Time-Multiplexed FPGA Overlay Architectures: A Survey

Article *in* ACM Transactions on Design Automation of Electronic Systems · July 2019
DOI: 10.1145/3339861

Xiangwei Li
The University of Sydney

Time-Multiplexed FPGA Overlay Architectures: A Survey

XIANGWEI LI and DOUGLAS L. MASKELL, Nanyang Technological University, Singapore

This article presents a comprehensive survey of time-multiplexed (TM) FPGA overlays from the research literature. These overlays are categorized based on their implementation into two groups: processor-based overlays, as their implementation follows that of conventional silicon-based microprocessors, and; CGRA-like overlays, with either an array of interconnected processor-based functional units or medium-grained arithmetic functional units. Time-multiplexing the overlay allows it to change its behavior with a cycle-by-cycle execution of the application kernel, thus allowing better sharing of the limited FPGA hardware resource. However, most TM overlays suffer from large resource overheads, due to either the underlying processor-like architecture (for processor-based overlays) or due to the routing array and instruction storage requirements (for CGRA-like overlays). Reducing the area overhead for CGRA-like overlays, specifically that required for the routing network, and better utilizing the hard macros in the target FPGA are active areas of research.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Hardware → Reconfigurable logic and FPGAs**;

Additional Key Words and Phrases: Reconfigurable system, FPGA overlay, time-multiplexing

ACM Reference format:
Xiangwei Li and Douglas L. Maskell. 2019. Time-Multiplexed FPGA Overlay Architectures: A Survey. *ACM Trans. Des. Autom. Electron. Syst.* 24, 5, Article 54 (July 2019), 19 pages.
https://doi.org/10.1145/3339861

1 INTRODUCTION

Modern FPGAs have seen a rapid growth in logic density along with the integration of CPU, GPU, and other hard silicon modules. To achieve the best accelerator performance, these FPGAs are often custom designed, using conventional RTL hardware design techniques, and as such, have only found mainstream applicability in specific applications such as digital signal processing and communications. This is because design productivity issues, particularly the difficulty of hardware design and the long compilation times, are major stumbling blocks to the widespread adoption of FPGA-based accelerators in general purpose computing [11, 33].

Traditionally, text-based hardware description languages (HDL) are used to define the behavior of the FPGA. However, getting the best performance from the HDL implementation still needs a good understanding of the target technology's capabilities and of basic hardware concepts such as pipelining and synchronization. Additionally, because of the fine granularity of the FPGA resource, design compilation time is significant. It takes hours or even days to compile a very large design

This work is supported by the Ministry of Education (MoE), Singapore under RG132/16 and MOE2017-T2-1-002.
Authors' addresses: X. Li and D. L. Maskell, Nanyang Technological University, 50 Nanyang Avenue, 639798, Singapore; emails: xli045@e.ntu.edu.sg, asdouglas@ntu.edu.sg.

1084-4309/2019/07-ART54 $15.00
https://doi.org/10.1145/3339861

due to the fine-grained placement and routing used in the FPGA implementation. Even for the case where just a few lines of HDL code change, the traditional FPGA CAD tools have to go through the whole process (including synthesis, mapping, placement, and routing) to generate a new bitstream to program the device. This design process greatly slows down the development progress of FPGA designs and, to some extent, hinders the widespread adoption of FPGAs.

High-level synthesis (HLS) has been widely adopted by EDA vendors to address some of the design productivity issues and provides a higher level of abstraction for the hardware, hiding much of the low-level detail. Typical HLS tools such as Xilinx Vivado HLS [21], Altera SDK for OpenCL [17], and LegUp [9] from the University of Toronto have been developed to interpret a high-level language description of a user application and convert it into low-level RTL. Using HLS tools, there is less of a requirement for hardware specialization as custom digital logic circuits can be generated automatically with high performance. However, while HLS techniques alleviate the design productivity problem to some extent, the back-end flow still requires very long compilation times, particularly for large designs, contributing to long design cycles and the lack of mainstream adoption of FPGAs by software designers who are used to rapid design iterations.

Because of these long design cycles, researchers have investigated other techniques for improving design productivity. One of these techniques is to use a virtual hardware representation which overlays the original FPGA fabric, referred to as an overlay architecture (or overlay).

This article is organised as follows: Section 2 gives a broad overview of FPGA overlays along with their advantages and disadvantages and classifies them, based on the run-time configurability, as either spatially configured or time multiplexed (TM). Section 3 looks at the most successful group of TM FPGA overlays, that is, processor-based overlays. Processor-based overlays range in complexity from simple single core (soft) processors to fully functional SIMD, VLIW or vector processors. Section 4 examines CGRA-like TM overlays which consist of an array of interconnected processing units. These processing units can range from complete processors down to medium-grained arithmetic units. Section 5 summarizes the various time multiplexed overlays and presents the conclusions.

2 OVERLAY ARCHITECTURES

An overlay is a virtual configurable architecture, implemented over the physical fine-grained FPGA fabric, thus enabling programmability at a higher level of abstraction [45]. Overlay architectures promise to tackle the "programmability wall" of FPGAs by avoiding the tedious fine-grained placement and routing process. Programming an overlay is similar to configuring an FPGA, except that configuration is also performed at a higher level, typically at the word and functional block level, rather than at the bit level. As such, the mapping tools for overlays can quickly generate an application bitstream in just a few seconds and configure the overlay in just a few microseconds, significantly faster than for FPGA. Figure 1 shows a typical automatic mapping tool flow targeting an overlay. The overlay is first designed using the FPGA vendors design tools, and a bitstream for configuring the FPGA is generated, as shown in the RHS dashed box of Figure 1. The remainder of the tool chain generates an overlay configuration based on a user application. As the overlay is located at a layer between the user application and the underlying physical FPGAs, it is not necessary to regenerate the FPGA bitstream for different target applications. If an application changes, all that a designer needs to do is to regenerate the new configuration for the overlay using the mapping tool flow (shown on the LHS of Figure 1) and reprogram the overlay. This flow (which is more like a software programming flow) achieves thousands of times reduction in the design cycle time compared to a traditional FPGA CAD flow [16].

While overlays allow high-level programmability with a significantly reduced compilation time, these advantages are not available for free. They generally come at the cost of a lower performance

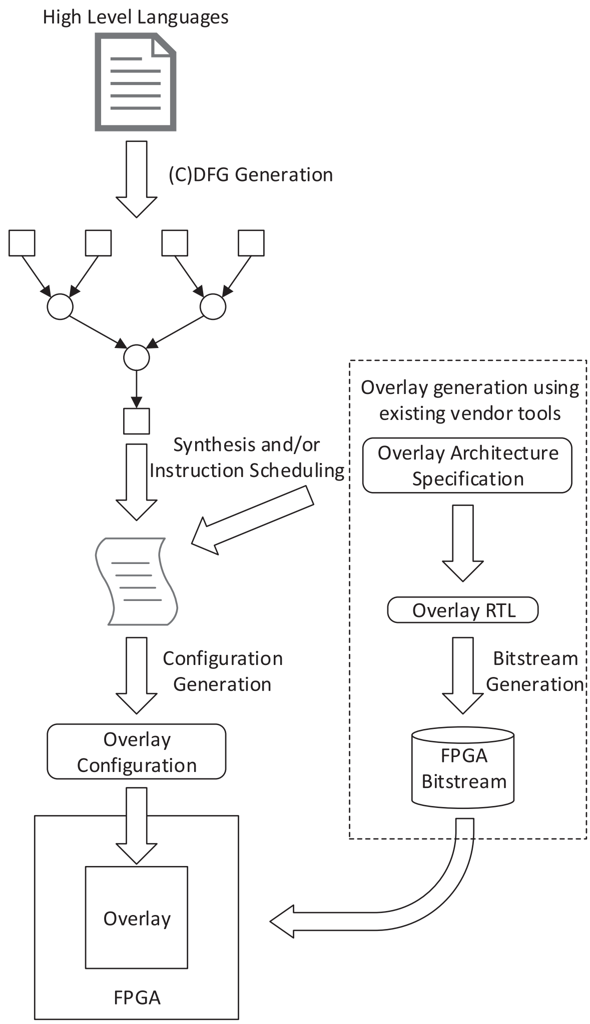


Fig. 1. A typical overlay tool flow.

with significantly more FPGA resource used than for an equivalent design mapped directly to FPGA. Even flexibility can be sacrificed as many overlays are specific to a set of applications [44, 68]. As such, a significant research effort has been applied to reducing the overlay area overhead and improving the throughput.

