An Improved FPGA Implementation of Sparse Fast Fourier Transform

Xiaohe Pei, Tao Shan, Shengheng Liu, Yuan Feng

School of Information and Electronics, Beijing Institute of Technology, Beijing, China
xiaohep12@gmail.com

Keywords: Digital signal processing, sparse fast Fourier transform (SFFT), hardware implementation architecture, field-programmable gate array (FPGA).

Abstract. The scale of the sequential data sets to be real-timely processed in most sectors of digital signal processing has substantially increased, making efficient numerical algorithms such as sparse fast Fourier transform (SFFT) more attractive than ever. This paper presents an improved FPGA-based SFFT implementation scheme, where a novel pipeline shift queue structure and a novel pipeline tracking filter structure are designed to significantly reduce the computational complexity. With the proposed scheme, millions of SFFT modules can be integrated on a single Virtex-6 FPGA chip, in addition to which, no assumptions or a priori knowledge on the spectrum distribution of the input signal is required.

1. Introduction

As one of the most fundamental numerical algorithms in digital signal processing, fast Fourier transform (FFT) plays a central role in a broad range of applications such as radar detection, medical imaging, radio astronomy, navigation, and etc. The time complexity of the FFT algorithm is proportional to the length of the signal, when analysing large data sets, the required processing capacity is too large, which makes it difficult to be real-timely implemented. However, generally most natural or desired signals can be sparsely represented. Such sparse signature of signals was exploited in many sub-linear algorithms for accelerating the computation of discrete Fourier transform, among which, SFFT [1,2] represents the most efficient alternative. The core idea of the SFFT algorithm is to first convert the FFT operation of a large number of points (N points) into that of much fewer points (B point) by partitioning the frequency coefficients into B buckets, then estimate K large coefficients of the original signal by following certain location-estimation-mapping rules. As a state-of-art efficient alternative of FFT, SFFT facilitates low-cost and low-consuming computing for portable and mobile embedded devices.

SFFT hardware implementations have been previously reported in [3,4], where 750,000 points and million points SFFT hardware realization is presented. On the basis of previous work, in this paper we propose a novel SFFT/sparse inverse fast Fourier transform (SIFFT) hardware implementation scheme, whose contribution is fourfold: (1) The design in [3] is suitable for the signals with a specific fixed sparsity degree, which is to say, the parameters cannot be changed once set. Whereas in our design, sparsity degree can be flexibly configured online; (2) In the maximum

selector, compared with the binary tree structure adopted in [4], a novel pipeline shift interchange queue structure is designed, which reduces intensive contrast and interchange operations. (3) In the candidate value voting module, this paper designs a new pipeline tracking filter structure, which saves massive storage resources and computing time compared with the "block memory + multiplier" structure in [4]. (4) In the cases where errors and omissions occur in the SFFT loops, [3] and [4] will inevitably output errors. In this paper, the fault tolerance threshold is added to the voting filter module, as such, the accuracy of SFFT operation is improved.

The rest of the paper is arranged as follows. In the second section, a brief overview of the SFFT algorithm principle and theoretical framework is given. In Section 3, the design architecture, working principle, interaction mode and hardware realization skill of various SFFT functional modules are described. The experiment results of the implementation and the performance comparisons among various implementation methods are provided in Section 4. The paper is concluded in Section 5.

2. Overview of SFFT algorithm

SFFT algorithm mainly consists of bucket mapping, sub-sampling FFT and reconstruction mapping. Bucket mapping resembles sorting N balls (all frequency coefficients) into the B bucket in sequence, where K balls are black (large frequency coefficients), and the remainders are white, as shown in Figure 1. If two or more black balls are divided into the same bucket, that is, if multiple large frequency points are mixed together, then the subsequent FFT/inverse FFT (IFFT) calculation results will inevitably lead to errors.

