Implementation of High-Throughput FFT Processing on an Application-Specific Reconfigurable Processor

Lei Liu, Ziyu Yang, Sikun Li, Ming Yan
School of Computer Science
National University of Defense Technology
Changsha, Hunan, China
alwater.liulei@gmail.com

***Abstract*—To meet the increasing requirements on both high-performance and high-flexibility for modern communication applications, this paper presents a novel Application Specific Reconfigurable Architecture called ASRA that tightly integrates a custom reconfigurable core with a very-long instruction word (VLIW) basic core. ASRA uses clustered register files and the two parts can work in parallel. The reconfigurable core used to execute multi-grained custom instructions offers improved performance by hardware acceleration for critical computation task, and the basic core for general instructions provides enough flexibility. The run-time context manager can dynamically select and bind custom instructions to appropriate reconfigurable fabrics while considering run-time changing scenarios. As a case study, we exploited variable-length and high-throughput fast Fourier transformation (FFT) processing which is a kernel data processing task in communication systems on ASRA processor. Using fast scratchpad memory and reconfigurable port-binding mechanism, we pipeline read/write operation and butterfly operation and connect them to support hardware pipeline execution of the loop kernel. Experiment results show that ASRA achieves a high performance improvement and a good flexibility.**

Keywords: Custom Instructions; Run-time Reconfigurable; Fast Fourier Transform.

I. INTRODUCTION

The fast Fourier transformation (FFT) and its inverse (IFFT) are essential processing blocks used for data conversion and represent the most computation-intensive tasks in orthogonal frequency-division multiplexing (OFDM) technique. OFDM has been widely used in various communication applications, such as digital video broadcasting (DVB-T), ultra-wideband (UWB), wireless local area network (WLAN), and worldwide inter-operability for microwave access (WiMAX).

Nowadays, the rapidly developing digital communication applications have raised increasingly stringent requirements on both high-throughput and high-flexibility for FFT computation. Different communication standards raise specific performance requirements. For example, the FFT/IFFT module must finish 64-point operation in 3.2us in a WLAN (802.11a/g) system, and the 802.15.3 multi-bandwidth UWB standard requires the highest data rates, ranging from 200 to 480 Mb/s[1]. In addition, processors should be well programmable or reconfigurable to adapt to modern wireless systems that integrate various communication standards and multiple operating modes. For example, in WiMAX/IEEE 802.16 standard, the FFT size has to range from 128 to 2048 points to scale the channel bandwidth from 1.25 to 20MHz for different applications [2].

Although there has been a lot of work on FFT processing in both software and hardware, it is challenging to provide both high throughput and flexibility. The existing designs for FFT processing can be divided into four categories. Digital signal processors (DSPs) offer good flexibility, but cannot meet the stringent high-throughput requirement. Application specific integrated circuits (ASICs) can be implemented to achieve high performance. However, they are short of programmability. Reconfigurable arrays (RAs) which is divided into fine-grained and coarse-grained are promising design choices since application can be mapped on them to meet both throughput and flexibility requirements. Most of present RAs are general purpose homogeneous arrays that use a large amount of configuration memories and a noticeable cost of time for reconfiguration. Compared with DSP, ASIC, and RA implementations, application specific instruction-set processors (ASIPs) can balance the tradeoff between flexibility and performance. ASIPs that have been customized to perform certain applications efficiently can provide both good flexibility with software control and high throughput with hardware acceleration.

The ASIP implementation of FFT algorithms typically falls into one of the three categories. One is based on the regular radix-2 Cooley-Turkey FFT algorithm and is exploiting pipeline parallelism in the data path to improve computing performance. It needs a large number of hardware resources. The second category employs higher-radix FFT and/or mixed-radix FFT algorithms to reduce the number of multiplications and additions. In the above two categories, butterfly operations are organized into several stages, and each stage cannot proceed until all the computations within the previous stage have finished. Therefore, a large amount of intermediate results are generated and need to be stored in each stage, which is facing a memory bottleneck for frequent access to main memory. The third category focuses on increasing the utilization of fast local memory to address the memory bottleneck. Baas [3] presented a cached-FFT algorithm to reduce the access of main memory. Guan [1,2] improved the cached-FFT algorithm by optimizing the organization of butterfly operations. Both Bass[3] and Guan [1,2] employ

register files as the cached-memory, but the register file size limits the number of FFT points that can be cached.

