A SLM-based Overlay Architecture for Fine-grained Virtual FPGA

Theingi Myint1, Motoki Amagasaki1, Qian Zhao2, and Masahiro Iida1

Abstract FPGA overlay technologies have been introduced to provide inter-FPGA bitstream compatibility by implementing virtual FPGA (vFPGA) layers on physical devices. Conventional LUT-based fine-grained vFPGAs have very large resource overheads. In this paper, we propose a fine-grained vFPGA overlay architecture that employs our previously proposed scalable logic module (SLM) as a logic cell. SLMs can cover most frequently used logics with far fewer hardware resources than LUTs. Evaluation results show that a 7-input SLM-based vFPGA can reduce LUT and flip-flop resource usage by up to 32% and 35% on an Artix-7 FPGA, 30% and 35% on a Kintex-7 FPGA, and 30% and 35% on a Kintex UltraScale+ FPGA respectively, as compared to a LUT-based vFPGA of the same input size.

key words: reconfigurable device, FPGA, overlay, virtualization, SLM

Classification: Circuits and modules for electronic instrumentation

1. Introduction

More and more different types of FPGAs are expected to be employed by a variety of applications in the future. However, configuration bitstreams of FPGAs are mutually incompatible, which will cause huge development, deployment, and management costs in future FPGA designs. We focus on a virtual FPGA (vFPGA) overlay architecture for providing bitstream compatibility between FPGA devices. A vFPGA is a soft core synthesized and implemented on a commercial FPGA.

Fig. 1 shows an example of vFPGA system. A vFPGA design is compiled with a vendor (Xilinx FPGAs are used in this work) CAD flow, then implemented on a commercial FPGA. Once a vFPGA is deployed, we can implement user designs incorporating high-level synthesis (HLS), domain-specific language (DSL), hardware-description language (HDL), or netlist with a virtual FPGA CAD flow to obtain a portable bitstream that can be dynamically loaded onto the vFPGA of any physical FPGA. Using a vFPGA has benefits such as design portability, resource abstraction, faster configuration times, and separated control from the physical FPGA.

There are two types of vFPGA overlay architectures: fine-grained and coarse-grained types [1]. Coarse-grained overlays commonly abstract DSP resources for computational applications, providing lower performance overhead but limited implementable applications. Fine-grained overlays can implement any application circuit but have large resource and performance overheads because a k-input LUT has $2^k$ configuration memory bits, which are all implemented with limited flip-flops (FFs) on the physical FPGA. Although [2] proposed a method to efficiently utilize physical RAM on an FPGA’s LUT to implement vFPGA LUTs, that method is heavily reliant on the physical FPGA structure (e.g., LUT size), limiting bitstream portability. According to our previous work [3], we developed the scalable logic module (SLM) architecture as a logic cell requiring fewer configuration memories. In that architecture, the minimum number of configuration memory bits for a k-input SLM can be less than $2^{k+1}$. In this work, we propose a fine-grained vFPGA overlay architecture based on SLM because SLM-based vFPGA can implement any user circuit (fine-grained), and then compare with LUT-based vFPGA for resource and delay.

Fig. 1. The vFPGA system as an example.

The remainder of this paper is structured as follows. Section 2 mentions related works. Fine-grained vFPGA is explained in section 3. The detail of our architecture and CAD flow is presented in section 4. Section 5 shows the evaluation conditions and performance results. And we will conclude our work, and mention the future work in section 6.

2. Related Works

FPGA overlays have been researched for a variety of purposes such as improving the productivity of application
design, portability of circuit, and providing other advanced capabilities in low cost such as high reliability and system debugging. Related works covering the above topics are introduced in this section.

2.1 Overlays for high productivity and portability
By enabling to target a software-programmable substrate instead of the underlying FPGA, overlays are used to improve productivity by reducing the application time of compiling. Coarse-grained overlays have benefits of high productivity for fast compilation and software-like programmability. But throughput oriented spatially configurable overlays implement one function unit for each compute kernel, which has large area overhead. So, Li et al. examined the sharing possibility of the functional units among kernel operations in order to reduce area overheads [4]. They proposed a linear interconnected array of time-multiplexed functional units as an overlay architecture, which reduces the instruction storage and interconnection of resource requirements.

