Chinese Journal of Electronics
Vol.25, No.6, Nov. 2016

Configurable Floating-Point FFT Accelerator on FPGA Based Multiple-Rotation CORDIC*

CHEN Jiyang, LEI Yuanwu, PENG Yuanxi, HE Tingting and DENG Ziye

(*College of Computer, National University of Defense and Technology (NUDT), Changsha 410073, China*)

Abstract — Fast Fourier transform (FFT) accelerator and Coordinate rotation digital computer (CORDIC) algorithm play important roles in signal processing. We propose a configurable floating-point FFT accelerator based on CORDIC rotation, in which twiddle direction prediction is presented to reduce hardware cost and twiddle angles are generated in real time to save memory. To finish CORDIC rotation efficiently, a novel approach in which segmented-parallel iteration and compress iteration based on CSA are presented and redundant CORDIC is used to reduce the latency of each iteration. To prove the efficiency of our FFT accelerator, four FFT accelerators are prototyped into a FPGA chip to perform a batch-FFT. Experimental results show that our structure, which is composed of four butterfly units and finishes FFT with the size ranging from 64 to 8192 points, occupies 33230(3%) REGs and 143006(30%) LUTs. The clock frequency can reach 122MHz. The resources of double-precision FFT is only about 2.5 times of single-precision while the theoretical value is 4. What's more, only 13331 cycles are required to implement 8192-points double-precision FFT with four butterfly units in parallel.

Key words — Fast Fourier transform (FFT), Coordinate rotation digital computer (CORDIC), FPGA, Floating-point.

I. Introduction

Fast Fourier transform (FFT) is a basic algorithm in digital communication, sensor signal processing, and Synthetic aperture radar (SAR). It is always the most time consumption part in real-time communication system. Higher requirements in hardware efficiency, size scalable, throughput and flexibility are put forward for FFT accelerator.

The FFT size is varied for different communication standards. For example, in IEEE 802.11A/G, HIPERLAN2 WLAN, 802.15.3 MB-UWB, IEEE 802.16 WiMAX, DVB-T standards, the FFT size is range from 64 to 8192. Therefore, it is basic requirement for FFT accelerator to provide high performance and efficiency for variable-size FFT.

With the increase of FFT size, the performance of FPGA will be reduced because of the storage consumption and resource utilization. Butterfly unit is the basic module in FFT structure. It's important to reduce its area, as well as to improve throughput. CORDIC has been shown to be an efficient way to implement butterfly operation because it can finish multiplication by adders and shifters instead of multipliers and reduce the storage consumption, however, large amount of iterations lead to great hardware cost. With the widely use of FFT, configurable FFT accelerator is needed to meet different systems.

In this paper, a configurable floating-point FFT accelerator is implemented to improve its capacity on FPGA and satisfy the demands of different applications. Based on CORDIC algorithm, segmented-parallel iteration and compress iteration are presented to reduce iterations. Redundant CORDIC is used to reduce the latency of each iteration, together with twiddle direction prediction to save hardware cost. Besides, data throughput is improved with built-in RAMs. What's more, twiddle angles are generated in real time so that the storage consumption can be greatly deduced.

The advantages of our design are as following:

1) FFT accelerator is flexible because the number of butterfly units is configurable, as well as the FFT size ranging from 64 to 8192 points.

2) FFT accelerator executes some butterfly units in parallel to reduce computing cycles and uses some schemes such as segmented-parallel iteration and compress iteration in CORDIC to reduce hardware cost and latency.

3) Overhead of double-precision is only about 2.5 times of single-precision while the theoretical value is 4.

*Manuscript Received Sept. 24, 2014; Accepted Mar. 17, 2015. This work is supported by Aerospace Science Fund of China (No.2013ZC88003), and the National Natural Science Foundation of China (No.61402499).

4) For the use of built-in RAMs of FPGA and real-time generation of twiddle angles, storing resources is small and advantage of our design is more prominent in large-scale FFT.

5) FFT accelerators can finish calculation of batch FFT.

II. Background and Related Work

1. Background

FFT transforms a discrete time-domain signal into a frequency-domain signal and the conversion formula is:

$$\begin{cases} X(K) = X_1(K) + W_N^k X_2(K) \\ X(K+N/2) = X_1(K) - W_N^k X_2(K) \end{cases} \tag{1}$$