Overlays can be broadly classified based on the run-time configurability of their FUs. If an FU has a single fixed functionality at run-time, the overlay is referred to as spatially configured (SC), while if the FU changes its operation on a cycle-by-cycle basis, the overlay is referred to as time-multiplexed (TM). Table 1 lists some overlays categorized in terms of FU and interconnect configuration.

From Table 1, it can be seen that overlays with SC FUs and SC interconnect networks [6, 10, 11, 15, 25, 30, 33, 36, 75] comprise a significant group. In an SC overlay, a single operation node is mapped to an individual FU and data is shifted between FUs over a programmable, but temporally dedicated, point-to-point link. That is, the FU and interconnect configuration are fixed while the

Table 1. Selected Overlay Architectures

Year	Overlay Name	FU SC	FU TM	Interconnect SC	Interconnect TM	Interconnect NoC
2005	SPREE [80]		✓			
2006	QUKU [75]	✓		✓		
2010	IF [15]	✓		✓		
2011	VDR [10]	✓		✓		
2011	Heracles [43]		✓			✓
2012	ZUMA [7]	✓		✓		
2012	Octavo [53]		✓			
2012	reMORPH [65]		✓		✓	
2013	VCGRA [30]	✓		✓		
2013	CARBON [8]		✓		✓	
2013	MXP [74]		✓			
2013	SCGRA [60]		✓		✓	
2013	TILT [64]		✓		✓	
2015	DSP-based [33]	✓		✓		
2016	Linear TM [57]		✓		✓	
2016	DeCO [35]	✓		✓		
2016	GRVI Phalanx [26]		✓			✓

kernel executes. The benefit of an SC overlay is that kernel execution achieves an initiation interval (II) [54] of one, with throughput just determined by the operating frequency of the overlay.

However, the area overheads of SC overlays, in particular their large interconnect resource requirements, have limited the practical use of these overlays in FPGA-based systems to very small compute kernels [6]. This means that as a large application executes a number of different kernels would need to be mapped to the overlay to achieve the best application acceleration. Thus, the overlay context switch time (the time required to switch between executing kernels) is also an important consideration in the efficient operation of an overlay [16, 37]. Some of the current overlays utilize partial reconfiguration to reduce the overlay area, in particular the interconnect resources, by trading off runtime connection flexibility [65]. However, while faster than a complete FPGA reconfiguration, partial reconfiguration still results in a significant context switch overhead, which will impact an application's runtime if multiple kernels are used.

As there is always a tradeoff between area and speed in hardware design, a number of research groups have shifted their attention to overlays which share the functional units among kernel operations in an attempt to reduce overlay resource requirements. Sharing or time-multiplexing the FU can significantly reduce the FU and interconnect resource requirements but at the cost of a higher II and hence a reduced throughput. TM overlays can be generally divided into two categories: processor-based overlays, and coarse-grained reconfigurable architecture (CGRA) like overlays. Although the development of TM overlays is still at the primary stage, some of the existing works have shown great potential in tuning the compute density (throughput per area) and achieving rapid hardware context switching compared to the SC alternatives. In the next section, we review the current state-of-the-art relating to TM overlays.

3 PROCESSOR BASED OVERLAYS

Most successful TM FPGA overlays are based on processor implementations. These implementations range from single-issue processors, through multithreaded processors, to parallel processors

Table 2. Soft Processors (32-bit)

Year	Name	Device	Fmax	Area
2005	CUSTARD [18]	Virtex-2	30MHz	2400 Slices
2005	UT Nios [66]	Stratix	77MHz	3000 LEs
2005	SPREE [80]	Stratix II	82MHz	1200 LEs
2007	Leon3 [24]	Virtex-2	125MHz	3500 LUTs
2010	MB-LITE [47]	Virtex-5	65MHz	1450 LUTs
2010	Leon4 [1]	RT4G150	150MHz	4000 LUTs
2012	iDEA [13]	Virtex-6	453MHz	335 LUTs
2012	Octavo [53]	Stratix IV	550MHz	900 ALUTs
2016	GRVI [26]	UltraScale	375MHz	320 LUTs

and processor arrays. Overlays based on a processor implementation have the advantage of a well-known, well-designed instruction set architecture (ISA) which makes them easy to use, however, they tend to utilize a large amount of FPGA resource with a significant power consumption.

3.1 Soft Processors

A soft processor generally refers to a processor architecture which can be implemented on FPGA, which then allows the ISA to be customized to suit a specific application. FPGA vendors provide commercial soft processors such as Xilinx MicroBlaze [79] and Altera Nios II [3], implementing a conventional MIPS-like architecture for software portability. These industrial soft processors allow non-hardware experts to better target FPGAs with dedicated tools such as Xilinx EDK and Altera Eclipse. However, these implementations are not portable between different FPGA vendor devices and their RTL source code is not freely available. To overcome this, open source clones of these commercial soft processors have been developed, such as the performance centric UT Nios from the University of Toronto [66] and the area-efficient MB-LITE [47]. While these implementations are open source and can be customized to a specific application, their ISAs are not. To address this issue, a number of open source soft processors with free ISAs, such as OpenSPARC [78], OpenRISC [55], Plasma [69], RISC-V [77], Leon3 [24], and Leon4 [1], were developed by industrial or independent groups. A recent survey of open source soft processors [38] showed that apart from Leon3, most had a larger area overhead and provided less performance compared to MicroBlaze and Nios II. Table 2 lists the latest versions of some typical soft processors in the last decade.

3.1.1 Single-Issue Processors. Many of the earlier soft-core processors were single-issue processors because of their simplicity and area efficiency. These processors were to some extent constrained by the limited resources available in earlier generations of FPGA devices. MicroBlaze [79], Nios II [3], OpenRISC [55], and Plasma [69] are all examples of single-issue processors. Single-issue processors also tend to have fewer pipeline stages than multi-issue (superscalar) processors [50]. Some other single-issue processors include:

SPREE. The Soft Processor Rapid Exploration Environment (SPREE) was developed to automatically generate synthesizable HDL implementations of soft processor architectures from textual descriptions of the ISA and datapath [80], facilitating the microarchitectural exploration of soft processors. The SPREE processor with a 3-stage pipeline demonstrates 9% less area and 11% speedup in wall-clock-time compared to the Nios II family of commercial soft processors. By customizing the microarchitecture to specific software applications, the tuned version of SPREE provides an average improvement of 11.4% over the fastest-on-average general purpose processor in terms of compute efficiency [81]. The complexity of SPREE can be reduced by using functional component abstractions, however, some practical issues such as combinational loops, false paths, and multi-cycle paths, which affect the functionality and performance of the soft processor, may arise due to the careless use of these components.

iDEA. iDEA [12, 13] is a lightweight soft processor based on the Xilinx DSP48E1 primitive and was developed to address the resource consumption issue while better targeting the underlying FPGA architecture. The 9-stage pipelined design with no data forwarding outperforms MicroBlaze in both resource consumption (a 59% reduction in LUTs with an 18% increase in FFs) and speed (a 92% increase in f_{max}). To reduce the execution time caused by NOP insertion due to data hazards, data-forwarding approaches applicable to the DSP48E1 primitive, such as internal loopback and external forwarding, were explored, resulting in an improvement of up to 25% for a set of benchmarks [32].

While iDEA was designed as a soft processor to handle integer operations, it cannot fully support 32-bit multiplication because of the limited width of the multiplier inputs in the DSP48E1 (25×18 bits). Only a single DSP block is used to implement the soft processor, however, as there are hundreds of DSP blocks available in the modern FPGAs, making better use of these resources within a multi-processor system would significantly improve the performance for large compute kernels.

3.1.2 Multi-Issue Processors. While most of the early generation of soft processors were single-issue cores, multi-issue or superscalar single processor implementations have also been developed. One of the best examples is the LEON3 processor [1] based on the 32-bit SPARC V8 processor architecture which was developed for space applications and is available as a soft core for FPGAs. Another example is the Intel Nehalem soft processor core [70] which was developed for emulation purposes and uses five FPGAs while running at a frequency of just 520 kHz. Unfortunately, a superscalar architecture requires significant hardware complexity to dynamically extract the instruction parallelism which when implemented in FPGA results in very high hardware costs.

3.1.3 Multithreaded Processors. While single-issue processors are expected to run at a higher frequency with a pipelined architecture, their area-efficiency and instruction-per-cycle (IPC) count can be improved significantly with minimal extra complexity to support multithreading [49]. UTMT II [23] and MT-MB [63] are two typical soft processors which support multithreading on the Altera Nios II/e and Xilinx MicroBlaze core, respectively. UTMT II achieved a 25% LE area reduction compared with Nios II/e, while MT-MB achieved a peak performance of 5× over that of MicroBlaze. Apart from the extension of commercial cores, there are a number of independent research efforts towards providing multithreading support on soft processors, such as CUSTARD [18] and Octavo [53].

