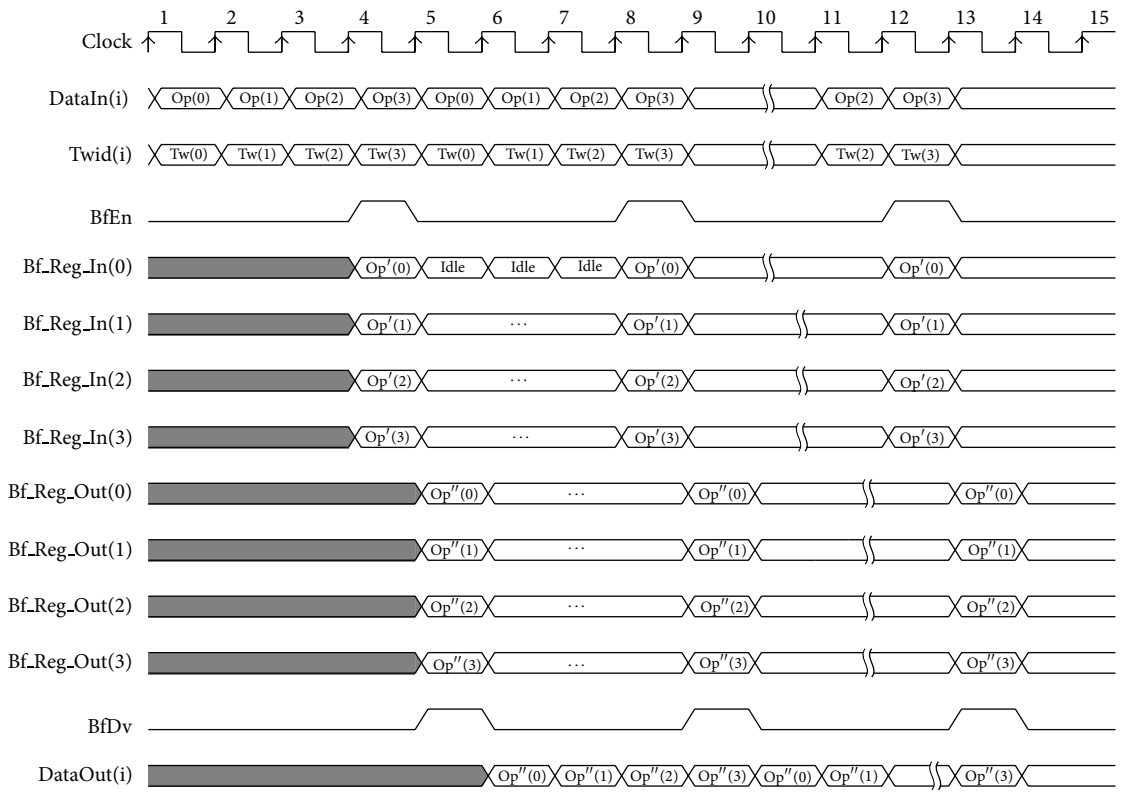


Figure 4: Timing of the input and the output of radix-4 butterfly unit in one memory.

(ii) The four operands from RAM1 are Op1(0)~Op1(3), and the complex multiplications with twiddle factors are $Op1'(0)$~$Op1'(3)$. The parallel outputs of butterfly unit are $Op1''(0)$~$Op1''(3)$.

(iii) The four operands from RAM2 are Op2(0)~Op2(3), and the complex multiplications with twiddle factors are $Op2'(0)$~$Op2'(3)$. The parallel outputs of butterfly unit are $Op2''(0)$~$Op2''(3)$.

(iv) The four operands from RAM3 are Op3(0)~Op3(3), and the complex multiplications with twiddle factors are $Op3'(0)$~$Op3'(3)$. The parallel outputs of butterfly unit are $Op3''(0)$~$Op3''(3)$.

In order to implement pipeline accesses for the four memories, reading data from RAM0~RAM3 are delayed for one cycle in turn. Because the four sets are independent of each other, the four sets need the same accessing addresses and twiddle factors. So the four FFTs share one set of twiddle factors and it can simplify FFT design.