As Eq.(1) shows, there is addition and multiplication in FFT calculation and multipliers usually increase complexity of the pipeline. To simplify the calculation, CORDIC algorithm is an alternative method. Rotation factor W_N^k can be decomposed according to Euler's formula:

$$W_N^k = \cos(-2\pi nk/N) + \mathrm{j}\sin(-2\pi nk/N) \tag{2}$$

so that the multiplication term $W_N^k X_2(k)$ can be expressed in plural form:

$$W_N^k X_2(k) = (x_2(k) + \mathrm{j}y_2(k))(\cos(-2\pi nk/N) + \mathrm{j}\sin(-2\pi nk/N)) \tag{3}$$

in which real and imaginary parts are:

$$\begin{cases} x(k) = x_2(k)\cos\theta - y_2(k)\sin\theta \\ y(k) = y_2(k)\cos\theta + x_2(k)\sin\theta \end{cases} \tag{4}$$

where $\theta = -2\pi nk/N$. The expression is the same with general rotation transform in CORDIC algorithm which is:

$$\begin{cases} X' = K(X\cos Z - Y\sin Z) \\ Y' = K(Y\cos Z + X\sin Z) \end{cases} \tag{5}$$

so that the complex multiplication in butterfly unit can be implemented with CORDIC. Here (X', Y') is the resulting coordinate after rotating though an angle of Z and K is a gain factor which tends to 0.60725 with the increase of iteration. X', Y', X, Y and Z are corresponding to $x(k)$, $y(k)$, $x_2(k)$, $y_2(k)$ and θ of the real and imaginary part, respectively.

2. Related work

With development of large-scale calculation, the performance of FFT on FPGA is reduced because of the storage consumption and latency of complex multiplication. To reduce hardware cost and ensure high performance, many studies have been done.

Some studies optimized FFT unit to improve efficiency. B.M. Bass[1] proposed cached FFT algorithm which increases performance, while it needs more caches and adds complexity; Z. Wang[2] simplifies multiplication by decreasing one real multiplication but increasing three real additions, there are still many multipliers and adders.

Different-radix FFT and configurable FFT are presented to meet different applications. C.M. Chen[3] proposed a 128-1024 point partial pipellined/cached FFT processor which results in energy efficiency but it's a bit complex; M. Hasan[4] implemented 64-point FFT based on low power pipelined radix-4 architecture, while it uses many complex multipliers; Q.H. Zhang[5] presented pipelined radix-2 FFT processor but it's based on memory.

Some works are presented to realize FFT based on CORDIC algorithm[6−8]. Optimization of CORDIC algorithm is mainly concentrated in three areas: computing time, hardware cost and convergence domain. Computing time is decided by iterations and each iteration time. To reduce the number of iterations, unidirectional CORDIC[9], scaling-free CORDIC[10] and radix-4 CORDIC[11] are presented, while the hardware cost of these methods are large. Redundant CORDIC[12,13] is the main way to reduce each iteration time, however, problems of sign detection of residual angle and change of scale factor are introduced. To reduce hardware cost of CORDIC, sign-prediction[12−14] in rotation is proposed, while this method is in serial and the latency is large. To calculate the elementary functions correctly, convergence domain should be met. Repetition of certain iteration steps[10,15] is an effective way to expand the range of convergence while it needs more iterations.

III. FFT Accelerator Structure

As shown in Fig.1, FFT accelerator is composed of four parts: address-generating unit, FFT controller, data-storing SRAM and butterfly unit. To improve the parallelism, SRAM and butterfly unit both include several independent sub-units.

Function of each module is as following:

1) Address-generating unit: generate extract-address and calculate twiddle angles in real time for CORDIC rotation.

2) FFT controller: harmonize run order of accelerator.

3) Data-storing SRAM: store initial data, update intermediate data and save final result.

4) Butterfly unit: predict twiddle direction, segmented-parallel iteration and compress iteration.

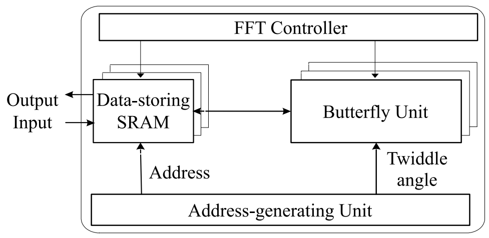

Fig. 1. Structure of floating-point FFT accelerator

1. Address-generating unit

Because of different extract-interval in each stage of FFT[16], address-generating unit is designed to generate the extract-address of data. As shown in Fig.2, the extraction rule of radix-2 FFT can be expressed as:

$$address = \{\mathrm{addr}[\log_2 N : n], \mathrm{addr}[0], \mathrm{addr}[n-1:1]\} \quad (6)$$

in which n is the stage number and N is the point of FFT.

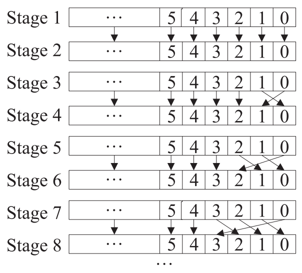

Fig. 2. Extraction Rule of Radix-2 FFT

Table 1. Correspondence of n, k and address (i=3,...,9)

Stage	n	k
1	0	0
2	Addr[8]	8'b0,Addr[9]
i	Addr[10-i]	(10-i)'b0,Addr[(11-i):9]
10	Addr[0]	Addr[1:9]

In conventional FFT accelerator, twiddle angles are stored in ROM in advance and the hardware cost increases rapidly. To save recourses, we propose to generate twiddle angles in real time based on extract-address. Twiddle factors in the same stage are fixed, that is, in twiddle angle $\frac{nk}{N} \cdot 2\pi$, n and k are two decisive factors. The correspondence of n, k and address of data (here Addr is the reverse order of extract-address) is shown in Table 1.

