Applied Mechanics and Materials Vols. 241-244 (2013) pp 2901-2909
© (2013) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMM.241-244.2901

Design and Implementation of Configurable FFT/IFFT Soft-Core Based on FPGA

Zeng Guigen[1,a], Ren Jiangzhe [1,b]

[1]Nanjing University of Posts and Telecommunication, Nanjing 210003, China

[a]zgg@njupt.edu.cn, [b]jiangzhe78@gmail.com

Keywords: FFT/IFFT, configurable, FPGA, pipeline, IP core

Abstract: As a basic transforming operation between time field and frequency field, FFT has been widely used in detection, telecommunication, signal processing, multimedia communication etc. The implementation of the FFT algorithms on FPGA is always the hot research spots. In order to overcome the shortcomings on the FPGA resource reusability used in FFT algorithm, this article discusses a new configurable and high efficient FFT/IFFT soft-core solution. The FFT/IFFT soft-core adopts radix-2^2 algorithm and Single-Path Delay Feedback (SDF) pipeline structure. Its configurable factors include: FFT/IFFT, FFT points ($2^n, n \in [3,12]$), fixed-point bit width, clock delay of complex multiplier. The design takes FPGA chip Stratix II *EP2S130F780C4* as hardware platform, and the complete simulation and synthesis is taken. The maximum operating frequency is up to *306.30MHz*. If 300MHz clock frequency used, *4096-point* FFT could be realized in *26.73us*, and the consumption of memory resources is only *148Kbit*. Compared with Altera FFT IP-core, our FFT/IFFT soft-core has a little bit longer computing time *(0.6%)*. However, the LE resource consumption is only 79% of Altera FFT IP-core.

Introduction

In the field of digital signal processing, FFT (Fast Fourier Transform) plays an important role and has wide applications. FFT is a fast algorithm of the DFT (Discrete Fourier Transform). When the transformation interval length N is large, we decompose the DFT computation using Radix-2, Radix-4 or Radix-2^2 to make the DFT long sequence broken down into short sequences. This greatly accelerates DFT processing speed and increases DFT computational efficiency by one to two orders of magnitude. FFT hardware solutions are generally divided into ASIC, FPGA, MCU, DSP or general-purpose processor, etc. From the factors of FFT processing speed, cost-effective and flexibility, the implementation based on FPGA is better than the other implementations, which can be easily scaled and parameterized when Hardware Description Language (HDL) is used in the design. Thus it has become the mainstream of the current application. In the actual development process, the program that has the same functions but different parameters is always developed to configured IP core, which has nothing to do with the process [14]. IP core has become a developing trend, because it not only facilitates program reuse and avoids duplication of efforts, but also greatly improves programming efficiency and reduces the burden on the engineers [6]. In this paper, architectures of configurable FFT/IFFT soft-core processor based on new structure oriented FFT algorithms are presented.

Algorithm Selection and Hardware Architecture Selection

FFT Algorithm Selection. Generally, we use the Radix-2 FFT algorithm of Cooley and Tukey to achieve the traditional FFT processor. Using the symmetry feature of the twiddle factors, this method can reduce the number of multiplication and addition operations, and it also has a simple hardware structure. In addition, it's convenient to control. Radix-4 FFT algorithm can further reduce the number of multiplication and addition operations. However, its hardware structure is relatively more complex than Radix-2 FFT algorithm, and the control mechanism is more complicated.

In order to achieve the purposes that reduce the usage of multiplication and addition operations and simplify hardware structure, the paper chooses the Torkelson *Radix-2^2 FFT* algorithm to implement the FFT processor. Radix-2^2 FFT algorithm not only retains the simple hardware structure of Radix-2 FFT algorithm, but also has the same advantage in Radix-4 FFT algorithm that consumes less computation. Combined with pipelining technology, it's very suitable to implement on the FPGA for high speed and efficiency FFT processor.

Generally, N-point DFT is defined by

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

where W_N, the so called "twiddle factor", denotes the Nth primitive root of unity, with its exponent evaluated modulo N . Most fast algorithms share the same general strategy, i.e. mapping the one-dimensional transform into a two or multi-dimensional representation, then exploiting the congruence property of its coefficients to simplify the computation.

