# SOFT-CORE STREAM PROCESSING ON FPGA: AN FFT CASE STUDY

*Peng Wang, John McAllister, Yun Wu*

Institute of Electronics, Communications and Information Technology (ECIT),
Queens University Belfast, UK

## ABSTRACT

The increasing design complexity associated with modern Field Programmable Gate Array (FPGA) has prompted the emergence of 'soft'-programmable processors which attempt to replace at least part of the custom circuit design problem with a problem of programming parallel processors. Despite substantial advances in this technology, its performance and resource efficiency for computationally complex operations remains in doubt. In this paper we present the first recorded implementation of a softcore Fast-Fourier Transform (FFT) on Xilinx Virtex FPGA technology. By employing a streaming processing architecture, we show how it is possible to achieve architectures which offer 1.1 GSamples/s throughput and up to 19 times speed-up against the Xilinx Radix-2 FFT dedicated circuit with comparable cost.

***Index Terms***— FPGA, Fast-Fourier Transform (FFT), streaming processing architecture

## 1. INTRODUCTION

The computational capacity and memory bandwidth capabilities of modern Field Programmable Gate Array (FPGA) make them ideal host devices to high performance DSP circuits, however the scale of modern devices makes the complexity of designing custom circuits for these applications increasingly unproductive. This has resulted in the emergence of software-programmable 'soft' processor architectures which attempt to replace the circuit design problem at least partially with a programming problem for parallel processor architectures. However, doubts remain about this design approach despite major advances in specific application domains, such as detection for Multi-Input Multi-Output (MIMO) wireless [1], where performance and cost are actually beyond that which dedicated circuits can achieve.

Specifically, the performance bounds and resource efficiency of this approach for a wide-range of DSP operations, such as the Fast-Fourier Transform (FFT) critical to modern signal processing applications, is unproven. Operations such as FFT are highly computationally demanding and their performance on soft processors such as [2, 3] is unrecorded.

This work is supported by UK Engineering and Physical Sciences Research Council contract number EP/H051155/1

In this paper, we conduct a case study to provide the first recorded implementations of FFT on softcore processors, and to measure its performance and resource efficiency as compared to dedicated circuits. Specifically, this paper extends previous work [1] which created the highest performance softcore processor on record to make two contributions :

1. A novel softcore stream processing architecture for FPGA DSP kernels is presented which enables factors of 7.4 speed-up and 5.4 improvement in resource efficiency compared to existing softcore architectures.

2. The proposed architecture is used to benchmark FFT implementations with up to a factor 19 speed-up with comparable resource efficiency.

The remainder of the paper is organised as follows. Section 2 motivates the background and related work, before Sections 3 and 4 detail the streaming processor and multicore architectures for FFT; implementation results are presented in Section 5.

## 2. BACKGROUND AND RELATED WORK

The DFT $X(k)$ of an $N$-element data sequence of data $x(n)$ is:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, 0 \leq k \leq N-1 \qquad (1)$$

where $W_N = e^{-j2\pi/N}$ is a twiddle factor. Directly computing (1) leads to $O(N^2)$ complexity; the FFT exploits the symmetry and periodicity of the twiddle factors to enable a computationally efficient $O(N \log_r N)$ alternative for a radix-*r* FFT. Despite this complexity reduction, the FFT is still a computationally demanding operation whose high-performance embedded implementation has prompted much research interest. The majority of this work addresses dedicated circuit implementations, e.g. [4, 5, 6, 7] or media processors [8, 9]. However, on FPGA only dedicated circuit-based designs such as [10] have been reported.

Modern FPGA hosts a vast array of computational and memory resources, to the extent that the complexity of the predominant dedicated-circuit design approach has become

very large. This has prompted the emergence of 'soft' processor architectures, hosted on the FPGAs programmable fabric, which enable a partial redefinition of the design approach to incorporate programmable processors customised to the application, rather than dedicated circuits. For instance, the vector processor in [3] is scalable to allow a varying number of processing lanes, whilst the approach in [11] adopts a multicore processing solution for FPGA. The work in [1] exploits massively parallel networks of fine-grained FPGA Processing Elements (FPEs) to achieve very high performance implementation of detectors for MIMO systems. However, to date there is no recorded realisation of FFT on these architectures.