In this paper, we propose a novel application specific reconfigurable architecture called ASRA that tightly integrates a custom reconfigurable core with a very-long instruction word (VLIW) basic core to meet the stringent requirements. As a case study, we exploit variable-length and high-throughput fast FFT processing which is a kernel data processing task in communication systems.

II. THE ASRA ARCHITECTURE

A. Clustered VLIW-like Hybrid Architecture

Using extensible/configurable processors in SoC designs has proven to be a good way to solve the conflicting flexibility and performance requirements of embedded systems design. While their usefulness has been demonstrated in a wide range of products such as Xtensa processor from Tensilica [3], a few challenges remain to be addressed. One of the shortcomings is the fixed hardware for hot spots along the whole application span. The hardware on which custom instructions are executed is fixed at design-time irrespective of the consideration of its utilization during the application run-time. To address this problem, dynamic reconfiguration which offers the adaptation at run-time is a potential approach.

The main feature of our work is the application specific reconfiguration at the application run-time. In ASRA design, we devote great efforts to find reuse of complex instructions as much as possible within an application or even across applications, and implement hardware acceleration in the form of custom instructions using reconfigurable rather than hardwired fabric.

As shown in Fig.1, ASRA is a clustered VLIW-like architecture and contains two heterogeneous clusters: basic core and reconfigurable core. This allows us to benefit from clustered register file scheme which results in increased speed, reduced area, and lower power consumption, as compared to a single register file based design. In the instruction pipeline of ASRA, the basic core constitutes a basic data path for execution of basic instructions, and the reconfigurable core composes a customizable data path for custom instructions.

The basic core and reconfigurable core share some identical instruction pre-processing modules, including Instruction Cache, Fetch, Dispatch and Decode Units. ASRA issues multiple independent instructions simultaneously. The two cores can work in parallel and communications between them can be achieved by executing special instructions to transfer data among clustered register files.

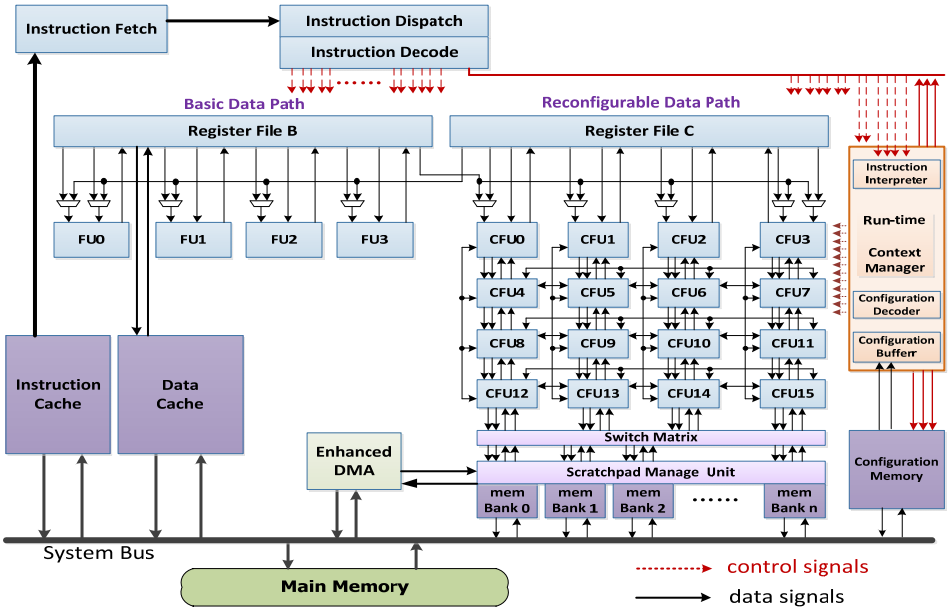

Figure 1. The ASRA Architecture

B. Custom Reconfigurable Core

Reconfigurable computing architecture has been researched for many years. Previous studies of reconfigurable architecture are different in various design points such as logic granularity, configuration control, interconnection and interface with host processor and so on.