3.3. Parallel Accessing for Memories. The outputs of the above 1024-point FFT multiple by the twiddle factors and they are new sequences. The new sequences are input into the butterfly unit. The last stage is to compute 1024 4-point DFTs, and the structure is shown as in Figure 6.

For the last stage, the same radix-4 butterfly is used. The 1024 4-point DFTs are computed. Because the operands of one butterfly belong to different memories, the four memories are accessed in parallel. The results are the 4096-point FFT.

4. Reconfigurable FFT Design

Because it needs IFFT computation in DPC, a configurable FFT is designed, which can be configured to FFT or IFFT.

According to the relationship between FFT and IFFT, the designed FFT can be configured to IFFT. Due to the inverse transformation,

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{-nk} = \mathrm{IDFT}[X(k), N]. \quad (8)$$

From (2) and (8), the main differences are the coefficients and the results being N times.

Therefore, modifying the FFT is as the following. First, we exchange real part with imaginary part of the input data and then compute FFT. Last, the real part and imaginary part are exchanged and then are divided by N. Finally the IFFT results are obtained.

Therefore, a control signal is set. When it is high, FFT computation is done; when it is low, IFFT computation is done. Thus, the proposed FFT processor is set as a reconfigurable FFT.

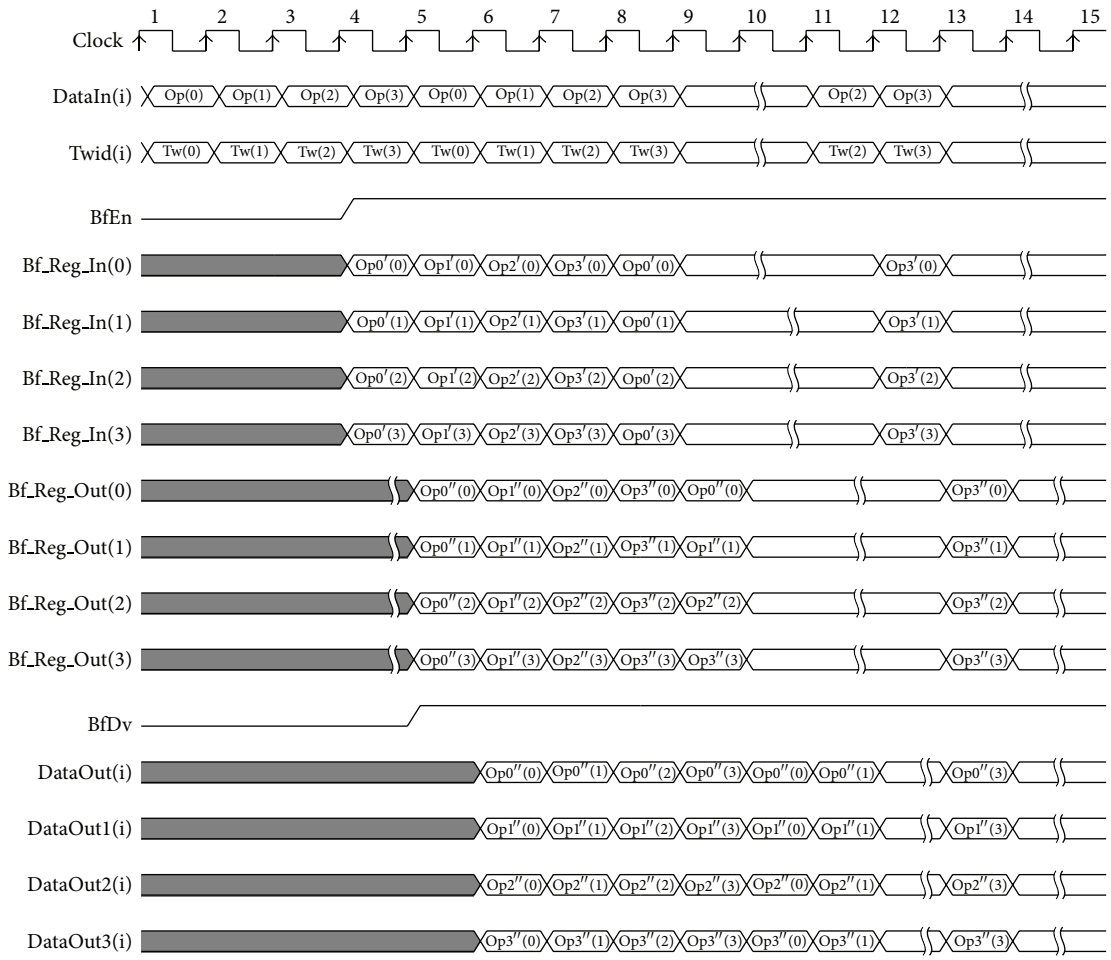


Figure 5: Timing for four memories in pipeline accessing.

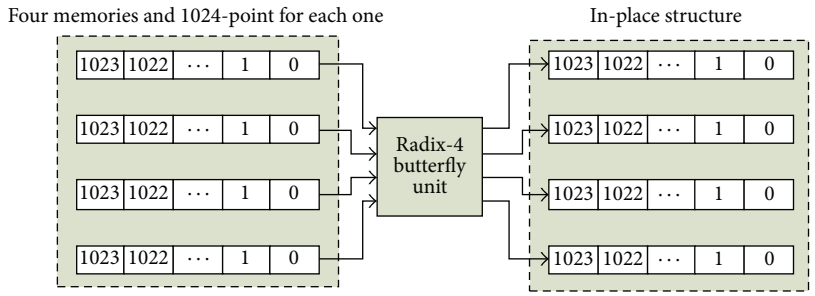


Figure 6: Structure of the last stage.

5. DPC Design

On demand of the actual engineering, there is 4096-point DPC to process. The process of DPC is to compute FFT and then multiply by coefficients and at last do IFFT. Radix-4 FFT is used to compute 4096-point FFT and four memories are accessed in "pipeline and parallel" structure.

First, 1024-point FFT should be implemented by one radix-4 butterfly unit, and the four sets are processed in pipeline; then the four memories are accessed in parallel. The stored results of 4096-point FFT in RAMs are in reversed order.

Second, the results of the above 4096-point FFT are read from memories, multiply by the matched filter coefficients and the results are stored in memories in pipeline. (In the example, hamming window is used for the matched filter coefficients.)

Last, data after multiplication with coefficients do IFFT. This step uses the reconfigurable FFT to process IFFT, followed by division by 4096. The division can be replaced

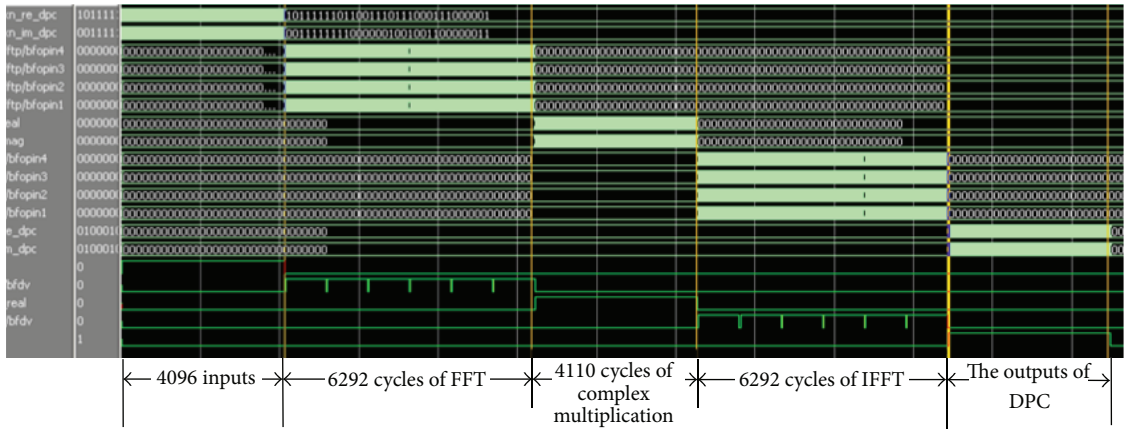


Figure 7: Timing of DPC.

