FPGA Overlays Hardware based Computing for the Masses

Conference Paper · February 2018
DOI: 10.15224/978-1-63248-144-3-12

3 authors, including:

Xiangwei Li
Nanyang Technological University
3 PUBLICATIONS **4** CITATIONS

SEE PROFILE

Proc. of the Eighth International Conference On Advances in Computing, Electronics and Electrical Technology - CEET 2018

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

FPGA Overlays:
Hardware–based Computing for the Masses.

Xiangwei Li, Cheng Fei Phung and Douglas L Maskell

***Abstract*—The hardware acceleration of compute intensive applications has definite advantages, particularly in terms of energy and application latency. Heterogeneous programmable system-on-chip (SoCs) FPGA devices, which combine general purpose processors with reconfigurable fabrics, provide a compelling platform for IoT applications. However, FPGA devices are constrained due to significant design productivity issues and a lack of suitable hardware abstraction. For FPGAs to compete as general purpose computing platforms they must be better virtualized, as eliminating the need to work with platform-specific details would make them more accessible to application developers who are accustomed to software API abstractions and fast development cycles. In this paper, we discuss the role of overlay architectures for enabling general purpose FPGA application acceleration.**

***Keywords*— FPGA, Overlay architecture**

I. Introduction

Many of today's embedded applications, particularly in the internet of things (IoT) domain, require significant low-power pre-processing. While some of this processing can be performed by low power processors, possibly with cloud offloading, more compute intensive applications, such as image pre-processing for real-time driver assistance, require a different approach. Heterogeneous programmable systems on chip (PSoC) platforms, which tightly couple general purpose CPUs with reconfigurable FPGA fabric [1], provide significantly better power/performance characteristics than high performance CPUs and/or GPUs, particularly within the tight power budget required by an embedded system.

This reconfigurable computing approach is not new [2], [3], and while promising has had limited success in the past. However, the more capable multicore processors in newer PSoC devices provide the ability to move the focus away from static (or quasistatic) accelerators to a more software oriented view, where reconfiguration is a key enabler for reusing available hardware resources among multiple tasks. But this requires that the traditional approaches to managing the execution and scheduling of hardware tasks be re-examined as they are generally inappropriate and cumbersome. In addition, the hardware design process is complex, requiring low-level device expertise and specialist knowledge of both hardware and software systems, resulting in major design productivity issues.

This is because there is no suitable abstraction at the hardware computing level which is able to fully eliminate the reliance on platform-specific detail. The lack of platform abstraction and the resulting application portability is one of the main impediments to the adoption of these platforms for mainstream computing. Enabling the virtualized execution of software and hardware tasks on PSoC platforms would make them more accessible to application developers who are used to software API abstractions and fast development cycles. Hence we require a revised look at how to effectively exploit the key advantages of reconfigurable hardware while abstracting implementation details within a software-centric processor-based system.

One possible solution is to treat the execution and management of software and hardware tasks in the same way, using an OS or hypervisor which treats the FPGA as just another software-managed task [4], [5]. This enables more shared use, while ensuring better isolation and predictability. Complete system run-time management, including FPGA configuration and inter-process data communication, has been implemented using a hypervisor [6] and within the Linux OS [7]. The programmable coarse-grained hardware abstraction layer (or overlay) which is mapped to the FPGA, resulted in better application portability across devices, better design reuse, and rapid reconfiguration that is orders of magnitude faster than other reconfiguration approaches on FPGA.

In this paper, we discuss an execution platform based on a virtual overlay sitting on top of the physical FPGA fabric of a commercial hybrid FPGA that not only abstracts the reconfigurable hardware details, such as the logic, memory, and I/O interfaces and their placement, but also provides runtime management support in order to facilitate virtualized execution of software and hardware tasks. This enables small, often used, sections of code to be mapped to dedicated hardware accelerators on demand.

The remainder of the paper is organized as follows: Section II examines some of the barriers that must be overcome before non-hardware practitioners can fully embrace FPGA design. Section III introduces the concept of coarse-grained overlays followed by a description of the two main types of overlay. Section IV examines the use of FPGA overlay for general purpose application acceleration within a hybrid FPGA, while Section V examines the characteristics and implementation details of some typical overlays used for application acceleration. Finally, we conclude in Section VI.

II. Barriers to Mainstream use of FPGAs

To understand why FPGA devices have not achieved mainstream adoption among the wider computing community, it is important to appreciate the differences between FPGAs and alternative solutions, specifically

Xiangwei Li, Cheng Fei Phung and Douglas L Maskell
School of Computer Science and Engineering, NTU Singapore
Singapore

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

traditional CPUs. The most fundamental difference relates to how an application is mapped to these platforms. A CPU provides functionality to execute a compute kernel as a list of sequential instructions, whereas an FPGA architecture implements compute kernels by mapping them to fine grained resources, such as configurable logic blocks, and medium grained hard DSP blocks, Block RAMs, etc. These resources are connected via a fine-grained programmable routing network to create a specialized datapath which implements the compute kernel. By exploiting parallelism in the algorithm, significant performance gains are possible.