The reconfigurable core in ASRA consists of Register Files, Custom Function Units (CFUs) array, reconfigurable interconnection, scratchpad manage unit, local data memory, configuration memory and run-time context manager. An enhanced direct memory access (DMA) controller is employed to transfer data between local data memory and main memory. Local data memory is on-chip scratchpad memory and consists of certain number of single-port SRAM banks. Scratchpad

manage unit contains a group of address generator and corresponding read/write control logic. Each memory bank can be accessed separately through its independent data port. Scratchpad manage unit is configurable and the port-binding with each memory bank is reconfigurable. Thus multiple addressing modes can be supported.

Traditional reconfigurable arrays generally take either fine-grain or coarse-grain homogeneous structure. ASRA differs from the traditional ones. One of our design purposes is to find reuse of complex instructions as much as possible within applications. Therefore, our solution allows both fine-grain and coarse-grain (called mix-grain) and heterogeneous design choices on application's demand. In ASRA, CFUs are reconfigurable application specific function units. This leads to reduced reconfiguration cost on time and storage and increased hardware resource utilization. Interconnections among CFUs are reconfigurable during run-time. CFUs can be organized in either parallel or sequential work mode. Thus several techniques that include VLIW, vector operations and fused operations can be exploited to improve performance.

All control information needed by the execution of custom instruction is encoded jointly in instruction and configuration. Execution of custom instruction is controlled by run-time context manager. The run-time context manager consists of instruction interpreter, configuration buffer, configuration decoder, and finite state machine. A custom instruction maybe needs two-level decoding.

C. ASRA Template Design Space

ASRA is a flexible template for application specific reconfigurable architecture. Architecture parameters that can be explored include: local data memory size, configuration memory size, register file size, CFU's function and number, interconnection, instruction and configuration encoding, etc.

ASRA is similar to ADRES [4] which tightly couples a coarse-grain reconfigurable array with a VLIW processor. The difference mainly lies in two aspects. One is the interface between basic processor and reconfigurable hardware. (1) In ADRES, the VLIW- and array mode are mutually exclusive; while in ASRA the basic core and reconfigurable core can work in parallel thus exploit more parallelism. (2) ADRES uses a centralized register file, while ASRA employs clustered register files. The other aspect is the structure of reconfigurable core. In ADRES, reconfigurable array is coarse-grain and is controlled by VLIW controller; while in ASRA reconfigurable core is mix-grain and is controlled by run-time context manager which can support more custom instructions and increase the utilization of hardware resource.

III. MAPPING AND OPTIMIZATION

A. FFT Algorithms

Given a sequence *x (n)*, an N-point discrete Fourier transform is defined as

$$Y(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk} \quad ,k,n\in[0,N-1] \tag{1}$$

where the *x(n)* and *Y(k)* are complex numbers. The twiddle factor is

$$W_N^{nk}=(W_N)^{nk}=e^{-j\frac{2pnk}{N}} \tag{2}$$

Butterfly operation is a basic computational block for FFT. Fig.2 shows the dataflow diagram of a butterfly operation in decimation-in-time (DIT) FFT algorithm.

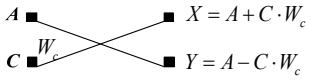

Figure 2. Radix-2 butterfly operation in DIT FFT

Taking a 32-point FFT implementation for example, the dataflow is shown in Fig.3. In Cooley-Turkey FFT algorithm, there are 5 ($\log_2 N$ =5, N=32) stages and 16 (N/2, N=32) butterfly operations within each stage. Each stage cannot proceed until all the computations within the previous stage have finished. In cached-FFT algorithm, the overall FFT computation is split into some small data-independent groups and the off-chip communications only occur at the beginning and end of the epochs. For each pass within an epoch, all intermediate results are stored in fast local memory.