In the classical divide and conquer procedure leading to a Radix-2 DIF FFT, consider the first 2 steps of decomposition together. Applying a 3-dimensional linear index map,

$$\begin{aligned} &n=\frac{N}{2}n_1+\frac{N}{4}n_2+n_3 \quad (n_1\,,n_2=0\,,1\,;\ n_3=0\,,1\,,\cdots,\frac{N}{4}-1)\\ &k=k_1+2k_2+4k_3 \qquad (k_1\,,k_2=0\,,1\,;\ k_3=0\,,1\,,\cdots,\frac{N}{4}-1)\end{aligned} \tag{2}$$

the Common Factor Algorithm (CFA) [2] takes the form of

$$\begin{aligned} X(k_1+2k_2+4k_3)&=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1}x(\tfrac{N}{2}n_1+\tfrac{N}{4}n_2+n_3)W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}\\ &=\cdots=\sum_{n_3=0}^{\frac{N}{4}-1}\{\underbrace{[\overbrace{B_{N/2}(n_3,k_1)}^{BFI}+\overbrace{B_{N/2}(\tfrac{N}{4}+n_3,k_1)}^{BFI}(-j)^{(k_1+2k_2)}]}_{BFII}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3}\}\end{aligned} \tag{3}$$

where the first butterfly structure can be written as

$$B_{N/2}(\tfrac{N}{4}n_2+n_3,k_1)=x(\tfrac{N}{4}n_2+n_3)+(-1)^{k_1}x(\tfrac{N}{2}+\tfrac{N}{4}n_2+n_3) \tag{4}$$

BFII in Eq.3 and $B_{N/2}(\frac{N}{4}n_2+n_3,k_1)$ represent the first two columns of butterflies with only trivial multiplications in the SFG of the Radix-2^2 algorithm. After these two columns, full multiplications are used to apply the decomposed twiddle factor $W_N^{n_3(k_1+2k_2)}$ in Eq.3. Note that the order of the twiddle factors is different from that of a Radix-4 algorithm [1].

Applying CFA with this cascade decomposition recursively to the remaining DITS of length N/4 in Eq.3, the complete radix-2^2 DIF FFT algorithm is obtained. An N=32 example is shown in Fig.1 [1] where ◇ represents trivial multiplications by $W_N^{N/4}=-j$, which involves only real-imaginary swapping and sign inversion. And ▷ represents complex multiplications by W_{32}^i.

