Streaming Elements for FPGA Signal and Image Processing Accelerators

Peng Wang and John McAllister, *Senior Member, IEEE*

***Abstract*— Field-programmable gate array (FPGA) devices boast abundant resources with which custom accelerator components for signal, image, and data processing may be realized; however, realizing high-performance, low-cost accelerators currently demands manual register transfer level design. Software-programmable soft processors have been proposed as a way to reduce this design burden, but they are unable to support performance and cost comparable to custom circuits. This paper proposes a new soft processing approach for FPGA that promises to overcome this barrier. A high-performance, fine-grained streaming processor, known as a streaming accelerator element, is proposed, which realizes accelerators as large-scale custom multicore networks. By adopting a streaming execution approach with advanced program control and memory addressing capabilities, typical program inefficiencies can be almost completely eliminated to enable performance and cost, which are unprecedented among software-programmable solutions. When used to realize accelerators for fast Fourier transform, motion estimation, matrix multiplication, and sobel edge detection, it is shown how the proposed architecture enables real-time performance and with performance and cost comparable with hand-crafted custom circuit accelerators and up to two orders of magnitude beyond existing soft processors.**

***Index Terms*— Fast Fourier transform (FFT), field-programmable gate array (FPGA), matrix multiplication (MM), motion estimation, processor, streaming.**

I. Introduction

RECENT years have seen rapid evolution in field-programmable gate array (FPGA) technologies, both in the scale of traditional devices and their extension to system-on-chip FPGA, incorporating heterogeneous multicore processor architectures. A key motivation for the use of FPGA is its ability to host components (known here as accelerators) that realize specific operations on the device's programmable logic, enabling that operation to be realized with high performance and/or low cost.

The resources available within modern FPGA, which may be used to compose the accelerators, are unprecedented: per-second access to trillions of multiply–accumulate (MAC) operations and bit-level memory locations via on-chip DSP units [1], [2] and block RAM (BRAM) [2], [3]. These mark FPGA as ideal hosts to high-performance custom computing architectures for signal, image, and data processing [4]. However, as the scale and the sophistication of FPGA devices increase with each passing generation, harnessing these resources becomes increasingly difficult. Traditionally, achieving requisite performance and cost has required manual design of custom circuits at register transfer level in a hardware design language. This is a highly effective approach, but imposes a heavy development load due to the low level of design abstraction.

Soft processors have been proposed to alleviate this design burden by employing a predominately software-based development route, but at present, adopting such an approach demands substantial compromise on performance and cost. No approach has been shown to support performance and cost even close to custom circuits designed via the traditional approach.

This paper proposes a resolution to this issue. A novel streaming accelerator element (SAE) is presented which enables software-based accelerator development, while maintaining the performance and cost of custom circuits. By application to accelerators for fast Fourier transform (FFT), matrix multiplication (MM), motion estimation, and sobel edge detection (SED) accelerators, the following contributions are made.

1) A novel streaming processor for FPGA, the SAE, is described and shown to overcome the performance limitations of existing soft processors.
2) It is shown how the SAE is unique among softcores in enabling real-time accelerators for standards, such as 802.11ac wireless and H.264 video.
3) It is shown how SAE-based accelerators are unique in exhibiting performance and cost which are highly competitive with custom circuits.
4) It is shown how SAE accelerators exhibit performance and cost up to two orders of magnitude beyond that of existing soft processors.

To the best of our knowledge, the SAE is the highest performance, lowest cost software-programmable component on record for FPGA and the first to enable signal and image processing accelerators with performance and cost comparable with custom circuits.

The remainder of this paper is as follows. Sections II and III survey FPGA soft processors. Sections IV and V propose the SAE to resolve the limitations outstanding in these. Section VI describes the SAE-based accelerators comparing these against real-time requirements, existing custom circuits, and soft processor solutions.

Manuscript received May 26, 2015; revised August 27, 2015 and October 16, 2015; accepted November 24, 2015. Date of publication January 6, 2016; date of current version May 20, 2016. This work was supported by the Engineering and Physical Sciences Research Council under Grant EP/H051155/1.

P. Wang is with ARM, Cambridge CB1 9NJ, U.K. (e-mail: pwang04@qub.ac.uk).

J. McAllister is with the Institute of Electronics, Communications and Information Technology, Queen's University Belfast, Belfast BT3 9DT, U.K. (e-mail: jp.mcallister@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2015.2504871

II. Background

Modern FPGA boasts enormous on-chip computation, memory, and communications resources—for instance the Virtex-7 family of Xilinx FPGAs offers up to 7×10^{12} MAC operations per second and 40×10^{12}-bits/s memory access rates. These are ideal for the construction of accelerators for signal, image, or data processing operations, either as coprocessors for high-performance computing [5], [6] architectures or standalone. However, realizing high-performance, low-cost accelerators has traditionally required manual custom circuit design at register transfer level—a process made increasingly unproductive by the scale of modern FPGA.

Attempts to overcome this productivity barrier are emerging. For example, high-level synthesis (HLS) approaches simplify this process by deriving accelerators from specifications in, for instance, C/C++ [7], CUDA [8], OpenCL [9], or Java [6]. The productivity benefits of HLS are unquestioned, but there is no compelling evidence that these can support performance and cost comparable with custom circuits, while the requirement for a host processor for any CUDA or OpenCL program imposes high resource and performance overheads.

An alternative approach is to realize software-programmable processors in the FPGA programmable logic. These soft processors allow their architecture to be tuned before synthesis to improve the performance and cost of the final result. Their use is desirable in instances where HLS approaches are constrained—for instance when specific programmatic constructs, such as pointers, are used which cannot be synthesized by HLS approaches. In addition, soft processors have been shown to enable performance scaling beyond that of HLS [10].

Soft general-purpose processors, such as MicroBlaze [11] and Nios-II [12], are performance-limited and a series of approaches attempt to resolve this issue. One approach uses soft vector coprocessors [13]–[16] employing either assembly level [13] or mixed C-macro and inline assembly programming. These have enabled orders of magnitude increases in performance relative to Nios-II and MIPS [13] but performance and cost still lag custom circuits by a significant margin—VIPERS [13] reports full block-search motion estimation with 1.2% of the throughput of a custom circuit, while consuming around 80 times the resources. An alternative approach is to redesign the architecture of the central processor architecture for performance/cost benefit—for instance, iDEA [17] reduces cost by ∼50% relative to MicroBlaze [17], but performance is, in general, little better. Finally, the multicore architectures incorporating up to 16 processors have been studied in [18]–[20] and up to even 100 processors in [20]. The work in [18] and [19] does show some speedup relative to standard MicroBlaze performance, but scalability is limited by basing the architecture on low-performance MicroBlaze processors. Similarly, while Hannig *et al.* [20] report the modest speedup of up to 25% relative to ARM Mali-T604 GPU and is based on a lower cost processing component, the demand for network-on-chip interconnect for grid-structured architectures imposes a high resource overhead. The result is that the performance of these architectures is only marginally beyond that of software-programmable devices and there is no evidence that these are competitive with custom circuits. It appears that if FPGA soft processors are to be a viable alternative to custom accelerators, then performance and cost must improve radically.

A series of alternative device architectures seek to overcome this limitation by sacrificing the fine-grained configurable nature of FPGA in order to enable more productive design. Data-centric (as opposed to control-centric) processors, such as the XPuter [21], replace the program counter prevalent in the traditional von Neumann architectures with a data counter, which supports the sequencing of data with low addressing overhead. This approach demonstrated substantial reductions in program execution overhead due to control and addressing operations, allowing generic memory address generators to iterate over patterns, known as scans, of data memory (DM) to realize computationally intensive operations with high performance and low cost. Reconfigurable processors, such as PACT XPP [22], take an alternative data-centric approach, with arithmetic logic unit (ALU)-based functional units combined via configurable interconnect to operate on abstract data streams with a specific emphasis on enabling fast, frequent reconfiguration via the use of novel reconfiguration circuitry. These data-centered processing philosophies enable high-performance realizations for typical signal, image, and data processing operations, but so far as we are aware, no one has attempted to apply similar techniques in FPGA soft processors, nor for the design of custom FPGA accelerators.

The work in [23] is a first step in that direction, sharing some characteristics of these non-FPGA-based architectures, specifically the use of large-scale multicore networks and operation on streams of data. It attempts to rethink these for the realization of custom FPGA accelerators. It uses so-called processing elements (PEs)—extremely fine-grained processors—combining these into large-scale *ad hoc* multi-PE architectures. A PE is not a general-purpose softcore processor, but is designed to enable software programmability with lowest resource cost, while maximizing performance and scaling. This approach is promising—it is the only processor-based real-time solution, and performance and cost were highly competitive with comparable custom circuits for any application, but still a general capability to enable performance and cost comparable with custom circuits is not in evidence.

This paper proposes an approach to resolve this issue.[1] Building on the PE concept, streaming soft processors are created and used to enable real-time performance with efficiency comparable with custom circuit accelerators and well in advance of existing soft processors, including that in [23].