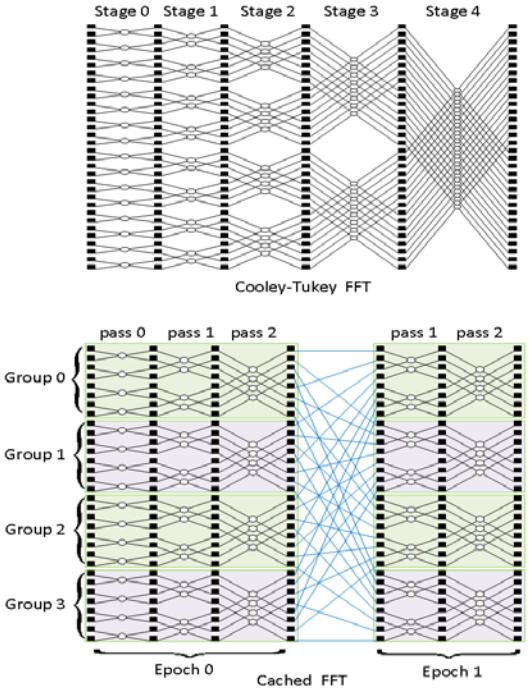

Figure 3. Structure for 32-point radix-2 FFT

B. Custom Instructions and Functional Units

To accelerate FFT computation, four custom instructions are added. DMA_DO instruction is used to transfer data between main memory and local memory in DMA mode. It has three operands: data block size, source address and destination address. BUT2_CONF instruction loads configurations for the following instruction. BUT2_DO instruction performs a butterfly operation. BUT2_LOOP instruction starts the process

of a group of butterfly operations and automatically executes a loop-level computation.

The CFUs are tailored for FFT computations and the reconfigurable data path is optimized to speed up the computation. As shown in Fig.4, a butterfly operation is mapped and implemented on a CFU. This CFU has two input and two output ports. Each port is 32-bit width, including 16-bit real and16-bit image parts. It contains four multipliers, eight adders. These arithmetic operation units are pipelined in four stages and four cycles are needed to process a butterfly operation. Both input and output operands are stored in local memory. Each read/write operation is pipelined into two cycles.

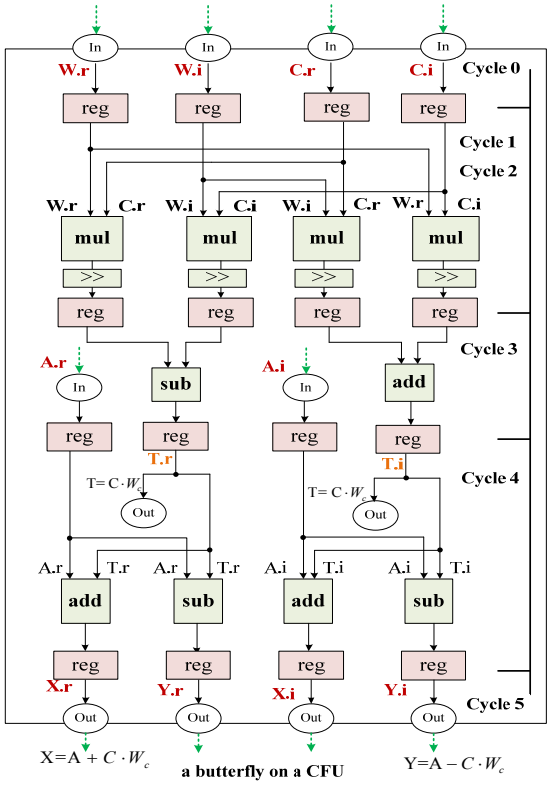

Figure 4. Mapping a Radix-2 butterfly onto a CFU

When the CFU is properly reconfigured, read/write data and butterfly operation can be connected to form a long operation pipeline which consists of six stages, shown in Fig.5.