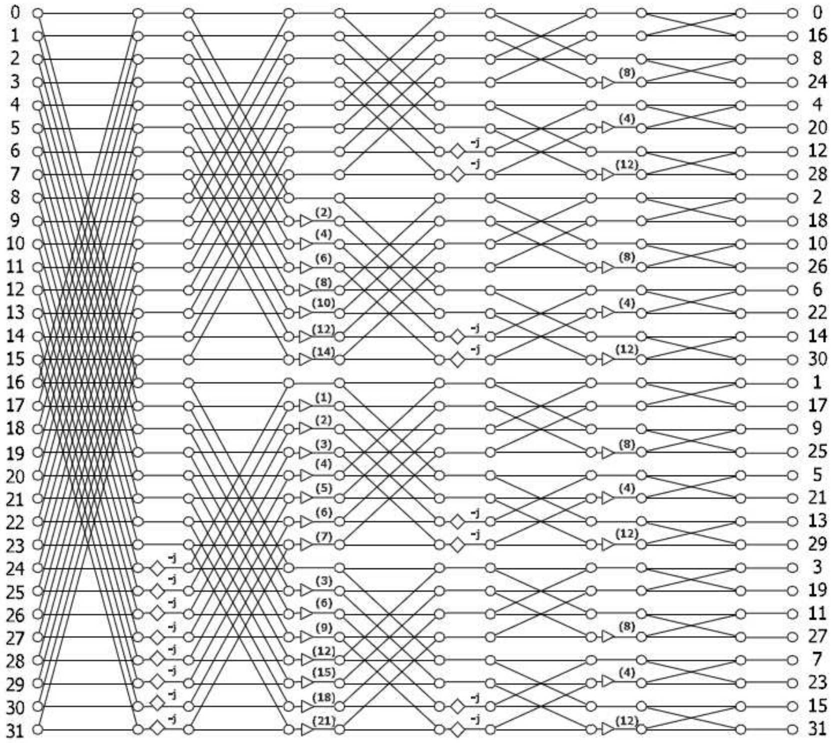


Fig.1. Radix-2^2 DIF FFT signal flow graph for N=32

The approach with cascade decomposition can be further exploited.

The Architecture Based on Fixed Points Radix-2^2 DIF FFT. The architecture design for pipeline FFT processor had been the subject of intensive research as early as in 80's when real-time processing was demanded in such applications as radar signal processing [3, 5]. Several architectures have been proposed over the last 2 decades. Here different approaches are put into functional blocks with unified terminology, such as R2MDC(Radix-2 Multi-path Delay Commutator), R2SDF(Radix-2 Single-path Delay Feedback), R4SDF(Radix-4 Single-path Delay Feedback), R4MDC(Radix-4 Multi-path Delay Commutator), R4SDC(Radix-4 Single-path Delay Commutator) [1], etc. These architectures reveal the distinctive merits and common requirements of the different approaches.

Mapping Radix-2^2 DIF FFT algorithm to R2SDF architecture, a new architecture of Radix-2^2 Single-path Delay Feedback *(R2^2SDF)* approach is obtained [4]. Fig.2 outlines an implementation of the R2^2SDF architecture for N=32 [1]. Note the similarity of the data-path to R2SDF and the reduced number of multipliers [9]. The implementation uses two types of butterfly: one identical to that in R2SDF, another contains also the logic to implement the trivial twiddle factor multiplication.

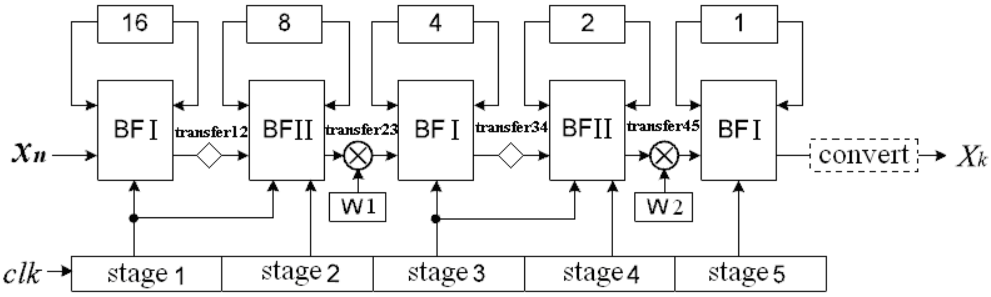


Fig.2. R2^2SDF pipeline DIF FFT architecture for N =32

Due to the spatial regularity of Radix-2^2 algorithm, the synchronization control of the processor is very simple. A ($\log_2 N$)-bit binary counter serves two purposes: synchronization controller and address counter for twiddle factor reading in each stage. The hardware requirement of proposed architecture as compared with various approaches is shown in Table 1, where not only the number of complex multipliers, adders and memory size but also the control complexity is listed for comparison. For easy reading, base-4 logarithm is used whenever applicable. It shows R2^2SDF has reached the minimum requirement for both multiplier and the storage, and only second to R4SDC for adder. This makes it an ideal architecture for VLSI implementation of pipeline FFT processors.