Section III describes the PE acceleration approach before the SAE is devised to overcome PE efficiency, program control, and memory addressing limitations in Sections IV and V. Finally, Section VI uses the SAE to realize FFT, motion estimation, MM, and SED, comparing the results against standard performance criteria as well as custom circuits and existing soft processors.

[1] Preliminary versions of this paper are reported in [24] and [25].

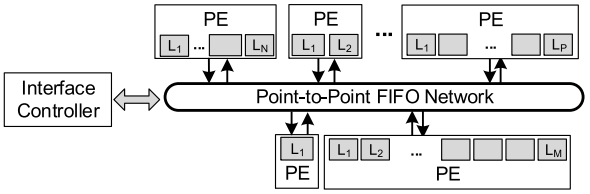


Fig. 1. Conceptual multicore PE accelerator architecture.

III. FPGA Processing Element Accelerators

Fig. 1 shows the conceptual multi-PE accelerator architecture proposed. As this shows, these are composed of multiple PEs communicating via a point-to-point network composed of first-input, first-output (FIFO) queues. A PE is a software-programmable single-instruction multiple-data (SIMD) component whose architecture is soft for configuration presynthesis in a number of aspects, most notably number of SIMD lanes. Each PE adopts a configuration independent of all others and PE execution is decoupled, such that the network is a heterogeneous multiple-instruction, multiple-data (MIMD) machine. Point-to-point links are made between communicating lanes whether housed within the same or disparate PEs, while the multicore point-to-point topology is tailored before synthesis appropriate to the operation being realized.

Realizing architectures of this kind efficiently demands two key PE features.

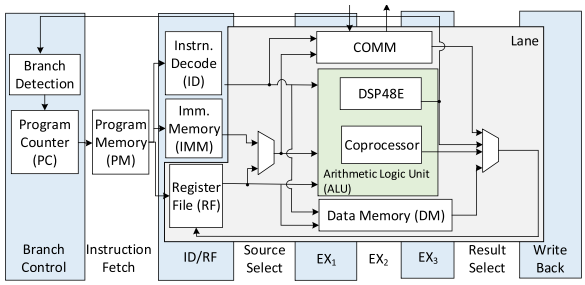
1) *Standalone:* A PE must be able to process data, manage memory access, and communicate externally under software control without a host processor.
2) *Lean:* Combined very high performance and low cost are demanded to support large-scale multicores.

None of the alternative soft processing approaches surveyed in Section II satisfy both of these criteria. One such which does is the FPGA PE (FPE).

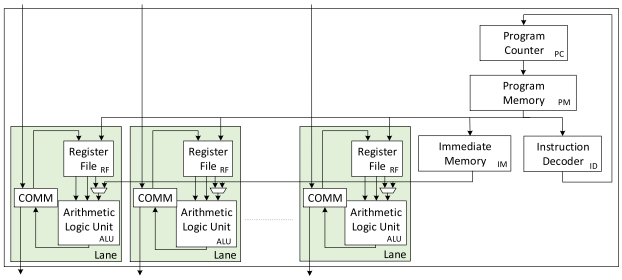
A. The FPGA Processing Element

The FPE [23] is a reduced instruction set computer load-store PE, SIMD, and single instruction, single data (SISD) (i.e., single-lane SIMD) variants of which are shown in Fig. 2. It houses a program counter, program memory (PM), instruction decoder, register file (RF), branch detection, DM, immediate memory, and an ALU based on the DSP48E in Xilinx FPGA [1]. A COMM module allows direct insertion/extraction of data into and out of the FPE pipeline.

The FPE is very lean, incorporating only those components critical to software programmability and in addition is highly configurable for low-cost realization—the reader is referred to [23] for detail on instruction set architecture (ISA) and configurability. The result is exceptionally high performance at low cost—a 16-bit SISD FPE on Xilinx Virtex 5 VLX110T supports 480 MMACs/s requiring 90 lookup tables (LUTs)—just 14% of the cost of a Microblaze and 35% of that of the iDEA lean processor on the same device [17], while the 10 × 10 tightly-coupled processor array (TCPA) multicore in [20] consumes 134 570 LUTs. This performance/cost combination



(a)



(b)

Fig. 2. FPE. (a) FPE–SISD mode. (b) FPE–SIMD mode.

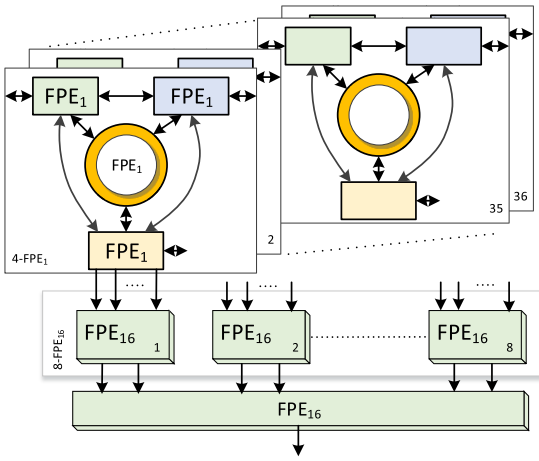


Fig. 3. FPE-based SD for 4 × 4 802.11n.

endowed a multi-FPE accelerator for sphere decoding (SD) in 4 × 4 multiple-input, multiple-output 802.11n, shown in Fig. 3, with two unique features—it is the only real-time software-programmable FPGA solution for this application, and performance and cost were highly competitive with custom circuit solutions.

The key feature of the FPE, which enables these capabilities, is extreme resource efficiency. By ensuring absolute lowest cost FPE structure, the economies of scale produce dramatic reductions in multicore resource cost. However, this extreme focus comes at the cost of flexibility: once synthesized, the FPE does not exhibit the same degree of flexibility as a general soft processor because the architecture is highly constrained at design time to support the desired operation with highest performance and lowest cost; hence, while it may be repro-

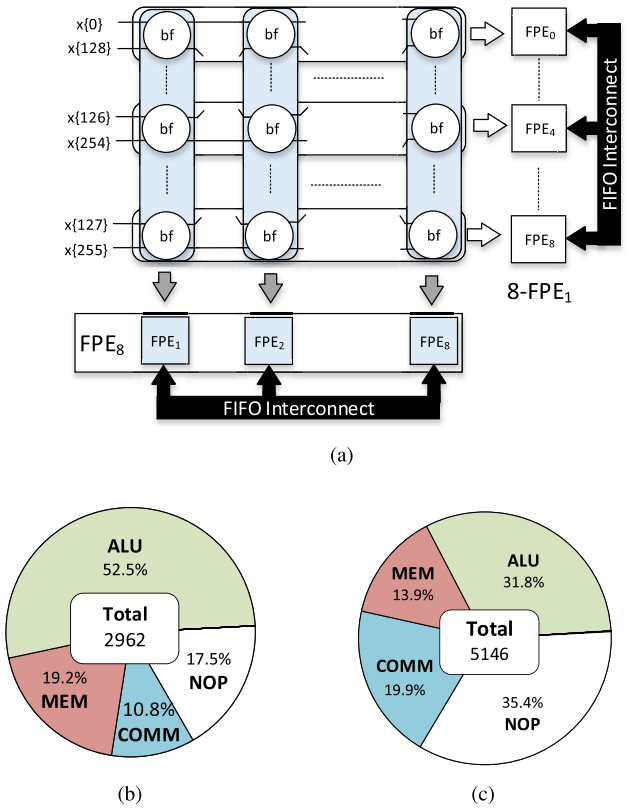


Fig. 4. FFT_{256}: FPE-based 256-point FFT. (a) FFT mappings. (b) 8-FPE_1. (c) FPE_8.

grammed after synthesis, it cannot enable general-purpose operation in the manner of a standard softcore. In addition, to minimize cost while supporting software programmability, the FPE operates under two substantial absolute restrictions.

1) *Processor and ISA:* The FPE is a load-store processor which can only source non-constant ALU operands and produce results to RF, with all memory and communications access via loads and stores to RF.
2) *Addressing Modes:* The FPE supports only direct memory addressing.

Sections IV and V illustrate the effects of these restrictions in the context of typical image and signal processing operations and proposes the SAE as their resolution.

IV. Stream Processing for FPGA Accelerators

A. Load-Store PEs

In common with all of the soft processors surveyed in Section II, the load-store FPE supports only register–register and immediate instructions; this means that all nonconstant operands and results access the ALU via RF. Consider the effect of this requirement in the context of a 256-point FFT (FFT_{256}) realized via two FPE configurations: an eight-way FPE SIMD (FPE_8) or an MIMD multi-FPE composed of eight SISD FPEs (8-FPE).[2] The FFT mappings and the itemized ALU, interprocessor communication, memory (MEM), and NOP instructions for each are shown in Fig. 4.

[2]Numeric prefix denotes the number of processors, with the subscript denoting number of lanes; 1 is assumed if either is absent.