CUSTARD. The Customizable Multithreaded Processor (CUSTARD) was one of the first customizable multithreaded soft processors, supporting a parameterizable number of threads, threading type, datapath bitwidths and custom instructions [18, 19]. CUSTARD is a RISC processor which has a fully bypassed architecture with a 4-stage pipeline. When implemented on a XC2V2000 FPGA and compared with MicroBlaze using five typical benchmarks, the CUSTARD processor achieved an average speedup of 2.41× across all benchmarks with custom instructions. However, CUSTARD, and its extended version, only achieved a clock frequency of 30MHz to 50MHz, which is far less than the 100MHz achieved by the MicroBlaze soft processor. Additionally, the custom instruction speedup came at a penalty of two times the area consumption and less I/O support compared to MicroBlaze.

Octavo. The Octavo soft processor [53] is a multithreaded 10-stage pipelined architecture designed to operate at the theoretical maximum BRAM frequency (550MHz) on a Stratix IV device. A method of self-loop characterization was adopted to collapse the conventional register/cache/memory hierarchy into one unified entity, which is beneficial to absorb the propagation delays and simplify the ISA. To support fast multiplication, a fast multiplier which consists of two half-pumped DSP blocks was designed to overcome the hardware timing restriction of 480MHz.

In summary, although single-core soft processors allow the benefits of software programmability and hardware re-usage, their performance is still significantly less than that of either hard processors or dedicated hardware accelerators, and cannot meet the requirements of very-high-speed applications. In order to improve the throughput, there is an increasing amount of research work exploring multi-core systems of soft processors with efficient routing technologies.

3.2 Parallel Processors

The sequential processing of single-issue soft processors has limited their use to specific lower performance applications. When large-scale applications are considered, parallel computing, using single instruction, multiple data (SIMD) execution or other parallel processing techniques, may be required.

3.2.1 Multithreaded Parallel Processors. The Octavo soft processor [53] was further extended to support SIMD by duplicating the datapath with a shared instruction stream [52]. SIMD-Octavo was compared with VectorBlox MXP [74] (discussed in Section 3.2.3) and operates at about double the clock frequency of MXP and generally achieves better performance (for an equal number of lanes) in terms of execution time, area, and area-delay product. It has been claimed that the execution time of multi-lane SIMD-Octavo is better than hand-crafted Verilog HDL, but requires one to two orders of magnitude more hardware resource [52].

3.2.2 VLIW Processors. Very long instruction word (VLIW) processors have been proposed to exploit instruction level parallelism (ILP) by executing different operations on multiple FUs simultaneously [40].

TILT. The 32-bit floating point TILT overlay [67, 68], was proposed as an FPGA-based VLIW processor comprised of multiple floating point FUs with configurable pipeline depths. To enhance the throughput, multiple TILT cores can be instantiated, working in parallel with a single shared instruction memory. This architecture is referred to as TILT-SIMD. TILT has a separate 256-bit memory fetcher unit which allows for data transfer between up to 8 TILT cores and the off-chip DDR memory. The TILT overlay was evaluated for a set of five application benchmarks against Altera OpenCL HLS implementations. The TILT overlay was able to achieve an operating frequency over 200MHz, which is close to that of the HLS implementations, with an area overhead of less than 2× for the same throughput.

Currently, the TILT-System is not customized to a general class of kernel applications, and as such, a kernel update for a different application requires instruction rescheduling, with an associated FPGA reconfiguration, resulting in a context switch time of 38 seconds on average. Another drawback of the TILT overlay is that, even though TILT is more flexible than OpenCL HLS for implementing very small designs, it has less compute density compared to the OpenCL implementation. This problem can be solved by customizing the number of FUs and their functionality for specific applications.

3.2.3 Vector Processors. While it remains a problem for soft processors to scale their performance, soft vector processors (SVPs) are able to exploit data-level parallelism. They are able to explore the tradeoff between performance and area, with a hybrid approach which shares the benefits of traditional vector processing and modern SIMD mode. Most of the proposed SVPs have a similar architecture, with a scalar soft processor acing as the controller for multiple vector lanes executing custom instructions on a local memory [51]. SVPs can achieve a significant speedup over soft processors by effectively unrolling loops into vector operations. However, there are a number of obstacles limiting the widespread adoption of SVPs. These include, difficulty in programming vector architectures [39], lack of a high-performance interface to external logic and limited support for data-dependent behaviors [71].

A number of SVP designs, including VESPA [82], VIPERS [85], VEGAS [14], VENICE [73], and MXP [74], have been proposed. VESPA and VIPERS were developed in parallel as the first generation of FPGA-centric SVPs, with VEGAS, which better utilizes the on-chip FPGA memory, being the second generation. VENICE is the latest version, targeting high frequency and low area, and led to the first commercial SVP, referred to as VectorBlox MXP.

VESPA. VESPA was proposed as a MIPS-based processor with a VIRAM [46]-compatible vector coprocessor, which results in a system combining the advantages of portability, scalability, and flexibility [82]. VESPA is portable across FPGA platforms, though the original design targeted Stratix III. The VESPA prototype achieved an average speedup from 1.8× (2-lane) to 6.3× (16-lane) over the scalar processor on EEMBC benchmarks. The flexibility of VESPA makes it possible to trade off area savings (up to 70%) by adjusting the vector lane length and width. To better target the FPGA, an improved VESPA with support for vector chaining and heterogeneous lanes [83] was implemented on a Stratix III FPGA. The modified VESPA achieved up to 34% better compute efficiency relative to VESPA in terms of performance-per-area for the full set of EEMBC benchmarks.

VIPERS. Similar to VESPA, VIPERS consists of a single-threaded (Nios II-compatible) scalar core referred to as UTIIe, a memory interface unit, and a vector processing unit [85]. Three typical data-intensive applications were used as benchmarks for VIPERS and the Altera Nios II/s processor using "push-button" C2H accelerators. Compared to Nios II, VIPERS demonstrated a scalable speedup ranging from 3× to 29×, at the cost of a reasonable (6× to 30×) area penalty. An improved version of VIPERS [84] offers double the vector registers and several new instructions (compared to VESPA), and is less strict about VIRAM compliance. Based on the same benchmarks as in [85], VIPERS with 16 lanes can achieve up to 25× better performance with a modest 14× area increase compared to the Nios II processor. It is possible to achieve a further 30% area savings by customizing VIPERS to the benchmarks, equal to 6× the logic area of the Nios II/s processor implementation.

Although both VESPA and VIPERS provide a wide range of granularity from 8-bit to 32-bit, the vector engine must be built to fit the largest width if mixed-width data processing is required. As a result, byte-sized data needs to be zero-extended or sign-extended to the full width, which unnecessarily adds overhead to the instruction memory and register files. Additionally, as the vector register file is connected to an on-chip memory (VIPERS) or on-chip data cache (VESPA), the memory/cache width must be large enough to support the traditional vector load/store operations. However, the amount of on-chip memory is limited by the capacity of a particular FPGA.

VEGAS. Though VESPA and VIPERS demonstrated the scalability and feasibility of SVPs, they were not specifically targeted to the underlying FPGA architecture. As such, a new SVP architecture, VEGAS, was presented as a vector core with a Nios II/f processor [14]. The most significant differences between VEGAS and the previous SVPs, is the use of a cacheless scratchpad memory and a fracturable ALU which can support byte, halfword or word operations efficiently, according to the data width. Instead of conventional vector load/store instructions, VEGAS adopted direct memory access (DMA) read/write commands to achieve better storage efficiency and less memory latency. VEGAS can achieve up to 2.8× better performance than VESPA and 3.1× better than VIPERS in terms of throughput-per-area, and outperforms a 2.66-GHz Intel X5355 processor on the integer matrix multiply benchmark.

Despite the high performance VEGAS achieves, there are some drawbacks to the design which result in an area/performance overhead. First, it is cumbersome to track and spill values from the

8-entry vector address register file (VARF), which also consumes additional ALMs and FFs. Second, while the alignment network grows super-linearly with the number of vector lanes, only one single alignment network is implemented on VEGAS, which may introduce a performance penalty if the operands are unaligned.

