

Parallel Implementation of Arbitrary-Sized Discrete Fourier Transform on FPGA

Lin Shu, Jie Hao, Chengcheng Li, Hui Feng, Donglin Wang

Institute of Automation, Chinese Academy of Sciences

Beijing, China

{lin.shu, jie.hao, chengcheng.li, hufeng, donglin.wang}@ia.ac.cn

Abstract—Discrete Fourier Transform(DFT) is one of the frequently used kernels in a variety of signal processing applications. Most previous state-of-the-art work has focused on the transform size of a power of 2. And many implementations of non-power-of-two sized DFT is customized for a specific application. In this paper, a parallel-processing architecture based on Field-Programmable Gate Array (FPGA) for arbitrary-sized DFT is proposed. In particular, it is attractive to use to compute DFT with arbitrary prime size. A memory efficient data mapping scheme for twiddle factors is proposed, which reduces the storage size of twiddle factors from n^2 to kn (k is a constant). We implement a design with 196 processing elements, which is available for any transform size n from 14 to 1024. For a transform size of 59, the throughput in this design can reach to 737.5 Msps. The design can be easily extended to be one with more processing elements if the hardware has enough resources. And we have built a code generator to automatically generate designs with different numbers of processing elements.

Keywords—DFT, arbitrary-sized, parallel, FPGA

I. INTRODUCTION

Discrete Fourier Transform(DFT) is one of the frequently used kernels in a variety of signal processing applications. As we all know, Fast Fourier Transform(FFT) is a fast algorithm of DFT. The parallel FFT algorithm has been implemented and optimized with high performance on different hardware platforms[1-3]. Open source Spiral DFT/FFT IP Generator[4, 5] can automatically generate customized DFT soft IP cores in synthesizable RTL Verilog, which is a state-of-the-art design. But the transform size is restricted to be a power of two. In some applications, the required DFT size is irregular. It can't be computed exactly from FFT by zero-padding it to a fixed length (e.g. a power of two). So an efficient implementation of arbitrary-sized DFT algorithm is significant.

The transform size of DFT can be classified into two classes: composite number and prime number. For the size of a power of two, many general implementations[1-5] based on Cooley-Tukey FFT and the optimized algorithms are state-of-the-art. And most previous work on implementations of non-power-of-two sized DFT has focused on producing a solution for a specific situation[6-8] like a given transform size and performance requirement. The transform sizes considered in[6-8] are composite numbers, which are consisted of different factorizations, e.g. $1920 = 15 \cdot 2^7$.

These cases are implemented by mixed-radix FFT, while it is not general. It needs the support of different kinds of radix r kernels, which is a huge work. [9] has proposed an implementation of large-prime-sized DFT using the Bluestein FFT algorithm[10]. The Bluestein FFT[10] is a convolution-based algorithm for any transform size n . For a given n , the algorithm divides the DFT to a circular convolution of two vectors of length m , while $m = 2^{\lceil \log_2(n) \rceil + 1}$ is the smallest power of two greater than or equal to $2n-2$. In the worst case, m is almost as large as $4n$. Although the total numbers of complex multiplication operations can be reduced to $2n + m + 2m\log_2^m$, the hardware implementation is complicated.

In this paper, a parallel-processing architecture based on Field-Programmable Gate Array (FPGA) for fixed point DFT is proposed. It is consisted of simple processing elements. With the change of application requirements or update of hardware, detailed hardware implementation is easily extensible. It is available for arbitrary-sized DFT. In particular, it is attractive to use to compute DFT with arbitrary prime size. The implementation is based on matrix-vector multiplication. In order to reduce communication overhead, block-parallel computing scheme is chosen. Besides, we propose a memory efficient data mapping scheme for twiddle factors. The storage size of twiddle factors is reduced from n^2 to kn (k is a constant). The main contributions in this paper are summarized as follows:

- 1) A parallel-processing architecture based on FPGA for arbitrary-sized DFT, which is easily extensible.
- 2) A memory efficient data mapping scheme for twiddle factors.
- 3) A parallel implementation with 196 processing elements on Virtex-6 FPGA board, which can compute DFT of any transform size n from 14 to 1024.
- 4) A code generator which can automatically generate designs with different numbers of processing elements.

The rest of the paper is organized as follows. Section II is about the background, especially to analyze the twiddle factors of DFT. The proposed architecture and its implementation is introduced in section III. Section IV

presents experimental results and analysis. Section V concludes the paper.

