A scalable, fixed-shuffling, parallel FFT butterfly processing architecture for SDR environment

Ting Chen, Hengzhu Liu, and Botao Zhang
College of Computer, National University of Defense Technology, Changsha, China
a) tingchen@undt.edu.cn

Abstract: This paper presents a programmable and high-efficient application specific instruction-set processor (ASIP) for fast Fourier transformation (FFT) processing based on software defined radio (SDR) methodology. It adopts single instruction multiple data (SIMD) architecture to exploit the parallelism of butterfly operations in FFT algorithm. The proposed ASIP features eight parallel radix-2 butterfly computations with fixed vector data shuffling pattern. In addition, a flexible vector address generation unit is proposed to support inner-and inter-group addressing mode. Experiment results show that the proposed FFT ASIP is much more flexible than previous works and outperforms state-of-the-art FFT ASIP architectures in term of energy-efficiency.

Keywords: FFT, parallel butterfly processing, SDR

1 Introduction

FFT processing is one of the most critical components in the orthogonal
frequency-division multiplexing (OFDM)-based systems. OFDM is used in a number of systems from WLAN, WiMAX to broadcast technologies including DVB and DAB. It directly affects the system throughput and accuracy of the channel estimation. Nowadays, multiple wireless standards are emerging and requiring different FFT size with various throughput. For example, in WiMax and LTE standards, the size of FFT processing can be configured from 128 to 2048 points according to the channel bandwidth scaling from 1.25 to 20 MHz under different operation environment. Further more, multi-band UWB standard proposed for short-range high-bandwidth communications requires the data rates ranging from 200 to 480 Mb/s. All of these systems need real-time implementation of FFT processing with different sizes. Hence, efficient implementation of the FFT with high throughput and appropriate flexibility is very important for multi-standard communication environment.

In the last decades, various efficient FFT architectures have been investigated. Generally, they can be divided into two categories: pipeline architecture and memory-based architecture. Pipeline architectures [1] have the advantage of high throughput, but demand high area cost especially for long-size FFT algorithms. Moreover, dynamic overflow protecting is difficult to be applied on the pipeline architecture, so it needs to extend the width of processing data to prevent overflow. Memory-based architectures [2] usually consist of a butterfly processing unit, a multi-bank memory and control unit. Though they have low area cost, their throughput are constrained by the size of processing unit and memory bandwidth.

Recently, platforms for SDR [3, 4] are proposed for wireless baseband signal processing. They feature multiprocessor systems with SIMD vector processing engines to exploit the parallelism of FFT processing and other algorithms. In case of AnySP architecture, it consists of a simple scale datapath and a SIMD datapath, in which a SRAM-based network is adopted to support different vector data shuffling modes among SIMD lanes when performing parallel butterfly processing. It can be programmed to support multiple standards and the SIMD architecture provides a highly energy-efficient way to exploit inherent parallelism in certain tasks. FFT algorithm has natural parallelism in butterfly processing. This paper adopts the similar idea as AnySP to exploit the parallelism of butterfly in FFT computation. However, the data flow is totally different. We construct a parallel butterfly processing unit with fixed data shuffling and computation pattern, which is easy for hardware implementation, reusability and control. Furthermore, a flexible and efficient vector address generation mechanism is proposed for both vector data and coefficient memory accessing. The proposed ASIP can be easily extended to support various-size FFT algorithms.

2 Parallel FFT algorithm

Implementing an efficient FFT processing requires a good exploitation of the computational complexity reduction and parallelism. The N-point Discrete Fourier Transform (DFT) of complex input \( x(n) \) can be defined as
Fig. 1. SFG of 128-point FFT algorithms with fixed data shuffle between stages

\[ X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad k = 0, 1, \ldots, N - 1; \quad W_N^{nk} = e^{-j2\pi nk/N} \quad (1) \]

Cooley-Turkey radix factorization is used to reduce the hardware cost. As shown in the equation (2), a 128-point DFT can be decomposed into one epoch [5] with eight radix-16 butterfly groups and one final epoch with sixteen radix-8 butterfly groups.