2. Data-storing SRAM

All the inputs and outputs of butterfly unit should go through the SRAM and the read-write speed of SRAM has a great impact on the performance of FFT accelerator. Built-in dual-port RAM of FPGA are used to improve the overall speed, so read and write operations can be done in the same RAM and data throughput be greatly improved.

Two complex data must be read from the RAM simultaneously and loaded to one butterfly unit. Meanwhile, two outputs of butterfly unit should be written into RAM at the same time, that is to say, there are total four ports (two for read and two for write) that one butterfly unit requires. So data-storing RAM should be divided into several sub-RAMs to meet the requirements of butterfly unit because built-in RAMs are dual port.

In case of radix-2 FFT which include one butterfly unit, as shown in Fig.3, to avoid address conflicts, two sub-RAMs are needed (port0 and port1 in RAM1, port2 and port3 in RAM2) and even-address data are stored in RAM1, as well as odd-address data stored in RAM2. As shown in Fig.4, in stage 1, data are read from RAM1 and RAM2 every cycle through port0 and port2 respectively because the extract interval is 1 and after butterfly operation, the results are written back to RAM1 and RAM2 through port1 and port3 respectively; in other stages, data are read from RAM1 in odd-cycle through port0 and port1 (written to RAM1 in even-cycle through port0 and port1) and read from RAM2 in even-cycle through port2 and port3 (written to RAM2 in odd-cycle through port2 and port3) because the extract interval is 2^{n-1} (n is the stage number). This dividing method ensures continuity of pipeline and makes full use of the built-in dual-port RAMs.

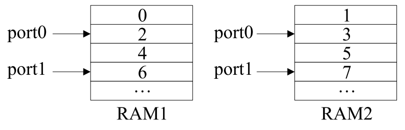

Fig. 3. Division of sub-RAMs

3. Butterfly unit

Butterfly unit, as the key part of FFT accelerator, is implemented by CORDIC rotation[17]. In conventional butterfly unit shown in Fig.5(b), there are four multipliers and six adders in one butterfly unit, while in our structure shown in Fig.5(a), it only costs two CORDIC units and four adders. So the butterfly unit based on CORDIC rotation will save a lot of resource because the hardware complexity of one CORDIC unit is the same with one multiplier[18] in the case of same data width. Our CORDIC structure will be introduced in detail in Section IV.

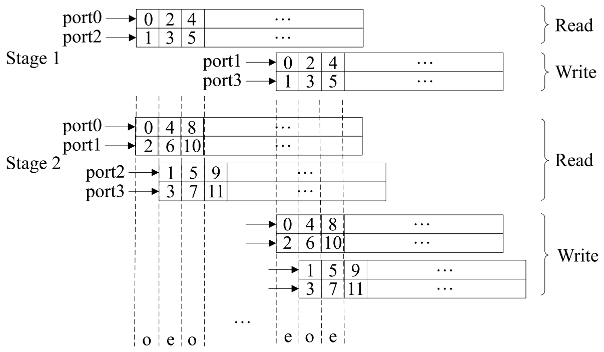

Fig. 4. Read and write process

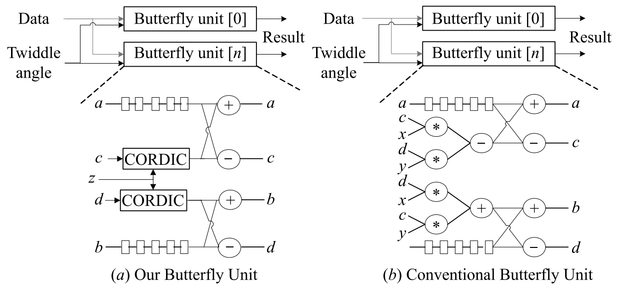

Fig. 5. Comparison of two butterfly units

IV. Multiple CORDIC Rotation for Butterfly Unit

Due to the large number of iterations, CORDIC algorithm usually yields slow and area demanding circuits. In order to overcome these drawbacks, twiddle direction prediction, segmented-parallel iteration and compress iteration are presented for CORDIC algorithm.

1. Twiddle direction prediction

Twiddle angle Z has two expressions as presented in Eq.(7), the first one is in binary and the second one is in trigonometric function[19].

$$\begin{cases} Z_0 = \sum_{i=0}^{N} c_i 2^{-i}, c_i \in \{0,1\} \\ Z_0 = \sum_{i=0}^{N} s_i \alpha^{-i}, \alpha_i = \arctan 2^{-i} \end{cases} \tag{7}$$

The gap between α_i and 2^{-i} will decrease with the increase of iteration i so that we can predict the direction of rotation according to the high bits of the binary expression. In the i^{th} iteration, suppose $Z_i = c_0.c_1 c_{i-1} c_i c_k$ $(c_0 = c_1 = ... = c_{i-1})$, we can predict the direction of rotation i to rotation k through c_{i-1}. The transformational rule for s_i to s_k is as following: if Z_i is positive, that is $c_{i-1} = 0$, then s_i=1, otherwise s_i=–1; if $j > i-1$ and c_j=0, then s_{j+1}=–1, otherwise s_{j+1}=1. Fig.6 gives the transformational rule in the i^{th} iteration.