II. BACKGROUND

Discrete Fourier Transform. The n point DFT of input sequence $x_i (i = 0, 1, \dots, n-1)$ is defined as:

$$y(k) = \sum_{i=0}^{n-1} x(i)w_n^{ki}, 0 \leq k \leq n-1 \quad (1)$$

Where the twiddle factor w_n^{ki} is given by:

$$w_n^{ki} = e^{-j2\pi \frac{ki}{n}} \quad (2)$$

It is obvious that the calculation of DFT can be the matrix-vector multiplication:

$$y = DFT_n x \quad (3)$$

$$DFT_n = [w_n^{ki}], 0 < k, i < n \quad (4)$$

$$x = [x_0 \ x_1 \ \dots \ x_{n-1}]^T \quad (5)$$

$$y = [y_0 \ y_1 \ \dots \ y_{n-1}]^T \quad (6)$$

A literal calculation of n point DFT by matrix-vector multiplication requires $O(n^2)$ operations. In this paper, efficient implementation of DFT is based on parallel matrix-vector multiplication. But the elements of matrix DFT_n is not irregular. Architecture in this paper will make full use of parallel matrix-vector multiplication to speed calculation, and consider the features of DFT to optimize performance as well.

Considering a 7 point DFT, as an example. The input sequence is given as $\{x_i, i = 0, 1, \dots, 6\}$. Then the equation (3) can be expressed as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w^1 & w^2 & w^3 & w^4 & w^5 & w^6 \\ 1 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} \\ 1 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} \\ 1 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} \\ 1 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} \\ 1 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad (7)$$

Alternatively the equation (7) can be written as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w^1 & w^2 & w^3 & w^4 & w^5 & w^6 \\ 1 & w^2 & w^4 & w^6 & w^1 & w^3 & w^5 \\ 1 & w^3 & w^6 & w^2 & w^5 & w^1 & w^4 \\ 1 & w^4 & w^1 & w^5 & w^2 & w^6 & w^3 \\ 1 & w^5 & w^3 & w^1 & w^6 & w^4 & w^2 \\ 1 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad (8)$$

So matrix DFT_n at most has n different elements. The storage size of matrix DFT_n can be reduced from n^2 to kn (k is a constant). And only a bit of logic about address generator is added. This will be introduced in section III-D.

III. ARCHITECTURE

A. System Overview and Specifications

The parallel-processing architecture is built upon parallel implementation of matrix-vector multiplication as well as features of DFT, using block-parallel computing scheme. As

showed in Fig.1, the system is consisted of 2D processing element array. Assuming that it has m processing elements, the array is divided into m_1 rows and m_2 columns, where $m = m_1 \times m_2$. The description in the rest of this section is based on this assumption. Each column of the array is seen as a subsystem, which has a shared memory. Shared memories store input sequences temporarily. Results from processing elements in the same row are transferred to adder tree unit in the same row, which accumulates these intermediate results. The adder tree units output final results in parallel.

System performance is related to the numbers of processing elements. Detailed analysis of the system will be described in section IV. The numbers of processing elements represent granularity of parallelism. With the change of application requirements or update of hardware n , the system is easily extensible.

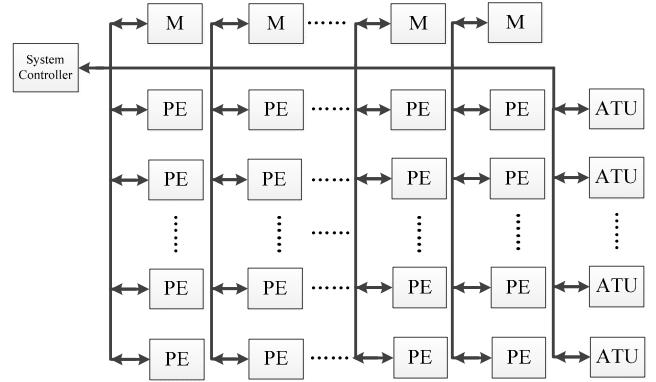


Fig. 1. System Overview. PE: Processing Element; ATU: Adder Tree Unit; M: Shared Memory.

B. Processing Element.

Processing element is a basic computation unit in this architecture, which shows in Fig.2. It can process matrix-vector multiplication in serial. There are two multipliers, two adders, two fifos, local memory, address generator and controller. When system is power on, parameters in the controller is configured and local memory is initialized with twiddle factors. The size of local memory limits DFT transform size processing element can handle. The length of fifo is equal to dimensions of submatrix. Address generator is especially customized to generate memory read address, which will be introduced in section III-D(algorithm 1).