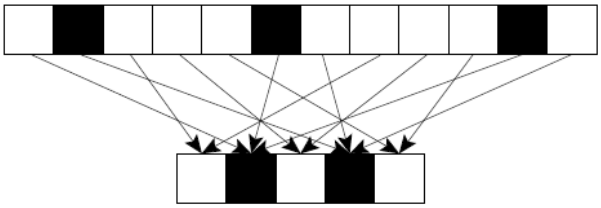

Figure 1 Bucket mapping.

This phenomenon is termed collision, in order to avoid which, one must ensure that the large frequency coefficients follow a uniform random distribution [5,6]. Nevertheless, the commonly encountered signals seldom satisfy this condition. Instead, the energy of these signals is usually very concentrated in a very narrow spectrum region. Hence, in order to avoid the collision problem, the first step of SFFT is to randomly permute the signal spectrum, which is achieved by permuting the time domain according to the scaling property of the Fourier transform [7,8].

After random permutation, the large frequency coefficients are separated from each other, followed by sub-sampling the frequency domain, N/B is the sampling interval. After sampling, the FFT of the B-point sequence is used to obtain the amplitude and position of the large frequency coefficients at the point B sequence. In this paper, the hash mapping reconstruction method is used to reconstruct the position of the large frequency coefficients on the N-point sequence. Hash mapping reconstruction method is characterized by simple structure, no iteration, but there are some requirements for the setting of parameters [1,2]. For example, it requires that the B be slightly larger than the square root of the product of N and K, and that N must be an integer power of two, thus zero-padding is needed in some cases.

3. Implementation scheme

3.1. Permutation and sub-FFT/IFFT

When the input data are tapped into the SFFT processing module, they need to be cached first. The addresses are then randomly permuted using a permutation factor. This process is conducted by modifying the time-domain signal as we do not have access to the input signal's Fourier spectrum. According to the scaling property of the Fourier transform, if the addresses, or the indexes of the time-domain signal are multiplied by a permutation factor, the indexes of the frequency-domain will be correspondingly multiplied by its inverse mod N. After permutation, the spectrum of the signal is spread over the entire frequency domain, which guarantees that all the large frequency coefficients can be sampled with a high probability during the sub-sampling process.

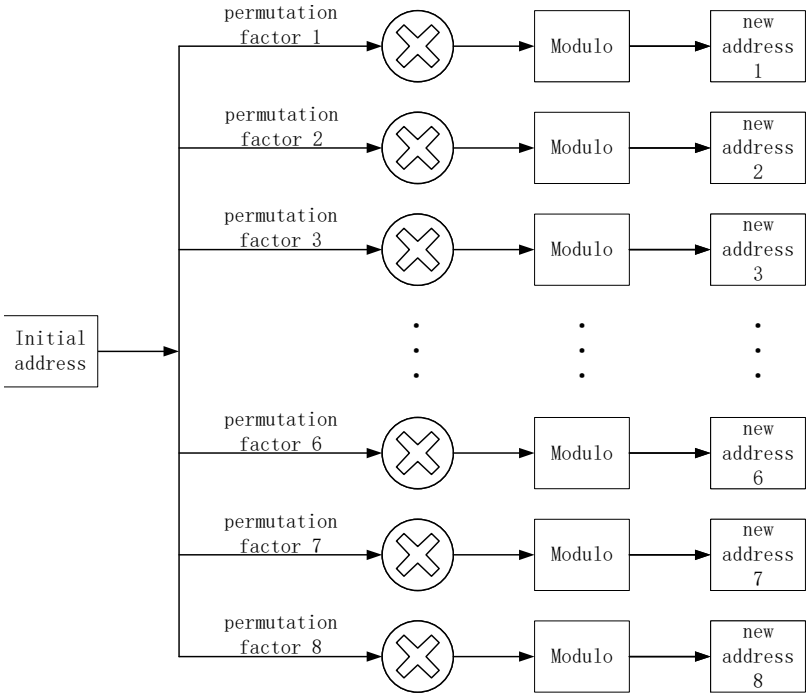

Figure 2 Permutation multiplier group.