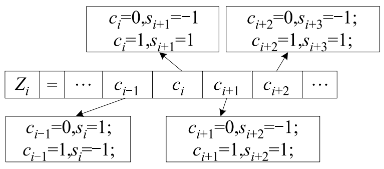

Fig. 6. Transformational rule for direction of rotation

Fig.7 gives the implementation structure of prediction module which composed of one twiddle-angle look-up-table (LUT), one adder and some selectors. In the i^{th} iteration (each iteration including four rotations), the four directions of CORDIC rotation $s_{4i+1} - s_{4i+3}$ can be generated by the rest angle Z_{4i} in the $i-1^{\text{th}}$ iteration and the angle value rotated in the i^{th} iteration can be searched from the look-up-table according to the high-4-bits in Z_{4i} so that the rest angle for the $i+1^{\text{th}}$ iteration can be calculated through adder. All directions after the 20th rotation can be generated by Z_{16} because the rest angle Z_{16} is small enough and the LUT and adder can be leaved out after the 20th rotation.

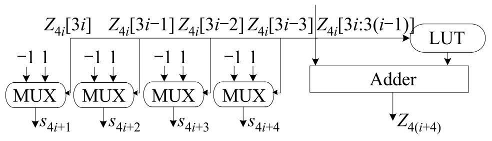

Fig. 7. Structure of prediction module

2. Segmented-parallel iteration based on CSA

In the first 32 rotations, compress rotation based on carry-save-adder(CSA4-2) is been used to reduce the delay. There are total 8 iterations for compress rotation and each iteration includes four rotations. The process of one compress iteration is given in Fig.8.

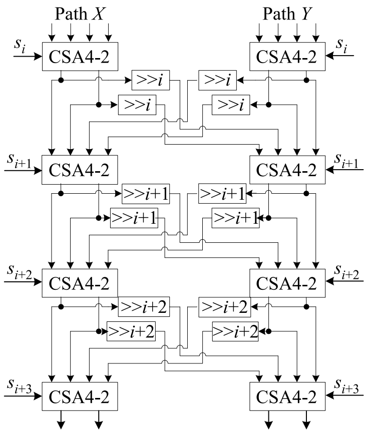

Fig. 8. Process of compress iteration including four rotations

3. Compress rotation based on CSA

Based on the formula of the i^{th} rotation, we can get the formula of the $i+1^{\text{th}}$ rotation, that is:

$$\begin{cases} X_{i+1} = X_i - s_i Y_i 2^{-i} \\ Y_{i+1} = Y_i + s_i X_i 2^{-i} \end{cases} \Rightarrow$$

$$\begin{cases} X_{i+2} = X_i(1 - s_i s_{i+1} 2^{-2i-1}) - Y_i(s_i 2^{-i} + s_{i+1} 2^{-i-1}) \\ Y_{i+2} = Y_i(1 - s_i s_{i+1} 2^{-2i-1}) + X_i(s_i 2^{-i} + s_{i+1} 2^{-i-1}) \end{cases}$$

So the formula from the m^{th} rotation to the $n-1^{\text{th}}$ rotation can be expressed as Eq.(8).

$$\begin{cases} X_n = X_m A_{m,n} - Y_m B_{m,n} \\ Y_n = Y_m A_{m,n} + X_m B_{m,n} \end{cases} \tag{8}$$

in which

$$A_{m,n} = 1 - \sum_i \sum_j s_i s_j 2^{-(i+j)} + ... \\ +(-1)^t \sum_{i(1)} ... \sum_{i(2t)} s_{i(1)} ... s_{i(2t)} 2^{-i(1)-...-i(2t)}$$

$$B_{m,n} = \sum_i s_i 2^{-i} - \sum_i \sum_j \sum_k s_i s_j s_k 2^{-(i+j+k)} + ... \\ +(-1)^t \sum_{i(1)} ... \sum_{i(2t+1)} s_{i(1)} ... s_{i(2t+1)} 2^{-i(1)-...-i(2t+1)}$$

When $m > N/2 (N = 64)$, addition items in $A_{m,n}$ are equal to 0 except the first item 1 and items in $B_{m,n}$ are also shifted to 0 except the first item $\sum_i s_i 2^{-i}$. Without impairing the required precision, the rotation formula from the 33th rotation to the 64th rotation can be simplified as following:

$$\begin{cases} X_{64} = (X_{C32} + X_{S32}) - (Y_{C32} + Y_{S32}) \sum_{i=33}^{N} s_i 2^{-i} \\ Y_{64} = (Y_{C32} + Y_{S32}) + (X_{C32} + X_{S32}) \sum_{i=33}^{N} s_i 2^{-i} \end{cases} \tag{9}$$

Here we total need four adders and two $(N/2) \cdot (N/2)$ multipliers to finish the last 32 times rotations. As the structure shown in Fig.9, the fixed-point multiplier is implemented based on CSA4-2. There are total 4 stages in the CSA-tree: Fig.9(a) gives the process of the first stage in which the 64 inputs (32 in path X and 32 in path Y) are compressed into 16 (8 in path X and 8 in path Y); Fig.9(b) gives the other 3 stages of the multiplier.