\[ X(k_1 + 16k_2) = \sum_{n_1=0}^{7} \left( \sum_{n_2=0}^{15} x(n_1 + 8n_2)W_{16}^{k_1n_2}W_{128}^{k_1n_1} \right)W_{8}^{k_2n_1} \quad (2) \]

where \( k_1 = 0, 1, 2, \ldots, 15 \) and \( k_2 = 0, 1, 2, \ldots, 7 \). Radix-16 and radix-8 can be replaced by radix-2^4 and radix-2^3 respectively to further reduce the number of complex multiplications [1]. Based on our further investigation of FFT signal flow graph (SFG), we reschedule the FFT signal flow structure, as shown in Fig. 1. The SFG within the butterfly group contains a fixed data shuffling.
and butterfly computation mode, which is suitable for parallel processing. In order to reduce power and area cost, the multiplications between butterfly stages can be implemented by constant multipliers, instead of the full complex multipliers. Radices beyond $2^4$ are impractical because the increasing adders required by constant multiplier are no longer beneficial compared to full multiplier.

3 Parallel butterfly computation architecture

Fig. 2 shows the architecture of proposed FFT ASIP. It consists of a scale processing and control unit (SPCU), a parallel butterfly processing unit (PBPU) and a vector load/store unit (VLSU). The SPCU is a C-compiler supported RISC processor executing program control and system parameter configuration instructions. In this section, we mainly introduce the PBPU and VLSU of the proposed processor.

3.1 Parallel butterfly processing unit

The PBPU structure is shown in Fig. 2, which consists of a 16-entry local vector registers (LVR) and eight radix-2 butterfly processing units (BPU0~7). In this design, Dual LVRs feed the data for the BPU0~7, and a complex data point consists of a 16-bit real part and a 16-bit imaginary part, so each LVR is $8 \times 32$-bit. Intermediate data within the butterfly group are stored in the LVR, so data communication between vector memory (VM) and LVR only occurs at the beginning and the end of computation of butterfly groups. C0~7 are constant multipliers. In order to support parallel butterfly computation of radix-2 to radix-$2^4$ groups, different constant parameters are needed for C0~7. Table I shows the parameters of C0~7 and its corresponding custom instructions. Besides butterfly computations within groups, full complex multiplications are required between epoches, thus every BPU also incorporates a complex multiplier. Fig. 3 shows the structure of BPU. The 32-bit full complex multiplier is divided into two pipeline stages for high frequency: one for four 16-bit real number multipliers and the other one for two 32-bit adders.

Constant multipliers in PBPU use canonic signed-digit (CSD) representation for area-efficient implementation [8]. Among them, C6 and C7 are

![Fig. 2. Architecture of FFT ASIP](image-url)
Table I. Constant parameters and corresponding instructions

| Instructions | R2BF0 | R2BF1 | R2BF2 | R2BF3 |
|--------------|-------|-------|-------|-------|
| C0           | 1     | 1     | 1     | 1     |
| C1           | 1     | -j    | -j    | -j    |
| C2           | 1     | 1     | $W_{8}^{1}$ | $W_{8}^{1}$ |
| C3           | 1     | -j    | $W_{8}^{3}$ | $W_{8}^{3}$ |
| C4           | 1     | 1     | 1     | $W_{16}^{1}$ |
| C5           | 1     | -j    | -j    | $W_{16}^{3}$ |
| C6           | 1     | 1     | $W_{8}^{1}$ | $W_{16}^{5}$ |
| C7           | 1     | -j    | $W_{8}^{3}$ | $W_{16}^{7}$ |

Fig. 3. Butterfly processing unit

Fig. 4. CSD representation and constant multiplier

the most complex constant multipliers since they have three constant real factors, as shown in Fig. 4 (a). Two of the factors 0.7071 and 0.3828 are not used at the same time in the computation, so they can share part of partial products. Fig. 4 (b) shows the constant multiplier of C6 and C7, it mainly consists of four constant real factor multipliers and two adders. Constant real
factor multipliers are directly realized by using partial products compression and trivial additions for low delay time. Resource sharing is exploited to reduce the area cost in each BPU, where adders in the constant multiplier are shared with the adders of compressed partial product of the 16-bit multipliers in complex multiplier, besides, four 16-bit adders for radix-2 butterfly processing can be reconfigured as two 32-bit adders for complex multiplier in the second pipeline stage. Resource sharing in BPU lead to maximum 9% area increase compared with a solo complex multiplier. Rather than different data shuffling patterns between SIMD lanes in [3, 4], the fixed computation and data shuffling mode in this work makes the PBPU easy to be implemented and extended.