VENICE. Based on the architecture of VEGAS, VENICE was proposed to maximize the throughput of SVPs with a small number of vector lanes [73]. While VEGAS achieved its best performance/area at 4-8 lanes, VENICE was tailored to 1-4 lanes without sacrificing performance. Removal of the vector address register file, adding a new conditional implementation, and streamlining the instructions, are the three major differences which reduce the area requirement and the complexity of programming, compared to VEGAS. 2D/3D vector instructions and operations on unaligned vectors were adopted to further improve the performance. VENICE can achieve over 2× better throughput-per-area than VEGAS, and a speedup of 5.2× higher than the fastest Nios II/f soft processor.

VENICE is much more area-efficient and easier to program compared with previous SVPs and further improves on the VEGAS ALU utilization. Since VENICE is designed as a small and fast SVP, the problem of efficiently integrating multiple VENICE components with high performance and interconnect simplicity remains a future problem.

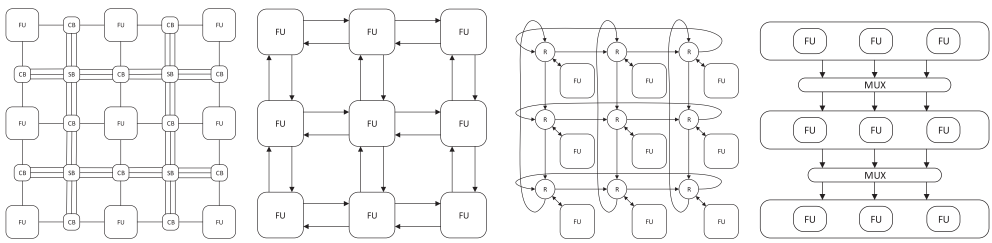
MXP. The VectorBlox MXP was developed as a commercial IP core which can interface to the Avalon and AXI on-chip bus protocols available in Altera or Xilinx FPGAs, respectively [74]. It is similar in design to VENICE, but with added features such as fixed-point arithmetic, 2D-DMA support, and a C++ object based application programming interface (API) for higher level programming. MXP can operate at over 200 MHz on a Stratix IV device with less than 16 vector lanes. A 64-lane configuration demonstrated a speedup of up to 918× that of a Nios II/f processor on matrix multiplication. Custom vector instructions (CVIs) were introduced for the latest SVPs to integrate streaming pipelines into the datapath with a minimum area overhead [72]. CVI-optimized SVPs achieved a 7200× speedup and over 100× improvement in terms of performance-per-ALM, compared to Nios II/f.

In general, SVPs achieve significant performance gains for data parallel applications. However, the scalability of SVPs is limited by the number of vector lanes, which is determined by the hardware resources on the FPGA. While increasing the number of vector lanes significantly increases the throughput, it also leads to clock frequency degradation. Additionally, compiler support for these processors is still at the primary stage as the repository of common operations and data types needs to be further improved.

3.2.4 Soft GPUs. Graphics processing units (GPUs) have a many-core architecture with considerable parallel processing capabilities. In general, GPUs and vector processors have many similarities with both supporting SIMD-style parallelism.

FlexGrip. FlexGrip [4] is a soft GPU based on the Nvidia G80 architecture targeting the Xilinx ML605 platform and provides direct CUDA compilation and execution. FlexGrip follows a single instruction multiple thread (SIMT) model with an instruction fetched and simultaneously mapped onto multiple scalar processors (SPs). FlexGrip with 32 SPs achieves a peak speedup of 30× compared to MicroBlaze, but with a significant area overhead, consuming 96% of the available LUTs.

MIAOW. MIAOW [5] is an open source RTL implementation of the AMD Southern Islands GPU ISA, which is compatible with OpenCL applications. The complete system was implemented on a VC707 evaluation board requiring a considerable amount of FPGA resource (195K LUTs and 137 BRAMs). MIAOW was validated by comparing it with commercial GPUs in terms of area, power, and performance.



(a) Island Style (b) Nearest Neighbor (c) NoC Torus (d) Linear

Fig. 2. Typical overlay topologies.

FGPU. A GPU-like SIMT soft processor, referred to as FGPU [2], was proposed as a flexible solution for software tasks. The VHDL implementation of FGPU did not use any FPGA specific IP cores or FPGA primitives, making it highly portable and customizable. It has a mixed ISA supporting both MIPS instructions and OpenCL functions. A speedup of 48.5× over MicroBlaze was achieved for a range of benchmarks on the ZC706 FPGA board, with a 17.7× area overhead. To achieve high performance, FGPU is designed with an 18-stage pipeline. Due to the complexity of the compute units, an 8 compute unit version of FGPU consumes 124K LUTs on the ZC706, corresponding to 57% of the available resource.

SCRATCH. An application-aware soft GPU, referred to as the SCRATCH framework [20], was developed as an upgraded version of the MIAOW GPU architecture. The main contribution of the SCRATCH system is the MIAOW-based architecture optimization to support additional instructions and the SCRATCH trimming algorithm which removed unnecessary architectural functionality to improvement performance. Similar to MIAOW, SCRATCH was evaluated on Xilinx Virtex 7 FPGAs. By applying architecture trimming along with multithread and multi-core parallelism, SCRATCH was able to achieve a peak speedup of 260× with a 250× better energy-efficiency compared to the original MIAOW system. In addition to the improvement in throughput and energy-efficiency, a significant reduction in FPGA resource was observed, specifically a 36% reduction in LUTs and a 41% reduction in FFs.

4 CGRA-LIKE OVERLAYS

Coarse-grained reconfigurable architectures (CGRAs) have been extensively researched due to their enhanced scalability, performance and power efficiency compared to CPUs. CGRAs typically fall within one of two classes: processor-centric arrays which are made up of individual processors connected via programmable interconnect; and CGRAs with coarse/medium-grained processing elements (also called medium-grained processing arrays).

4.1 Interconnect Topology

Irrespective of the computational element (be it a processor or a dedicated processing element), CGRA-like overlays are characterized by an array structure of computational elements connected using programmable interconnect. A number of interconnect strategies exist, with the most common being: island style [6, 15, 25, 33, 36], nearest neighbor (NN) [11, 59], network-on-chip (NoC) [26, 41, 42] and to a lesser extent linear interconnect [10, 16], as shown in Figure 2. Other interconnect strategies are possible, including circuit switched [31] networks, but these typically consume significant hardware resource and are less suited for FPGA-based overlays. There are also variations in the more common interconnect strategies. For example, for NN, alternative topologies include torus [59], mesh plus [61] and fully connected [76], while for NoC, many different typologies such as bidirectional mesh, unidirectional torus and deflection-routed torus have been investigated [41]. The deflection-routed torus proves to be 3.5× more area-efficient than the bidirectional mesh by adopting a deflection routing technique [62] to the directional torus.

Island style and NN interconnects are a 2-D mesh structures which to some extent have a similar architecture to the interconnect on FPGAs. These interconnect strategies are highly flexible to fully support direct communication between the adjacent FUs. However, they require a considerable amount of the FPGA routing to implement and as a result consume a significant amount of the FPGA resource [34]. In contrast, the resource requirement for a linear interconnect is significantly less because of its 1-D feed-forward array structure. For example, the DeCO overlay [35], which has a cone-shaped linear array of FUs which maps well to the feed-forward DFGs being accelerated, has an 87% reduction in LUT utilization compared to the island-style overlay.

4.2 CGRA-like Processor Arrays

Large CGRA-like processor arrays have seen a resurgence in recent years due to the higher capacity of modern FPGAs. This larger FPGA capacity, along with more efficient NoC implementations [41] has meant that they are able to accommodate more complex designs. These processor arrays have similarities to ASIC-based processor-centric CGRAs. Some examples include:

Heracles. Heracles [43] is an open-source integer-based 7-stage MIPS-III processor array with a 2D-mesh topology, which consists of a NoC architecture for data communication. Synthesis results showed that one processor element with cache memory consumed 5562 LUTs and 2695 FFs on a Virtex-5 LX330T, running at a frequency of 155MHz. The Heracles virtual-channel router consumed 2058 LUTs, 2806FFs and operated at a frequency of 71MHz. Compared to the classic unbalanced fat-tree [56] topology, the proposed virtual-channel router consumed only 1.7% of the fabric logic, with a 2.3× higher clock frequency. However, LUT consumption became the bottleneck when scaling due to the attached memory subsystem, thus Heracles was restricted to a 4×4 array on Virtex-5.