TABLE I
256-Point FFT Performance/Cost Comparison

	Cost		T	T/LUT
	LUTs	DSP48E	(MSamples/s)	($\times 10^3$)
8-FPE_1	2,296	8	30.5	13.3
Xilinx	621	6	61.9	99.7

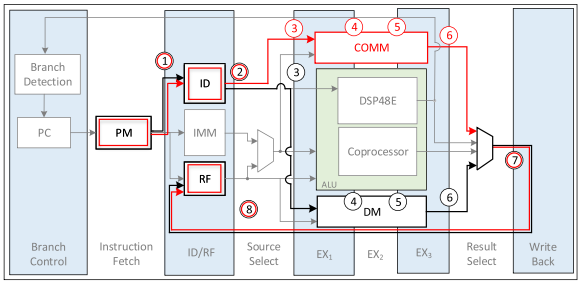


Fig. 5. Load-store paths in the FPE.

As shown in Fig. 4, the efficiency of each of these programs is low—only 52.5% and 31.8% of the respective cycles in 8-FPE_1 and FPE_8 are used for ALU instructions. The resulting effect on accelerator performance and cost is clear from Table I, which compares 8-FPE_1 with the Xilinx Core Generator FFT [26] component. As this shows, for this operation, the FPE is not competitive with the custom circuit Xilinx FFT, which exhibits twice the performance at a fraction of the LUT cost.

These results are a direct consequence of permitting only register–register instructions. Each FFT_{256} stage, and hence each FPE iteration, consumes/produces 512 complex words. Since RF is the most resource-costly element of the FPE, buffering this volume of data requires BRAM DM; in order for these operands to be processed and results stored, a large number of loads (stores) are required between BRAM and RF. Given the simplicity of the FFT butterfly operation, the proportion of the program occupied by these instructions is significant. However, in the context of the FPE, the situation worsens still: since the FPE is standalone and handles its own communication, further cycles are consumed transferring incoming and outgoing data between DM and COMM, reducing program efficiency still further. Finally, each of these transfers induces a latency between source and destination—as shown in Fig. 5, each FPE DM-RF (black) and COMM-RF (red) transfer takes eight cycles, imposing the need for NOP instructions. Ultimately, these factors combine to severely limit the efficiency of the FPE.

This situation is not unique to the FPE—all of the processors surveyed in Section II adopt load-store architectures and accordingly all suffer the same limitation. In order to overcome this inefficiency, two properties should be supported.

1) Direct instruction access to any combination of RF, DM, and COMM for either instruction source or destination.
2) In cases where local buffering is not required, data streaming through the PE should be enabled.

The design challenge is to enable these features without compromising on the resource efficiency required to compete with custom circuitry.

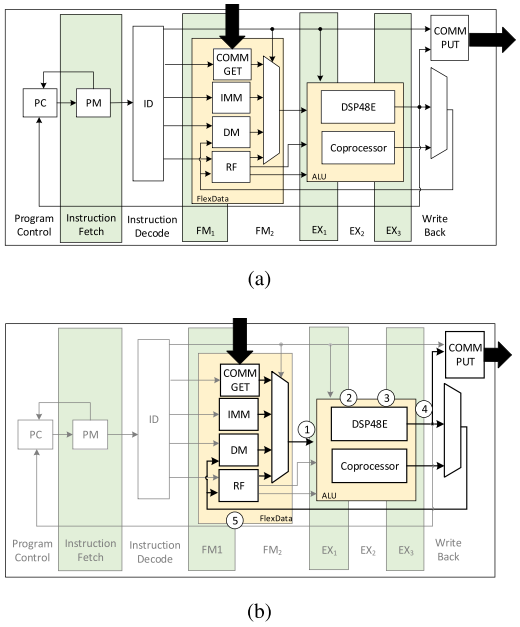


Fig. 6. SAE SISD architecture and ALUE access paths. (a) SISD SAE_1 architecture. (b) SAE ALU operand access paths.

B. Stream Accelerator Elements

To support these streaming features, a novel SAE is proposed. The SAE maintains standalone behavior and a software-programmable lean architecture, but supports advanced data streaming—i.e., the ability to stream data into and out of operation sources and destinations and through the ALU without the need for load and store cycles. This streaming takes two forms.

1) *Internal:* Peer access to RF, DM, COMM, and IMM without the need for load-store cycles.
2) *External:* Unbuffered streaming of data from input FIFOs to output FIFOs via only ALU.

The architecture of an SISD SAE_1 is shown in Fig. 6. There are three main architectural features of note:

1) the dedication of an entire pipeline stage to ID;
2) the FlexData data manager;
3) decoupled off-SAE read (COMMGET) and write (COMMPUT) components.

In the SAE, ID and FlexData dominate full pipeline stages. The ID determines the source/destination of any instruction operand/result, with all of the potential sources or destinations of data incorporated in FlexData to allow each to be addressed with equal latency; this flat memory architecture is unique to the SAE and distinct from that employed by any other soft processor. Its effect is to reduce the complexity of accessing each of the distinct operand sources via a regular dataflow. If these were not in the same pipeline stage, instruction decode and pipeline management would be substantially complicated to align the data arriving at the ALU with variable latency. As a result, data operands and results may be sourced/produced to any of IMM, RF, DM, or COMM with identical pipeline control and without the need for explicit load and store cycles or instructions for DM or COMM.

TABLE II
ALU OPERAND/DESTINATION INSTRUCTION CODING

Op	Source/Sink	x
`Rx`	RF	Register location
`&x`	DM	DM address
`^x`	COMMGET/COMMPUT	IPC channel no.
`x`	IMM	Constant value

In addition, in order to allow unbuffered streaming operation from input FIFOs to output FIFOs via ALU, simultaneous read/write to external FIFOs is required, with direct access to ALU in both directions. In order to support this capability, decoupled COMMGET and COMMPUT components are deployed in the SAE within FlexData. Note that these both reside in the same pipeline stage and, hence, conform to the regular dataflow pipeline maintained across the remainder of FlexData. In addition, since all of COMMGET, COMMPUT, DM, RF, and IMM access distinct memory resources (with separate memory banks employed within the SAE and an FIFO employed per off-SAE communication channel), there is no memory bandwidth bottleneck resulting from decoupling these accesses in this way—all could be accessed, simultaneously if needed. SAE operand and result read/write cycles are shown in Fig. 6(b).

C. Instruction Coding

To allow input (output) of data from (to) the appropriate source (destination), both the physical source component (RF, COMMGET, COMMPUT, DM, and IMM) and the appropriate addresses within each (i.e., memory location or communication channel) have to be relayed within the instruction. To accommodate this, SAE ALU instructions are expressed in the following format:

```
INSTR dest, opA, opB, opC
```

where `INSTR` is the instruction class, `dest` identifies the result destination, and `opA`, `opB`, and `opC` identify the source operands. The possible encodings of each of `dest`, `opA`, `opB`, `opC`, and the destination are described in Table II.

As described in Table II, all of RF, DM, COMMGET, and COMMPUT are addressed directly via the absolute addresses of the source/sink registers, memory locations, or external channel, respectively. Constant operands are hard-coded into the instruction and IMM locations allocated by the assembler. The sizes of the address fields in the final instructions are dynamically generated to match the configuration of the processor and program—i.e., five bits are assigned for register location for a 32-element RF, six bits for a 64-element RF, and so on. Instruction fields for RF, DM, COMMGET, and COMMPUT addresses are similarly determined at compile-time by the SAE assembler.

D. Configuration

The highly customizable nature of the FPE is maintained in the SAE, with the addition of parameters specific to the configurable FlexData. The sizes of FlexData's constituent

TABLE III
FLEXDATA CONFIGURATION PARAMETERS

Parameter	Meaning	Values
`data_ws`	Data Wordsize	16, 32
`imm_depth`	IMM Capacity	$\mathbb{N} \in [1, 2^{32} - 1]$
`dm_depth`	DM Locations	$\mathbb{N} \in [1, 2^{32} - 1]$
`rf_depth`	RF Capacity	$\mathbb{N} \in [1, 2^{32} - 1]$
`pp_depth`	Pipeline Depth	0, 1, 2
`dm_thr`	DM Threshold	$\mathbb{N} \in [1, 2^{32} - 1]$

components can be defined presynthesis such to enable operation-specific cost optimization via the configuration parameters described in Table III.

These configuration parameters enable substantial customization: `data_ws` controls the data word size for the SAE, while the depth of each the IMM, DM, and RF is set by `imm_depth`, `dm_depth`, and `rf_depth`, respectively. In addition, the number of physical delay cycles inserted by FlexData is defined pre-synthesis via `pp_depth`: any of zero, one or two cycles may be adopted.

There are two implicit issues of note in regard to configuration parameters. In the case where any of `imm_depth`, `dm_depth`, and `rf_depth` are zero, the associated component (IMM, DM, and RF) is absent from the synthesized version of FlexData; this allows the absolute minimum set of resources required to realize a given operation to be realized, minimizing cost, with the added benefit of reducing the size of the FlexData multiplexer whose size is configurable according to the same parameters.