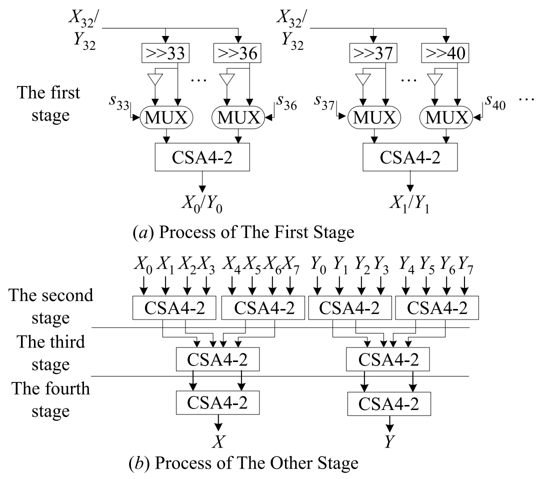

Fig. 9. Fixed-point multiplier

4. Structure of multiple CORDIC rotation

The structure of multiple CORDIC rotation unit based on CSA, as shown in Fig.10, is aimed at reducing the delay and implementing multiplication of twiddle factors. There are three main parts in the structure: pre-processor, CORDIC processor and post-processor.

Pre-processor transforms the inputs of floating-point format into fixed-point and does pretreatment with the twiddle angle. As shown in Fig.11, after the shift operation, fixed-point M should be multiplied by gain $K = 0.607253$. Meanwhile, to ensure the twiddle angle Z in the range of CORDIC rotation, which is $[-99.911\,°, 99.911\,°]$, pretreatment should be done and the output in the post-processor also should follow the rule shown in Table 2.

Table 2. Rule of pretreatment

Twiddle angle	Pre-twiddle angle	X_{out}	Y_{out}
[0,π/2]	0	X	Y
[π/2,π]	π/2	Y	-X
[π,3π/2]	π	-X	-Y
[3π/2,2π]	3π/2	-Y	X

CORDIC processor finishes the multiplication of twiddle angle. In double-precision CORDIC rotation, 64 rotations are divided into two parts: the first part includes 8 iterations (Iteration 1-Iteration 8) to finish the first 32 rotations and the second part implements the other rotations using fixed-point multiplier. In the first part, the first 5 iterations have three data paths: path X, path Y and path Z. In path Z, direction prediction unit predicts the twiddle directions based on the rest angle of last iteration and generate the rest angle of current iteration. In the 5$^{\text{th}}$ iteration, the other twiddle directions can be generated according to Z_{16}, so the last 3 iterations only have two data paths. In the second part, we just need four adders and two fixed-point multipliers to finish the rotations.

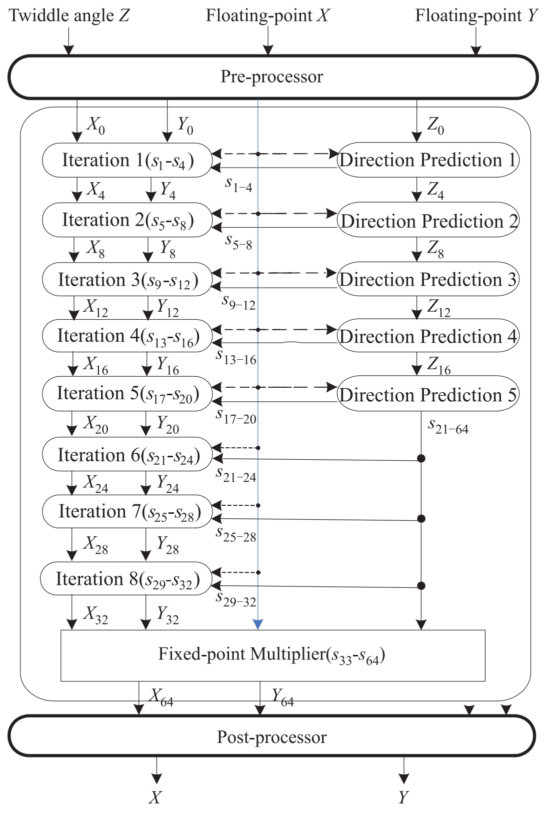

Fig. 10. Structure of multiple CORDIC rotation

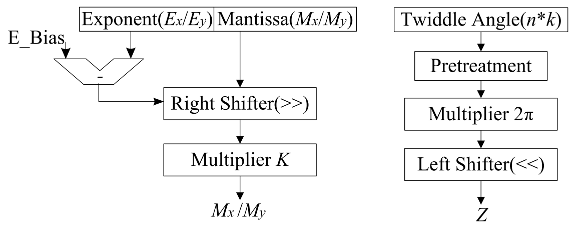

Fig. 11. Process of pre-processor

As shown in Fig.12, post-processor transforms the output from fixed-point format into IEEE-754 floating-point and the outputs should be changed depending on Table 2.