Rashid et al. compared the performance, development and scalability of two C-to-FPGA approaches: TILT overlay processor and Altera’s OpenCL high-level-synthesis (HLS) [5]. TILT [6] is a highly configurable overlay compute engine for FPGAs, which has varied and deeply pipelined floating point functional units. Their application-customized TILT was evaluated using five data-parallel benchmarks, and the results showed only 41% to 80% compute density can be achieved when compared with their best OpenCL HLS designs. However, after every code change, OpenCL HLS applications require full recompilation. TILT enables smaller, more area-efficient designs than OpenCL HLS when low to moderate throughput is sufficient.

A highly efficient processor are commonly required by overlay architectures. Ng et al. presented an open-source soft processor designed to tightly-couple with FPGA accelerators as part of an overlay framework [7]. RISC-V was chosen to be the instruction set for its openness and portability, and a four-stage pipelined soft processor was implemented to balance resource consumption and performance in the implementation of FPGAs. Especially, the processor is implemented to improve the portability and compatibility of design across different FPGA platforms. In their experiments, the proposed soft processor is small and efficient, and the tightly-coupled architecture can provide a unified programming model for software designers when they compared with the existing design.

C. Wang et al. proposed a novel coarse-grained reconfigurable array (CGRA) with hierarchical context cache structure and efficient cache management approaches, including time-frequency weighted (TFW) context cache replacement strategy and context multi-casting (CMC) mechanism. CGRA was implemented under the process of TSMC 65nm, frequency 200MHz, and area 23.2 mm² [8]. For fine-grained overlay implementation, Brant et al. proposed ZUMA in [2]. In this paper, the authors presented an open-source, cross-compatible embedded FPGA architecture, also called ZUMA open FPGA overlay architecture, intended to overlay on top of an existing FPGA (vFPGA on a physical FPGA). Such architecture has benefits including bitstream compatibility between different vendors and parts, compatibility with open FPGA tool flows, and the ability to embed some programmable logic into systems on FPGAs without requiring for releasing or recompiling the master netlist. ZUMA reduced area overhead by 40x than conventional vFPGAs, which requires 40 host LUTs per ZUMA embedded LUT (eLUT). ZUMA employs configurable LUTRAMs as basis for implementing both programmable LUTs and routing MUXs, the design of a Clos-style Input Interconnect Block (IBB) network.

The hardware virtualization of FPGA resources based on overlay architectures is proposed in [1]. They demonstrated virtualization techniques to improve portability, to speed up reconfiguration, and to promote resource abstraction. They extended the overlay architecture on top of commercial-off-the-shelf (COTS) FPGAs to bring novel features for improving management of hardware tasks and binary compatibility for heterogeneous FPGAs. A deployment platform that is capable of node-to-node application migration, and a software stack offering an operating system service is proposed. The proposed software stack has backward compatibility with a new overlay architecture.

For higher flexibility and portability, the virtualized FPGA resources are provided for user applications. To access FPGA accelerators, authors mentioned that multiple-user services are provided to a paravirtualized Xen Virtual Machine (VM) environment [9]. Authors presented dynamic reconfiguration of the FPGA depended on most of hardware virtualization schemes. The idea is decomposing the physical FPGA into several reconfigurable regions by using dynamic partial reconfiguration (DPR) [10, 11].