Table 1: Hardware requirement comparison

	Multiplier #	Adder #	Memory size	Control
R2MDC	2($\log_4$ N-1)	4$\log_4$ N	3N/2-2	Simple
R2SDF	2($\log_4$ N-1)	4$\log_4$ N	N-1	Simple
R4SDF	$\log_4$ N-1	8$\log_4$ N	N-1	Medium
R4MDC	3($\log_4$ N-1)	8$\log_4$ N	5N/2-4	Simple
R4SDC	$\log_4$ N-1	3$\log_4$ N	2N-2	Complex
R2^2SDF	$\log_4$ N-1	4$\log_4$ N	N-1	Simple

Design of Configurable FFT/IFFT Soft-Core

Architecture of Configurable Points Radix-2^2 DIF FFT. It can be pointed out from Fig.2 that the architecture using SDF for FFT has not only similarity and independence, but also high programmability. Through adding or deleting some stages, we can obtain the core to compute dynamic length of FFT sequence, which is configurable. The outline of the architecture is shown in Fig.3.

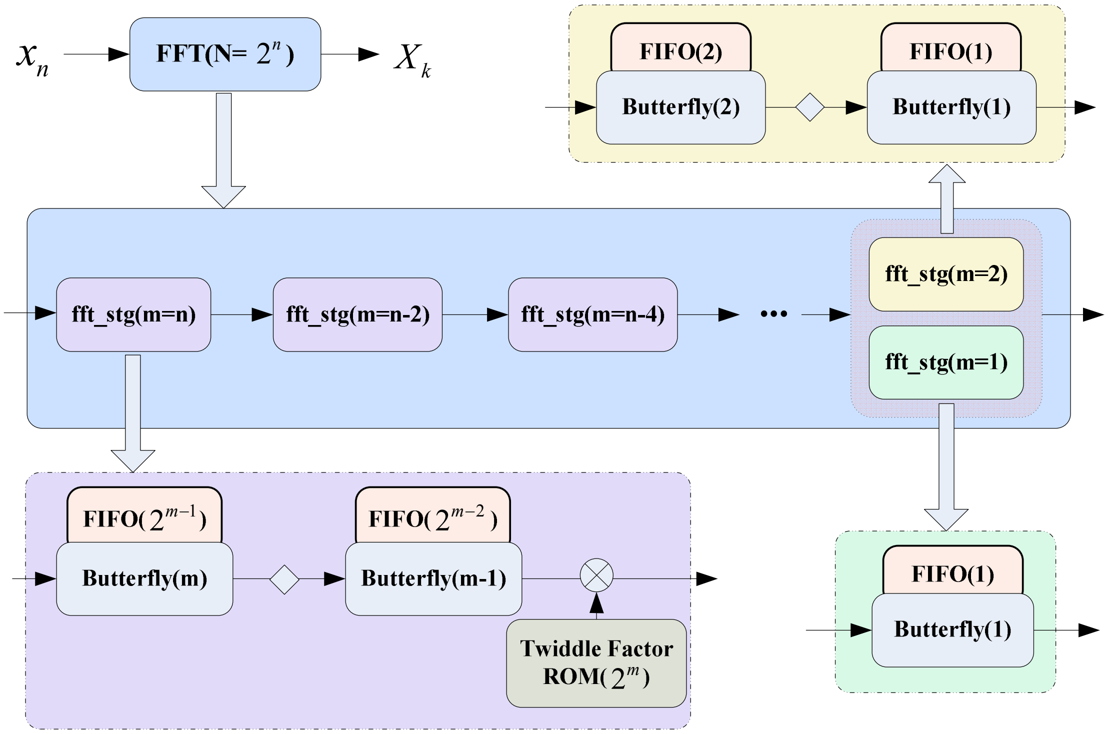


Fig.3. Architecture of Configurable points Radix-2^2 DIF FFT

When FFT order n is odd, the tail module should use fft_stg(m=1). And when n is even, the tail module should use fft_stg(m=2).

Implement of FFT/IFFT Configurability. The relationship between IFFT and FFT is like this:

$$IFFT[F(k)] = \{FFT[f^*\{n\}]\}^* / N \qquad (5)$$

So we can obtain IFFT results indirectly by using FFT module. According to Eqn.5, take the conjugate of the input data before computing FFT and take the conjugate of the output data after FFT, we can obtain the enlarged results of IFFT so far [12]. The magnification is just the FFT points N. Fortunately, FFT points are always the power of 2. So we can simply add a shift operation module to get the real IFFT results. The diagram of these operations is shown in Fig.4.