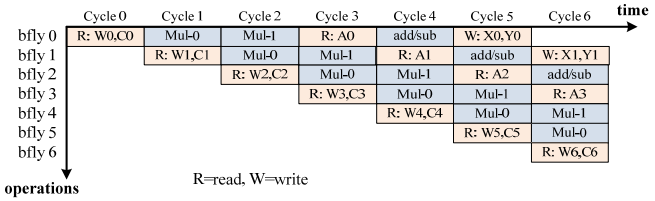

Figure 5. The pipeline of a BUT_LOOP intruction

C. *Mapping and Optimization*

To support multiple butterfly operations perform in parallel, certain number of memory banks are needed and the scratchpad manage units should be properly reconfigured. The execution of a BUT2_LOOP instruction needs two input and two output data and one twiddle factor every cycle when the pipeline is filled. To address the data bottleneck, ten local memory banks are employed and corresponding access logics are contained in scratchpad manage unit, thus ten operands can be accessed in parallel. Two CFUs are employed to execute two BUT2_LOOP instructions simultaneously. Therefore, while two loops are performing automatically on hardware pipeline, there are up to twelve butterfly operations that are executing concurrently.

IV. EXPERIMENT RESULTS

A. *Experiment*

As a case study, the FFT specific processor based on ASRA is implemented to estimate the performance and area cost. A functional RTL model is firstly designed in Verilog HDL hardware description language. Then it is synthesized in Synopsys Design Compiler to generate the CMOS standard cell ASICs using TSMC 0.13 μm technology.

TABLE I. FFT IMPLEMENTATIONS ON ASRA

Scheme	***Usage of Instructions***	***Speedup***
Basic core	Basic	1
Software Pipeline	BUT2_CONF, BUT2_DO	2.7
Hardware Pipeline	BUT2_CONF,BUT2_LOOP	3.8

We exploit several software implementations of FFT on ASRA. One scheme only employs basic instructions, and its results are used as the baseline. In another scheme, CFUs are configured to work in VLIW way that exploits software pipeline techniques, and it achieves a 2.7 speedup compared to the baseline. The best scheme is to use BUT2_LOOP instruction which perform kernel loops in automatic hardware pipeline, and its speedup is up to 3.8.

TABLE II. RESULTS OF VARIOUS-LENGTH FFT

Points	***Cycles***	***Throughput(Msample/s)***
128	284	47.7
256	568	47.7
512	1188	45.6
1024	2496	43.4
2048	6192	35.1
4096	25474	17.1
8192	53762	16.2

Table II presents the data throughput results for different FFT sizes, from 128 to 8192 points. For a 1024-point FFT computation, the data throughput can reach up to 43.4 Msample/s, which meets UWB-OFDM specifications.

B. Comparison and Analysis

To compare various related designs, the results data of some representative ones are listed in Table 3.

FPGA or ASIC implementations perform FFT computation in dedicated hardware, and generally adopt the delay-feedback structure such as the multiple delay feedback (MDF)[9], dual delay feedback(DDF)[10] and serial delay feedback (SDF). These implementations can achieve high performance, but are short of programmability. Both DSP[7]and ASIP[1,2,6,8]are programmable architecture, and can offer good flexibility. However, DSP cannot meet the stringent high-throughput requirement.

Compared with Guan [1], our work achieves a performance improvement about 48.2% at the cost of increased area about 24.5% which is mainly used by on-chip fast local memory. Our work is comparable to Guan [1] which designs an ASIP based on cached-FFT algorithm. In Guan [1], two custom register files are employed as cached-memory. Each custom register file is 128-bit width and contains 32 entries. Its custom instructions perform read/write and butterfly operation separately. It has two shortcomings. One is that cached-memory size is limited by register file. The other is that at the beginning and end of each butterfly group, it has to perform a number of load/store operations to transfer data between register file and main memory. We addressed the two issues in our reconfigurable core. The multi-bank memories offer enough data bandwidth and larger cached-memory size. The run-time configurability support BUT2_LOOP instruction to perform a kernel loop in automatic hardware pipeline.