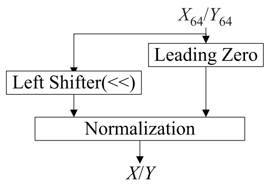

Fig. 12. Process of psost-processor

V. Experiments

1. Synthesize result

We have implemented our FFT accelerator on a Xilinx Virtex-5 XC6VLX760 FPGA chip.

Experiment 1 presents the resource usage and clock frequency of double-precision floating-point FFT accelerator with different numbers of butterfly unit. As shown in Table 3, FFT accelerator with four radix-2 butterfly units ("R2B4" means four butterfly units in parallel based on radix-2) occupies 33230(3%) REGs and 143006(30%) LUTs and reaches the clock frequency of 122MHz. Comparing the three rows, with more butterfly units running in parallel, the more resources accelerator needs because the read-write control of RAM is more complex and the address-generating unit should generate more extract-address, as well as rotation angles every cycle. To better apply to different applications, the tradeoff between speed and resources should be sought.

Table 3. Double-precision performance with XC6VLX760

Double-Precision	REG	LUT	Frequency(MHz)
R2B1	8211	35547	127
R2B2	11288	42828	126
R2B4	33230	143006	122

Experiment 2 presents the synthesis results of single-precision. From Tables 3 and 4, the overhead of double-precision floating-point is just about 2.5 times of single-precision floating-point while the theoretical value is 4 times. Our improvement of CORDIC rotation in the last 32 rotations with fixed multiplier saves a lot of resources.

Table 4. Single-precision performance with XC6VLX760

Single-Precision	REG	LUT	Frequency(MHz)
R2B1	3476	13176	131
R2B2	4856	16384	129
R2B4	13889	53558	125

In Experiment 3, performance of our design is compared with other structures. As shown in Table 5, K.Kalyani[20] and R.Bhakt[21] does not use built-in RAMs and uses conventional CORDIC algorithm which should store twiddle factors, so the cost is very large. G.Zhang[22] implements the 8-bits 128-points FFT calculation with four butterfly units based on conventional CORDIC algorithm. Though the structure can achieve a high frequency, it consumes a lot of resources. Our FFT accelerator is flexible because the computing scale and the number of butterfly units are both reconfigurable. At the same time, through the improved CORDIC algorithm, a high calculation speed can be achieved. Besides, by using built-in RAMs and real-time generation of twiddles, hardware cost can be saved a lot and in particular, it's suitable for large-scale computing.

Table 5. Performance of different structures

Method	Bits	Points	REG	LUT	Freq(MHz)
Our R1B1	64	8192	8211	35547	127
Our R1B1	32	8192	3476	13176	131
K.Kalyani[20]	64	64	50722	62955	
R.Bhakt[21]		8	3003	5983	
R.Bhakt[21]		16	13934	27742	
G.Zhang[22]	8	128	5115	5740	219

In Experiment 4, in order to evaluate the performance of the proposed FFT module, a batch-FFT prototype is built on our self-designed development board containing one Xilinx Virtex-5 XC5VLX330 FPGA chip and two 2GB Double Data Rate(DDR) DRAM modules, as illustrated in Ref.[23].

The batch-FFT prototype is composed of four FFT accelerator modules, two DDR controllers running at 200MHz on 128-bit data width, as shown in Fig.13. This prototype can calculate a batch of FFT with the same size, which is the kernel part in synthetic aperture radar and two-dimension FFT. The read controller reads the initial data from DDR 0 and sends the data into FFT modules with cyclic form. After each FFT module finishes the calculation, the write controller writes the final result into DDR 1. For the batch of FFT, the overhead of read initial data and write result is overlapped with FFT calculation. As shown in Table 6, the LUT is the most constrained resource and the batch-FFT prototype with four FFT modules consumes 153106 LUTs available. The achievable maximum frequency of batch-FFT prototype is 109.4MHz, so we can run at 100MHz in our prototyped.