In each cycle of the SFFT hardware implementation, a prime number less than N should be randomly generated as the permutation factor. Then the new addresses are obtained by executing the N modulus operation against the product of the original address and the permutation factor. In order to deal with the SFFT operation in real time, the total data length N is set to 1048576 points, the sub-sampling length B is set to 4096 points, and the SFFT cycle numbers are set to 8 times. In order to reduce the computing time, this paper will rearrange the multiplier group and reconstruction multiplier group from the traditional serial execution to parallel execution, as shown in Figure 2. Since N is set to an integer power of 2 and the addresses are both positive integers, modulo operations can be replaced by truncation operations.

Since the random permutation is actually just a rearrangement of the virtual address for data access, it has no impact on the input data cache, so this step can be completed in advance. Specifically, we write the cached data to the permuted random access memory (RAM) address. Next, the aliasing operation is executed. In this paper, an improved pipeline-type mixer is adopted, which is shown in Figure 3.

When the writing process of the permuted data ends, we constantly retrieve the data from the RAM, and tap them into buffer A. The buffer A transfers the data to the shift register A and the buffer B at the next clock cycle. After buffer B receives the data, the current data is superimposed on the far-right side of the shift register B at the next clock cycle. For each clock cycle, the data in shift registers A and B are shifted to the left. When all the data within a pulse repetition time (PRT) have entered, the aliasing operation is completed, and the result is a 4096-point short sequence that is aliased from along sequence of 1048576 points.

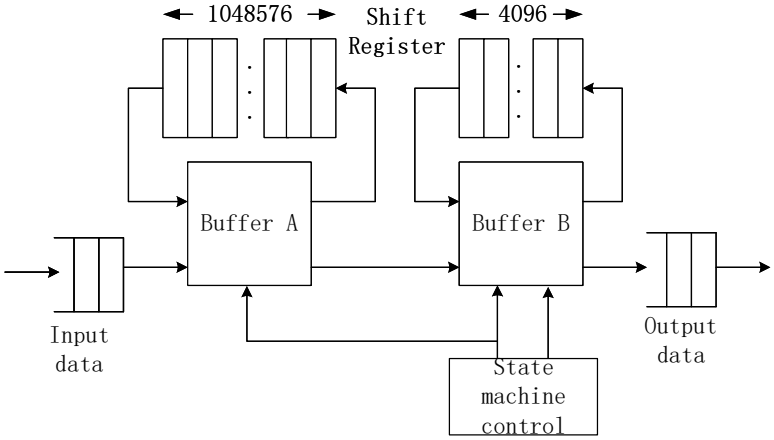

Figure 3 Pipeline mixers.

The algorithm flows of SFFT and SIFFT are very similar. The sparse transforms focus on the exploitation of the sparse nature of the signal in the frequency/time domain to reduce the computational complexity. As long as the result of SFFT/SIFFT is sparse, it is applicable. In the presented hardware implementation, the switch between SFFT and SIFFT is realized by setting a transform-type parameter. On the other hand, the sub FFT/IFFT is implemented by calling Xilinx intellectual property (IP) core, where data format is set as 25-bit fixed-point signed number that meets the practical accuracy requirement. Afterwards, the algorithm flow comes to the maximum selection and candidate vote modules, which are extremely important in the SFFT/SIFFT architecture, as shown in Figure 4.