A. *Low Level Hardware Design*

The biggest difference between programmable hardware devices and other hardware implementations, such as ASIC devices, is that the former are user configurable, and in the case of SRAM-based FPGAs, dynamically reconfigurable. This allows the FPGA to adapt to changing processing requirements, thus better utilizing the FPGA resources, while providing a more software centric approach to hardware design. The typical flow starts with a large software application, which is profiled and partitioned into hardware and software components, with the resulting hardware accelerator running on the FPGA fabric and the remaining software running on the CPU. Significant performance improvements are possible due to hardware parallelization and pipelining even though the FPGA may run at a much slower clock speed than the CPU.

The hardware accelerator is then typically designed at a low level of abstraction (register-transfer-level (RTL)) to achieve an efficient implementation. This can consume significant time and make design reuse difficult when compared to a similar software only design. The designer manually converts the compute kernel into a fully pipelined datapath, specified using a hardware description language (HDL) such as Verilog or VHDL. The detailed structure of the datapath and the control needed for reading inputs from memories into buffers, stalling the datapath when buffers are full or empty, writing outputs to memory, etc., must be defined. In FPGA, a datapath implementing just several lines of C code may require 2-3 orders of magnitude more lines of HDL code, but results in much better performance by pipelining and exploiting parallelism. However this performance comes at the cost of significant design effort.

High-level synthesis (HLS) has been widely adopted by EDA vendors to address design productivity. HLS has helped to raise the level of programming abstraction from RTL to high level languages, such as C or C++, allowing designers to focus on high-level functionality instead of low-level details. However, low-level design effort is often still required to achieve the desired performance, making FPGA design difficult for non-experts. Also, while HLS tools have simplified the design process, the back-end flow (specifically the FPGA place and route) requires very long compilation times, particularly for large designs, further contributing to a lack of productivity and the main-stream adoption of FPGAs. In many cases, the time required to change an FPGA configuration limits hardware accelerators to predesigned static (fixed) implementations, negating the fundamental benefit of FPGAs.

Additionally, FPGA designs do not necessarily port well to the next hardware generation, making reconfigurable systems more difficult to work with. The designer must make a number of decisions, such as how to best fit the application to the device, including the datapath structure and the amount of parallelism. Applications are normally optimized for a specific target device, and are unable to execute on a smaller device or cannot take full advantage of the additional resources on a larger device.

B. *Reconfiguration Latency*

As mentioned earlier, SRAM-based FPGAs are able to partially and dynamically reconfigure the functionality of the FPGA fabric. However, despite the popularity and inherent capability of FPGAs for partial reconfiguration, whereby the FPGA operation is dynamically adapted to changing application requirements, this feature is not well supported by FPGA vendors and is hampered by slow reconfiguration times, poor CAD tool support, and large configuration file sizes. These issues make dynamic reconfiguration difficult and inefficient, resulting in most FPGAs being used with just a single configuration.

Dynamic partial reconfiguration (DPR) reduced the configuration time by allowing a smaller region of the FPGA fabric to be dynamically reconfigured at runtime. This provided a way of virtualizing the FPGA to allow the implementation of applications that are larger than the FPGA. DPR improved reconfiguration performance [8], however the efficiency of the traditional design approach for DPR is heavily impacted by how a design is partitioned and floor planned [9], tasks that again require FPGA expertise. Furthermore, the commonly used configuration mechanism is highly sub-optimal in terms of throughput [10]. Despite numerous efforts in reducing reconfiguration times and improving CAD tool support for dynamic reconfiguration of the FPGA fabric [11], [8], the implementation of rapidly reconfigurable hardware accelerators is still difficult and time consuming with application kernel swap times orders of magnitude more than that of a CPU context switch.

III. Coarse-Grained Overlays

Coarse-grained reconfigurable overlays implemented on top of a commercial FPGA devices, as shown in Fig. 1, have recently been explored as a means for addressing some of the problems seen with established FPGA-based hardware design. These overlays allow coarse-grained components, specifically the functional units (FUs) and interconnect to be modified at runtime according to application requirements. Coarse-grained overlays have several potential advantages, including: improved designer productivity, better design portability, software-like programming and fast application switching. This is because programs can be compiled to the overlay several orders of magnitude faster than that for the fine grained FPGA on which the overlay is implemented. That is, instead needing a full cycle through the FPGA vendor tools, overlay architectures present a simpler problem, that of programming an interconnected array of FUs. However, overlays are not intended to replace HLS tools and vendor implementation tools and are instead intended to support FPGA usage models where programmability, abstraction, resource sharing, fast compilation, and design productivity are critical issues.