### 3.2 Vector address generation unit

The VLSU controls the VM reading and writing operations. It generates both data memory and coefficient addresses. VM is a static random access memory (SRAM) decomposed into eight banks for parallel data accessing. In-place scheme which stores the butterfly output at the same memory locations used by the input of same butterfly is adopted to minimize the memory size. The VLSU can load or store eight data points per clock. The strides (the address offset between two date points) of butterfly are different epoch by epoch that may cause bank conflicts in parallel accesses. Conflict-free memory access strategy is adopted to avoid bank conflicts [6]. The row and bank address of $2^n$-point FFT can be expressed as follows.

\[ \text{row address} = b[n-1]b[n-2] \cdots b[4]b[3]. \]

\[ \text{bank address} = b[n-1]b[n-2]b[n-3] \oplus \cdots \oplus b[2]b[1]b[0]. \]  \hspace{1cm} (3)

where $\oplus$ means the modulo-8 addition. It is a inner-group linear addressing mode.

Besides data, inter-epoch twiddle coefficients are stored in the VM too. Taking 2048-point FFT for example, it is decomposed into two radix-2⁴ epoches and one radix-2³ epoch. Between the first two epochs, there are 128 different inter-epoch coefficient groups \( \{W_N^0, W_N^{8k}, W_N^{16k}, W_N^{128k}, \ldots, W_N^{158k}\} \), where k is the radix-2⁴ butterfly group number in the first epoch and \( k = 0, 1, 2, \ldots, 127 \). The sixteen twiddle factors in each coefficient group is stored in two aligned rows of VM. Between the second epoch and third epoch, coefficient groups with \( k = 0, 16, 32, 48, \ldots, 112 \) are used circularly. Obviously, it is a inter-group linear addressing mode with a fixed offset.

Fig. 5 shows the hardware implementation of vector address generation logic and corresponding instructions. The address consists of three elements: pt1 indexes the base address, pt2 indexes the vector offset addresses that provide small offset values for each data-point within groups, and imm12 delivers a unified offset between groups for linear group addressing. All the three elements are added together to get the final addresses for vector memory access. The flexible address generation mechanism supports both inner- and inter-group linear addressing mode.
With all the custom hardware extensions and new instructions in place, an N-point FFT computation can be easily implemented on the proposed ASIP. Fig. 6 illustrates the mapping flow of FFT computation which is decomposed of several radix-2^4 epochs. Software pipeline can be applied in the inner loop to greatly improve the usage of functional units and execution pace, about 6 cycles per radix2\(^4\) butterfly group on average.

### 4 Experiment results

A maximum 2048-point FFT ASIP is setup to demonstrate the efficiency of proposed technique, the VM size is 4k words (including coefficient memory). The whole project is synthesized with Design compiler under a 65 nm low-power low-leakage standard cell library, and gate-level dynamic power is evaluated using Synopsys PrimePower, which considers both switching and leakage components. Fig. 7 (a) shows the chip micrograph and features are summarized in Fig. 7 (b). The core size is 0.75 \(mm^2\). With careful pipeline partition and gate-clock enabled, power consumption is about 51 mW when running 2048-point FFT at 470 MHz.

In order to validate the efficiency of the proposed scheme, normalized Energy/FFT and normalized area \cite{7} are used to compare the energy and area efficiency of various FFT processors respectively. Table II shows the comparison of the proposed FFT processor with other previous FFT ASIPs, our proposed FFT ASIP outperforms in energy-efficiency compared with other works. Among them, \cite{2} adopts the similar epoch idea to implement parallel butterfly operations, but needs extra data shuffle operations at the beginning and end of butterfly group. It utilizes large register file (RF) to store the in-
Fig. 7. Layout and features of the proposed processor