The FPE-based architecture in [1] offers the highest computational performance of all these options; the performance of a 256-point FFT on a 8-FPE Multiple Instruction Multiple Data (MIMD) architecture is compared to that of the Xilinx Radix-2 FFT component in Table 1.

**Table 1**. 8-FPE 256-point FFT

| | **LUTs** | **DSP48Es** | **T (MSample/s)** | **T/LUT ($\times 10^{-3}$)** |
|---|---|---|---|---|
| FPE | 2,296 | 8 | 30.5 | 13.3 |
| Xilinx | 621 | 6 | 61.9 | 99.7 |

The performance and efficiency of this realisation as compared to the Xilinx component is disappointing, exhibiting less than 50 % of the absolute throughput and only 13% of of the resource efficiency. There are three major limitations of the FPE-based architecture which lead to this situation:

1. The load-store architecture and lack of forwarding facility results in a large number wasted cycles due to pipeline bubbles; specifically, 48% of the instructions are wasted on NOPs and load/store instructions.

2. The 16-bit real ALU requires 10 cycles for a butterfly.

3. The simple FIFO-based communication network is not flexible enough to support the complex communication pattern of FFT, resulting in 24 inter-FPE FIFOs costing 28% of total LUTs.

Apparently, even the highest performing softcore processing solution for Xilinx FPGA on record is still limited in performance and resource efficiency compared to dedicated circuit solutions. In Section 3 we propose a modified softcore architecture which resolves this situation.

## 3. SOFT-CORE STREAM PROCESSING UNIT FOR FPGA FFT

### 3.1. SPU Architecture

To overcome the wasted cycle inefficiencies in the FPE-based architecture when implementing the FFT, a Stream Processor Unit (SPU) is proposed as the foundation unit for implementation of applications of this type. The SPU is a width-configurable Single Instruction Multiple Data (SIMD) processor composed of between 1 - 32 Processing Elements (PEs), shown in Fig. 1.

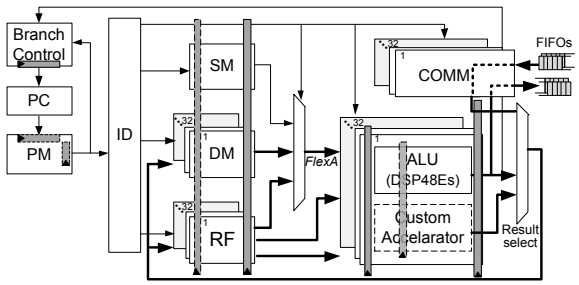



**Fig. 1**. SPU Processing Core

The ALU is implemented using Xilinx DSP48E slices; arithmetic operations employ four-address instructions to match the three-input, single-output DSP48E multiply-add computation pattern. Furthermore, the inherent forwarding path in the DSP48E can be exploited to enable instruction-controlled data forwarding.

In contrast to the load-store FPE architecture, the SPU incorporates three kinds of memory, which can be optionally included at design time to best match the architecture to the needs of the application.

1. 32-entry or 64-entry 3-read 1-write Register File (RF) in each SPU PE;

2. Dual-port RAM Data Memory (DM) in each SPU PE;

3. Shared Memory (SM) RAM accessible by all SPU PEs.

In addition, the SPU memory architecture is directly-accessed; hence, rather than all ALU data access proceeding via the RF, as is the case in the FPE-based architecture, arithmetic instructions can choose from either RF, DM or SM storage directly, with the result written to DM. Note that in Fig. 1 the flexible operand *FlexA* is connected to DSP48E port *A* and applicable for both add and multiply operations. This extends the use of data forwarding to memory operations, further reducing the potential for pipeline bubbles.

### 3.2. Complex ALU Design

The real ALU employed by the FPE architecture requires 6 cycles to compute a butterfly's multiply-add operation; given the abundance of butterfly operations in the FFT, accelerating this operation specifically will have a corresponding accelerating effect on the FFT implementation. Specifically, the SPU extends the ALU architecture to employ four DSP48E slices, as shown in Fig. 2 to enable a single-cycle complex-valued multiply-add operation via two half operations:

1. Real: single-cycle $C_r \pm (A_i \times B_i) \pm (A_r \times B_r)$
2. Imaginary: single-cycle $C_i \pm (A_i \times Br) \pm (A_r \times B_i)$