GRVI Phalanx. GRVI Phalanx [26] is a massively parallel overlay based on an FPGA-efficient implementation of the RISC-V [77] soft processor. The GRVI processor uses just 320 LUTs and runs at a frequency of up to 375MHz on a Kintex UltraScale FPGA. Multiple GRVI processors with shared memory and local interconnect, are formed as clusters, which efficiently communicate with each other via a Hoplite NoC [41]. Implementations with 400 and 1680 RISC-V cores on a Kintex UltraScale KU040 and a Virtex UltraScale+ VU9P have been reported. Currently there is minimum tool support for this platform with no application performance comparisons with other overlays.

120-Core MIPS Overlay. A 120-core MIPS overlay [48] was developed to optimize a silicon-tested microAptiv MIPS processor for FPGA implementation. The design achieved a significant reduction to the original μaptiv MIPSfpga [27], by replacing the complex instruction/data cache with dedicated scratchpads, adopting DSP blocks for multiplication and a NoC-specific modification to the decoder. The improved MIPS processors with a Hoplite NoC [41] increased the maximum array size from 30 to 120 cores on the DE5-NET board, while achieving a higher frequency (94MHz).

4.3 CGRA-Like Medium-Grained Overlays

CGRAs with medium-grained processing elements have an number of advantages compared to CPUs, including better scalability, performance and power efficiency [28]. Additionally, compared to fine-gained reconfigurable architectures, such as FPGAs, which typically consist of an array of logic blocks at the bit-level (or a small number of bits), CGRAs are reconfigurable at the word-level (8-bit, 16-bit, 32-bit, etc.). In CGRAs, the processing elements are typically much larger than the FPGA's fine-grained lookup tables (LUTs), and can be an arithmetic logic unit (ALU) or word-level multiplier, or even a DSP primitive. This coarse granularity results in a reduction in the configuration memory, the configuration time, and the placement and routing complexity, compared to

Table 3. Selected CGRA-like Overlays

Year	Name	Granularity Arithmetic	Device	Fmax Size	FPGA Resource
2010	Heracles [43]	32-bit Integer	Virtex 5	155MHz 4×4	12K LUTs, 8.8K FFs
2011	MIN Overlay [22]	8/32/64-bit Integer & FP	Virtex 6	100MHz 30	22K LUTs, 4.8K FFs, 40 DSPs
2011	CARBON [8]	32-bit Integer	Stratix III	150MHz 2×2	3K ALMs, 517 FFs, 15Kb BRAM, 4 DSPs
2012	reMORPH [65]	32-bit Integer	Virtex 6	400MHz[1] 40	196 LUTs, 41 FFs, 3 BRAMs, 1 DSP[2]
2013	SCGRA [60]	32-bit Integer	Zynq-7000	250MHz 2×2	5K LUTs, 9K FFs, 50 BRAMs, 12 DSPs
2016	GRVI Phalanx [26]	32-bit Integer	UltraScale	$10 \times 5 \times 8$ 375MHz	177K LUTs, 1200 BRAMs
2016	Linear TM [57]	32-bit Integer	Zynq-7000	286MHz 8	1.7K LUTs, 1.9K FFs, 8 DSPs
2017	MIPS Overlay [48]	32-bit Integer	Stratix V	94MHz 60×2	2.4K ALMs, 2.1K FFs, 2 DSPs, 3 M20Ks[2]

[1]Reported Fmax is only for an FU.
[2]Reported Resource is only for a single FU.

fine-grained FPGAs [29]. Although there has been a significant amount of CGRA research over the last few decades, only a few CGRAs have been commercialized, mainly because they are less flexible compared to FPGAs and lack a well-defined design flow [75].

An alternative to an ASIC CGRA is the CGRA-like FPGA overlay, which implements a CGRA as a virtual configurable architecture on top of a reconfigurable FPGA. Initially, mapping CGRAs to FPGA was performed to demonstrate their functionality before ASIC implementation. More recently, specific dedicated CGRA-like FPGA overlays were developed mainly to improve the design productivity of FPGA. Many of these initial CGRA-like overlays were more throughput-oriented SC overlays which mapped each operation to a single FU to achieve an II of one. However, as mentioned earlier, these overlays were relatively small due to the limited hardware resources available in the underlying FPGA and were unable to accommodate larger compute kernels. Recently, researchers have shifted to more area-efficient overlay architectures which are able to time-multiplex the operations to an FU on a cycle-by-cycle basis. This makes it possible to map larger application kernels to the overlay, but at the cost of throughput. A summary of some of the TM CGRA-like overlays is given in Table 3.

4.3.1 TM Overlays with Homogeneous FUs. Time-multiplexed CGRA-like overlays with Homogeneous FUs have the advantage that they can be more easily tiled to the FPGA architecture due to their regularity. Additionally, applications can be more easily scheduled as operations can be arbitrarily mapped to FUs. However, having only homogeneous FUs can restrict application flexibility. Some examples include:

CARBON. CARBON [8] is a CGRA-like overlay which was implemented as a 2×2 array of tiles on an Altera Stratix III FPGA. Each tile has an FU with a programmable ALU and instruction memory, supporting up to 256 instructions. An FU consumed 3K ALMs, 517 FFs, 15.6Kb BRAM, and 4 DSP blocks, achieving an operating frequency of 150MHz. Compared to the other TM overlays discussed here, CARBON has a large resource requirement with a relatively slow speed which limits the scalability of the architecture. Additionally, the BRAMs were not effectively used to read the instruction memory, which results in the need for an additional bypass register to avoid the extra latency.

reMORPH. The reMORPH overlay [65] better targeted the FPGA fabric, with an FU consuming 1 DSP Block, 3 block RAMs, 196 LUTs, and 41 registers. This low footprint makes it possible to implement around 40 tiles on the Xilinx Spartan 6 LX45 FPGA. A reMORPH tile uses the Xilinx DSP primitive as a 5-stage pipelined ALU with a BRAM as its instruction memory, which ensures a high operating frequency (400MHz). To reduce the overhead due to routing and multiplexers, the reMORPH FU does not use decoders resulting in a 72-bit-wide instruction memory (supporting up to 512 instructions) which causes an over utilization of BRAMs, thereby limiting the possible size of this overlay. Tiles are interconnected using an NN style of non-programmable interconnect, which is adapted using partial reconfiguration at runtime, and hence changing between application kernels is relatively slow (that is, the overlay has a large hardware context switch time).

SCGRA. The SCGRA overlay [60] was proposed to address FPGA design productivity, demonstrating a 10× to 100× reduction in compilation time compared to the AutoESL HLS tool. Application-specific SCGRA overlays were subsequently implemented on the Xilinx Zynq platform [59], achieving a speedup of up to 9× higher than the standalone Zynq ARM processor. The FU used in the Zynq-based SCGRA overlay operates at 250MHz and consists of an ALU, multi-port data memory (256 × 32 bits) and a customizable depth instruction ROM (Supporting 72-bit wide instructions) which results in the excessive utilization of BRAMs. As the full FPGA bitstream needs to be reconfigured for a compute kernel change, very fast context switching between applications is not possible.

Although the SCGRA overlay allows for different size implementations, there is a significant performance drop for larger implementations due to the following reasons. First, the higher BRAM requirement for instruction memory means that there needs to be a tradeoff in the number of BRAMs for the I/O buffer, which has a negative effect on data reuse. Second, a larger SCGRA overlay will increase the routing cost between PEs, therefore reducing the compute performance. Finally, the operating frequency drops as the overlay size increases, resulting in a degradation in the overall performance.

Linear TM Overlay. An area efficient time-multiplexed overlay with linear interconnect [57, 58] was proposed to reduce the interconnect requirements of array-based overlays. It consists of a streaming data interface made up of Distributed RAM (DRAM) acting as a FIFO, which feeds a cascade of time-multiplexed FUs, with another DRAM-based FIFO at the output. Tasks are scheduled to the overlay using ASAP scheduling, which allows data flow graph (DFG) nodes from the same scheduling time step to be allocated to individual FUs. The FU uses the same principle as the iDEA DSP-based processor [12], and requires 1 DSP block, 212 LUTs, and 228 FFs and runs at 323MHz on a Xilinx Zynq. Cascading 8 FUs into a linear overlay consumes 1,747 LUTs and 1,954 FFs (814 logic slices) and 8 DSPs, and operates at a frequency of 286MHz. While this represents a 21% reduction in resource utilization compared to DeCO [35], it comes at the expense of a significant reduction in the throughput and the II.