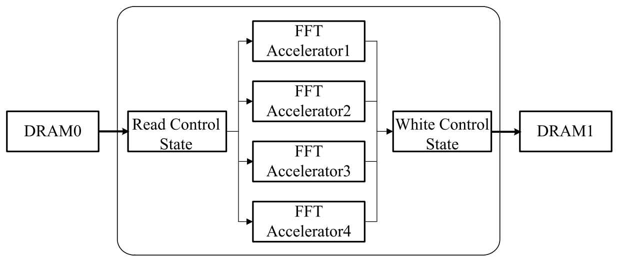

Fig. 13. Batch FFT with four FFT accelerators

Table 6. Hardware consume of batch-FFT prototype

	REG	LUT	Frequency(MHz)
FFT Module	10357	37562	120.1
DDR Controller	3841	3336	257.7
Batch-FFT Prototype	49305	153106	109.4

The total run time of our implementation includes computation time and the time for sending the initial data to FPGA as well as receiving the results back to the host PC. For $1K \times 1K$ points batch-FFT, the total run time is 80.4ms and the time for FFT calculation is 25.1%.

2. Performance evaluation

To improve speed calculations in the butterfly unit, the pipeline registers are located after each addition and subtraction. Hence, the pipeline butterfly unit keeps the final result in the register to transfer it to the RAM by the next clock. The total cycles of N-point FFT can be expressed as:

$$Totalcycles = \frac{N \cdot \log_2 N}{2B} + L \tag{10}$$

in which N is the points of FFT, B is the number of butterfly units and L is created by pipeline registers in a serial butterfly unit to emptying the pipeline.

The actual calculation cycles of our FFT structures are shown in Table 7 and they are proved the complex calculation of $\frac{N \cdot \log_2 N}{2B} + L$. The L is 19 in double-precision and 17 in single-precision because the fixed multiplication for the last 32 bits costs 2 cycles. With butterfly units increasing, the calculation cycles can be reduced by half. Thus, parallel design of butterfly units decreases the calculation time significantly.

Table 7. Actual cycles for different parallel structures

		64	128	1024	4096	8192
Double Precision	R2B1	179	467	5139	24595	53267
	R2B2	99	243	2579	12307	26643
	R2B4	59	131	1299	6144	13331
Single Precision	R2B1	177	465	5137	24593	53265
	R2B2	97	241	2577	12305	26641
	R2B4	57	129	1297	6142	13329

To compare with other studies, Ren-Xi Gong[24] and N. Mahdavi[25] both cost more than 5000 cycles to finish 1024-points FFT calculations because butterfly units in their structures are in serial, while with our "R2B4" structure, only about 1000 cycles are required. A.Banerjee[26] costs 10690 cycles to finish 128-points FFT for the process is totally in serial but not pipeline and the maximum frequency is also very low which is only 10MHz. G.Zhang[22] finish 8-bits 128-points FFT with four butterfly units in parallel based on radix-4, however, 99 cycles are required to finish the calculation because conventional CORDIC rotation is used in the structure so the latency is very long and the pipeline emptying cycle L is very large.

VI. Conclusions

This paper presents the design and implementation of floating-point FFT accelerator based on multiple-rotation CORDIC. With several improved schemes such as built-in RAMs, real-time generation of twiddle angles, twiddle direction prediction, segmented-parallel iteration and compress iteration, our FFT accelerator achieves high frequency and small cost. Experiments show that more butterfly units in parallel can speed up the calculation with reasonable increase of resources. Besides, our FFT structure is suitable for both single-precision and double-precision floating-point and the cost of double-precision is only 2.5 times of single-precision which is 4 times in theory. What's more, our FFT accelerator can be used in batch-FFT calculations.

References

[1] B.M. Bass, "A generalized cached-FFT algorithm", *Proc. of Acoustics, Speech and Signal Processing*, California, USA, pp.89–92, 2005.

[2] Z. Wang, M. Dong and Y. Zhao, "Design and implementation of efficient FFT processor for multicarrier system", *Proc. of Electrical and Computer Engineering*, Saskatoon, Canada, pp.1384–1387, 2005.

[3] C. Chen and Y. Huang, "Partial cached-FFT algorithm for OFDMA communications", *IEEE Region 10 Conference*, Taipei, China, pp.1–4, 2007.

[4] M. Hasan, "A novel low power pipelined architecture for a MC-CDMA receiver", *Proc. of Image and Signal Processing and Analysis*, Edinburgh, UK, pp.1048–1053, 2003.

[5] Q. Zhang and N. Meng, "A Low Area Pipelined FFT Processor for OFDM-Based Systems", *Proceeding of Wireless Communications, Networking and Mobile Computing*, Beijing, China, pp.1–4, 2009.

[6] C.S. Wu and A.Y. Wu, "Modified vector rotational CORDIC (MVR CORDIC) algorithm and architecture", *IEEE Transaction on Circuits and Systems-II (TCAS-II): Analog and Digital Signal Processing*, Vol.48, No.6, pp.548–561, 2001.

[7] C.H. Lin and A.Y. Wu, "Mixed-scaling-rotation CORDIC (MSRCORDIC) algorithm and architecture for high-performance vector rotational DSP applications", *IEEE Transaction on Circuits and SystemsI (TCAS-I)*, Vol.52, No.11, pp.2385–2396, 2005.

[8] P.K. Meher, J. Valls, T. Juang, *et al.*, "50 years of CORDIC: Algorithms, architectures and applications", *IEEE Transaction on Circuits and Systems*, Vol.56, No.9, pp.1893–1907, 2009.