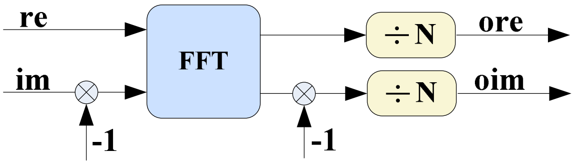


Fig.4. Diagram of using FFT to compute IFFT

Implement of fft_stg(m) Module. As the core module of the whole FFT/IFFT soft-core, fft_stg(m) adopts SDF pipeline structure. FFT module with different points has different stages of fft_stg module. Every fft_stg module is composed of 2 butterfly operations and 2 results processing module (shown in Fig.3). The first results processing module is just multiple the results by *–j* in right place. This operation can be simplified. We can just exchange the real and imaginary parts of the complex results and negate the imaginary part symbol. The second results processing module must use a complex multiplier to compute the results of second butterfly operation multiple by the twiddle factors. The twiddle factors are in advance stored in *ROM*.

As an example, we'll introduce the implementation of 32-point butterfly. The 32 complex input data is serial. To save RAM resource, we use *FIFO* structure, whose depth is 16, to store the first 16 complex data. When the other 16 complex data input, the previous data stored in FIFO output and take addition and subtraction operations with the new input data. The result of addition output directly and the result of subtraction is stored in FIFO. After all 32 complex data input, the whole 32-points butterfly operation completes.

Clock Delay of Complex Multiplier. The complex multiplier we used is the IP-core provided by Altera which is shown in Fig.5. Since this IP-core adopts the pipeline architecture, there will be some clocks delay before getting the result. More clocks delay, higher working frequency we will obtain. However, it will consume more logic resources. And vice versa. Fortunately, the delay is configurable. Through test, for the soft-core which is configured as 1024-points FFT, 18-bit fixed-point width, when the delay is 1-clock, the maximum operating frequency is up to *184.57MHz*. When the delay is 6-clock, the maximum operating frequency is up to *306.30MHz*. The latter one consumes more *314 register units* than the former one.

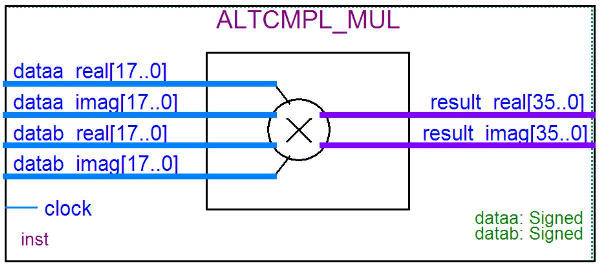


Fig.5. Complex Multiplier IP Core

Frequently, the complex multiplier maximum operating frequency directly determines the maximum operating frequency of the FFT soft-core. So we can use less clock delay multiplier unit in the FFT/IFFT soft-core on the occasion of strain resources. And when it is highly requirement to the operating frequency, a more clocks delay multiplier unit should be applied.

The Optimization of Memory Resources for Twiddle Factors. Twiddle factor storage units are used to store *pre-calculated* twiddle factors. In this module, we mainly focus on storage capacity and addressing issues [10]. The Fourier transform twiddle factor has clear features of periodicity and symmetry [7]. The periodicity is shown in Eqn.6~Eqn.8.