4.3.2 *TM Overlays with Heterogeneous FUs.*

Time-multiplexed CGRA-like overlays with Heterogeneous FUs have the advantage that they can support a wider range of applications, including mixed integer and floating point applications. Some examples include:

MIN Overlay. The MIN overlay consists of heterogeneous FUs, which are connected by a global multi-stage interconnection network (MIN) [22]. The heterogeneous FUs can support up to 64-bit floating point computations. MIN uses a global interconnect network instead of the traditional 2-D array topology, which significantly reduces the routing resource requirements, resulting in better hardware resource utilization. Compared with the crossbar network in TILT, the proposed two parallel blocking MINs reduce the cost complexity from $O(n^2)$ to $O(n \log n)$. A number of different parameterized architectures, chosen to evaluate the impact of the number and types of FU, memory and I/O, were implemented on a Virtex 6 FPGA. The smallest architecture (called A1) has 30 FUs and a 64 I/O global network and consumes around 1% of registers, 4% of DSPs and 15% of LUTs, while running at a frequency of 100MHz. A heuristic scheduling, placement and routing algorithm was used and could achieve just-in-time compilation in less than 300ms. While MIN has relatively good FPGA resource utilization, the LUT usage due to the routing network in larger designs will eventually become the limiting factor. Additionally, in some cases, routing fails due to missing registers which could be overcome by adding a register file in some of the FUs.

5 DISCUSSION AND CONCLUSIONS

TM overlays for FPGA are reasonably mature with processor-based TM overlays being better accepted compared to CGRA-like overlays. This is because the processor-based overlays have the advantage of well understood ISAs and easily accessible compilation tool chains making application development much easier for non-hardware designers. Furthermore, processor-based overlays using parallel processing techniques, such as multi-issue, multithreading, VLIW, and vector processing, have been developed and shown to improve overlay performance. However, these overlays suffer from similar problems to processors implemented directly in silicon, such as being complex with significant resource utilization and power consumption, which tends to negate some of the designer productivity advantages (such as their software programmability).

On the other hand, CGRA-like TM FPGA overlays have only really appeared within the last several years. These overlays are again targeted at improving FPGA designer productivity, and are better tailored towards area-efficient higher speed processing than processor-based overlays, although they still suffer from a lower speed and higher FPGA resource utilization than direct HDL- or HLS-based application implementation on FPGA. Recent CGRA-like overlays better utilize the coarse-grained modules present in modern FPGAs, such as DSP blocks and BRAMs. These overlays are particularly targeted towards the acceleration of compute intensive loops [59].

A selection of the TM overlays from the literature (both processor-based and CGRA-like) are summarized in Table 4. Table 4 categorizes the different overlays based on the overlay type and provides an indication of the computational throughput and the relative FPGA resource consumed. So that the overlay's implementation technology does not overly impact the throughput and resource utilization, both these metrics have first been nominally normalized to that of a Virtex 7 implementation. The throughput is normalized by multiplying by the ratio of the maximum Virtex 7 BRAM frequency divided by the maximum BRAM frequency of the original target device, while the resource consumption is determined by considering the total system resource utilization (adjusted to account for technology changes, such as the transition from 4-LUTs to 6-LUTs) divided by the number of cores/FUs. However, it should be noted that it is difficult to compare the performance of the various overlays due to the different architectures involved. Processor-based overlays can be compared to existing processors, such as to soft core processors from the major FPGA vendors. The array based overlays are more difficult to compare as they are relatively newer and less established, with limited system level support available to make a general comparison to other overlays. Where possible a comparison between the existing overlays from the literature is presented (as the Speedup column) in Table 4. The advantages and disadvantages of the different overlay types are also presented.

In conclusion, this article introduces FPGA overlay architectures and classifies them into two categories: SC overlays and TM overlays. Existing TM FPGA overlays are then focused upon, with a comprehensive survey of these overlays from the research literature being presented. TM

Table 4. A Summary of Selected TM Overlays

Name	Overlay Type[1]	Throughput[2] / Speedup	Resource[3]	Advantages	Disadvantages
UT Nios [66]	Single-issue μp	Low / 1% over Nios II	Medium	Well understood processor ISA; good tool support; easy to use; area efficient; high flexibility when configuring the core and tuning to specific applications	Relatively low performance; relatively high power consumption; Some processor designs attempt to be general and do not make good use of a specific FPGA architecture
SPREE [80]	Single-issue μp	Low / 11% over Nios II	Low		
Leon3 [24]	Single-issue μp	Low / Close to MicroBlaze	Medium		
MB-LITE [47]	Single-issue μp	Very low / Lower than MicroBlaze	Low		
Leon4 [1]	Single-issue μp	Medium / NA	Medium		
iDEA [13]	Single-issue μp	Medium / 92% over MicroBlaze	Very low		
CUSTARD [18]	MT Processor	Medium / 2.4× over MicroBlaze	Medium	Well understood processor ISA; good/adequate tool support; high performance; supports parallelism; low power consumption	Resource hungry; higher code complexity; inefficient if the data cannot be executed in a highly parallel manner
SIMD-Octavo [52]	MT Processor	High / Comparable to MXP [74]	Low		
TILT [68]	VLIW Processor	Medium / Lower than OpenCL HLS	High		
MXP [74]	Vector Processor	High / 918× over Nios II	Medium		
FGPU [2]	Soft GPU	High / 48.5× over MicroBlaze	Very high		
SCRATCH [20]	Soft GPU	Very high / 260× over MIAOW [5]	Very high		
Heracles [43]	CGRA-like μp Array	Medium / NA	Medium	Well understood processor ISA; high performance; high scalability; efficient NoC routing network	Limited tool support; resource hungry; high power consumption; lack of application benchmark evaluations
GRVI Phalanx [26]	CGRA-like μp Array	Very high / NA	Very low		
MIPS Overlay [48]	CGRA-like μp Array	High / NA	Medium		
MIN Overlay [22]	CGRA-like MG Overlay	Medium / NA	Low	Moderate performance; low area consumption; low power consumption, makes good use of coarse-grained modules such as DSPs and BRAMs	Limited tool support; large routing area overhead; long context switching time; only suitable for acceleration of small kernels (except SCGRA); lack of a pipeline-aware scheduling strategy
CARBON [8]	CGRA-like MG Overlay	Low/Medium / NA	Low		
reMORPH [65]	CGRA-like MG Overlay	Medium / NA	Very low		
SCGRA [60]	CGRA-like MG Overlay	Medium / 9× over Zynq ARM	Low		
Linear TM [57]	CGRA-like MG Overlay	Medium / NA	Very low		

[1] μp is short for microprocessor and MG is short for medium-grained.
[2] Normalized throughput in Virtex 7 device. Very low: $\leqslant$500 MB/s. Low: 0.5–1 GB/s. Medium: 1–5 GB/s. High: 5–10 GB/s. Very high: $\geqslant$10GB/s.
[3] Total system resource utilization / Number of cores. Very low: $\leqslant$500 LUTs. Low: 500–2K LUTs. Medium: 2K–5K LUTs. High: 5K–10K LUTs. Very high: $\geqslant$10K LUTs.

overlays are further categorized as processor-based overlays (as their implementation follows that of conventional silicon-based processors) and CGRA-like overlays (with both processor-based FUs and medium-grained FUs).

Time-multiplexing the overlay allows it to change its behavior, cycle by cycle, during the compute kernel execution, thus allowing better sharing of the limited FPGA resources. However, most of the TM overlays described still suffer from relatively large area overheads, due to either their underlying processor-like architecture or, for CGRA-like overlays, due to the routing resources and instruction storage requirements. Reducing the area overhead for CGRA-like overlays, specifically for the routing network, and utilizing the fast context switch capabilities of these overlays are likely to result in better usability with corresponding improvements in design productivity.

REFERENCES

[1] COBHAM GAISLER AB. 2017. GRLIB IP core user's manual. (2017).

[2] Muhammed Al Kadi, Benedikt Janssen, and Michael Huebner. 2016. FGPU: An SIMT-architecture for FPGAs. In *Proc. 24th Int. Symp. Field Program. Gate Arrays (FPGA)*. 254–263.

[3] Altera. 2016. Nios II processor reference handbook.

[4] Kevin Andryc, Murtaza Merchant, and Russell Tessier. 2013. FlexGrip: A soft GPGPU for FPGAs. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. 230–237.

[5] Raghuraman Balasubramanian, Vinay Gangadhar, Ziliang Guo, Chen-Han Ho, Cherin Joseph, Jaikrishnan Menon, Mario Paulo Drumond, Robin Paul, Sharath Prasad, Pradip Valathol, et al. 2015. Enabling GPGPU low-level hardware explorations with MIAOW: An open-source RTL implementation of a GPGPU. *ACM Trans. on Archit. and Code Optimization (TACO)* 12, 2 (2015), 21.