[9] S. Ravichandran and V. Asari, "Implementation of unidirectional CORDIC algorithm using precomputed rotation bits", *The 45th Midwest Symposium on Circuits and Systems*, Virginia, USA, pp.453–456, 2002.

[10] K. Maharatna, A. Troya, S. Banerjee, *et al.*, "Virtually scaling-free adaptive CORDIC rotator", *Computers and Digital Techniques*, pp.448–456, 2004.

[11] P. Rao and I. Chakrabarti, "High-performance compensation technique for the radix-4 CORDIC algorithm", *IEEE Proc. of Computers and Digital Techniques*, Guwahati, Indai, pp.219–228, 2002.

[12] E. Antelo, J. Bruguera, J. Villalba, *et al.*, "Redundant CORDIC rotator based on parallel prediction", *Proc. of Computer Arithmetic*, Bath, UK, pp.172–179, 1995.

[13] D. Timmermann and H. Hahn, "Low latency time CORDIC algorithms", *IEEE Transaction of Computers*, Vol.41, No.12, pp.1010–1015, 1992.

[14] S.F. Tso, B. Juang and M.Y. Tsai, "Para-CORDIC: Parallel CORDIC rotation algorithm", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol.51, No.8, pp.1014–1024, 2004.

[15] G.H. Xiaobo and C.B. Steven, "Expanding the range of convergence of the cordic algorithm", *IEEE Transactions on Comput-*

ers, Vol.40, No.1, pp.12–21, 1991.

[16] J.F. Zhang, "Research and implementation of DDFS and FFT based on improved CORDIC", *Master Thesis*, National University of Defense Technology, China, 2011.(in Chinese)

[17] J.S. Walther, "A unified algorithm for elementary functions", *Proc. of Spring Joint Computer Conference*, New York, USA, pp.379–385, 1971.

[18] T.Y. Wang and Q.X. Jiang, "Design and implementation of parallel FFT based on CORDIC", *Computer Engineering and Applications*, Vol.7, No.3, pp.51–54, 2005. (in Chinese)

[19] Y.W. Lei and J. Zhou, "Research of the parallel CORDIC algorithm and its implementation in FPGA", *Computer Engineering and Science*, Vol.30, No.8, pp.75–78, 2008. (in Chinese)

[20] K. Kalyani and D. Sellathambi, "Reconfigurable FFT using CORDIC based architecture for MIMO-OFDM receivers", *Information Communication and Embedded Systems*, Chennai, India, pp.670–675, 2013.

[21] R. Bhakthavatchalu and N.A. Kareem, "Comparison of reconfigurable FFT processor implementation using CORDIC and multipliers", *IEEE Transaction on Recent Advances in Intelligent Computational Systems*, Trivandrum, India, pp.343–347, 2011.

[22] G.P. Zhang and F. Chen, "Parallel FFT with CORDIC fir Ultra wide band", *Personal, Indoor and Mobile Radio Communications*, Singapore, pp.1173–1177, 2004.

[23] Y.W. Lei, Y. Dou, S. Guo, *et al.*, "FPGA accelerating quad-double high precision floating-point applications for exa-scale computing", *Proc. of the 24th ACM International Conference on Supercomputing*, New York, USA, pp.325–336, 2010.

[24] R.X. Gong, J.Q. Wei and D. Sun, "FPGA implementation of a CORDIC-based radix-4 FFT processor for real-time harmonic analyzer", *2011 7th International Conference on Natural Computation*, Shanghai, China, pp.1832–1835, 2011.

[25] N. Mahdavi and R. Teymourzadeh, "On-Chip implementation of high speed and high resolution pipeline radix 2 FFT algorithm", *International Conference on Intelligent and Advanced Systems*, Kuala Lumpur, Malaysia, pp.1286–1288, 2012.

[26] A. Banerjee and A.S. Dhar, "FPGA realization of a CORDIC based FFT processor for biomedical signal processing", *Microprocessors and Microsystems*, Vol.25, No.3, pp.131–142, 2001.

CHEN Jiyang received the B.S. degree and M.S. degree in computer science in 2012 and 2015, respectively, from National University of Defense and Technology (NUDT), China. Her research interests are high performance computing, multi-core architectures and on-chip networks. (Email: 824750961@qq.com)

LEI Yuanwu is an associate professor of National University of Defense and Technology (NUDT), China. His research interests include high performance computer architecture and computing engineering.

PENG Yuanxi is a professor of National University of Defense and Technology (NUDT), China. His research interests are in the areas of high performance computing, multi- and many-core architectures, on-chip networks and architectural support for parallel programming.

HE Tingting is a student in microelectronics and solid-state electronics in National University of Defense and Technology (NUDT), China. Her research interests include high performance computer architecture and computing engineering.

DENG Ziye is a student in microelectronics and solid-state electronics in National University of Defense and Technology (NUDT), China. His research interests include high performance computer architecture and computing engineering.