Proc. of the Eighth International Conference On Advances in Computing, Electronics and Electrical Technology - CEET 2018

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

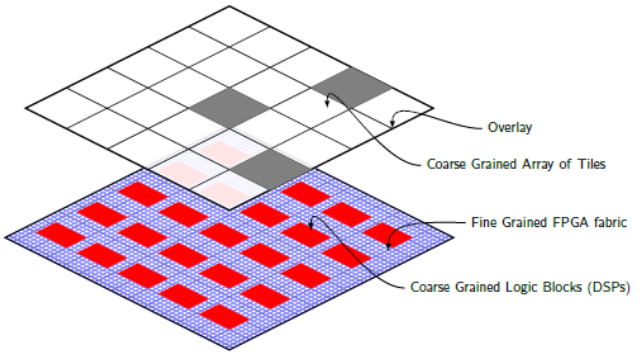


Figure 1. A course-grained overlay on top of a fine-grained FPGA.

Overlays come in many forms, with the majority falling within one of just two classes: spatially configured (SC) and time multiplexed (TM), with both the FU and interconnect also falling within one of these two categories.

A. *Spatially Configured Overlays*

SC overlays fully unroll the computational kernel and allocate each kernel operation to an individual FU such that an FU executes a single arithmetic operation and data is transferred over a dedicated point-to-point link between FUs. That is, both the FU and the interconnect are fixed while a compute kernel executes. Pipelining can then be used, both between and within FUs, to provide high throughput data computations which result in a fully pipelined, throughput oriented programmable datapath executing one kernel iteration per clock cycle. Thus an SC overlay has an initiation interval (II) of one.

SC overlays are usually characterised by their interconnect type, with the most common being: island style (IS) [12], [13], nearest neighbour (NN) [14], and to a lesser extent linear interconnect [15], [16], [17]. However, many IS and NN connected overlays suffer from very high FPGA resource overheads due to the interconnect network complexity, and are unsuitable for large compute kernels due to the limited size of the overlay that can be mapped onto the FPGA fabric. However, SC overlays do have a number of advantages, such as the ability to exploit larger FPGAs to deliver scalable performance for data-parallel and throughput oriented applications. They are able to maintain extremely high throughput by employing deep pipelining within the architecture, as well as having drastically reduced compilation times and configuration data sizes due to the requirement for just one instruction per functional unit. But this flexibility comes at a cost in terms of area and performance overheads. Hence, a significant amount of research effort has recently been aimed at reducing area overheads and improving performance.

B. *Time-Multiplexed Overlays*

TM overlays, on the other hand, are able to change their behaviour at each clock cycle, thereby reusing the resources allocated to the FU and interconnect. However, this multiplexing of the resource leads to a higher II and hence a reduced throughput.

Most successful TM overlays are based on soft processors, and include single-issue, multi-threaded and parallel processors. Single-issue processors, such as Micro-Blaze [18], Nios II [19], RISC-V [20] and Leon-4 [21] provide the benefits of software programmability and hardware re-usage. However, compared to hard processors and dedicated FPGA accelerators, they have significantly worse power and performance characteristics and may not meet the requirements of high speed applications. To improve power consumption and throughput, smaller and faster processor architectures, such as the iDEA processor [22], have been proposed. Examples of multi-threaded and parallel processors include: CUSTARD [23], Octavo [24] and SIMD-Octavo [25], The VectorBlox MXP soft vector processor [26] and the TILT VLIW processor [27].

Current processor-based overlay research is exploring multi-core systems of soft processors with efficient routing to improve processor throughput. Examples include GRVI Phalanx [28], a massively parallel overlay based on the RISC-V processor and the Hoplite NOC [29] which mapped 1680 RISC-V cores onto an UltraScale+ VU9P, and the 120-core microAptiv MIPS Overlay targeting the Stratix V GX FPGA [30]. These overlays have the advantage of a well-known, well-designed ISA which makes them easy to use.

An alternative solution is to build arrays of customized TM FUs and interconnect on the FPGA, similar to CGRAs [31]. As with SC overlays, array-based TM overlays mainly utilise IS [12], [13], NN [32], [33] and to a lesser extent linear interconnect [3], [9], for connecting between the TM FUs. Again, as with SC overlays, the overhead of the interconnect network, particularly for IS and NN interconnects, contribute to a significant FPGA resource utilization. Examples of CGRA-like TM overlays include: CARBON [34], reMORPH [32], SCGRA [33], the MINs Overlay [35] and a TM DSP-based overlay with linear interconnect [36]. Many of these overlays do not support fast application context switching as they rely on either full or partial reconfiguration at runtime when the compute kernel needs to change.

CGRA-based overlays aimed at addressing FPGA design productivity have only appeared in the last 6 to 8 years, but have already shown potential in terms of speed and area-efficiency. This is likely to improve as more and more coarse-grained modules, such as DSP blocks and BRAMs, are fabricated in modern FPGAs.