[6] Jesse Benson, Ryan Cofell, Chris Frericks, Chen-Han Ho, Venkatraman Govindaraju, Tony Nowatzki, and Karthikeyan Sankaralingam. 2012. Design, integration and implementation of the DySER hardware accelerator into OpenSPARC. In *Proc. 18th Int. Symp. High Performance Comput. Archit. (HPCA)*. 1–12.

[7] Alexander Brant and Guy GF Lemieux. 2012. ZUMA: An open FPGA overlay architecture. In *Proc. 20th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 93–96.

[8] Alexander Dunlop Brant. 2013. *Coarse and Fine Grain Programmable Overlay Architectures for FPGAs*. Ph.D. Dissertation. University of British Columbia.

[9] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In *Proc. 19th Int. Symp. Field Program. Gate Arrays (FPGA)*. 33–36.

[10] Davor Capalija and Tarek S. Abdelrahman. 2011. Towards synthesis-free JIT compilation to commodity FPGAs. In *Proc. 19th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 202–205.

[11] Davor Capalija and Tarek S. Abdelrahman. 2013. A high-performance overlay architecture for pipelined execution of data flow graphs. In *Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL)*. 1–8.

[12] Hui Yan Cheah, Fredrik Brosser, Suhaib A. Fahmy, and Douglas L. Maskell. 2014. The iDEA DSP block-based soft processor for FPGAs. *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)* 7, 3 (2014), 19.

[13] Hui Yan Cheah, Suhaib A. Fahmy, and Douglas L. Maskell. 2012. iDEA: A DSP block based FPGA soft processor. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. 151–158.

[14] Christopher H. Chou, Aaron Severance, Alex D. Brant, Zhiduo Liu, Saurabh Sant, and Guy G. F. Lemieux. 2011. VEGAS: Soft vector processor with scratchpad memory. In *Proc. 19th Int. Symp. Field Program. Gate Arrays (FPGA)*. 15–24.

[15] James Coole and Greg Stitt. 2010. Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing. In *Proc. Int. Conf. Hardware/Software Codesign and Syst. Synthesis (CODES+ ISSS)*. 13–22.

[16] James Coole and Greg Stitt. 2015. Adjustable-cost overlays for runtime compilation. In *Proc. 23rd Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 21–24.

[17] Tomasz S. Czajkowski, Utku Aydonat, Dmitry Denisenko, John Freeman, Michael Kinsner, David Neto, Jason Wong, Peter Yiannacouras, and Deshanand P. Singh. 2012. From OpenCL to high-performance hardware on FPGAs. In *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*. 531–534.

[18] Robert Dimond, Oskar Mencer, and Wayne Luk. 2005. CUSTARD-a customisable threaded FPGA soft processor and tools. In *Proc. 15th Int. Conf. Field Program. Logic Appl. (FPL)*. 1–6.

[19] R. Dimond, O. Mencer, and W. Luk. 2006. Application-specific customisation of multi-threaded soft processors. *IEE Proc.-Comput. and Digital Tech.* 153, 3 (2006), 173–180.

[20] Pedro Duarte, Pedro Tomas, and Gabriel Falcao. 2017. SCRATCH: An end-to-end application-aware soft-GPGPU architecture and trimming tool. In *Proc. 50th Int. Symp. Microarchitecture*. 165–177.
[21] Tom Feist. 2012. Vivado design suite. *White Paper* 5 (2012).
[22] Ricardo Ferreira, Julio Goldner Vendramini, Lucas Mucida, Monica Magalhaes Pereira, and Luigi Carro. 2011. An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture. In *Proc. Int. Conf. Compilers, Archit. and Synthesis for Embedded Syst. (CASES)*. 195–204.
[23] Blair Fort, Davor Capalija, Zvonko G. Vranesic, and Stephen D. Brown. 2006. A multithreaded soft processor for SoPC area reduction. In *Proc. 14th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 131–142.
[24] Jiri Gaisler, Edvin Catovic, Marko Isomaki, Kristoffer Glembo, and Sandi Habinc. 2007. GRLIB IP core user's manual. *Gaisler Research* (2007).
[25] Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. 2011. Dynamically specialized datapaths for energy efficient computing. In *Proc. Int. Symp. High Performance Comput. Archit. (HPCA)*. 503–514.
[26] Jan Gray. 2016. GRVI-Phalanx: A massively parallel RISC-V FPGA accelerator. In *Proc. 24th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 17–20.
[27] Sarah L. Harris, Robert Owen, Enrique Sedano, and Daniel Chaver Martinez. 2016. MIPSfpga: Hands-on learning on a commercial soft-core. In *Proc. 11th European Workshop on Microelectronics Education (EWME)*. 1–5.
[28] Reiner Hartenstein. 2001. Coarse grain reconfigurable architectures. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*. 564–569.
[29] Reiner Hartenstein. 2001. A decade of reconfigurable computing: A visionary retrospective. In *Proc. Conf. Design, Autom. and Test in Europe (DATE)*. 642–649.
[30] Karel Heyse, Timothy N. Davidson, Elias Vansteenkiste, Karel Bruneel, and Dirk Stroobandt. 2013. Efficient implementation of virtual coarse grained reconfigurable arrays on FPGAs. In *Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL)*. 1–8.
[31] Clint Hilton and Brent Nelson. 2006. PNoC: A flexible circuit-switched NoC for FPGA-based systems. *IEE Proc.-Comput. and Digital Tech.* 153, 3 (2006), 181–188.
[32] Cheah Hui Yan, Suhaib Fahmy, and Nachiket Kapre. 2015. On data forwarding in deeply pipelined soft processors. In *Proc. 23rd Int. Symp. Field Program. Gate Arrays (FPGA)*. 181–189.
[33] Abhishek Kumar Jain, Suhaib A. Fahmy, and Douglas L. Maskell. 2015. Efficient overlay architecture based on DSP blocks. In *Proc. 23rd Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 25–28.
[34] Abhishek Kumar Jain, Xiangwei Li, Suhaib A. Fahmy, and Douglas L. Maskell. 2016. Adapting the DySER architecture with DSP blocks as an overlay for the Xilinx Zynq. *ACM SIGARCH Comput. Archit. News* 43, 4 (2016), 28–33.
[35] Abhishek Kumar Jain, Xiangwei Li, Pranjul Singhai, Douglas L. Maskell, and Suhaib A. Fahmy. 2016. DeCO: A DSP block based FPGA accelerator overlay with low overhead interconnect. In *Proc. 24th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 1–8.
[36] Abhishek Kumar Jain, Douglas L. Maskell, and Suhaib A. Fahmy. 2016. Throughput oriented FPGA overlays using DSP blocks. In *Proc. Conf. Design, Autom. and Test in Europe (DATE)*. 1628–1633.
[37] Abhishek Kumar Jain, Khoa Dang Pham, Jin Cui, Suhaib A. Fahmy, and Douglas L. Maskell. 2014. Virtualized execution and management of hardware tasks on a hybrid ARM-FPGA platform. *J. of Signal Process. Syst.* 77, 1-2 (2014), 61–76.
[38] Rui Jia, Colin Yu Lin, Zhenhong Guo, Rui Chen, Fei Wang, Tongqiang Gao, and Haigang Yang. 2014. A survey of open source processors for FPGAs. In *Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL)*. 1–6.
[39] Soh Jun Jie and Nachiket Kapre. 2014. Comparing soft and hard vector processing in FPGA-based embedded systems. In *Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL)*. 1–7.
[40] Alex K. Jones, Raymond Hoare, Dara Kusic, Joshua Fazekas, and John Foster. 2005. An FPGA-based VLIW processor with custom hardware execution. In *Proc. 13th Int. Symp. Field Program. Gate Arrays (FPGA)*. 107–117.
[41] Nachiket Kapre and Jan Gray. 2015. Hoplite: Building austere overlay NoCs for FPGAs. In *Proc. 25th Int. Conf. Field Program. Logic Appl. (FPL)*. 1–8.
[42] Nachiket Kapre, Nikil Mehta, Michael Delorimier, Raphael Rubin, Henry Barnor, Michael J. Wilson, Michael Wrighton, and André Dehon. 2006. Packet switched vs. time multiplexed FPGA overlay networks. In *Proc. 14th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 205–216.
[43] Michel A. Kinsy, Michael Pellauer, and Srinivas Devadas. 2011. Heracles: Fully synthesizable parameterized mips-based multicore system. In *Proc. 21st Int. Conf. Field Program. Logic Appl. (FPL)*. 356–362.
[44] Dirk Koch, Christian Beckhoff, and Guy G. F. Lemieux. 2013. An efficient FPGA overlay for portable custom instruction set extensions. In *Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL)*. 1–8.
[45] Dirk Koch, Frank Hannig, and Daniel Ziener. 2016. *FPGAs for Software Programmers*. Springer.
[46] Christoforos Kozyrakis and David Patterson. 2002. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proc. 35th Int. Symp. Microarchitecture*. 283–293.