Furthermore, the nature of individual components can change. For instance, the DM component is configured to allow realization as either distributed RAM (DisRAM) realized in the FPGA programmable logic LUTs, or by using the dedicated on-chip BRAM. The threshold for this decision is configurable as `dm_thr`. Specifically, if the DM capacity does not exceed `dm_thr`, then it will be realized using DisRAM; otherwise, it will be realized using BRAM. This configurability allows this substantial architectural decision to be made in an application specific manner. For the remainder of this paper, this threshold is taken to be 256 words.

The SAE forms the basic building block of large-scale streaming multicore architectures in a manner similar to that of the FPE—it is a configurable-width SIMD with direct external communication capability from which networks may be composed via FIFO queues.

E. SAE-Based FFT

To illustrate the effect of the streaming architecture on program size and efficiency, consider SAE-based FFT_{256} in the same configurations analyzed in Section IV-A. Fig. 7 itemizes the program instructions for both 8-SAE and SAE_8 and compares with those of the FPE.

As shown in Fig. 7(b), the streaming capabilities of the SAE enable realizations which are much superior to the FPE. In MIMD 8-SAE form, the total number of instructions required is 257, a decrease of ∼91%. In addition, the

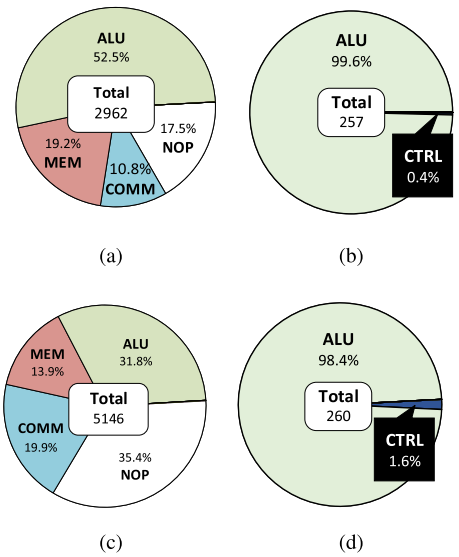


Fig. 7. FFT_{256}: FPE and SAE comparison. (a) 8-FPE_1. (b) 8-SAE_1. (c) FPE_8. (d) SAE_8.

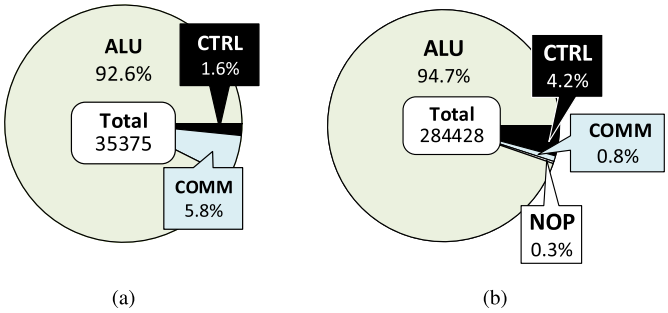


Fig. 8. Itemized SAE MM and motion estimation (ME) operations. (a) MM. (b) Motion estimation.

efficiency of this realization is now 99.6%, with only a single non-ALU instruction required for control. Similarly, SAE_8 requires 95.9% fewer instructions and operates with an efficiency of 98.4%. Given these metrics, it is reasonable to anticipate increases in throughput for 8-SAE and SAE_8 by factors of 20 and 30 over their FPE counterparts. Section VI-A measures the absolute performance and cost of SAE-based FFT accelerators and compares these with custom circuit architectures.

V. STREAMING BLOCK PROCESSING

The efficiency increases resulting from the streaming nature of the SAE are highly encouraging, but in many operations, addressing modes other than simple direct addressing are vital; for instance, an itemized instruction breakdown for the multiplication of two 32×32 matrices and full-search ME (FS-ME) with a 16×16 macroblock on a 32×32 search window are shown in Fig. 8.

These report the same high efficiency as the FFT detailed in Section IV-E, but also extremely large programs—35 375 instructions for MM and 284 428 for FS-ME. This places a heavy demand on FPGA memory resources for PM—in the case of FS-ME, this would require 241 BRAMs for

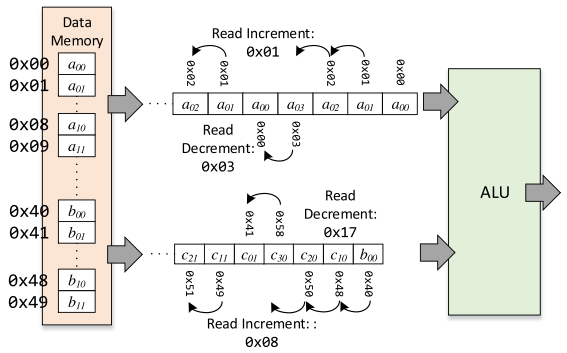


Fig. 9. SPE block matrix multiply operand addressing.

```
repeat(k; M){                                          1
 repeat(j; P){                                         2
     repeat(l; N){                                     3
        C[k][j]+=A[k][l]*B[l][j];                      4
      }      }     }   }                               5
```

Listing. 1. Streaming MM C code.

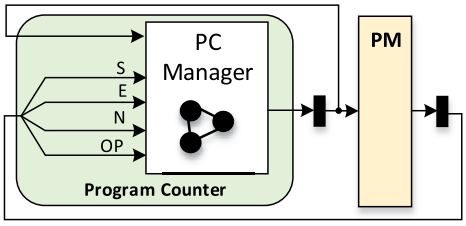
PM storage alone. These extreme sizes follow from the restriction to direct addressing, which dictates that the number of instructions is bounded below, by the number of ALU operations; for MM and ME, this translates a very large number of instructions.

Both MM and FS-ME perform a given operation many times, repeatedly, on small subsets of the input data at regularly spaced memory locations leading to highly repetitive behavior on regularly spaced memory locations. For instance, consider block-MM of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ when $m = n = p = 8$ via four 4×4 submatrices. Assuming that A and B are stored in DM contiguously and in row-major order and that C is derived in row-major order, and the operand memory access is shown in Fig. 9.

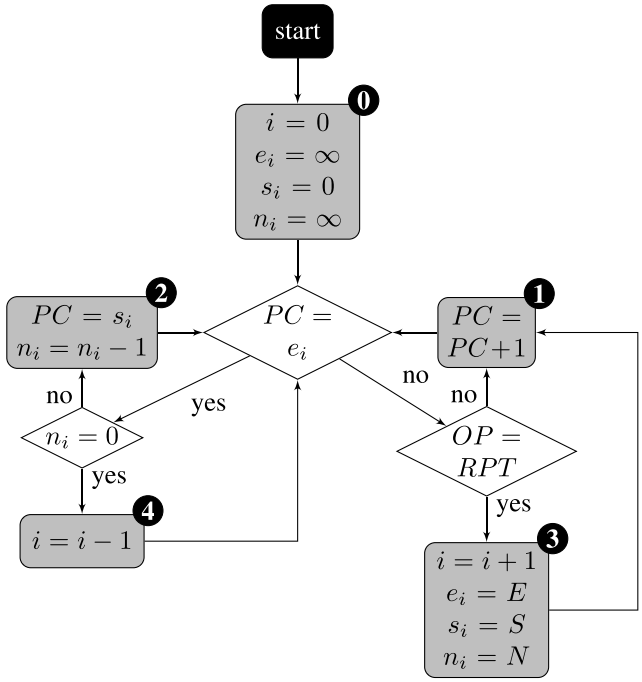
To compute an element of a submatrix of C, the inner product of a four-element vector of contiguous locations in A (a row of the submatrix) and a four-element vector of elements spaced by eight locations in B (a column of the submatrix) is formed. Afterward, either or both of the row of A or column of B are incremented to derive the next element of C, before the operation proceeds to the next submatrix. The resulting memory accesses are highly predictable: a regular repeated increment along the rows of A and the columns of B and periodic realignment to a new row of A and/or column of B repeated multiple times before realigning for subsequent submatrices.

This behavior is compactly represented using a loop-based code—for example, MM pseudocode is given in Listing 1. Each `repeat` operation realizes a predefined number of iterations over its body, with affine combinations of the iterators `j`, `k`, or `l` indexing the operands.

To support this highly compact expression of behavior for operations, such as MM and FS-ME, the SAE needs to combine two facilities: `repeat`-style behavior with the ability for a single instruction to address blocks of memory



(a)



(b)

Fig. 10. SAE loop manager elements. (a) PC architecture. (b) PCM behavior.

at regularly-spaced locations when invoked multiple times by a `repeat`. While repeat-type instructions are evident in conventional processors, there is no record of a softcore processor for FPGA which realizes these capabilities and as such their realization within the stringent cost constraints of the FPGA accelerators.

A. Zero-Overhead Loop Execution

In order to support low overhead loop operation, the SAE is augmented with the ability to perform `repeat`-type behavior. This means managing the PC, such that in the event of such an instruction, the body of the repeat statement is executed a number of times. This task is fulfilled by a PC manager (PCM). The structure of the SAE PC and PCM and the behavior of the PCM are shown in Fig. 10.