$$W_N^{(m+1)N} = W_N^m \tag{6}$$

$$W_N^{-m} = W_N^{N-m} \quad \text{or} \quad (W_N^{N-m})^* = W_N^m \tag{7}$$

$$W_N^{m+N/2} = -W_N^m \tag{8}$$

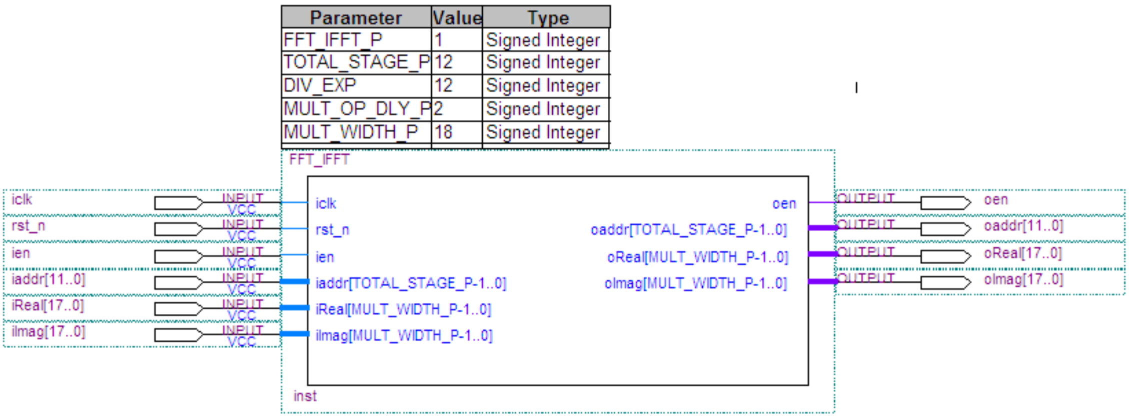
In addition, the twiddle factor is a combination of sine and cosine function. It can be decomposed into

$$W_N^{nk} = e^{-j2\pi nk/N} = \cos(2\pi nk/N) - j\sin(2\pi nk/N) \tag{9}$$

With these features, we can just store the sine and cosine values in ROM. And the actual storage capacity is just one quarter of the total. This is undoubtedly a huge save for the FPGA RAM resources. Of course, the complexity of addressing and symbols control will relatively increase.

Implement of Configurable FFT/IFFT Soft-Core

Combined with the modular thinking, we adopt the method of Top-down hierarchical approach to design the whole module [13]. From the perspective of system, the FFT/IFFT soft-core is composed of a kind of configurable Radix-2^2 module and two variable- radix modules. And the configurable Radix-2^2 module is composed of 3 kinds of small modules. Similarly, these small modules can be further refined again. Overall, the interface of the whole FFT/IFFT soft-core which is programmed in Verilog HDL is shown in Fig.6.



Parameter	Value	Type
FFT_IFFT_P	1	Signed Integer
TOTAL_STAGE_P	12	Signed Integer
DIV_EXP	12	Signed Integer
MULT_OP_DLY_P	2	Signed Integer
MULT_WIDTH_P	18	Signed Integer

Fig.6. Interface of FFT/IFFT Soft Core

Its main configuration parameters are shown in Table 2.

Table 2: List of configuration parameters

Parameter name	Default	Range	Remarks
FFT_IFFT_P	1	0,1	1: FFT; 0: IFFT
TOTAL_STAGE_P	10	3~12	FFT/IFFT Order
MULT_OP_DLY_P	2	2,6	Delay clk of Multiplier
MULT_WIDTH_P	18	9,12,18	Width of complex

Testing and Results Analysis

To test the function of FFT/IFFT soft-core, we use Modelsim to make functional simulation. And then we use Matlab to analysis the results. Firstly, we need to program the Testbench, which is used to call FFT/IFFT soft-core by using specific parameters. The Testbench reads the input data which is produced by Matlab randomly from the stimulus file and writes the results into another file [11]. In the stage of results analysis, we compare the simulation results and the results computed by Matlab. In order to achieve the completeness of the test, the tests cover all permutations and combinations of all the configuration parameters of the FFT/IFFT soft-core. Since these tests are highly repetitive, here we take 4096-points, 18-bit fixed-point width and 6-clock complex multiplier delay as an example to describe.

Functional Simulation and Error Analysis. When the Testbench is finished, we need compile all source files in Modelsim. And then we use Matlab to produce stimulus data randomly. After that we can run the simulation. Through test, when the soft-core is configured as *4096-points* FFT, it needs *4119 clocks* from inputting the first data to outputting the first data (shown in Fig.7). The maximum operating frequency is up to *306.30MHz*. If we use 300MHz as the clock frequency, we can get the result of 4096-point FFT in *26.73us*. And the consumption of memory resources is only *148Kb*.