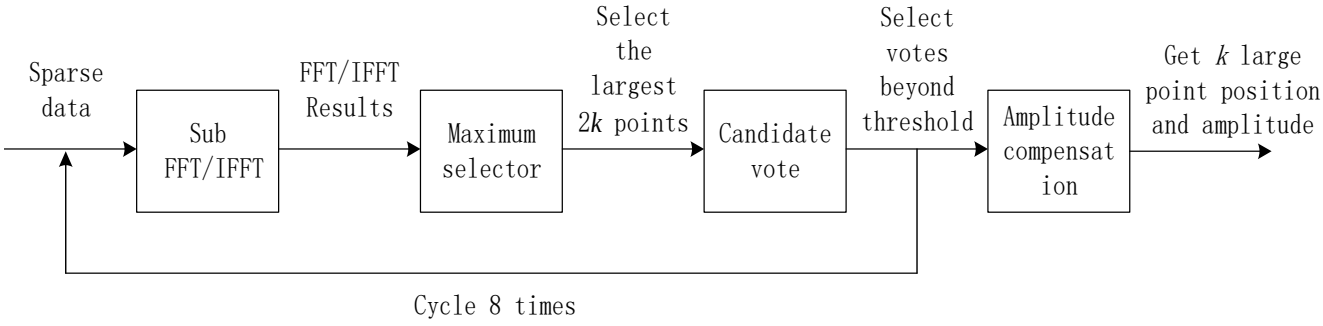

Figure 4 SFFT/SIFFT main function module.

3.2. Maximum selector

In our design, the sparsity degree is a key parameter, and the subsequent operations adjust to its variation. This value can be flexibly configured on the field-programmable gate array (FPGA) chip based on the actual measurements. If there is no manually interference, this parameter will be

automatically set to its default 16. In this case, to ensure the error-tolerant rate of the algorithm, the maximum selector will filter out the 32 largest amplitudes from the IP core output. In this paper, we design a novel pipeline shift queue architecture, which is able to achieve high-speed data filtering and sorting. The maximum selector is mainly composed of a maximum register and a shift register, they are in the form of a pipeline, both with the length of 32, the following describes its operating principle.

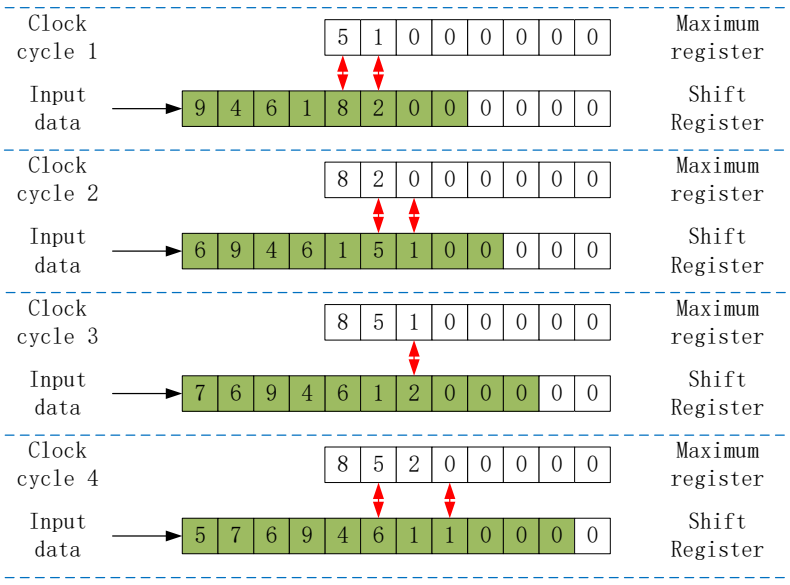

Figure 5 Pipeline shift interchange queue architecture.

As shown in Figure 5, in each clock cycle, the new input data is assigned to the leftmost point in the shift register, whilst the shift register is shifted by one point to the right, and the original rightmost point is discarded due to overflow. At the same instant, we compare each values of the same position in the maximum register and shift register, and swap them if the value in the shift register is greater. As such, the 32 largest amplitudes are kept in the maximum register after all the 4096-point output data have passed through the shift register. Compared to the conventional binary tree implementation structure, for one point at each instant, the proposed structure only takes one compare-and-swap operation to realize the selection and sorting of the K largest coefficients.

3.3. Candidate vote

After obtaining the amplitudes and positions of these large frequency coefficients, then the addresses of these points are used to perform the location operations, and the values are estimated with the amplitudes of these points. After obtaining the location information of the large frequency coefficients on the B-point sequence, the large value frequency points can be mapped to the N-point sequence by the hash function to determine their true location. In this paper, N is set as 1048576, B is 4096, which means that in this stage, each large frequency coefficients corresponding to 256 points in the N-point sequence.