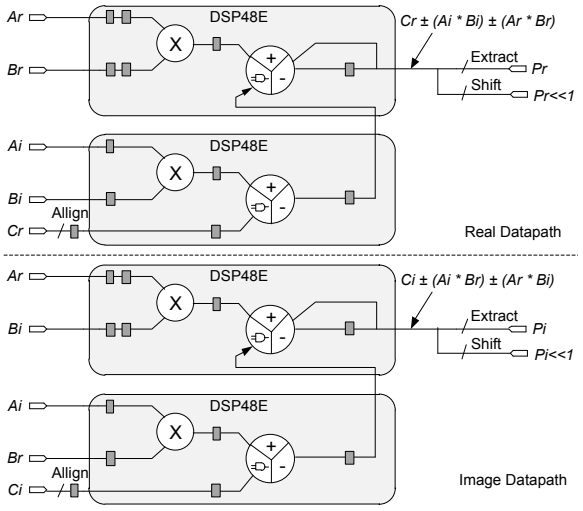



**Fig. 2**. Complex ALU in SPU

This change has been made within the framework of the existing FPE instruction set and so is transparent from the programmer's perspective. There is an extra *shift-right-by-one* output which enables the scaling operation merged with the computation. This complex ALU enables 472 MMAC/s when mapped to Virtex-5 and enables a two-cycle radix-2 butterfly.

## 4. MULTICORE-SSP FFT

### 4.1. Multicore SSP Architecture

Exploiting the SPU as a basic building block, the overall structure of the Softcore Streaming Processor (SSP) is shown in Fig. 3. This architecture exploits both Task-Level Parallelism (TLP) by varying the number of SPUs, and Data-Level Parallelism (DLP) within the SPU by varying the number of SIMD ways. Communication between SPUs in SSP uses flexible point-to-point FIFOs to provide maximum throughput with single-cycle latency and avoiding the need for data shuffling within the SPU.

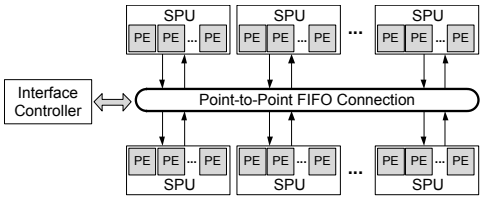
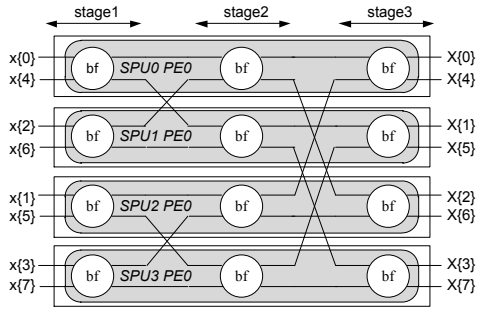
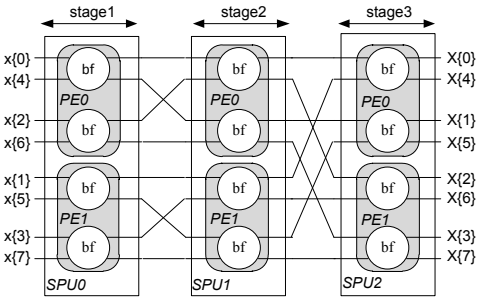



**Fig. 3**. SSP Architecture

The SSP is used as the basic computation unit of a mutlicore FFT implementation. To demonstrate, we propose two mapping approaches, one of which results in a MIMD implementation and the other a SIMD implementation of an 8-point FFT is used for illustration. The MIMD implementation is derived as shown in Fig. 4(a) by mapping butterfly operators to a multicore MIMD.




(a) MIMD Partition




(b) Multi-SIMD Partition

**Fig. 4**. FFT Graph Partition

The SIMD mapping is shown in Fig. 4(b) and employs pipelined processing, partitioning the graph vertically stage by stage such that each SIMD stage is implemented on an SPU. In order to enable the multi-SIMD processing, the rotation factors of butterfly are made private to the SPU PEs to be stored in their DM. The butterflies inside a computation stage are mapped to SPU PEs. Compared to MIMD partition, the multi-SIMD partition computes tasks in a pipelined way, resulting in higher latency, further, cutting the FFT graph vertically involves more communication cost (see edges cut by partitions). However, multi-SIMD partition can save considerable resource cost by enabling SIMD processing and simple inter-SPU communications.