FPGA resources sharing is a requirement and promising technique to reuse resources for different configuration circuits. Feilen et al. proposed to switch between signal processing DVB-T2 tasks in a time multiplexed by using DPR [12]. To share circuit-specific DSP blocks from Xilinx, the authors described that multiple sets of DSP blocks are controlled by state machines to ensure that each set achieves a high initiation interval instead of reconfiguring a set of DSP blocks to implement all operations [13]. An approach based on FaRM (Fast Reconfigurable Manner) controller for an advanced scheduling of hardware tasks on a DPR-based resource to enable the pre-loading of the partial bitstream, and to reduce PR configuration time overhead is proposed in [14]. FPGA virtualization can be classified into three types: temporal partitioning, virtualized execution, and virtual machine [15]. To define the approach of splitting up applications into multiple communicating tasks and using a run-time system to manage, authors used the virtualized execution in order to support device independence within a device family [16], and it is used not only for higher productivity of design, isolation and resource management but also for device independence between different devices.
The classification of FPGA virtualization is composed of three levels: node level, multi-node level, and resource level. In node level, a node is referred to as a single FPGA. A cluster of two or more FPGAs is considered as multi-node. The architecture virtualization and I/O virtualization are applied in this level that includes overlays [2, 17] for architecture abstraction and transparent I/O sharing.

2.2 Overlays for other capabilities
Psarakis addressed main features and challenges of designing reliability-aware overlay architectures that building fault tolerant SRAM-based FPGA systems [18]. Although many fault tolerant ways of SRAM-based FPGAs had been developed in the past [19], FPGA-based systems with the reliable design for critical environment applications were still evolving in research area. To build a fault tolerant design using triple-modular redundancy (TMR) technique, if an FPGA overlay is employed, FPGA designer will integrate with the design features of the specific overlay architecture. So, the authors investigated the reliability-aware overlay architectures with SRAM-based FPGA designs by analyzing the benefits of a customized FPGA overlay architecture.

In order to exploit the benefits of reconfigurable technology, a debug ecosystem is essential to provide effective visibility into the working design. So, overlays are the main role in the ecosystem. Eslami et al. discussed that an overlay fabric can be created and exploited for user to add debugging instrumentations to a design [20]. Overlays can provide a flexible and adaptable debugging system, and the overlay can be reconfigured without changing the underlying circuit of user. Their overlay can be optimized not only for debugging instrumentation for applications but also for underlying user circuits.

Overlays can support another level of programmability implemented on top of the low level FPGA resources, and overlay application can be translated into configurations of the underlying FPGA architectures [21, 22]. Although most researchers had made coarse-grained reconfigurable architectures (CGRAs) to achieve energy efficiency, the ease of programming and fast compilation, fine-grained FPGA overlays are requiring the design portability and resources reduction. Therefore, we propose SLM-based overlay architecture for fine-grained virtual FPGA in order to improve design portability and to reduce resources usage in our work.

3. Fine-grained vFPGA Architecture
Fine-grained virtual FPGAs commonly adopts the island-style architecture, in which reconfigurable elements are made up of configurable logic blocks (LBs) and routing resources [23, 24, 25]. For the design tool, the versatile place and route (VPR) [26] is widely used, which supports generic LUT-based architectures that are compatible with the standard architectures. As shown in Fig. 2, each logic cluster, also called cluster, has I inputs and N outputs. There are N basic logic elements (BLEs) included in a logic cluster. The inputs of BLEs are from local connection block (LCB) of I+N inputs, in which I inputs are from logic block (LB) inputs and N inputs are from feedbacks of BLE outputs. A BLE consists of a k-inputs LUT and one flip-flop. A look-up table (LUT) is used in conventional FPGAs as logic cell.

The SLM-based and LUT-based fine-grained vFPGAs are similar to the standard FPGA architecture with a tile array and input/output blocks (IOBs), as shown in Fig. 3. One tile includes one switch block (SB), two connection blocks (CBs: CB_X and CB_Y), and one logic block (LB). IOB is connected to routing channels with programmable switches. The designing, implementing and deployment of vFPGA hardware applications are not depending on physical FPGA devices or tools. A vFPGA corresponds to a set of reconfigurable elements available to the application, and the number of available reconfigurable elements are the main limitation for implementing large user circuits. In this paper, we propose SLM-based overlay architecture for fine-grained vFPGA that can reduce the resource overhead.