As the location reconstruction of the large frequency coefficients is very similar to a voting process, this function is called the candidate voting module. In this paper, the location loop is performed eight times, i.e., eight rounds of voting. In each voting round, each large frequency coefficient cast 256 votes for the candidates selected from the 1048576-point long sequence, that is to say, 32 large frequency coefficients make a total of 8192 voting cast. The candidates whose votes

exceed the threshold (the threshold is set to 5 in this paper) are considered the true locations of the large frequency coefficients.

A novel pipeline tracking filter structure is developed in this paper. After obtaining the candidates' location in each voting round, this new structure does not count the number of votes using the traditional method, but tracks the candidate frequency. In the hardware implementation, the voting module is mainly composed of a ticket counter and a candidate value register, as shown in Figure 6. After the first round of the cycle, its 8192 candidate locations are registered in the candidate register. After the second round of the cycle, the candidate frequency positions of each new mapping are pipelined through the candidate register. If the corresponding value is the same, the number of votes in the position is increased by one, and the candidate's position is reserved. If they do not match, the original value in the candidate value register is replaced with the location of the candidate frequency calculated by the current iteration. In the subsequent rounds of voting, if a position already has two votes or more, even if they do not match, skip this round and resume to the next round. Otherwise, if the number of votes in a location is less than two, and the result does not match in this round, the candidate position will be cleared directly and will not be replaced.

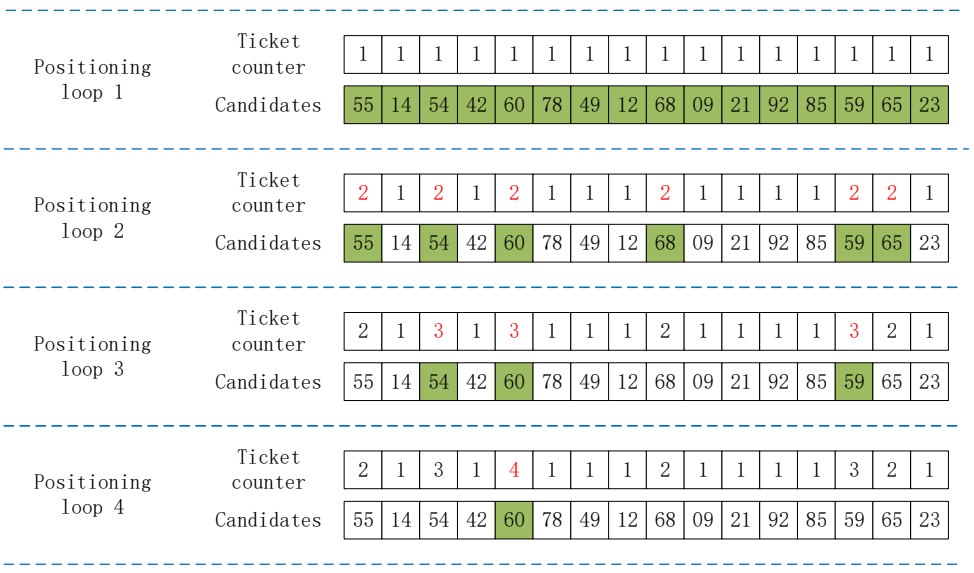

Figure 6 Pipeline tracking filter structure.

The advantage of this design are that the computational complexity can be significantly reduced, for the vast majority of false candidates will be removed in three cycles. In addition, in the cases where omission or error occurs accidentally, the proposed structure is able to still derive the correct results. After the candidate voting ends, the true number of large coefficients and their locations are obtained. Then the algorithm flow returns to the maximum selector, where the values of the large coefficients are obtained by computing then compensating the median of all the individual amplitudes from each estimation loop.