IV. Application Acceleration on FPGA Overlays

These overlays can then be integrated into a complete system for application acceleration. In this section, we will examine two possible scenarios.

Hypervisor Control: The first uses a modified CODEZERO hypervisor [5], [6] running on the host processor (in this case the dual-core ARM processor on the Xilinx Zynq FPGA) to provide run-time management, including overlay configuration and data communication, as in Fig. 2. This provides a significant advantage over conventional FPGA accelerators as it now allows the use of multiple independent accelerator kernels, which can be very quickly mapped to the overlay on demand, with software-like context switch times, as the application runs. Due to the long FPGA configuration times, conventional FPGA accelerators usually require all accelerator cores to be present on the FPGA fabric. This results in the need for a large FPGA device, negating any power and cost advantages

Proc. of the Eighth International Conference On Advances in Computing, Electronics and Electrical Technology - CEET 2018

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

associated with the use of these hardware accelerators. Even if using dynamic partial reconfiguration, the delay in swapping between accelerator implementations, is much too slow for many applications.

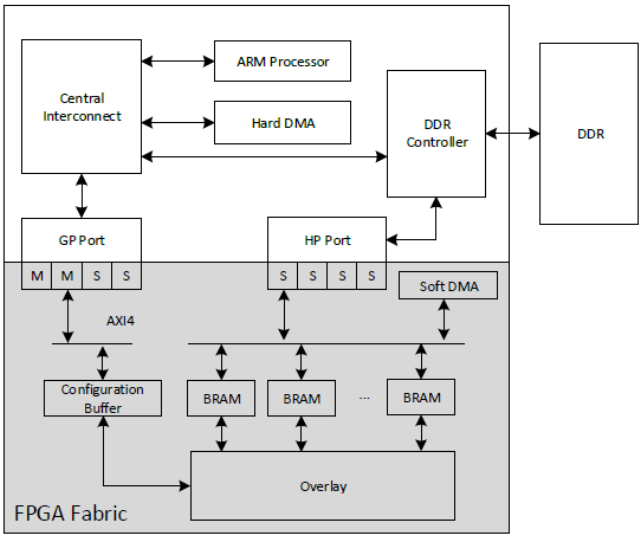


Figure 2. The hypervisor based overlay on Zynq.

The ARM-based hypervisor was able to support multiple hardware and software tasks running in different hypervisor containers. The FPGA bitstream describing the 5×5 overlay of [13], the four dual-port 512×64-bit input BRAMs, the single dual-port 512×64-bit output BRAM, the configuration buffer and an FPGA-based Xilinx softcore DMA engine connected to the 64-bit HP port, described in Fig. 2, are loaded (once only) at power-on as the hypervisor boots. The total configuration time is approximately 31ms. Multiple applications are then able to be scheduled to the overlay which has a configuration size of 287 Bytes (independent of the application kernel). The overlay is configured using the configuration buffer, via the GP port, and takes 11.5µs.

The execution profile for tasks running on the overlay is shown in Fig. 3. In this case, the first kernel requires 20.5µs for input data transfer, while data processing takes 2µs and output data transfer takes a further 5.12µs. This data transfer process repeats until the task is finished or the kernel is preempted. Upon kernel preemption, the hypervisor unlocks the overlay, performs a hardware context switch and locks it for the next task, which takes 5.4µs (the worst case when the two applications are in containers which are both running on the same core). The hypervisor then schedules the second kernel to the overlay, which again requires 287 Bytes to be sent to the configuration buffer and takes 11.5µs. Here, the second kernel has an input data transfer which takes 10.25µs, while data processing takes 2µs and output data transfer takes 5.12µs. Again, the data transfer process repeats until the task is finished or the FFT kernel is preempted.

From this simple example, it can be seen that the time to configure the overlay, perform a hardware context switch, and reconfigure the overlay for the next kernel is relatively insignificant (assuming that multiple data packets would be processed before a kernel pre-emption). However, this data transfer/process/transfer cycle reveals that the DMA based data transfer is a major bottleneck. This is using a relatively fast FPGA soft core DMA engine which is 4–5× faster than the ARM processor's hard DMA. This transfer time could be improved by replicating DMA controllers and using all four HP ports, overlapping communication and computation, or by implementing a streaming interface directly to/from the FPGA using PCIe interfaces. However, the important point to take away is that the overlay is not the bottleneck, and is now able to adequately support general purpose hardware acceleration on FPGA. This has been achieved by removing the need for dynamic reconfiguration of the overlay (as is needed by many of the other overlays proposed in the literature).

The PYNQ Project: Python productivity for Zynq (PYNQ) [37] is an open-source project which provides a simple platform independent method to design high performance embedded applications on the Zynq FPGA. It consists of API accessible reconfigurable libraries (or overlays) which are programmable using Python in a browser based Jupyter Notebook.