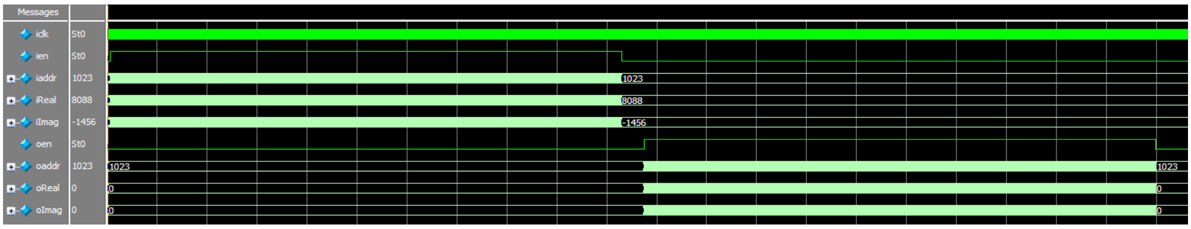


Fig.7. Functional Simulation of FFT(N=4096) using Modelsim

During the simulation, calculation results are written to file in specific format. Since we use fixed-point censored handling mechanism, error is inevitable. Here we only analyze the error of real part because the real part is familiar to imaginary part. The error result which is shown in Fig.8 can be obtained by subtracting the simulation results from the results computed by Matlab.

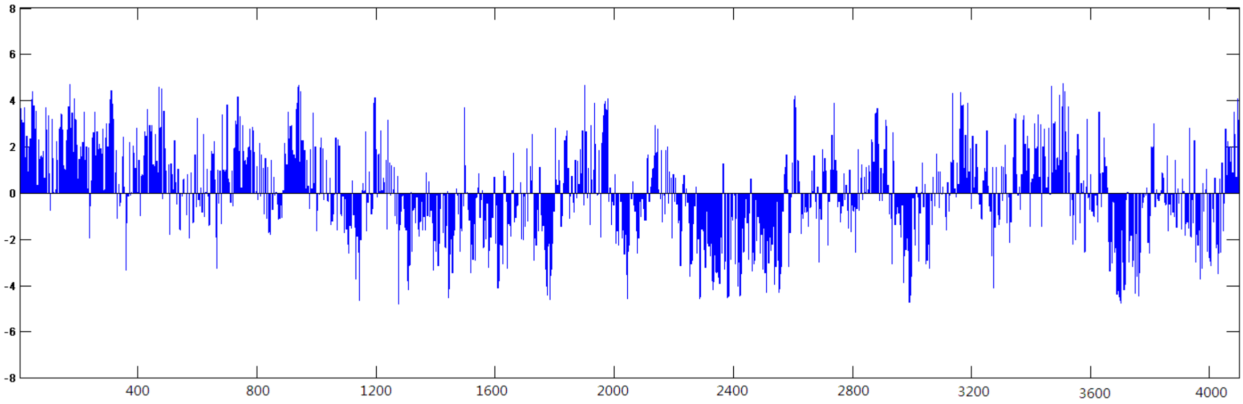


Fig.8. Computation Error analysis of N=4096 FFT

The raw input data is random, and its range is [-2^{17}, 2^{17}]. The result in Fig.8 can be seen as a snapshot of the numerous results. In this error result, the error range is *[-4.92, 4.86]*. And the expectation for the absolute value of error result is *1.783*. It indicates that the result of calculation can reach equivalent precision. On one hand, the error is resulting from the fixed-point quantization error. And on the other hand, it comes from multiplication and addition operations.

Resource Usage Analysis. The FPGA device we used is chip *EP2S130F780C4* which belongs to Altera *Stratix II* family. In software *Quartus II 9.1*, the FFT/IFFT soft-core was configured as *4096-points* FFT, *18-bit* fixed-point width and *6-clock* complex multiplier delay. Through integration, we obtain the resource usage result in Table 3.