4. Proposed SLM-based vFPGA and CAD Flow
A SLM consists of an input controller, a small LUT core, and an output controller as shown in Fig. 4 [3]. In this figure, Programmable NAND (PN) indicates programmable NAND logic for controllers, and M is configuration memory. A SLM structure can be described as k-SLM (k-m, o), which means a k-input SLM with a (k-m)-LUT core, and the number of inputs with m-level controller (multilevel
controller) is \( \omega \), where \( 1 \leq m \leq k-2 \) and \( 0 \leq \omega \leq k-m \). Level means the number of extracted variables for Shannon expansion because there was a tradeoff between hardware resources and coverage [3]. 5-LUT is the best in the part of LUT for 2-level controller. The proposed 7-SLM (5, 0) structure as an example is shown in Fig. 5. For 6-input SLM using 4-input LUT as a logic core, and 7-input SLM using 5-input LUT as a logic core, there are no controllable inputs, that means \( \omega=0 \), and the other two inputs (\( \text{In}_0 \) and \( \text{In}_1 \)) are only connected to PN of output controller. We can see that two inputs (\( \text{In}_0 \) and \( \text{In}_1 \)) in this example are connected to 2 levels of PN, respectively. For determined number of inputs, the level of PNs will be deeper if a smaller LUT is used. Our previous research in [3] showed such SLM structure can cover most frequently used logics. Therefore, a \( k \)-SLM thus has fewer configuration memory cells than a \( k \)-LUT, and so can reduce resource usage for a vFPGA implementation. Similar to LUT-based FPGAs, there is a trade-off between area (resource usage) and delay (logical depth) when determining the input size of the logic cell for user circuit implementation. If we use a smaller SLM as a logic cell, the area overhead is smaller but the logical depth is larger. If we use a larger SLM as a logic cell, more resources are wasted when mapping small logics to SLMs. However, in most cases, the logical depth is same because one SLM can implement more logics.

Fig. 4. Example of SLM structure (\( k \)-SLM (\( k-1, k-1 \))) [3].

Fig. 5. Example of proposed SLM structure (7-SLM (5, 0)).

Fig. 6 shows developed CAD flow for vFPGA architecture exploration, vFPGA implementation, and user design. We used the open-source academic FPGA design tools ABC (for technology mapping) and VPR 7.0 (for clustering and placement) [29]. We modified our previously proposed EasyRouter [30] for vFPGA routing, vFPGA HDL generation, and user design bitsream generation.

5. Evaluation
In this section, we will show the architecture exploration results of SLM-based and LUT-based vFPGAs, then compare resource usage and delay performance of explored vFPGAs by implementing them on three different Xilinx FPGA devices.

5.1 Condition
We used a common FPGA architecture exploration flow, as shown in Fig. 6 (a), to find the optimal architecture parameters such as array size and channel width of SLM-based and LUT-based vFPGAs. The twenty largest MCNC circuits were used as benchmarks. The preset parameters, explored optimal parameters and property of explored architecture used to define a tile structure are listed in Table I. These parameters are commonly used for FPGA architecture definition [29, 31]. The cluster size indicates the number of logic cells implemented in one logic tile, which was set to 4 for all architectures. The number of cluster inputs is calculated with a common method, \( I=k(N+1)/2 \), in which \( k \) is the number of the logic cell inputs and \( N \) is the cluster size. \( F_c \) and \( F_r \) are routing structure parameters. \( F_c \) is 0.5, which means 50% of tracks in a routing channel are connected to the logic cluster. \( F_r \) is 3, which means an output multiplexer in the switch block can select an input from three directions for its output. Channel width (CW) means the number of routing tracks in a routing channel. The array size indicates the required number of logic tiles to implement a circuit. The total number of configuration memories in a tile is denoted as \#Conf mem.