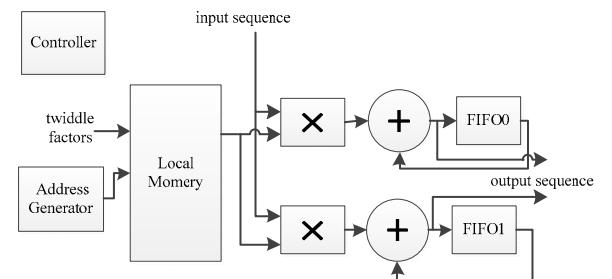


Fig. 2. Data Path and Framework of Processing Element. FIFO: First Input First Output

C. Parallel Computing Scheme

Parallel schemes of matrix-vector multiplication can be classified into three classes: row-parallel scheme, column-parallel scheme and block-parallel scheme. If using row-parallel scheme, it needs to reduplicate the input sequences to each processing element(this is defined as pre-communication). Considering a n point DFT, the total communication volume is mn if there are m processing elements. Column-parallel scheme needs post-communication instead. The total communication volume of a n point DFT is also mn , using the same hardware architecture.

Block-parallel scheme needs both pre-communication and post-communication. The pre-communication volume of a n point DFT is $\frac{m_1 n}{m_2}$. Because each m_1 processing elements have shared memory, the pre-communication volume can be reduced to zero. The post-communication volume of a n point DFT is $m_2 n$. So the total communication volume is $\frac{m_1 n}{m_2} + m_2 n$, which is much smaller than mn if $m_1 = m_2$. In term of communication volume, block-parallel scheme is a better choice. In this paper, block-parallel scheme is used.

D. Data Mapping Scheme

In the matrix-vector multiplication based implementation for DFT, there are two kinds of data resources: twiddle factors and input sequences. This section discusses storage mapping of twiddle factors and input sequences.

1) Twiddle Factors' Mapping Scheme. Considering a n point DFT, the twiddle factors can be represented as a matrix DFT_n in section II. And the matrix DFT_n is initialized and mapped to processing elements' local memory, when system starts up. According to the block-parallel scheme, elements of matrix DFT_n should be divided into m blocks if the system has m processing elements. The total memory size for twiddle factors is $32n^2$ bits. But as mentioned in section II, matrix DFT_n at most has n different elements. So a memory-efficient data mapping scheme for twiddle factors is proposed: 1) Each processing element needs $32n$ bits of memory to store the n different elements. For a system of m processing elements, the total memory size for twiddle factors is $32mn$ bits. The storage size of matrix DFT_n can be reduced from $32n^2$ to $32mn$ (m is usually a much smaller constant than n). 2) A bit of logic about address generator is added to each processing element. Algorithm 1 shows operation of the customized address generator.

Algorithm 1. Assuming that the system has m processing elements, it is divided into $m_1 \times m_2$ 2D-array where $m = m_1 \times m_2$. Different processing element is marked as $PE(I,J)$, $0 \leq I < m_1$, $0 \ll J < m_2$. Operation of the customized address generator of $PE(I,J)$ is listed below.

Parameter: m_1 , m_2 , I , J , DFT's transform size n

Variables: memory write address $addr_w$, memory read address $addr_r$, intermediate variable $symbol_addr$ and $interval$, column number col

```

1: {Initialization}
2:  $addr\_w = 0$ ;  $addr\_r = (I \times J) \% n$ ;

```

```

3: for  $i = 0$  to  $n-1$  do
4:    $addr\_w = addr\_w + 1$ ;
5: end for
6: { During each computation of DFT }
7:  $symbol\_addr = (I \times J) \% n$ ;  $col = J$ ;  $interval = (m_1 \times J) \% n$ ;
8: for  $i = 0$  to  $\left\lceil \frac{n}{m_2} \right\rceil - 1$  do
9:   for  $j = 0$  to  $\left\lceil \frac{n}{m_1} \right\rceil - 1$  do
10:     $addr\_r = symbol\_addr$  if  $j = 0$ ;
11:     $addr\_r = addr\_r + interval$  if  $(addr\_r + interval) < n$  and  $j > 0$ ;
12:     $addr\_r = 0$  if  $(addr\_r + interval) = n$  and  $j > 0$ ;
13:     $addr\_r = addr\_r + interval - n$  if  $(addr\_r + interval) > n$  and  $j > 0$ ;
14: end for
15:  $col = col + m_2$ ;
16:  $interval = 0$ ;
17: for  $j = 0$  to  $m_1 - 1$  do
18:    $interval = interval + col$  if  $(interval + col) < n$ ;
19:    $interval = 0$  if  $(interval + col) = n$ ;
20:    $interval = interval + col - n$  if  $(interval + col) > n$ ;
21: end for
22: for  $j = 0$  to  $m_2 - 1$  do
23:    $symbol\_addr = symbol\_addr + I$  if  $(symbol\_addr + I) < n$ ;
24:    $symbol\_addr = 0$  if  $(symbol\_addr + I) = n$ ;
25:    $symbol\_addr = symbol\_addr + I - n$  if  $(symbol\_addr + I) > n$ ;
26: end for
27: end for