Table 1: FFT processing time comparisons (cycles).

FFT processors	Time	Comparisons with the proposed design
In place with single butterfly	24259	17967↑
Xilinx FFT IP	10427	4135↑
C6701 floating-point DSP	12000	5708↑
The proposed design	6292	—

Table 2: FFT resources comparisons.

FFT processors	Number of FFTs	LUTs
Pipeline FFT [9]	4096	16561
FFT design [15]	4096	12467
The proposed design	4096	10427

by subtracting 12 because of floating-point operation being used.

6. Implementations and Analyses

FPGA is a large-scale programmable logic device with flexible logic cell, high integration, short developing time, and low developing cost; it is widely used in prototype design and prophase of new product development. Xilinx series FPGA is used for the proposed FFT and DPC verifications. The timing simulations are done in ModelSim and the resources are estimated by the device of Xilinx V6 series Xc6v1x240t (–1).

6.1. Implementation of the FFT Processor. The processing time periods and resources of the proposed FFT are listed in Tables 1 and 2, respectively.

From Table 1, the computation time of the proposed FFT is shorter than the compared methods. If 4096-point FFT is implemented with one memory and one butterfly, the processing time is 24259, which is 4 times as many as the proposed scheme. The running time of the other two methods which are based on the in-place architecture in spite of different platforms, is longer than the time of the proposed one. Therefore, the proposed design adopts "pipeline and parallel" accessing for memories to implement FFT and indeed reduces the processing time based on in-place architecture.

From Table 2, the LUTs of the proposed design are less than the two compared methods. Because the accessing addresses for the operands and the twiddle factors are generated for 1024-point FFT, not for 4096-point FFT, the waste resources can be reduced.

Therefore, the proposed method with "pipeline and parallel" structure is not only keeping less resources but also having higher processing speed to compute FFT and the above results show its advantage.

Table 3: Device utilization summary of DPC.

Logic resources	Used	Utilization
Slice registers	16251	5%
Slice LUTs	21418	14%
RAM/FIFO	36	8%
DSP48E1s	60	7%

6.2. Implementation of DPC. The resources of DPC are listed in Table 3. The maximum frequency is 122.128 MHz and satisfies the engineering demand.

The timing simulation of DPC is shown in Figure 7. The data source of 4096 numbers is generated by Matlab tool and the matched filtering coefficients are stored in ROM.

Figure 7 shows that the computing cycles of FFT or IFFT are 6292, and the cycles of multiplications with coefficients are 4110. So the total of processing cycles of DPC is 16694. If the clock period is set 100 MHz, the running time of 4096-point DPC computation is 166.94 μs.

Table 4 shows the processing time of DPC with different schemes. If one butterfly is used with in-place structure and one memory, the running clocks are 52656 and if Xilinx FFT IP core is used to compute FFT and IFFT in DPC, the

Table 4: Comparisons of DPC's processing cycles with methods.

FFT processors	Time	Comparisons with the proposed design
In place with single butterfly	52656	35934↑
Xilinx FFT IP	24992	8270↑
The proposed design	16722	—

time periods of DPC are 24992. There are many arithmetic operations, such as additions and multiplications, in DPC computation. Because the processing delays of the operations are not the same in the first two schemes, the processing time is different for them. As the processing delay of arithmetic operations of the proposed scheme is the same as the first one, from the result, the proposed method wastes less processing time than the first one. Therefore, the proposed FFT processor can improve the processing speed of DPC and has its advantages in engineering.

7. Conclusions

The proposed scheme for FFT structure based on "pipeline and parallel" access can improve processing speed and can be applied to DPC design for single-channel data. According to the processing flow of DPC, the stages of FFT and IFFT adopt the proposed FFT structure to achieve the goals of high speed and less resources, so the DPC system can obtain high real-time performance.