| Architecture | Cluster size | Preset parameters | Exploded optimal parameters | Property of arch. |
|--------------|-------------|-------------------|----------------------------|-------------------|
| 5-LUT        | 4           | 0.5/3             | 52, 31 × 31               | 604               |
| 5-SLM(3, 0)  | 4           | 0.5/3             | 52, 32 × 32               | 532               |
| 6-LUT        | 4           | 0.5/3             | 56, 28 × 28               | 791               |
| 6-SLM(4, 0)  | 4           | 0.5/3             | 52, 29 × 29               | 625               |
| 7-LUT        | 4           | 0.5/3             | 54, 27 × 27               | 1,065             |
| 7-SLM(5, 0)  | 4           | 0.5/3             | 52, 27 × 27               | 603               |

Fig. 6 (b) shows the vFPGA implementation flow for both LUT-based and LUT-based vFPGAs. Three target physical FPGAs of different grades and generations were selected for evaluation, which are Xilinx Artix-7 xc7a200tffg1156-1
device, Kintex-7 xc7k420tffv11561 device and Kintex UltraScale+ xcku5p-fvb676-2-e device. All evaluation results are obtained from Xilinx Vivado 2018.2 routed design reports with an area-oriented compilation constraint.

5.2 Architecture exploration results
In this evaluation, we compare SLM-based vFPGA and LUT-based vFPGA tiles with input sizes from 5 to 7. The 5-SLM (3, 0), 6-SLM (4, 0), and 7-SLM (5, 0) are selected as SLM-based overlay architectures. By performing architecture exploration for all architectures with the same flow, preset parameters, and benchmark circuits, we derived explored optimal parameters and property listed in Table I. The explored CWs and array sizes listed in Table I are the largest values of all implemented circuits on a certain architecture. These parameters also mean the minimum values for implementing all circuits on that architecture. For CW results, the explored CWs of 5-LUT and 5-SLM (3, 0) are the same, but the total number of configuration memory bits of the 5-SLM (3, 0) is smaller, because fewer memories are required for logic cell implementation. For array size, the array size of 7-LUT and 7-SLM (5, 0) are the same, but 6-SLM (4, 0) and 5-SLM (3, 0) have larger array size than 5-LUT and 6-LUT. This is because SLMs have lower logic expression capabilities than LUTs of the same input size, so more tiles have to be used to implement the same circuit.

5.3 Implementation results
Table II lists resource usage and delay results for the evaluated tile architectures of Artix-7. We can see that the 5-SLM (3, 0), 6-SLM (4, 0) and 7-SLM (5, 0) architectures use 3%, 21% and 32% fewer physical LUTs than 6-LUT and 7-LUT architectures, respectively. Similarly, FF usage reductions are 12%, 21% and 35% respectively. The results show that an SLM-based tile reduces resource requirements as compared to a LUT-based tile of the same logic cell input size. Also, the FFs usage of the tile are less than the LUT tile when the input size are same. To evaluate critical path delays in vFPGA tiles, we measured critical path starts from tile inputs, which pass through logic elements in combination mode, then end at the tile outputs for each architecture. The results show that 5-SLM (3, 0), 6-SLM (4, 0), and 7-SLM (5, 0) are 1%, 2.7%, and 2% faster than 5-LUT, 6-LUT, and 7-LUT.

For the configuration path, all designs were successfully implemented at 100MHz, indicating that the configuration time is 10ns per configuration bit. Because configuration FFs in our vFPGA design are serially connected as shift registers, the configuration time is proportional to the total number of configuration memories. Configuration time can be improved by dividing configuration paths into more segments in the vFPGA design.

Table III shows the evaluation results of Kintex-7 xc7k420tffv1156-l, in which 5-SLM (3, 0), 6-SLM (4, 0) and 7-SLM (5, 0) architectures use 3%, 21% and 30% fewer physical LUTs than 5-LUT, 6-LUT and 7-LUT architectures respectively. FF usage reductions of 5-SLM (3, 0), 6-SLM (4, 0) and 7-SLM (5, 0) are 12%, 21% and 35% with the compared results of 5-LUT, 6-LUT and 7-LUT. We can see that 6-SLM (4, 0) is 0.59% faster than 6-LUT, and 7-SLM (5, 0) is 0.76% faster than 7-LUT although 5-SLM (3, 0) is 3% slower than 5-LUT in delay results of Kintex-7.