The PCM controls the update of the PC given its previous value and the instruction referenced in PM given pieces of information—the start and end lines of the body statements to be repeated `S` and `E`, the number of repetitions `N`. These are encoded in an `RPT` instruction added to the SAE instruction set. These instructions are encoded as

```
RPT N S E.
```

```
RPT 5 2 4                    1
  INSTR1...                  2
  INSTR2...                  3
  INSTR3...                  4
```

Listing. 2. RPT instruction coding.

TABLE IV
PC CONFIGURATION PARAMETERS

Parameter	Meaning	Values
pcm_en	Enable/disable PCM	boolean
pcm_depth	Max. repeat nest depth	$\mathbb{N} \in [1, 2^{32}-1]$

The intended use of RPT is shown in Listing 2, which dictates five repetitions of lines 2–5. Any number of repeat instructions can be nested to allow efficient execution of loop nests with static and compile-time known loop bounds.

The PCM arbitrates the PC to ensure the correct number of repetitions of the body statement and to support the construction of nested repeat operations by enacting the flowchart in Fig. 10. Specifically, for an n-level nest, it maintains $n+1$-element lists of metrics, with an additional element added to support infinite repetition of the top-level program, considered to be an implicit infinite repeat instruction. For layer i of the loop nest, the start line, end line, and number of repetitions are stored in element $i+1$ of the lists s, e, and n, respectively. In all cases $s_0 = 0$, $e_0 = \infty$ and $n_0 = \infty$ to represent the start line, end line, and number of repetitions of the top-level program [❶ in Fig. 10(b)].[3] Every time a repeat instruction is encountered i, the current index into s, e, and n is incremented and the values of the new element initialized using S, E, and N from the decoded instruction in ❸. Regular PC updating then proceeds (❶) until either another repeat instruction is detected or until e_i is encountered. In the latter case, the number of iterations of the current statement is decremented (❷), or if $n_i = 0$, all of the iterations of the current repeat statement have been completed and control of the loop nest reverts to the previous level (❹).

The PCM operation described in Fig. 10 was realized using behavioral VHDL and synthesized. The cost of the basic PCM component is 36 LUTs, a cost which should only be accepted in cases where it can enable a substantial cost/performance benefit, for instance when it can substantially reduce program size and, therefore, PM cost, with the saving preferably outweighing the cost of the PCM. To allow minimum cost for every application, the PC is configurable via the parameters listed in Table IV.

As shown in Table IV, pcm_en dictates whether the PCM is included in the synthesized architecture or otherwise it takes a Boolean value. In the case where a PCM is included, the maximum depth of loop nest is configurable via pcm_en which can take, hypothetically, any value. As such, the PCM may be included or excluded and, hence, imposes no cost when it is not required; furthermore, when it is included, its cost can be tuned to the application at hand by adjusting the maximum depth of loop nest.

[3]Note that this assumes that the end line of the program is a JMP instruction with the start line as the target.

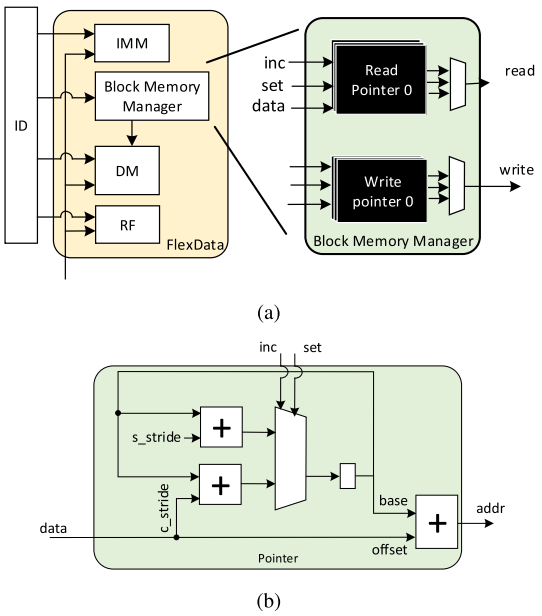


Fig. 11. SAE block memory management elements. (a) SAE FlexData. (b) Pointer architecture.

B. Block Data Memory Access

To enable block memory access capability, three important capabilities are required:

1) autoincrement with any constant stride;
2) manual increment with any stride;
3) custom offset.

Each of these behaviours is evident in MM: auto-increment traverses along rows and colmuns with a fixed stride. There are many such operations and so eliminating the need for an individual instruction for each reduces the overall instruction count considerably. Manual increment is required for movement between rows/columns, while custom offset is used to identify the starting point for the increments, such as the first element of a submatrix.

To enable these capabilities, a block memory manager (BMM) is incorporated in the SAE FlexData, as shown in Fig. 11. As shown, the BMM arbitrates access to DM via read pointers (RPs) and write pointers (WPs). The architecture of FlexData and a pointer is shown in Fig. 11(b).

Each pointer controls access to a subset (block) of the SAE DM and addresses individual elements of that block via a combination of two subaddress elements as follows.

1) *Offset* selects the root block data element.
2) *Base* iterates over elements relative to the offset.

A pointer operates in one of three modes. Either the base autoincrements, or it is incremented by explicit instruction, or the offset increments by explicit instruction. All three modes are supported under the control of the set, inc, and data interfaces. The offset is used to select the root data element of the submatrices of A, B, and C, with the base added to address elements relative to the offset. The base is updated via two mechanisms, under the control of inc. The first autoincrements by a value [s_stride in Fig. 11(b)], the

TABLE V
BMM CONFIGURATION PARAMETERS

Parameter	Meaning	Values
mode	Addressing mode	direct, block
n_rptrs / n_wptrs	No. of read /write pointers	$\mathbb{N} \in [1, 2^{32}]$
s_stride	Constant stride	$\mathbb{N} \in [1, 2^{32}]$

TABLE VI
BMM INSTRUCTIONS

Operand Field	Meaning
INC_RP / INC_WP	Increment base of RP / WP n to val
SET_RP /SET_WP	Set offset of RP / WP n to val

TABLE VII
ALU BLOCK OPERAND INSTRUCTION CODING

Operand Field	ofs	idx	!
Meaning	Offset	Pointer reference	Autoincrement base

value of which is set as a constant presynthesis. To enable manual increment of the base, c_stride is set via the data signal. Finally, when update of the offset is required, the value of data is accepted on the assertion of set.

The BMM and pointer architectures were expressed in behavioral VHDL and synthesized, with a total resulting cost of 40 LUTs per pointer. In common with the PCM, this is a substantial overhead and one which should be incurred only when the performance/cost benefit makes it sustainable. To allow absolute minimum cost for any operation, the configuration parameters for the SAE FlexData, BMM, and pointer components are included as listed in Table V.

As shown in Table V, addressing mode is now a configuration parameter of the SAE, with direct and block modes supported. In the former, the BMM is absent and imposes no cost on synthesis, while it is included in the latter. In the case of block addressing mode, the cost can be tuned to the application at hand by defining the number of RP and WP via n_rptrs and n_rptrs. Finally, the autoincrement stride s_stride for each pointer is fixed at the point of synthesis.

To support custom increment of the base and offset for each pointer, the SAE instruction set is updated to BMM instructions of the form

INSTR n val

where n denotes the pointer to which the instruction is to be applied. The permitted values of INSTR and the corresponding behavior are as described in Table VI.

To control the extra functionality enabled by the BMM, ALU operands accessing DM have an alternate encoding, which takes the form

&<ofs><idx><!>.

This encoding is elaborated in Table VII.

C. Off-SAE Communications

The COMMGET and COMMPUT components are shown in Fig. 12, with the aspects in which both are configurable pre-synthesis summarized in Table VIII.

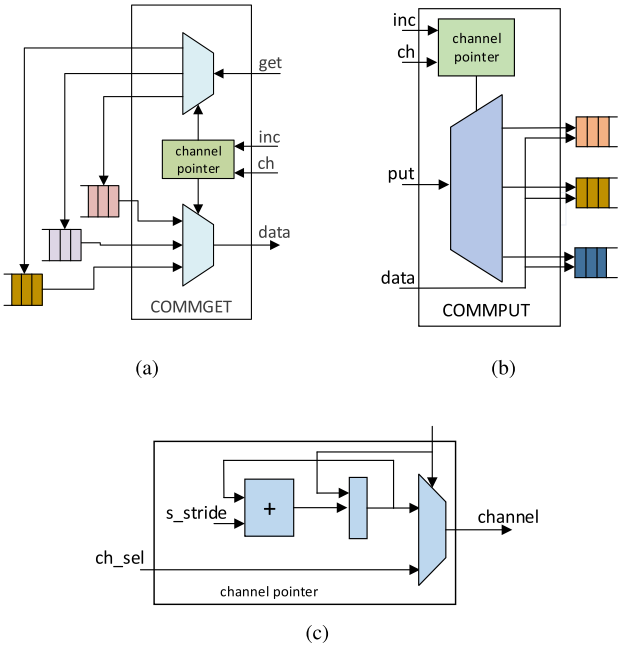


Fig. 12. SAE COMM adapters. (a) COMMGET. (b) COMMPUT. (c) COMM pointer.