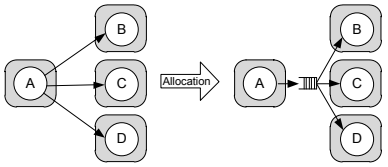
### 4.2. Communications Architecture

The MIMD mapping illustrated in Fig. 4(a) resolves the abstract channels represented by edges between nodes in dataflow graph into physical FIFOs. The default allocation scheme simply allocates all edges of nodes between any two SPU PEs into a physical FIFO. As a result, there is at most one FIFO between any two SPU PEs. In the 4-SPU implementation illustrated in Fig. 4(a), each SPU communicates with the other 2 SPUs requiring 8 FIFOs. However, no two of

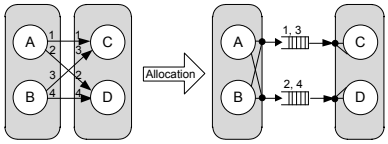
**Table 2**. Area and Performance Results for FFT

| | Config. | LUTs (x$10^2$) | BRAMs | DSP48Es | F (MHz) | T (MSample/s) | L ($\mu$s) | Speedup | T/LUT | T/DSP48E |
|---|---|---|---|---|---|---|---|---|---|---|
| 256-point FFT | 4xV1 | 18.8 | 4 | 16 | 319 | 109.5 | 2.3 | 1.8 | 5.8 | 6.6 |
| | 8xV1 | 31.1 | 8 | 32 | 316 | 224.7 | 1.1 | 3.6 | 7.2 | 6.8 |
| | 16xV1 | 61.9 | 16 | 64 | 321 | 428.0 | 0.6 | 6.9 | 6.9 | 6.4 |
| | 8xV1 | 21.2 | 9 | 32 | 355 | 142.2 | 3.2 | 2.3 | 6.7 | 4.3 |
| | 8xV2 | 36.9 | 8 | 64 | 335 | 297.8 | 1.8 | 4.8 | 8.1 | 4.5 |
| | 8xV4 | 61.4 | 10 | 128 | 310 | 551.1 | 1.8 | 8.9 | 9.0 | 4.1 |
| | Xilinx Radix -2 | 6.2 | 3 | 6 | 404 | 61.9 | 4.1 | 1.0 | 10.0 | 10.0 |
| 1024-point FFT | 8xV1 | 27.4 | 49 | 32 | 305 | 118.5 | 8.6 | 2.0 | 6.7 | 6.1 |
| | 16xV1 | 75.0 | 25 | 64 | 306 | 303.6 | 3.4 | 5.2 | 6.3 | 7.9 |
| | 32xV1 | 136.1 | 32 | 128 | 310 | 620.0 | 1.7 | 10.7 | 7.0 | 8.0 |
| | 10xV2 | 75.1 | 21 | 80 | 342 | 342.0 | 6.2 | 5.9 | 7.0 | 3.8 |
| | 10xV4 | 136.7 | 15 | 160 | 325 | 650.0 | 3.4 | 11.2 | 7.4 | 3.6 |
| | 10xV8 | 175.5 | 10 | 320 | 311 | 1105.7 | 2.0 | 19.0 | 9.3 | 3.3 |
| | Xilinx Radix-2 | 9.0 | 3 | 6 | 417 | 58.0 | 17.7 | 1.0 | 10.0 | 10.0 |

these FIFOs are used at once; hence by analysis of the read-write access patterns, these can be shared as illustrated in Fig. 5(a), reducing the number of FIFOs to 4. By sharing, the number of required inter-SPU FIFOs for a $K$-SPU partition is reduced from $K * \log_2 K$ to only $K$.




(a) Shared FIFO Allocation




(b) Conflict-free FIFO Allocation

**Fig. 5**. SSP FIFO Allocation Configurations

Likewise, the default FIFO allocation causes large amounts of access conflicts, in the early stages of the FFT (when each SPU PE compute one or more butterfly groups), as the data is consumed by a sink processor in a different order from that in which it is produced. This requires data to be buffered in the processor's local storage space, resulting in unwanted spills and reduced duty factor. This conflict can be resolved by allocating up to four FIFOs to each SPU PE for the conflicting stages. The effect of this process is illustrated in Fig. 5(b).