PYNQ has been quickly adopted by the reconfigurable computing research community. A framework for SPARK execution on PYNQ has been proposed to accelerate machine learning, achieving up to a 11x speedup compared to the application running on just the ARM processor [38]. PYNQ was used to implement a deep recurrent neural network using the AXI Stream interface, which achieved a throughput of 20 Giga operations per second (GOPS) [39]. Dynamic partial reconfiguration of PYNQ overlays resulted in a 40% reduction in the resource consumption [40].

v. Overlay Case Studies

A number of different FPGA overlays developed by the research group which can be integrated into the hypervisor based ZYNQ system for supporting hardware application acceleration are described.

DISO: A DSP block based Island-Style Overlay: An island-style overlay using Xilinx DSP48E1 primitives to implement a programmable FU in an efficient overlay architecture targeting data-parallel compute kernels was presented in [41]. The overlay consists of tiles and borders,

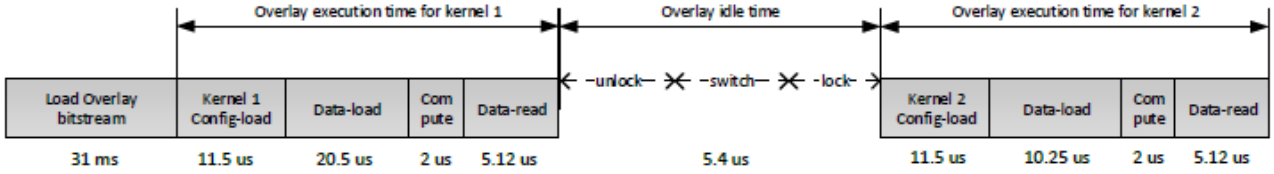
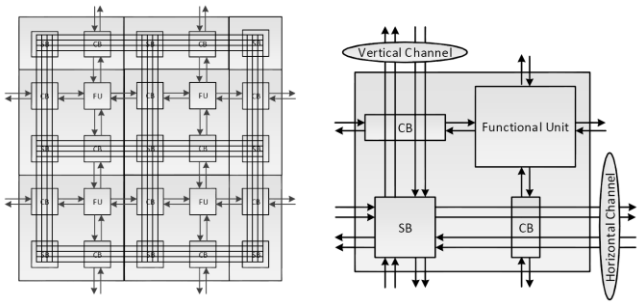


Figure 3. The execution profile of tasks on overlay under hypervisor control.

Proc. of the Eighth International Conference On Advances in Computing, Electronics and Electrical Technology - CEET 2018

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

where a tile contains virtual routing resources and an FU, as shown in Fig. 4. The simple 2×2 overlay of Fig. 4, requires 4



FUs, 9 SBs and 8 CBs. Extrapolating, an N×N overlay requires N^2 FUs, $(N+1)^2$ SBs and N^2+2N CBs.

(a) The 2x2 overlay (b) The tile architecture

Figure 4. The DISO overlay architecture.

The 16-bit DSP-based FU consists of DSP block, MUX based reordering logic and variable length synchronization logic for balancing pipeline latencies, as shown in Fig. 5. The FU has 4 input and 4 output ports. The reordering logic is a mux-based 4x4 crossbar switch providing connectivity between the FU inputs and the DSP block. The variable length synchronization logic is implemented using SLICEM shift register LUTs (SRLs) for maximum performance.

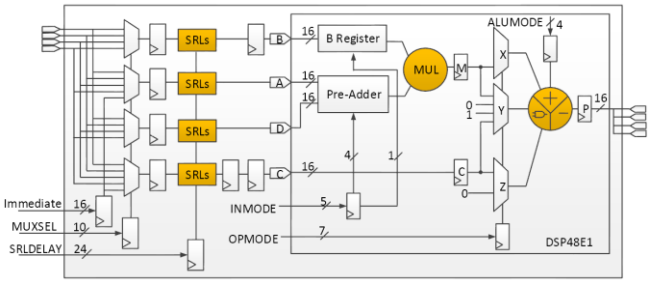


Figure 5. The DISO FU.

The operating frequency, peak throughput and FPGA resource usage for different size overlays on the ZYNQ device are given in Fig. 6. DISO is able to achieve an operating frequency close to the maximum frequency possible on the ZYNQ device, but for an 8x8 overlay, it uses approximately 52% of the LUTs available while achieving a peak throughput of 65 GOPS. The DISO overlay has an overhead of 430 LUTs/GOPS, which while significantly less than other overlays from the literature [12], [42], still represents a very significant hardware utilisation.