TABLE VIII
COMM CONFIGURATION PARAMETERS

Parameter	Meaning	Values
mode	Addressing mode	direct, block
n_chan	No. channels	$\mathbb{N} \in [1, 64)$
s_stride	Constant stride	$\mathbb{N} \in [1, 64)$

As shown in Fig. 12, each of COMMGET and COMMPUT can operate under direct and block addressing modes as determined via the mode parameter. In direct mode, individual FIFO channels via addresses encoded within the instruction. Instructions for either COMM unit are encoded as

^<p><ofs/idx><!>

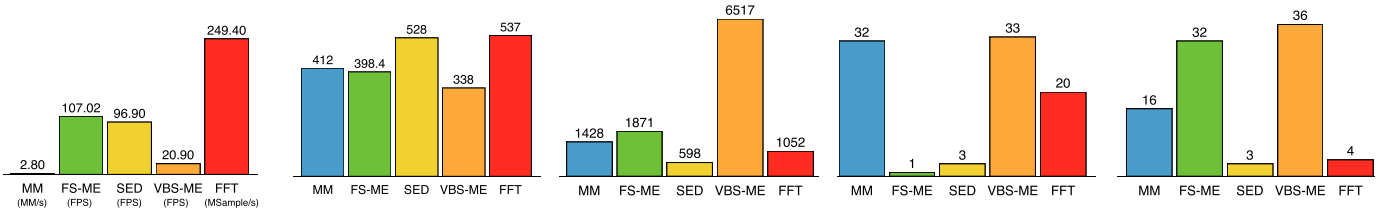
where p differentiates peek (read-without-destroying) and get (read-and-destroy) operations, ofs denotes the offset, idx denotes the pointer reference, and ! denotes the autoincrement flags.

D. Stream Frame Processing Efficiency

Consider the impact of adopting the repeat and block memory access facilities of the SAE on the realizations of MM and FS-ME detailed previously. The number of instructions in each category for the direct (SAE) and block-based (SAE-B) SAE modes is described in Table IX.

As shown in Table IX, the substantial reductions in program size have resulted, with SAE-B requiring fewer than 1% of the number of instructions required by SAE. As such, adopting a stream processing model and enabling advanced program control and memory addressing have had a clear beneficial effect on program efficiency and scale.

It should be noted that the only side effect of employing zero-overhead loop execution and/or block DM accesses is



(a) (b) (c) (d) (e)

Fig. 13. SAE accelerators. (a) T. (b) Clk (MHz). (c) LUTs. (d) DSP48e. (e) BRAM.

TABLE IX
SAE-BASED MM AND ME: ITEMIZED PM

Class	Matrix Multiply			Motion Estimation		
	SAE	SAE-B	δ (%)	SAE	SAE-B	δ (%)
ALU	32768	32	**-99.9**	268353	26	**-99.9**
COMM	2048	6	**-99.7**	2467	14	**-99.4**
CTRL	559	4	**-99.7**	12582	12	**-99.9**
NOP	0	6		1026	6	**-99.6**
Total	35375	54	**-99.8**	284428	58	**-99.9**

TABLE X
SAE-BASED ACCELERATOR CONFIGURATIONS

	MM	FS-ME	SED	VBS-ME	FFT
Config.	SAE_8	SAE_{32}	3-SAE_3	2-SAE_{32}+SAE	5-SAE
`data_ws`	32	16	16	16	16
`data_type`	real	real	real	real	complex
`dm_depth`	1024	1009	1800	[1024,82,32]	[0,32,32, 128,512]
`pm_depth`	64	64	113	[64,200,32]	[68,78,190, 758,1949]
`rf_depth`	0	0	32	[0,32,32]	0
`n_rptrs`	2	2	1	[2,1,0]	1
`n_wptrs`	1	1	1	[1,1,0]	1

that of the extra resource cost required to integrate these components within the SAE. There is no effect on the timing or behavior of a program beyond that made explicit in the program by deploying `INC_RP`, `INC_WP`, `SET_RP`, or `SET_WP` in the program instruction stream. For instance, in a typical indirect addressing scheme, the use of an indirect addressing instruction implies lookup of another value from, for example, a specific register or memory address, implying multicycle latency. This scheme, in contrast, contains all information required to properly handle `RPT` instructions, and block memory accesses are either available in the instruction or locally in the handling PCM or BMM units. No multicycle or variable latency instructions are required.

Section VI quantifies and compares the SAE-based accelerators for a number of typical signal and image processing operations against real-time performance criteria and custom circuit and soft processor alternatives.

VI. EXPERIMENTS

SAE-based accelerators for five typical signal and image processing operations were created:

1) 512-point FFT;
2) 1024×1024 MM;
3) SED on 1280×768 image frames;
4) FS-ME with 16×16 macroblock and 32×32 search window on common intermediate format (CIF) 352×288 images;
5) variable block size ME (VBS-ME) with 16×16 macroblock and 32×32 search window on CIF 720×480 images.

These applications have been chosen to gauge the ability of the SAE configurations for each of these accelerators is detailed in Table X, which enable real-time performance for communications and video standards (FFT, FS-ME, and VBS-ME), and to allow objective comparison with fixed-function custom circuits (FFT). The SAE assembler has been updated to support all of the new instructions detailed in Sections IV and V. All accelerators target Xilinx Kintex-7 XC7K70TFBG484 using Xilinx ISE 14.2; all quoted performance and cost results are post-place-and-route.

These configurations reveal much about the SAE; particularly notable is the absence of any RF component in MM, FS-ME, and FFT, a very substantial saving enabled directly by the dual-streaming nature of the SAE exploiting only COMM and DM components. The wide flexibility of the SAE enables a number of notable firsts in the performance and cost results in Fig. 13. Specifically, throughput for FS-ME is sufficiently high as to enable real-time performance for H.264, while VBS-ME can support real-time processing of 480p video in H.264 Level 2.2. To the best of our knowledge, this is the first time an FPGA-based software-programmable component has demonstrated this capability. These performance and cost metrics are compared with custom circuit and softcore realizations in the remainder of this section.

A. IP Comparison Case Study: FFT for 802.11ac

To objectively measure and compare the performance and cost of SAE-based accelerators relative to custom circuits, SAE-based FFTs for IEEE 802.11ac transmitters have been developed and compared with realizations using the Xilinx FFT and those generated by Spiral [27]. The IEEE 802.11ac standard [28] mandates eight-channel FFT operations on 20-, 40-, 80-, and 160-MHz frequency bands with FFT size and throughput requirements as outlined in Table XI.

The multi-SAE accelerator configurations are summarized in Table XII.[4, 5] The performance and cost of the

[4]In the case where more than one SAE is used, the configurations of each are presented in vector format.

[5]Note that FFT_{512} takes a different configuration to the 512-point FFT previously addressed.

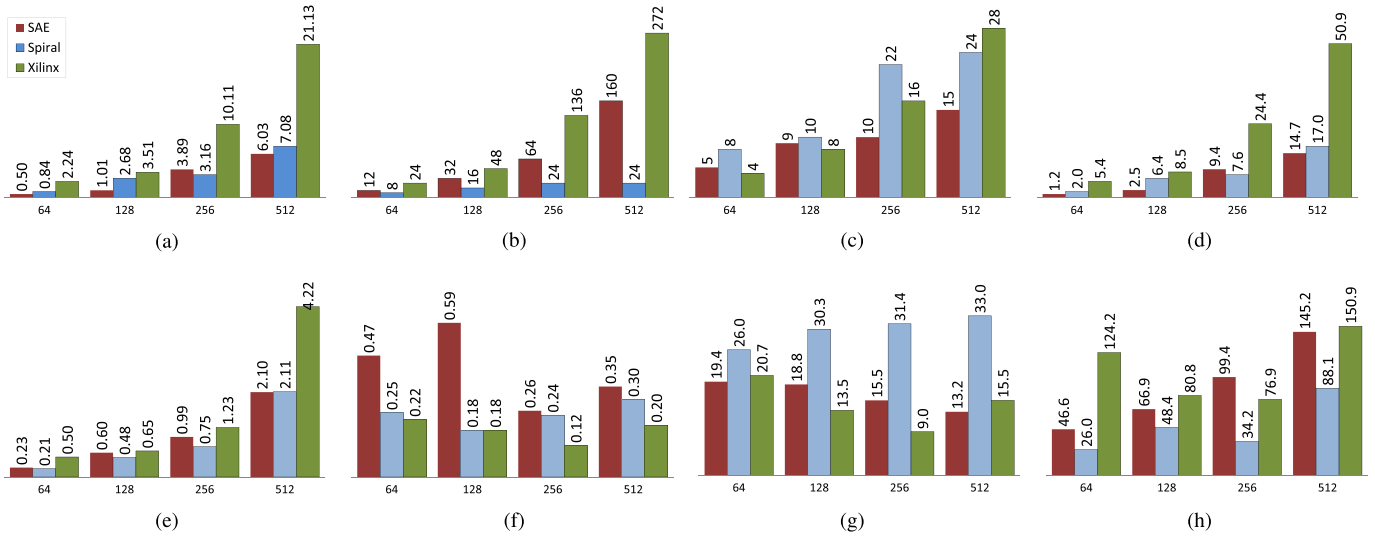


Fig. 14. FPGA-based FFT: performance and cost. (a) LUT cost ($\times 10^3$). (b) DSP48E cost. (c) BRAM cost. (d) % Device occupied. (e) T ($\times 10^9$ samples/s). (f) T/LUT ($\times 10^6$). (g) T/DSP48E ($\times 10^6$). (h) T/BRAM ($\times 10^6$).