### Table II. Tile implementation results for Xilinx Artix-7 (Values in parentheses indicate improvement rates).

| Architecture | LUT usage | FF usage | Critical path delay (ns) |
|--------------|-----------|----------|-------------------------|
| 5-LUT        | 836       | 610      | 26.314                  |
| 5-SLM(3,0)   | 806 (3%)  | 538 (12%)| 27.194 (3%)             |
| 6-LUT        | 1,360     | 797      | 29.977                  |
| 6-SLM(4,0)   | 1,068 (21%)| 629 (21%)| 29.8 (0.59%)            |
| 7-LUT        | 1,692     | 1,071    | 25.178                  |
| 7-SLM(5,0)   | 1,188 (30%)| 699 (35%)| 24.987 (0.76%)          |

### Table III. Tile implementation results for Kintex-7 (Values in parentheses indicate improvement rates).

| Architecture | LUT usage | FF usage | Critical path delay (ns) |
|--------------|-----------|----------|-------------------------|
| 5-LUT        | 813       | 610      | 13.587                  |
| 5-SLM(3,0)   | 795 (2%)  | 538 (12%)| 12.895 (5%)             |
| 6-LUT        | 1,333     | 797      | 15.468                  |
| 6-SLM(4,0)   | 1,058 (21%)| 629 (21%)| 14.587 (5.7%)           |
| 7-LUT        | 1,642     | 1,071    | 12.432                  |
| 7-SLM(5,0)   | 1,151 (30%)| 699 (35%)| 12.528 (0.8%)           |

The evaluation results for Kintex UltraScale+ KCU116 xcku5p-fvb676-2-e device are shown in Table IV. We can see that the 5-SLM (3, 0), 6-SLM (4, 0) and 7-SLM (5, 0) architectures use 2%, 21% and 30% fewer physical LUTs than 5-LUT, 6-LUT and 7-LUT architectures. Similarly, FF usage reductions are 12%, 21% and 35% respectively. In the critical path delay results on Kintex UltraScale+, we can see that 5-SLM (3, 0) and 6-SLM (4, 0) is 5% and 5.7% faster than 5-LUT and 6-LUT. Delay of 7-SLM (5, 0) is almost the same with 7-LUT. So, although the architectures of Xilinx FPGAs are different, the SLM architectures can achieve significant resource reduction with the comparison of LUT architectures with similar delay performance. When comparing the LUT resource usage of an entire vFPGA (array size × array size × LUT usage of one tile), we can see 7-SLM also performs better than 6- and 5-SLMs. In addition, the vFPGA bitstream can be portable between Xilinx FPGAs without modification through the proposed vFPGA architecture.

5.4 Discussion about SLM-based physical FPGA and vFPGA
In our previous work [3], we compared SLM-based and LUT-based physical FPGAs. As a result, the best performed 8-SLM (6, 2) architecture can reduce 65% silicon area usage than an 8-LUT architecture. However, in this research, the logic cells of smaller input sizes (less than 8) have better performance because larger virtual cells require more levels
of logic cells of the physical FPGA to express, which will increase the critical path delay significantly. Therefore, as the implementation technologies of the physical FPGA and the vFPGA are different. In this paper, we have to explore SLM architecture again for vFPGAs.

6. Conclusion and Future Work

We found that SLM-based vFPGA architecture can significantly reduce resource utilization, and delay results are almost same with the comparison of LUT-based vFPGA architecture for Xilinx FPGA devices. These advantages can be achieved by simply replacing LUTs with SLMs. In future, we will evaluate performance on FPGA devices from different vendors. Also, we will research methods for using LUTRAM on a physical FPGA to improve the vFPGA’s SLM implementation efficiency without losing the bitstream compatibility of vFPGAs. In the future works, we will try to implement vFPGA on Intel FPGA device.