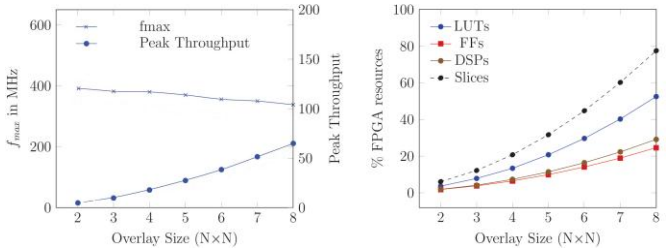


Figure 6. The frequency, throughput and resource usage for different size DISO overlays.

A significant advantage of the coarse-grained DSP-based DISO overlay is that it can perform a context switch in just 45.5µs using just 1137 bytes of configuration data.

DeCO: A DSP enabled Cone-shaped Overlay: Many of the existing overlays use a fully flexible general-purpose interconnect which in many cases represents an over-provision and is not normally required for implementing accelerators based on feed-forward pipelined datapaths. Instead, a linear array of interconnected FUs was proposed [17] to improve resource utilization. Additionally, it was observed that the greater majority of these feed-forward applications had a triangular (or cone shaped) FU pattern. By analysing a number of simple computational kernels, we derived a processing pipeline consisting of 20 DSP block based FUs and four layers of simple interconnection, as shown in Fig. 7. The FU is similar to the DISO FU except that the SLICEM shift register LUTs are not needed the dataflow through the overlay is self-synchronising.

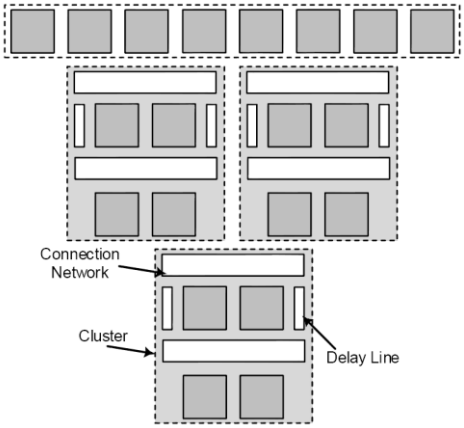


Figure 7. The DeCO feed-forward overlay.

The 16-bit DeCO overlay requires 1368 LUTs, 2348 FFs (1032 logic slices) and 20 DSP blocks, with an operating frequency of 395 MHz and a peak throughput of 23.7 GOPS. This represents an overhead of just 58 LUTs/GOPS (an order of magnitude less than the DISO overlay) and just 2-3 times that when the application is directly implemented on FPGA. A hardware context switch can be performed in just 2µs requiring just 54 bytes of configuration data

TM Overlay with Linear interconnect: While DeCO represents a significant step forward in terms of overlay resource efficiency, we also explore low resource TM overlays [36] with linear interconnect, to minimize the FPGA resource usage. The major advantage of TM overlays is that an application kernel can be mapped to fewer FUs, reducing resource consumption at the expense of II.

In the linear TM overlay there is a direct connection between FUs. It consists of a streaming data interface made up of Distributed RAM (DRAM) acting as a FIFO, which feeds into a cascade of time-multiplexed FUs, with another DRAM-based FIFO at the output, as shown in Fig. 8. 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 [22], and requires 1 DSP block, 212 LUTs and 228 FFs and runs at 323 MHz on a Xilinx Zynq. It consists of a LUTRAM-based instruction memory (IM) and register file (RF), and a DSP-based ALU, as shown in Fig. 9.