TABLE XI
802.11ac FFT CHARACTERISTICS

Frequency (MHz)	20	40	80	160
FFT	64	128	256	512
Throughput ($\times 10^6$ Samples/s)	160	320	640	1280

TABLE XII
SAE FFT CONFIGURATIONS

Parameter	**FFT_{64}**	**FFT_{128}**	**FFT_{256}**	**FFT_{512}**
Config.	1-SAE_3	1-SAE_8	3-SAE_8	5-SAE_8
data_ws			16	
data_type			complex	
dm_depth	192	128	[32,256]	[0,32,32,128,512]
pm_depth	1184	902	[134,1852]	[68,78,190,758,1949]
rf_depth			0	
sm_depths	32	64	[32,128]	[0,32,32,64,256]

resulting architectures are described in Fig. 14. The LUT, DSP48E, BRAM, and overall device occupation costs are given in Fig. 14(a)–(d), while throughput T and resource efficiency are shown in Fig. 14(e) and Fig. 14(f)–(h), respectively.

Fig. 14 shows that the SAE FFT accelerators for 802.11ac, supported by the clock rates of 528 MHz (FFT_{64} and FFT_{128}), 506 MHz (FFT_{256}), and 512 MHz (FFT_{512}), the real-time throughput requirements listed in Table XI are satisfied. In addition, the performance and cost are highly competitive with the Xilinx and Spiral custom circuits. As shown in Fig. 14(a)–(c), the LUT, DSP48E, and BRAM costs are lower than the Xilinx FFT in 9 out of 12 cases, with the savings of up to 69%, 53%, and 56%. When measured as the proportion of device occupied (Fig. 14), the SAE is up to 36% smaller. The Xilinx FFT offers consistently higher throughput but at disproportionately greater cost, with SAE resource efficiency superior in 7 out of 12 cases in Fig. 14(f)–(h); increases in LUT, DSP48E, and BRAM efficiency exceed the factors of 3.2, 1.7, and 1.3.

Relative to the Spiral FFT, the performance and cost of the SAE accelerators are similarly encouraging, enabling increased throughput in all but one case and reduced LUT and BRAM costs in 7 out of 8 cases; savings reaching 62.8% and 55%, respectively. The Spiral FFTs have consistently lower DSP48E cost; however, the total proportion of the device occupied by each, reported in Fig. 14, remains in favor of the SAE in all but one instance. These metrics describe consistently more efficient use of LUT and BRAM by the SAE by factors of up to 3.3 and 2.9, respectively; although Spiral is consistently more efficient in its use of DSP48E, the SAE still exhibits superior efficiency in 8 out of 12 cases.

These are highly encouraging results, repeated when compared with commercial FFT components—e.g., the 6.4-Gsample/s 512-point FFT in [29] exhibits LUT, DSP48E, and BRAM efficiency metrics of 0.17, 22.2, and 100.2—in only the DSP48E case are these figures superior, with the SAE exhibiting substantially higher LUT and BRAM resource efficiency. The generally more favorable cost, performance, and efficiency metrics in all the alternatives surveyed here suggest that the SAE enables architectures, which are highly competitive with custom circuits. Given that the SAE is a software-programmable processor, this is a substantial breakthrough—indeed to even get within an order of magnitude is a unique result. To the best of our knowledge, this is the first record of a software-programmable components with this capability for FPGA, or indeed, any other technology.

B. Soft Processor Comparison

The performance and cost of SAE-based MM and FS-ME are compared with other soft processors in Figs. 15 and 16.

When applied to MM, the performance and cost advantages relative to 32-way VEGAS ($VEGAS_{32}$) [15] and 4-way VENICE ($VENICE_4$) [16] are clear. Relative to $VEGAS_{32}$, throughput is increased by a factor of 2, despite requiring only 25% of the number of datapath lanes. The accompanying LUT, DSP48E, and BRAM cost reductions of 96%, 75%, and 50% lead to LUT, DSP48E, and BRAM efficiency increases by factors of 66.7, 8.8, and 4.5, respectively. Compared with $VENICE_4$, throughput is increased by

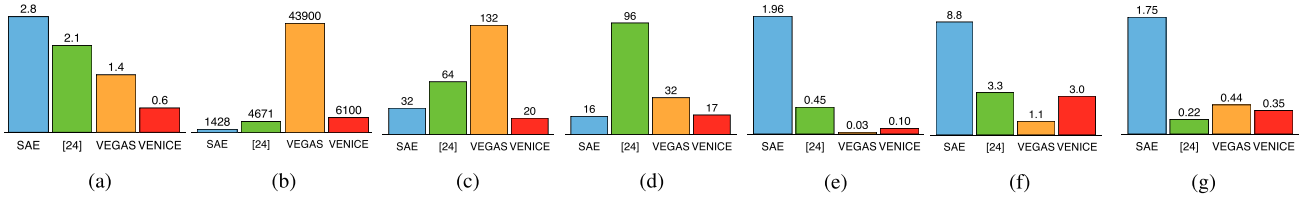


Fig. 15. Softcore MM: performance and cost comparison. (a) T (MM/s). (b) LUTs. (c) DSP48E. (d) BRAM. (e) T/LUT ($\times 10^{-3}$). (f) T/DSP ($\times 10^{-3}$). (g) T/RAM ($\times 10^{-3}$).

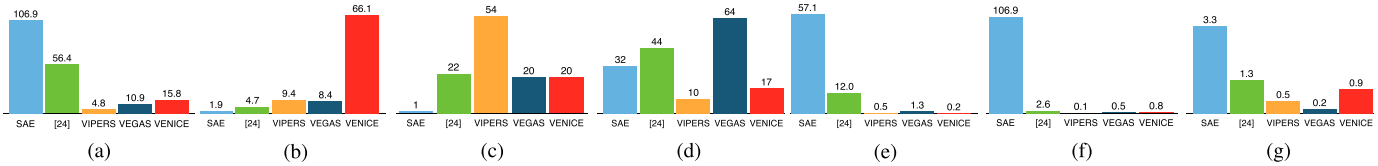


Fig. 16. Softcore FS-ME: performance and cost comparison. (a) T (frames/s). (b) LUTs ($\times 10^3$). (c) DSP48E. (d) BRAM. (e) T/LUT ($\times 10^{-3}$). (f) T/DSP48E. (g) T/BRAM.

a factor of 4.7, while LUT and BRAM costs are reduced by 76% and 5%, respectively; correspondingly, the efficiency of LUT, DSP48E, and BRAM exploitation reaches the factors of 20, 2.9, and 5.1, respectively. The increased throughput over [24] is accompanied by LUT, DSP48E, and BRAM resource cost reductions by 69.4%, 50%, and 83.3% with efficiency increased by factors of 5, 2.7, and 9, respectively.

SAE-based ME is compared with the VIPERS_{16}, the VEGAS_4 and VENICE_4, and the FPE in Fig. 16. SAE_{32} is the only realization capable of supporting the 30-frames/s throughput requirement for standards, such as H.264, with absolute throughput increased by factors of 22.3, 9.8, and 6.8 relative to VIPERS_{16}, VEGAS_4, and VENICE_4. This increased performance is accompanied by very strong reductions in resource cost. LUT and DSP48E costs are reduced by 80.1% and 98.1% relative to VIPERS_{16} increasing LUT, DSP48E, and BRAM efficiency by factors of 112, 1203.9, and 7, respectively. Similarly, LUT, DSP48E, and BRAM costs are reduced by 77.7%, 95%, and 50% relative to VEGAS_4, with efficiency increased by factors of 44.1, 196.4, and 19.6. Finally, relative to VENICE_4, LUT and DSP48E costs are reduced by 97.2% and 95%, indicating efficiency increases in LUT, DSP48E, and BRAM resources by factors of 239.3, 135.5, and 3.6, respectively. In addition these performance and cost metrics are well beyond those of the early works presented in [24], with throughput increased by a factor of 1.9 with associated LUT, DSP48E, and BRAM costs reduced by 60.2%, 95.5%, and 27.3% leading to efficiency increases by factors of 4.8, 41.7, and 2.6, respectively. As a reference, Li and Leong [30] quote an FS-ME circuit for Xilinx Virtex-II FPGA with T/LUT of 0.053.