## 5. IMPLEMENTATION AND RESULTS

To verify the effect of these architectural manipulations on the performance and cost of the softcore-based FFT implementation, implementations of 256-point FFT on $4 \times 1$, $8 \times 1$ and $16 \times 1$ SPU MIMD allocations and $8 \times 1$, $8 \times 2$, $8 \times 4$ multi-SIMD SPU allocations are reported. This is extended for 1024 point FFT employing $8 \times 1$, $16 \times 1$, and $32 \times 1$ SPU MIMD partitions and $10 \times 2$, $10 \times 4$ and $10 \times 8$ SPU allocations.

Table 2 shows the area and performance results of the SSP implementations on Xilinx Virtex 5 VSX240T along with a radix-2 Xilinx FFT core for comparison [12]. As this shows, the performance and efficiency (measured in terms of throughput per LUT (T/LUT) and throughput per DSP48E (T/DSP48E)) of these implementations is well in advance of the original FPE-based implementation and scales well with the number of SPU PEs increases.

The MIMD FFT implementation exhibits latency less than 50% of the SIMD implementations with similar throughput but has higher BRAM requirements. Overall the SSP approach was observed to offer speed-ups of a factor of up to 8.9 (19) times than the radix-2 core for 256-point (1024-point FFT). In addition, for higher throughput FFT operations the resource efficiency (normalised to Xilinx Radix-2 FFT core in Table 2) approaches that of even dedicated circuitry for high throughput implementations, and never exhibits efficiency more than 40% lower than the Xilinx component.

In summary, the SSP-based implementation has formed an effective realisation approach for large scale FFT architectures, enabling high throughput implementations and resource efficiency comparable to dedicated circuit implementations.

## 6. CONCLUSION

This paper presented the first recorded implementation of FFT operations on FPGA-based softcore processor architectures. By extending the FPE-based processing infrastructure to include more complex memory, datapath and communications infrastructures, the resulting implementations enable speed-ups by factors of up to 19 times whilst presenting resource efficiency measures which approach those of dedicated circuit implementations, despite operating a software-programmable processing paradigm.

## 7. REFERENCES


[1] X. Chu and J. McAllister, "Software-defined sphere decoding for FPGA-based MIMO detection," *IEEE Trans. on Signal Processing*, vol. 60, no. 11, pp. 6017 –6026, Nov. 2012.

[2] J. Yu, C. Eagleston, C.H. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Trans. Reconfigurable Technology and Systems*, vol. 2, no. 2, pp. 12:1–12:34, June 2009.

[3] C. H. Chou, A. Severance, A.D. Brant, Z. Liu, S. Sant, and G. Lemieux, "VEGAS: Soft vector processor with scratchpad memory," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA, 2011, FPGA '11, pp. 15–24, ACM.

[4] B.M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 380 –387, Mar 1999.

[5] W.C. Yeh and C.W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. on Signal Processing*, vol. 51, no. 3, pp. 864 – 874, Mar 2003.

[6] Y.W. Lin, Y.C. Tsao, and C.Y. Lee, "A 2.4-gsample s DVFS FFT processor for MIMO OFDM communication systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, May 2008.

[7] C. Cheng and K.K. Parhi, "High-throughput VLSI architecture for FFT computation," *IEEE Transs on Circuits and Systems II: Express Briefs*, vol. 54, no. 10, pp. 863 –867, Oct 2007.

[8] U.J. Kapasi, W.J. Dally, S. Rixner, J.D. Owens, and B. Khailany, "The Imagine stream processor," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2002, pp. 282–288.

[9] R. Thomas, "An architectural performance study of the fast fourier transform on vector IRAM," Tech. Rep., UC Berkeley, Aug. 2000.

[10] I.S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast fourier transforms for real-time signal and image processing," *IEE Proc. Vision, Image, and Signal Processing*, vol. 152, no. 3, pp. 283–296, 2005.

[11] M.A. Kinsy, M. Pellauer, and S. Devadas, "Heracles: Fully synthesizable parameterized mips-based multicore system," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sept. 2011, pp. 356 –362.

[12] Xilinx Inc., "Xilinx LogiCORE IP fast fourier transform v7.1," Tech. Rep., 2010.