Compared to general-purpose reconfigurable array such as ADRES [4], the application specific reconfigurable array in ASRA which is designed on the resource sharing strategy greatly reduces configuration information. The reconfiguration cost on both time and storage is smaller.

V. CONCLUSION

This paper presents an application specific reconfigurable architecture called ASRA. It tightly integrates a custom reconfigurable core with a VLIW basic core. It adopts clustered VLIW-like hybrid architecture approach, and the two clusters work in parallel. Using run-time context manage technique, reconfigurable core executes multi-grained custom instructions, which improves effectively performance by hardware acceleration for critical computation task. As a case study, we exploited variable-length and high-throughput FFT processing Using fast scratchpad memory and reconfigurable port-binding mechanism, we pipeline read/write operation and butterfly operation and connect them to support hardware pipeline execution of the loop kernel. Experiment results show that our work achieves a high performance improvement and a good flexibility.

TABLE III. COMPARISON AMONG VARIOUS IMPLEMENTATIONS OF FFT PROCESSING

Reference	Scheme	Technology (nm)	Points	Word Length	Time for 1024 Points	Area	Frequency （MHz）
TI [7]	DSP,R^2-CFFT	-	-	-	77.55us	-	320
Tang[9]	ASIC,R^X MDF	180	64-1024	10	512Points：2.4GS/s	3.2mm^2	300
Chen[10]	ASIC,$R^{2/4}$DDF	180	128-1024	13	61us	1.47mm^2	51
Jacobson[8]	ASIP,$R^{2/4}$-CFFT	65	-4096	-	3.1us	-	866
Hassan[6]	ASIP,R^2-CFFT	130	-4096	13	42.2us	-	100
Guan[3]	ASIP,R^2-CFFT	130	16-1024	16	14.1us	147 KGate	320
Our work	rASIP,R^2-CFFT	130	16-8192	16	7.3us	183 KGate	320

ACKNOWLEDGMENT

This work was supported by National Natural Science Foundation of China under grant 61076020

REFERENCES

[1] Guan X, Fei Y S, Lin H, Hierarchical design of an application-specific instruction set processor for high-throughput and scalable FFT processing[J]. IEEE Transactions on VLSI Systems, 2012,20(3):551-563.

[2] Guan X, Lin H, Fei Y S, Design of an application-specific instruction set processor for high-throughput and scalable FFT [C]. IEEE International Symposium on Circuits and Systems,Taipei,2009:2513-2516.

[3] Baas B M. An approach to low power, high performance, fast fourier transform processor design [D].Stanford University, 1999.

[4] Bouwens F, Berekovic M, Kanstein A, et.al. Architecture exploration of the ADRES coarse-grained reconfigurable array[J]. Springer Reconfigurable Computing: Architectures, Tools and Applications, 2007(1): 1–13

[5] Chakrapani L, Gyllenhaal J, Scott A,et.al.Trimaran: an infrastructure for research in instruction-level parallelism[J]. Languages and Compilers for High Performance Computing, 2005(1):922-932

[6] Hassan H M, Mohammed F, Shalash A, Implementation of a reconfigurable ASIP for high throughput low power DFT/DCT/FIR engine[J]. Journal on Embedded Systems,2012(1):1-18

[7] Texas Instruments. TMS320C6713 floating-point digital signal processor[M]. Dallas:2005.

[8] Jacobson A T, Truong D N, Baas B M. The Design of a Reconfigurable Continuous-Flow Mixed-Radix FFT Processor[C]. IEEE International Symposium on Circuits and Systems. Taipei, 2009:1133-1136.

[9] Tang S N, Liao C H, Chang T Y, An area-and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems[J]. IEEE Journal of Solid-State Circuits, 2012(7): 1419-1435.

[10] Chen C M, Hung C C, Huang Y H, An energy-efficient partial FFT processor for the OFDMA communication system[J]. IEEE Trans. Circuits Syst. II: Express Briefs, 2010,57(2):136–140.