While it is very difficult to precisely directly compare the SAE approach with that of VIPERS, VESPA, or VEGAS owing to differences in the FPGA technologies device generations,[6] it is clear that very substantial cost and performance benefits have accrued from the use of SAE. Similarly, assuming performance and cost scale linearly with matrix size, the TCPA described in [20] would require ~21 000 LUTs for a 4×4 array, which would support the multiplication of 1024×1024 at ~8 MM/s. The resulting T/LUT metric of 3.81×10^{-4} is almost an order of magnitude lower than that of the SAE.

These results demonstrate the benefit of the SAE relative to other soft processors—coupled performance/cost increases of up to three orders of magnitude. Of course, the softcores to which the SAE is compared here are general-purpose components and, hence, offer substantially greater run-time processing capability than the SAE, which is highly tuned to the operation for which it was created. In that respect, the SAE is more a component for constructing fixed-function accelerators than a general-purpose softcore. However, despite employing similar multilane processing approaches as VIPERS, VEGAS, and VENICE, the SAE's focus on extreme efficiency, multicore processing, stream processing, and novel block memory management has enabled very substantial performance and cost benefits.

VII. Conclusion and Future Work

Soft processors for FPGA suffer from substantial cost and performance penalties relative to custom circuits handcrafted at register transfer level. Performance and resource overheads associated with the need for a host general-purpose processor, load-store processing, loop handling, addressing mode restrictions, and inefficient architectures combine to amplify cost and limit performance.

This paper describes the first approach, which challenges this convention. The SAE presented realizes accelerators using multicore networks of fine-grained, high-performance, and standalone processors. The SAE enables performance and cost unprecedented among soft processors by adopting a streaming operation model to ensure high efficiency combined with advanced loop handling and addressing constructs for very compact and high-performance operation on large data sets. These enable efficiency routinely in excess of 90% and performance and cost which are comparable with custom circuit accelerators and well in advance of existing soft processors.

[6]VIPERS and VEGAS both exploit Altera Stratix III FPGA, while VEGAS exploits Altera Stratix IV.

Specifically, the real-time accelerators for 802.11ac FFT and H.264 FS-ME VBS-ME are described; the former of these exhibits performance and cost, which are highly competitive with custom circuits. In addition, it is shown how SAE-based MM and ME accelerators offer improvements in resource/cost by up to three orders of magnitude. To the best of our knowledge, these capabilities are unique, not only for FPGA, but for any semiconductor technology.

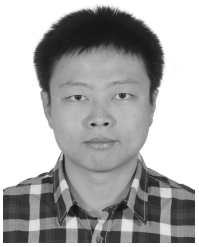
This paper lays a promising foundation for the construction of complete FPGA accelerators, but in addition may be used to further ease the design process. For example, in the case where off-chip memory access is required, the programmable nature of the SAE means that it may also be used as a memory controller to execute custom memory access schedules and highly efficient block access. However, resolving this and other accelerator peripheral functions is left as future work.

ACKNOWLEDGMENT

The authors would like to thank Dr. X. Chu and Dr. Y. Wu for their valuable input to this paper.

REFERENCES

[1] *7 Series DSP48E1 Slice User Guide*, Xilinx, Inc., San Jose, CA, USA, Aug. 2013.
[2] *Stratix V Device Handbook*, Altera, Inc., San Jose, CA, USA, Jan. 2014.
[3] *7 Series FPGAs Memory Resources User Guide*, Xilinx, Inc., San Jose, CA, USA, Jan. 2014.
[4] J. McAllister, "FPGA-based DSP," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. New York, NY, USA: Springer, 2010,
[5] B. Klauer, "The convey hybrid-core architecture," in *High-Performance Computing Using FPGAs*, W. Vanderbauwhede and K. Benkrid, Eds. New York, NY, USA: Springer, 2014, pp. 431–451.
[6] O. Pell, O. Mencer, K. H. Tsoi, and W. Luk, "Maximum performance computing with dataflow engines," in *High-Performance Computing Using FPGAs*, W. Vanderbauwhede and K. Benkrid, Eds. New York, NY, USA: Springer, 2014, pp. 747–774.
[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.
[8] A. Papakonstantinou, K. Gururaj, J. A. Stratton, D. Chen, J. Cong, and W.-M. W. Hwu, "FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs," in *Proc. IEEE 7th Symp. Appl. Specific Processors*, Jul. 2009, pp. 35–42.
[9] S. O. Settle, "High-performance dynamic programming on FPGAs with OpenCL," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2013, pp. 1–6.
[10] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core CPU accelerator," in *Proc. 16th Int. ACM/SIGDA Symp. Field Program. Gate Arrays (FPGA)*, 2008, pp. 222–232.
[11] *MicroBlaze Processor Reference Guide*, Xilinx, Inc., San Jose, CA, USA, Apr. 2014.
[12] *Nios II Processor Reference Handbook*, Altera, Inc., San Jose, CA, USA, Feb. 2014.
[13] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 2, Jun. 2009, Art. ID 12.
[14] P. Yiannacouras, J. G. Steffan, and J. Rose, "Portable, flexible, and scalable soft vector processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1429–1442, Aug. 2012.
[15] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. F. Lemieux, "VEGAS: Soft vector processor with scratchpad memory," in *Proc. 19th ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2011, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950420
[16] A. Severance and G. Lemieux, "VENICE: A compact vector processor for FPGA applications," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2012, pp. 261–268.
[17] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, "The iDEA DSP block-based soft processor for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 3, Aug. 2014, Art. ID 19.
[18] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer, "An FPGA-based soft multiprocessor system for IPv4 packet forwarding," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug. 2005, pp. 487–492.
[19] D. Unnikrishnan, J. Zhao, and R. Tessier, "Application specific customization and scalability of soft multiprocessors," in *Proc. 17th IEEE Symp. Field Program. Custom Comput. Mach. (FCCM)*, Apr. 2009, pp. 123–130.
[20] F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche, "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4s, Jul. 2014, Art. ID 133.
[21] R. W. Hartenstein, A. G. Hirschbiel, M. Riedmuller, K. Schmidt, and M. Weber, "A novel ASIC design approach based on a new machine paradigm," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, pp. 975–989, Jul. 1991.
[22] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP—A self-reconfigurable data processing architecture," *J. Supercomput.*, vol. 26, no. 2, pp. 167–184, 2003.
[23] X. Chu and J. McAllister, "Software-defined sphere decoding for FPGA-based MIMO detection," *IEEE Trans. Signal Process.*, vol. 60, no. 11, pp. 6017–6026, Nov. 2012.
[24] P. Wang and J. McAllister, "Soft-core stream processor for sliding window applications," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2013, pp. 213–218.
[25] P. Wang, J. McAllister, and Y. Wu, "Soft-core stream processing on FPGA: An FFT case study," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2013, pp. 2756–2760.
[26] *LogiCORE IP Fast Fourier Transform v7.1*, Xilinx, Inc., San Jose, CA, USA, Mar. 2011.
[27] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer generation of hardware for linear digital signal processing transforms," *ACM Trans. Design Autom. Electron. Syst.*, vol. 17, no. 2, Apr. 2012, Art. ID 15.
[28] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 4: Enhancements for Very High Throughput for Operation in Bands Below 6 GHz*, IEEE Standard P802.11ac/D2.2, 2012.
[29] *Dillon Engineering—Dual Parallel FFT*. [Online]. Available: http://www.dilloneng.com/dual-parallel-fft.html, accessed Dec. 1, 2015.
[30] B. M. H. Li and P. H. W. Leong, "Serial and parallel FPGA-based variable block size motion estimation processors," *J. Signal Process. Syst.*, vol. 51, no. 1, pp. 77–98, Apr. 2008.



Peng Wang received the B.S. degree in electronics engineering from Shandong University, Shandong, China, in 2008, the M.Sc. degree in instrumentation science from Beihang University, Beijing, China, in 2011, and the Ph.D. degree in electronics engineering from Queen's University Belfast, Belfast, U.K., in 2014.

He has been a Design Engineer with ARM, Cambridge, U.K., since 2014. His current research interests include processor architectures and signal processing.

John McAllister (S'02–M'04–SM'13) received the Ph.D. degree in electronics engineering from Queen's University Belfast, Belfast, U.K., in 2004.

He is currently a Senior Lecturer with Queen's University Belfast, where he leads a group of researchers in embedded architectures, and electronic system level design technologies for streaming applications, with a specific focus on field-programmable gate array targets. He is the Co-Founder of Analytics Engines Ltd., Belfast.

Dr. McAllister is a member of the IEEE Signal Processing Society and its Technical Committee on Design and Implementation of Signal Processing Systems. He is the Chief Editor of SigView, the IEEE SPS tutorial library, and an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING and the *Journal of Signal Processing Systems* (Springer). He serves on the program committees of a number of IEEE conferences, including the International Conference on Acoustics, Speech and Signal Processing and the Workshop on Signal Processing Systems: Design and Implementation.