Meanwhile, the proposed method is also applied to multiple-channel data. The data of each channel can be stored in one memory and the multiple sets of multiple-channel data are stored in multiple memories.

Furthermore, the novel way can be used to other radix FFTs and simple mixed-radix FFTs. When radix-r FFT adopt the scheme, usually r memories are taken and N data are divided into the r memories and single radix-r FFT is employed. Simply mixed-radix FFT, for example, radix-2/4 FFT, can use the proposed method. We can store N data into four memories; thus there exists four memories for radix-2 butterfly computation. Two radix-2 butterflies can be computed in parallel. Because radix-4 butterfly can reconfigure into two radix-2 butterflies, radix-2/4 FFT can only use one radix-4 FFT to compute.

Therefore, the proposed method can have wide applications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. O. Coleman, "Cascaded coefficient number systems lead to FIR filters of striking computational efficiency," in *Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS '01)*, pp. 513–516, September 2001.

[2] L. Pang and H. Chen, "Implementation of digital pulse compression system based on FPGA," *Modern Electronics Technique*, vol. 33, no. 14, pp. 190–194, 2010.

[3] F. Li and T. Long, "A high-speed real-time digital pulse compression system based on TMS320C6201," in *Proceedings of the CIE International Conference on Radar*, pp. 557–561, Beijing, China, October 2001.

[4] S. G. Qadir, J. K. Kayani, and S. Malik, "Digital implementation of pulse compression technique for x-band radar," in *Proceedings of the 5th International Bhurban Conference on Applied Sciences and Technology (IBCAST '07)*, pp. 35–39, Islamabad, Pakistan, January 2007.

[5] F. Li and T. Long, "A high-speed real-time digital pulse compression system based on TMS320C6201," in *Proceedings of the CIE International Conference on Radar*, pp. 557–561, October 2001.

[6] H.-G. Yang, J.-B. Sun, and W. Wang, "An overview to FPGA device design technologies," *Journal of Electronics and Information Technology*, vol. 32, no. 3, pp. 714–727, 2010.

[7] W.-X. Liu, R.-H. Tang, H. Li, and X. Hu, "Design of an IPv4/IPv6 translator based on SOPC technology," in *Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC '08)*, pp. 786–790, Sanya, China, April 2008.

[8] C. Ma, H. Chen, J. Yu, and T. Long, "A novel conflict-free parallel memory access scheme for FFT constant geometry architectures," *Science China Information Sciences*, vol. 56, no. 4, pp. 1–9, 2013.

[9] C. Yu, M.-H. Yen, P.-A. Hsiung, and S.-J. Chen, "A low-power 64-point pipeline FFT/IFFT processor for OFDM applications," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 40–45, 2011.

[10] C. M. Ma, Y. Z. Xie, H. Chen, Y. Deng, and W. Yan, "Simplified addressing scheme for mixed radix FFT algorithms," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '14)*, pp. 8355–8359, Florence, Italy, May 2014.

[11] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Transactions on Circuits and Systems. I. Regular Papers*, vol. 52, no. 5, pp. 911–919, 2005.

[12] C. Ma, H. Chen, and L. Ma, "An efficient design of high-accuracy and low-cost FFT," in *Proceedings of the IET International Radar Conference*, pp. 1–4, April 2013.

[13] H. Chen, X. Qu, L. Pang, J. Yu, and T. Long, "An improved constant coefficient multiplication algorithm based on cascaded adder graph," *Science China Information Sciences*, vol. 56, no. 6, pp. 1–7, 2013.

[14] T. Lu, P.-K. He, H. Chen, and Y.-Q. Han, "A fast address generation scheme for FFT processor," *Transaction of Beijing Institute of Technology*, vol. 26, no. 1, pp. 68–71, 2006.

[15] H. X. Liu, J. Huang, and S. T. Huang, "A strategy of address generating for programmable high throughout FFT processor," *Signal Processing*, vol. 2, no. 25, pp. 185–193, 2009.