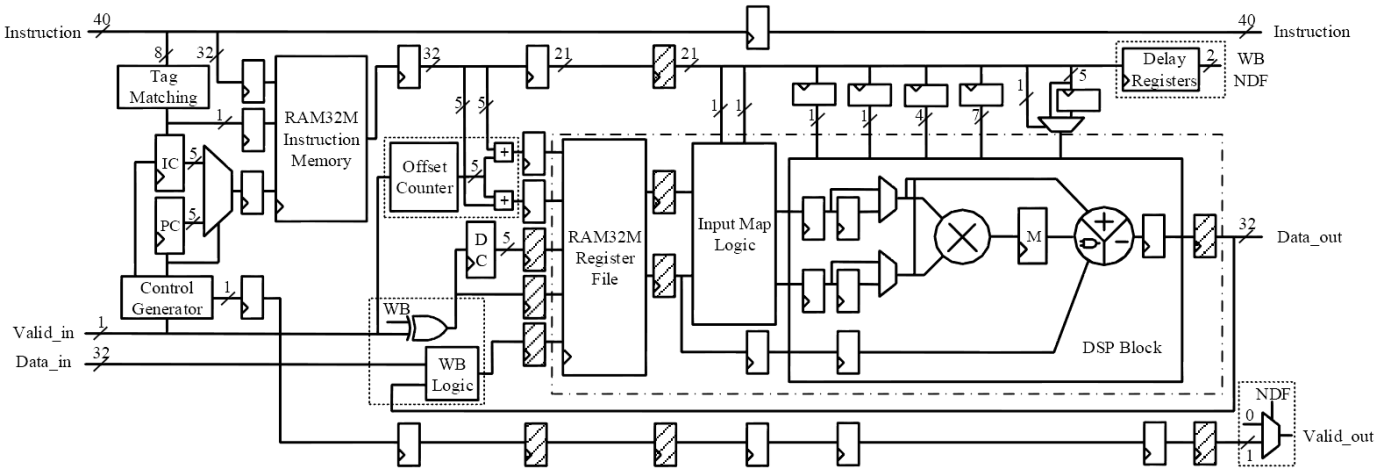


Figure 9. The DSP block based time-multiplexed functional unit.

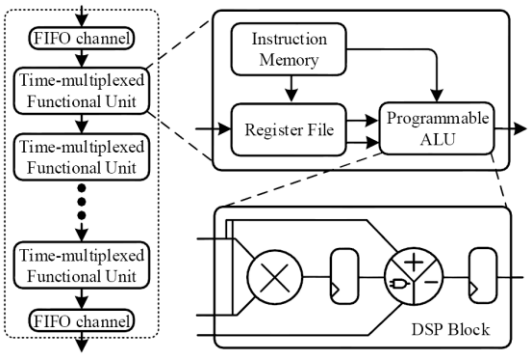


Figure 8. The TM overlay with linear interconnect

Cascading 8 FUs into a linear overlay results in a resource consumption of 1747 LUTs and 1954 FFs (814 logic slices) and 8 DSPs, and operates at a frequency of 286MHz. While this represents a 21% reduction in resource utilisation compared to DeCO, it comes at the expense of a significant reduction in the throughput and the II. The TM overlay has an average throughput of 0.63 GOPS compared to an average throughput of 6.1 GOPS for DeCO, representing an order of magnitude reduction in throughput.

VI. Conclusions

We have examined the use of overlays, a virtual abstraction on top of the conventional FPGA fabric, for general purpose on-demand application acceleration. We examiner a hypervisor-based implementation targeting the ZYNQ platform which embedded a DSP-based overlay within the FPGA platform for use as a rapidly reconfigurable general purpose accelerator under software control. Here we saw that even with an efficient DMA controller, data transfer, and not the overlay, was the bottleneck, clearly showing the benefits of an overlay for supporting hardware acceleration of tasks. We presented a number of overlays with varying characteristics and efficiencies (both in terms of throughput and FPGA resource utilisation) In the future, we plan to investigate techniques for overcoming the data communication bottleneck, and examine the power and cost benefits of accelerator overlays.

Acknowledgment

This research is supported by the Singapore Ministry of Education under research grants MOE_RG183/14 and MOE2017-T2-1-002.

References

[1] S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, "A 16-nm multiprocessing system-on-chip fieldprogrammable gate array platform," IEEE Micro, vol. 36, no. 2, pp. 48–62, 2016.

[2] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures, " Proceedings of the IEEE, vol. 103, no. 3, pp. 332–354, 2015.

[3] S. M. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," Proceedings of the IEEE, vol. 103, no. 3, pp. 318–331, 2015.

[4] N. W. Bergmann, S. K. Shukla, and J. Becker, "QUKU: a duallayer reconfigurable architecture," ACM Transactions on Embedded Computing Systems (TECS), vol. 12, no. 1s, pp. 63:1–63:26, Mar. 2013.

[5] A. K. Jain, K. D. Pham, J. Cui, S. A. Fahmy, and D. L. Maskell, "Virtualized execution and management of hardware tasks on a hybrid ARM-FPGA platform," J. Signal Process. Syst., vol. 77, no. 1–2, pp. 61–76, 2014.

[6] K. D. Pham, A. K. Jain, J. Cui, S. A. Fahmy, and D. L. Maskell, "Microkernel hypervisor for a hybrid ARM-FPGA platform," in Proceedings of the International Conference on Application-Specific Systems, Architecture Processors (ASAP), 2013, pp. 219–226.

[7] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou, "A fully pipelined and dynamically composable architecture of CGRA," in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2014, pp. 9–16.

[8] K. Vipin and S. A. Fahmy, "Mapping adaptive hardware systems with partial reconfiguration using CoPR for Zynq," in Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS), June 2015, pp. 1–8.

[9] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC), 2012, pp. 13–25.

[10] K. Vipin and S. A. Fahmy, "A high speed open source controller for FPGA partial reconfiguration," in Proceedings of International Conference on Field Programmable Technology (FPT), 2012, pp. 61–66.

ISBN: 978-1-63248-144-3 doi: 10.15224/ 978-1-63248-144-3-12

[11] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq, " IEEE Embedded Systems Letters, vol 6(3), pp. 41–44, 2014.