Table II. Performance comparison of Various FFT ASIP

|                      | Proposed | Guan [2] | Bo [6] | GAEA [3]*2 | AnySP [4]*2 |
|----------------------|----------|----------|--------|------------|-------------|
| Technology           | 65 nm    | 130 nm   | 130 nm | 90 nm      | 65 nm       |
| Supported FFT        | variable | 16-1024  | 16-4096| variable   | variable    |
| Wordlength           | 16-bit   | 16-bit   | 16-bit | 16-bit     | 16-bit      |
| Core voltage         | 0.9 V    | 1.2 V    | 1.2 V  | 1.2 V      | 0.9 V       |
| Clock rate           | 470 Mhz  | 320 Mhz  | 100 Mhz| 350 Mhz    | 300 Mhz     |
| Cycles for 1k-point FFT | 1024     | 4526     | 1280   | 1950       | About 556   |
| Power (mW)           | 51       | 60.7     | 87.2   | 96.9       | 258.2       |
| Area (mm²)           | 0.75     | 0.85*1   | 2.23   | 1.47       | 2.18        |
| Norm. Energy <sub>FFT</sub> | 111 nJ   | 120.7 nJ | 157 nJ | 151.9 nJ   | 478.5 nJ    |
| Norm. Area <sub>FFT</sub> | 0.75     | -        | 0.56   | 0.77       | 2.18        |

*1 Not include the data memory  
*2 The performance is estimated on one core, and memory size is normalized to 4k-word  
*3 \( \text{Norm. Energy}_{\text{FFT}} = \frac{\text{Power} \times T_{\text{clock}} \times N_{\text{FFT}}}{(V_{\text{dd}})^2} \), \( \text{Norm. Area} = \frac{\text{area}}{(V_{\text{dd}})^2} \)

Intermediate data and reduce the number of memory access, but RF is accessed much more frequently than the memory and costs a lot of power itself. Our proposed processor only uses a 16-entry RF with two write and one read ports. Besides, [2] did not make use of the constant multiplier within the butterfly groups. In [6], it adopts radix-4/2 butterfly structure and exploits limited parallelism in FFT computation, so the power-consuming memory accessing is much more frequent. Our processor has higher area cost than [6], that is because it has almost four times radix-2 butterfly computation units of that in [6], and the SPCU occupies about 20% area of the processor. However, our FFT ASIP is much more flexible than [6].

As for processors designed for SDR environment, both GAEA [3] and AnySP [4] adopt multi-processing cores that employ wide VLIW and SIMD architecture to exploit instruction and data parallelism in FFT computation. The processing core of GAEA employs a four-way VLIW and four-lane SIMD architecture with complex-number supported instructions for FFT computa-
tion in 4G wireless communication, and the vector register file in each lane has 8 read ports and 4 write ports. The wide VLIW and multiple-port RF enhanced the power and area cost. The single processing core of GAEA has similar normalized area cost with our proposed processor, but our processor delivers a better energy-efficiency than it. In AnySP, the processing core adopts five-way VLIW and 64-wide 16-bit SIMD lanes to exploit the parallelism of different algorithms. The fine-grain 16-bit ALUs in each lane need to run multiple iterations to implement the complex-number operations, like complex multiplication and addition in FFT computation, which increases the number of RF accessing and power consumption. Besides, both GAEA and AnySP use a dedicated shuffling network between SIMD lanes to regulate the vector data. They need to reconfigure the shuffle pattern in every butterfly stage to regulate the vector data in FFT computation, which further increases the power cost and hardware complexity. Power and area consumption of one processing core in AnySP is about 258.2 mW and 2.18 \( \text{mm}^2 \) respectively under the same condition. The single processing core consumes triple area of our processor, and needs about 556 clock cycles when implementing 1k-point FFT, which is about half the cycle number needed in our FFT processor. However, the proposed architecture is much more energy-efficient than it. The fixed shuffle pattern, complex multiplier, fused constant multiplier and butterfly processing instructions enable our proposed architecture to be much more efficient for FFT computation than GAEA and AnySP.

5 Conclusion

This paper presents an energy-efficient FFT processor based on parallel butterfly processing. It implements eight butterflies simultaneously with a fixed data shuffling and butterfly computation pattern. Resource sharing, constant multipliers and simple control structure minimize system area and power consumption. Besides, this paper provides a scalable parallel butterfly processing architecture to support various-point FFT algorithms, which is suitable for future evolving standards in SDR environment.