[47] Tamar Kranenburg and Rene Van Leuken. 2010. MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture. In *Proc. Conf. Design, Autom. and Test in Europe (DATE)*. 997–1000.

[48] Chethan Kumar HB, Prashant Ravi, Gourav Modi, and Nachiket Kapre. 2017. 120-core microAptiv MIPS overlay for the Terasic DE5-NET FPGA board. In *Proc. 25th Int. Symp. Field Program. Gate Arrays (FPGA)*. 141–146.

[49] Martin Labrecque and J. Gregory Steffan. 2007. Improving pipelined soft processors with multithreading. In *Proc. 17th Int. Conf. Field Program. Logic Appl. (FPL)*. 210–215.

[50] Martin Labrecque, Peter Yiannacouras, and J. Gregory Steffan. 2008. Scaling soft processor systems. In *Proc. 16th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 195–205.

[51] Charles Eric LaForest. 2015. *High-speed Soft-processor Architecture for FPGA Overlays*. Ph.D. Dissertation. University of Toronto.

[52] Charles Eric Laforest and Jason H. Anderson. 2017. Microarchitectural comparison of the MXP and Octavo soft-processor FPGA overlays. *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)* 10, 3 (2017), 19.

[53] Charles Eric LaForest and John Gregory Steffan. 2012. Octavo: An FPGA-centric processor family. In *Proc. 20th Int. Symp. Field Program. Gate Arrays (FPGA)*. 219–228.

[54] Monica Lam. 1988. Software pipelining: An effective scheduling technique for VLIW machines. In *ACM Sigplan Notices*, Vol. 23. 318–328.

[55] Damjan Lampret. 2001. OpenRISC 1200 IP core specification. https://opencores.org/.

[56] Charles E. Leiserson. 1985. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. on Comput.* 100, 10 (1985), 892–901.

[57] Xiangwei Li, Abhishek Jain, Douglas Maskell, and Suhaib A. Fahmy. 2016. An area-efficient FPGA overlay using DSP block based time-multiplexed functional units. In *Proc. 2nd Int. Workshop on Overlay Archit. for FPGAs (OLAF)*.

[58] Xiangwei Li, Abhishek Kumar Jain, Douglas L. Maskell, and Suhaib A. Fahmy. 2018. A time-multiplexed FPGA overlay with linear interconnect. In *Proc. Conf. Design, Autom. and Test in Europe (DATE)*. 1075–1080.

[59] Cheng Liu, Ho-Cheung Ng, and Hayden Kwok-Hay So. 2015. QuickDough: A rapid FPGA loop accelerator design framework using soft CGRA overlay. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. 56–63.

[60] Cheng Liu, Colin Lin Yu, and Hayden Kwok-Hay So. 2013. A soft coarse-grained reconfigurable array based high-level synthesis methodology: Promoting design productivity and exploring extreme FPGA frequency. In *Proc. 21st Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*. 228–228.

[61] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. 2003. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *Proc. 13rd Int. Conf. Field Program. Logic Appl. (FPL)*. 61–70.

[62] Thomas Moscibroda and Onur Mutlu. 2009. A case for bufferless routing in on-chip networks. *ACM SIGARCH Comput. Archit. News* 37, 3 (2009), 196–207.

[63] Roger Moussali, Nabil Ghanem, and Mazen A. R. Saghir. 2007. Supporting multithreading in configurable soft processor cores. In *Proc. Int. Conf. Compilers, Archit. and Synthesis for Embedded Syst. (CASES)*. 155–159.

[64] Kalin Ovtcharov, Ilian Tili, and J. Gregory Steffan. 2013. TILT: A multithreaded VLIW soft processor family. In *Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL)*. 1–4.

[65] Kolin Paul, Chinmaya Dash, and Mansureh Shahraki Moghaddam. 2012. reMORPH: A runtime reconfigurable architecture. In *Proc. 15th Euromicro Conf. Digit. Syst. Design (DSD)*. 26–33.

[66] Franjo Plavec, Blair Fort, Zvonko G. Vranesic, and Stephen Dean Brown. 2005. Experiences with soft-core processor design. In *Proc. 19th Int. Parallel and Distrib. Process. Symp. (IPDPS)*. 4–pp.

[67] Rafat Rashid. 2015. *A Dual-Engine Fetch/Compute Overlay Processor for FPGAs*. Ph.D. Dissertation. University of Toronto.

[68] Rafat Rashid, J Gregory Steffan, and Vaughn Betz. 2014. Comparing performance, productivity and scalability of the TILT overlay processor to OpenCL HLS. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. 20–27.

[69] Steve Rhoads. 2009. Plasma-most MIPS I(TM) opcodes. https://opencores.org/project,plasma.

[70] Graham Schelle, Jamison Collins, Ethan Schuchman, Perrry Wang, Xiang Zou, Gautham Chinya, Ralf Plate, Thorsten Mattner, Franz Olbrich, Per Hammarlund, et al. 2010. Intel nehalem processor core made FPGA synthesizable. In *Proc. 18th Int. Symp. Field Program. Gate Arrays (FPGA)*. 3–12.

[71] Aaron Severance. 2015. *Broadening the Applicability of FPGA-based Soft Vector Processors*. Ph.D. Dissertation. University of British Columbia.

[72] Aaron Severance, Joe Edwards, Hossein Omidian, and Guy Lemieux. 2014. Soft vector processors with streaming pipelines. In *Proc. 22nd Int. Symp. Field Program. Gate Arrays (FPGA)*. 117–126.

[73] Aaron Severance and George Lemieux. 2012. VENICE: A compact vector processor for FPGA applications. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. 261–268.

[74] Aaron Severance and Guy G. F. Lemieux. 2013. Embedded supercomputing in FPGAs with the VectorBlox MXP matrix processor. In *Proc. Int. Conf. Hardware/Software Codesign and Syst. Synthesis (CODES+ ISSS)*. 1–10.

[75] Sunil Shukla, Neil W. Bergmann, and Jürgen Becker. 2006. QUKU: A coarse grained paradigm for FPGAs. In *Proc. Dagstuhl Seminar.*
[76] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, et al. 1998. MorphoSys: A reconfigurable architecture for multimedia applications. In *Proc. XI Brazilian Symp. Integr. Circuit Design*. 134–139.
[77] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. 2011. The RISC-V instruction set manual, Volume I: Base user-level ISA. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62* (2011).
[78] David L. Weaver. 2008. *OpenSPARC Internals: OpenSPARC T1/T2 CMT Throughput Computing*. Sun Microsystems.
[79] Xilinx. 2017. MicroBlaze processor reference guide.
[80] Peter Yiannacouras, Jonathan Rose, and J. Gregory Steffan. 2005. The microarchitecture of FPGA-based soft processors. In *Proc. Int. Conf. Compilers, Archit. and Synthesis for Embedded Syst. (CASES)*. 202–212.
[81] Peter Yiannacouras, J. Gregory Steffan, and Jonathan Rose. 2006. Application-specific customization of soft processor microarchitecture. In *Proc. 14th Int. Symp. Field Program. Gate Arrays (FPGA)*. 201–210.
[82] Peter Yiannacouras, J Gregory Steffan, and Jonathan Rose. 2008. VESPA: Portable, scalable, and flexible FPGA-based vector processors. In *Proc. Int. Conf. Compilers, Archit. and Synthesis for Embedded Syst. (CASES)*. 61–70.
[83] Peter Yiannacouras, J Gregory Steffan, and Jonathan Rose. 2009. Fine-grain performance scaling of soft vector processors. In *Proc. Int. Conf. Compilers, Archit. and Synthesis for Embedded Syst. (CASES)*. 97–106.
[84] Jason Yu, Christopher Eagleston, Christopher Han-Yu Chou, Maxime Perreault, and Guy Lemieux. 2009. Vector processing as a soft processor accelerator. *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)* 2, 2 (2009), 12.
[85] Jason Yu, Guy Lemieux, and Christpher Eagleston. 2008. Vector processing as a soft-core CPU accelerator. In *Proc. 16th Int. Symp. Field Program. Gate Arrays (FPGA)*. 222–232.

Received September 2018; revised April 2019; accepted June 2019