[12] G. Stitt and J. Coole, "Intermediate fabrics: Virtual architectures for near-instant FPGA compilation," IEEE ESL, vol. 3(3), pp. 81–84, 2011.

[13] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient Overlay architecture based on DSP blocks, " in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2015, pp. 25–28.

[14] D. Capalija and T. S. Abdelrahman, "A high-performance overlay architecture for pipelined execution of data flow graphs," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2013, pp. 1–8.

[15] J. Coole and G. Stitt, "Adjustable-cost overlays for runtime compilation, " in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2015, pp. 21–24.

[16] D. Capalija and T. Abdelrahman, "Towards synthesis-free JIT compilation to commodity FPGAs," in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2011, pp. 202-205.

[17] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmyb "DeCO: A DSP block based FPGA accelerator overlay with low overhead interconnect," in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2016, pp. 1-8.

[18] Xilinx, "Microblaze processor reference guide," 2017.

[19] Altera, "Nios II processor reference handbook," 2016.

[20] A. Waterman. Design of the RISC-V Instruction Set Architecture. University of California, Berkeley, 2016.

[21] Cobham Gaisler AB, GRLIP IP Core User's Manual, 2017.

[22] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, "The iDEA DSP block-based soft processor for FPGAs," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 7, no. 3, pp. 19:1– 19:23, 2014.

[23] R. Dimond, O. Mencer, and W. Luk, "CUSTARD-a customisable threaded fpga soft processor and tools," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2005, pp. 1–6.

[24] C. E. LaForest and J. G. Steffan, "Octavo: an fpga-centric processor family, " in Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA), 2012, pp. 219–228.

[25] C. E. Laforest and J. H. Anderson, "Microarchitectural comparison of the MXP and Octavo soft-processor FPGA overlays,"ACM TRETS, vol. 10, no. 3, p. 19, 2017.

[26] A. Severance and G. G. Lemieux, "Embedded supercomputing in fpgas with the vectorblox mxp matrix processor, " in Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013, pp. 1–10.

[27] K. Ovtcharov, I. Tili, and J. G. Steffan, "Tilt: a multithreaded vliw soft processor family," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2013, pp. 1–4.

[28] J. Gray. "Grvi phalanx: A massively parallel RISC-V FPGA accelerator accelerator," in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2016, pp. 17–20.

[29] N. Kapre and J. Gray. "Hoplite: Building austere overlay NoCs for FPGAs," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2015, pp. 1–8.

[30] K. HB Chethan, P. Ravi, G. Modi, and N. Kapre. "120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board." In Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), 2017, pp. 141–146.

[31] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2003, pp. 61–70.

[32] K. Paul, C. Dash, and M. S. Moghaddam. "reMORPH: a runtime reconfigurable architecture," In Euromicro Conference on Digital System Design (DSD), 2012, pp. 26–33.

[33] C. Liu, C. L. Yu, and H. K.-H. So. "A soft coarse-grained reconfigurable array based high-level synthesis methodology: Promoting design productivity and exploring extreme FPGA frequency," in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 2013, pp. 228–228.

[34] A. D. Brant, "Coarse and fine grain programmable overlay architectures for FPGAs," PhD thesis, University of British Columbia, 2013.

[35] R. Ferreira, J. G. Vendramini, L. Mucida, M. M. Pereira, and L. Carro, "An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture," in Proceedings of the 14th International Conference on Compilers, architectures and synthesis for embedded systems (CASES), 2011, pp. 195–204.

[36] X. Li, A. K. Jain, D. L. Maskell and S. A. Fahmy, "A Time-Multiplexed FPGA Overlay with Linear Interconnect," to appear in Proceedings of the Design, Automation and Test in Europe Conference (DATE), 2018.

[37] Pynq: Pyhton productivity for Zynq. Technical report, 2017.

[38] K. Elias, I. Stamelos, C. Kachris, and D. Soudris, "Spark acceleration on FPGAs: A use case on machine learning in Pynq," in Proceedings of the international conference on Modern Circuits and Systems Technologies (MOCAST), 2017, pp. 1–4.

[39] Y. Hao, , and S. Quigley, "The implementation of a Deep Recurrent Neural Network Language Model on a Xilinx FPGA," arXiv preprint arXiv:1710.10296 (2017).

[40] J. Benedikt, P. Zimprich, and M. Hübner, "A dynamic partial reconfigurable overlay concept for PYNQ," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–4.

[41] A. K. Jain, X. Li, S. A. Fahmy, and D. L. Maskell, "Adapting the DySER architecture with DSP blocks as an Overlay for the Xilinx Zynq," ACM SIGARCH Computer Architecture News vol. 43(4), pp. 28–33, 2016.

[42] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "DySER: Unifying functionality and parallelism specialization for energy-efficient computing," IEEE Micro, vol. 32, no. 5, pp. 38–51, 2012.