4. Implementation results

In this paper, we implement the synthesis, place and route of the SFFT/SIFFT and its peripheral processing modules on an FPGA model XC6VLX240T chip using Xilinx ISE 13.4 Design Suite. The entire design occupies a total of 42% of the slice register, 57% of the LUT, 16% of the BRAM and 11% of the DSP48E resources.

The resource consumption and runtime comparisons among different implementation approaches are illustrated in Figure 7 and 8. As can be seen from the figures, the advantage of the SFFT in computational complexity becomes more evident when the data length is increased to a moderate level, and compared with the recent SFFT implementation approaches in [4] and [9], the improved architecture still consumes less time and hardware resources.

In hardware implementation, the total length of the signal N and the length of the sub-FFT B, and the number of loops jointly determine the computation of the entire system. The number of large frequency coefficients that the SFFT/SIFFT module finally outputs is the number of points that have passed the threshold, which may be less or greater than the pre-configured K. In the experimental trials, the FPGA performs 100,000 times SFFT/SIFFT computing on different input signals with different sparsity degree. The statistical result is shown in Figure 9. It can be seen that the hardware modules in this paper is able to maintain high estimation accuracy for input signals with different sparsity.

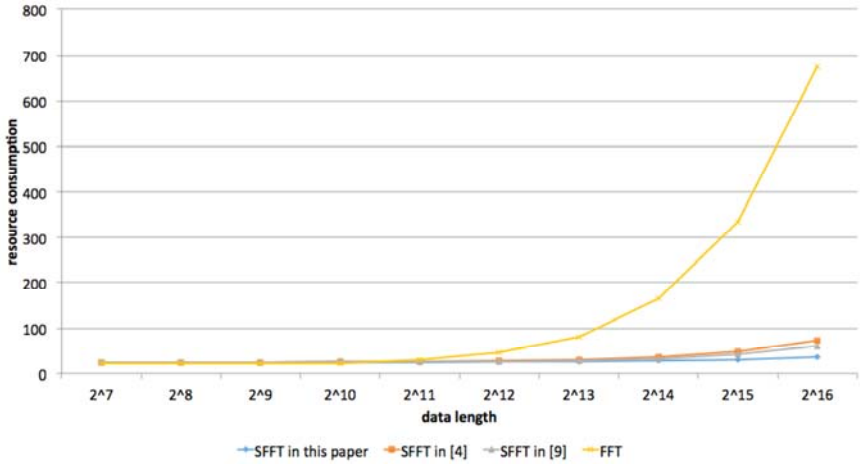

Figure 7 Comparison of resource consumption of different approaches.

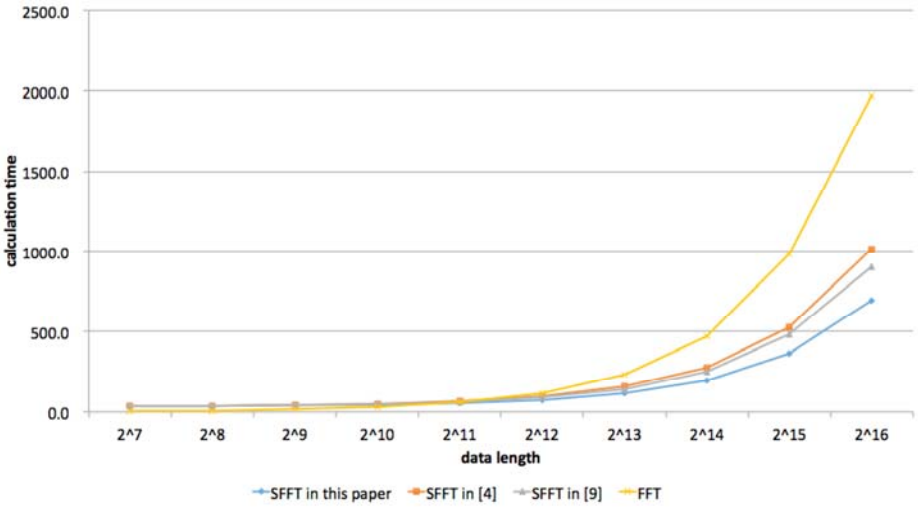

Figure 8 Runtime comparison of different approaches.