Table 3: Resource usage of 4096-points FFT using FFT/IFFT soft core

Device type	EP2S130F780C4
Logic Elements	3,044/106,032(3%)
Memory Bits	147,912/6,747,840(2%)
9×*9*bit Multiplier	40/504(8%)
f_{max}	306.30MHz

As we can see, the resource usage is very efficient. The total LEs(Logic Elements) usage is only about 3K and Memory Bits usage is about *148Kb*. All of these merits attribute to the SDF pipeline architecture and our efforts to optimize the performance according to particular chips.

Compared with the FFT IP-core from Altera [8], our soft-core f_{max} is *smaller* than Altera FFT IP-core, and RAM resource consumption is almost equal to Altera FFT IP-core. However, our FFT/IFFT soft-core LE consumption is only *79%* of Altera FFT IP-core. Altera FFT IP-core resource usage is shown in Table.4.

Table 4: Resource usage of 4096-points FFT using Altera IP-core [8]

Device type	EP3SE50F780C2
Logic Elements	3,852
Memory Bits	147,712
18×*18*bit Multiplier	12
f_{max}	410MHz

Conclusions

In this paper, a multiple parameters configurable, high speed and low storage FFT/IFFT soft-core is presented. In practical applications, since it has great flexibility and adaptability, it can be applied in many such areas as detection, communications, signal processing and multimedia communication etc. If we use 300MHz as the clock frequency, we can get the result of *4096-point* FFT in *26.73us*. The total LEs usage is only about *3K* and the consumption of memory resources is only 148Kb. Compared with Altera FFT IP-core, our FFT/IFFT soft-core has a little bit longer computing time *(0.6%)*. However, the LE resource consumption is only *79%* of Altera FFT IP-core. Currently, this FFT/IFFT processor has been successfully used in an OFDM communication system.

Acknowledgements

This work was financially supported by Jiangsu Provincial Science and Technology Support Program (Industry) (No. SBE201201230).

References

[1] S. He and M. Torkelson: *Designing pipeline FFT processor for OFDM (de)modulation*, pp.257–262, (1998).

[2] C. **S.** Burrus: *Index mapping for multidimensional formulation of the DIT and convolution*, p.239-242, (1977).

[3] L.R. Rabiner and B. Gold: "*Theory and Application of Digital Signal Processing".* Prentice-Hall, (1975).

[4] Swedish patent application No. 95/01371, (1995).

[5] S. Lee and S.C.Park: *Modified SDF architecture for mixed DIF/DIT FFT*, p.2590–2593, (2007).

[6] Y. W. Lin, H. Y. Liu, and C. Y. Lee: *A dynamic scaling FFT processor for DVB-T applications*, p.2005–2013, (2004).

[7] T. Lenart and V. Owall: "*Architectures for dynamic data scaling in 2/4/8 K pipeline FFT cores"*, p.1286–1290, (2006).

[8] *FFT MegaCore Function User Guide*. www.altera.com. (2006).

[9] S. He and M. Torkelson *"A New Approach to Pipeline FFT Processor,"* Proceedings of the IPPS, (1996).

[10] K. Nakos, D. Reisis, N.Vlassopoulos, *"Addressing Technique for Parallel Memory Accessing in Radix-2 FFT Processors"* ICECS, pp.52-56, (2008).

[11] T. H. Yu, C. Z. Zhan, Y. J. Cho, C. L. Yu, and A. Y. Wu, *"Efficient fast Fourier transform processor design for DVB-H system,"* in Proc. 18th VLSI/CAD Symp., pp. 65–68, (2007).

[12] C. T. Lin, Y. C. Yu, and L. D. Fan, *"A low-power 64-point FFT/IFFT design for IEEE 802.11a WLAN application,"* in Proc. IEEE Int. Conf. Circuits Syst. pp.4523–4526, (2006).

[13] S Sukhsawas and K Benkrid, *"A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs"*, pages 229–232, (2004).

[14] John F. Wakerly, *"Digital Design Principles and Practices",* Pearson Education, (2006).