```

2) Input Sequences' Mapping Scheme. Just like FFT, input sequence can be reordered firstly, then pushed into computation system. Or it goes to the computation system in sequence, then the output sequence should be reordered. The $m_1 \times m_2$ 2D-array computation system can be regarded as m_2 subsystems. Each subsystem processes related calculations about different columns of matrix DFT_n . It means that matrix DFT_n is divided into m_2 submatrix for columnwise decomposition scheme. Each subsystem has m_1 processing elements. Each of the m_1 processing elements processes related calculations about different rows of related submatrix. Elements of the input sequence can be mapped to different subsystems in turn. So owe to this parallel-processing architecture, there is no extra overhead for reorder of input sequence. And the output sequence is in order, using this input sequences' mapping scheme.

IV. EXPERIMENTAL RESULT

A. Experimental Setup

The parallel-processing architecture is implemented on Xilinx Virtex-6 FPGA(XC6VLX240T, speed grade -2) using Xilinx ISE 14.4. And the processing element is customized to be a parameterized IP core. We implement a design with 196 processing elements on the Virtex-6 FPGA board, processing at the frequency of 250 MHz. The 196-processing-element array is divided into 14 rows and 14 columns. The utilization of hardware resource is showed in table 1. Memories in the implementation are all used block rams on chip. In the experiment, the DFT transform size and twiddle factors are configured by X86 computer through gigabit ethernet.

Besides, a generator using Visual Studio 2012 is built to automatically generate Verilog based designs of the proposed architecture. We use it to generate designs with different numbers of processing elements in Verilog. Then we evaluate these designs on Xilinx Virtex-7(XC7VX980T, speed grade -2), which has much more configurable logic and DSPs, using Xilinx ISE 14.4.

TABLE 1 the Utilization of Hardware Resource

Slice Logic Utilization	Used	Available	Utilization
slices	32,360	37,680	86%
DSP48E1s	392	768	51%
BlockRam	800	832	96%

B. Performance Analysis

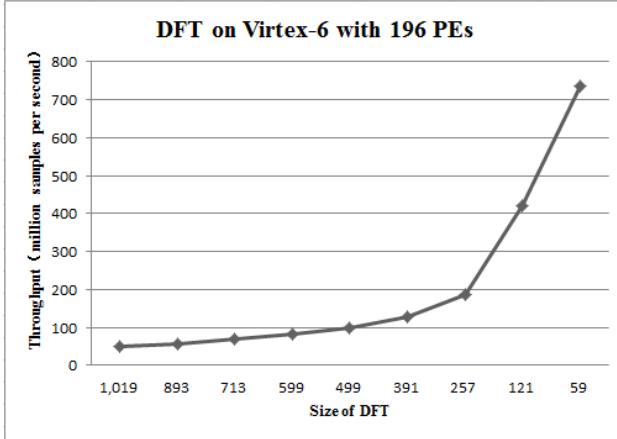


Fig. 3.Throughput for Different Sized DFT on Virtex-6 with 196 Processing Elements. The sizes are prime numbers.

As is showed in Fig.3, a customized design with 196 processing elements on FPGA is available for computing different sized DFTs. This implementation can compute any transform size from 14 to 1024. If off-chip memory is also used, the transform size will be much more flexible. In particular, the test size are prime numbers. As the transform size reduces, the throughput is increased. For a transform size of 59, the throughput can reach to 737.5 Msps. So a

customized design based on the proposed parallel architecture is compatible with different applications' requirement.

If higher throughput is required, increasing the numbers of processing elements is a good choice. Fig.4 shows that throughput of DFT is almost increased linearly with the increasing of numbers of processing elements. The implementation can be easily extended to different numbers of processing elements if the hardware resources are enough, using the code generator we built.