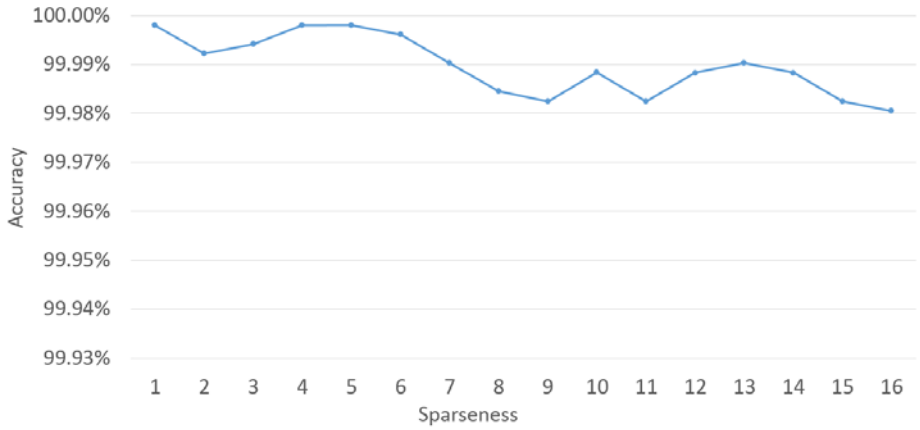

Figure 9 The relationship between computational accuracy and sparsity degree.

5. Conclusions

This paper presents an improved SFFT/SIFFT hardware architecture that implements millions of SFFT operations on a Virtex-6 FPGA chip, where a novel pipeline shift interchange queue structure and a novel pipeline tracking filter structure are designed. Experimental results demonstrates that the proposed architecture outperforms the existing SFFT schemes in terms of computational complexity and estimation accuracy.

Acknowledgements

This research was supported in part by the National Natural Science Foundation of China under Grant Nos. 61671060, 61331021 and Beijing Natural Science Foundation under Grant No. 4172052.

References

[1] Hassanieh, H., Indyk, P., Katabi, D. (2012) Nearly optimal sparse Fourier transform. Proceedings of the 44th Symposium on Theory of Computing Conference, STOC, pp. 563–578.

[2] Hassanieh, H., Indyk, P., Katabi, D. (2012) Simple and practical algorithm for sparse Fourier transform. Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1183-1194.

[3] Abari, O., Hamed, E., Hassanieh, H. (2014) A 0.75-million-point Fourier-transform chip for frequency-sparse signals. Solid-State Circuits Conference, 2014 IEEE International, pp. 458–459.

[4] Agarwal, A., Hassanieh, H., Abari, O. (2014) High-throughput implementation of a million-point sparse Fourier Transform. International Conference on Field Programmable Logic and Applications IEEE, pp. 1-6.

[5] Liu, S.H., Shan, T., Tao, R. (2014) Sparse discrete fractional Fourier transform and its applications. IEEE Transactions on Signal Processing, vol. 62, no. 24, pp. 6582-6595.

[6] Liu, S.H., Shan, T., Zhang, Y.D. (2015) A fast algorithm for multi-component LFM signal analysis exploiting segmented DPT and SDFrFT. IEEE International Radar Conference, pp. 1139-1143.

[7] Schumacher, J., Puschel, M. (2014) High-performance sparse fast Fourier transforms. Signal Processing Systems IEEE, Vol.20, pp.1-6.

[8] Wang, X. (2015) Fast Demodulation Algorithm of Underwater Acoustic Based on the Sparse Fourier Transform. Master thesis, Beijing Institute of Technology.

[9] Schumacher, J., Püschel, M. (2014) High-performance sparse fast Fourier transforms. Computer Journal, 20(1), 78-83.