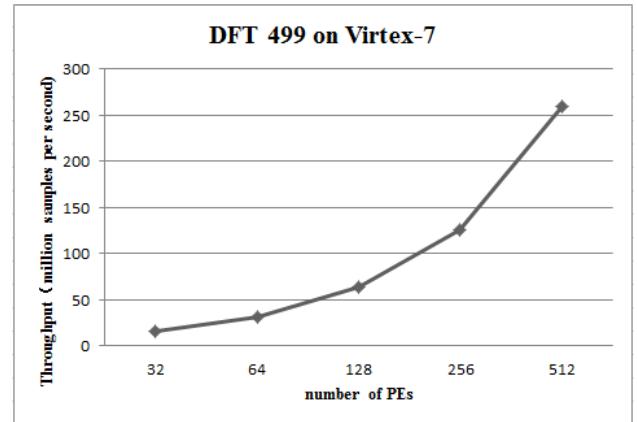


Fig. 4.Throughput for DFT 499 on Virtex-7 with different numbers of processing elements.

V. CONCLUSION

In this paper, we propose a parallel processing architecture on FPGA for arbitrary-sized DFT. In particular, it is attractive to use to compute DFT with arbitrary prime size. And a block-parallel computing scheme is used to reduce communication overhead. Besides, a memory efficient data mapping scheme for twiddle factors is proposed, which reduces the storage size of twiddle factors from n^2 to kn (k is a constant). Then, we implement a design with 196 processing elements, which is available for any transform size n from 14 to 1024. If off-chip memory is also used, the transform size will be much more flexible. For a transform size of 59, the throughput in this design can reach to 737.5 Msps. The performance of the design is mostly linear to the numbers of processing elements. The design can be easily extended to be one with more processing elements if the hardware has enough resources. And we have built a code generator to automatically generate designs with different numbers of processing elements.

ACKNOWLEDGEMENT

This work is supported by Chinese Academy of Sciences (NO. XDA06010402, NO. YCZB201302).

REFERENCES

- [1] J. Barhen, C. Kotas, T. Humble, P. Mitra, N. Imam, M. Buckner, *et al.*, "High performance FFT on multicore processors," in *Cognitive Radio Oriented Wireless Networks & Communications (CROWNCOM), 2010 Proceedings of the Fifth International Conference on*, 2010, pp. 1-6.
- [2] F. Franchetti, M. Püschel, Y. Voronenko, S. Chellappa, and J. M. Moura, "Discrete Fourier transform on multicore," *Signal Processing Magazine, IEEE*, vol. 26, pp. 90-102, 2009.

- [3] G. Inggs, D. Thomas, and S. Winberg, "Exploring the latency-resource trade-off for the Discrete Fourier Transform on the FPGA," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, 2012, pp. 695-698.
- [4] M. Püschel, J. M. Moura, J. R. Johnson, D. Padua, M. M. Veloso, B. W. Singer, *et al.*, "SPIRAL: Code generation for DSP transforms," *Proceedings of the IEEE*, vol. 93, pp. 232-275, 2005.
- [5] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing DSP transforms," in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 385-390.
- [6] K. Dong-Sun, L. Sang-Seol, S. Jae-Yeon, W. Kyu-Yeul, and C. Duck-Jin, "Design of a mixed prime factor FFT for portable digital radio mondiale receiver," *Consumer Electronics, IEEE Transactions on*, vol. 54, pp. 1590-1594, 2008.
- [7] M. D. Van de Burgwal, P. T. Wolkotte, and G. J. Smit, "Non-power-of-two FFTs: Exploring the flexibility of the Montium TP," *International Journal of Reconfigurable Computing*, vol. 2009, p. 4, 2009.
- [8] B. Duan, W. Wang, X. Li, C. Zhang, P. Zhang, and N. Sun, "Floating-point mixed-radix FFT core generation for FPGA and comparison with GPU and CPU," in *Field-Programmable Technology (FPT), 2011 International Conference on*, 2011, pp. 1-6.
- [9] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Puschel, "Hardware implementation of the discrete fourier transform with non-power-of-two problem size," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, 2010, pp. 1546-1549.
- [10] L. I. Bluestein, "A linear filtering approach to the computation of discrete Fourier transform," *Audio and Electroacoustics, IEEE Transactions on*, vol. 18, pp. 451-455, 1970.