References

[1] M. Najem, T. Bollengier, J. Christophe Le Lann, and L. Lagadec: “Extended Overlay Architectures for Heterogeneous FPGA Cluster Management,” Journal of Systems Architecture 78, Elsevier, June. (2017), pp. 1-14.
[2] A. Brant, and G. G. F. Lemieux: “ZUMA: An Open FPGA Overlay Architecture,” IEEE 20th International Symposium on FCCM, July. (2012), INSPEC Accession Number: 12866207.
[3] M. Amagasaki, R. Araki, M. Iida, and T. Sueyoshi: “SLM: A Scalable Logic Module Architecture with Less Configuration Memory,” IEICE Trans. on Fundamentals of Electronics, Communications, and Computer Sciences, Vol. E99-A, No.12, Dec. (2016), pp. 1745-1337.
[4] X. Li, A. K. Jain, D. L. Maskell, and S. A. Fahmy: “Area-Efficient FPGA Overlay using DSP Block based Time-multiplexed Functional Units,” 2nd International Workshop on OLAF, Monterey, CA, USA, Feb. (2016), (paper number: OLAF/2016/02).
[5] R. Rashid, J. G. Steffan, and V. Betz: “Comparing Performance, Productivity and Scalability of the TILT Overlay Processor to OpenCL HLS,” FPT, (2014).
[6] K. Ovtcharov, I. Tili, and J. G. Steffan: “TILT: A multithreaded VLIW soft processor family,” FPL, Sept. (2013), pp. 1–4.
[7] H. Ng, C. Liu, H. K. So: “A Soft Processor Overlay with Tightly-coupled FPGA Accelerator,” 2nd International Workshop on OLAF, Monterey, CA, USA, Feb. (2016), (paper number: OLAF/2016/07).
[8] C. Wang, P. Cao, B. Liu, and J. Yang. “Coarse-grained Reconfigurable Architecture with Hierarchical Context Cache Structure and Management Approach”, IEICE Electronics Express, Vol.14, No.6, pp. 1-10, 2017, DOI: 10.1587/exle.14.20170090.
[9] W. Waang, M. Bolic, and J. Parri: “PVFPGA: Accessing an FPGA-based Hardware Accelerator in a Paravirtualized Environment,” International Conference on CODES+ISSS, (2013), pp. 1-9, (DOI: 10.1109/CODES-ISSS.2013.6658997).
[10] J.D. Dondo, J. barba, F. Rincen, F. Moya, and J.C. Lpez: “Dynamic Objects: Supporting Fast and Easy Run-time Reconfiguration in (FPGAs)”, System Architecture. 59 (1), (2013), pp. 1-15, (DOI: 10.1016/j.sysarc.2012.09.001).
[11] S. Byma, J.G. Steffan, H. Bannazadeh, A.L. Garcia, and P. Chow: “FPGAs in the cloud: Boating Virtualized Hardware Accelerators with Openstack,” IEEE 22nd Annual International Symposium on FCCM, (2014), pp. 109-116, (DOI: 10.1109/FCCM.2014.42).
[12] M. Feilen, M. Ihmig, C. Schwarzbauer, and W. Stechle: “Efficient dvb-t2 Decoding Accelerator Design by Time-multiplexing FPGA Resources,” 22nd International Conference on FPL, (2012), pp. 75-82.
[13] B. Ronak, and S.A. Fahmy: “Improved Resource Sharing for FPGA DSP Blocks,” 26th International Conference on FPL, (2016), pp. 1-4, (DOI: 10.1109/FPL.2016.7577373).
[14] F. Duhem, F. Muller, P. Lorenzini: “Reconfiguration Time Overhead on Field-programmable Gate Arrays: Reduction and Cost Model,” IET Comput. Digital Tech. 6(2), (2012), pp. 105-113, (DOI: 10.1049/iet-cdt.2011.0033).
[15] A. Vaishnav, K. Dang Pham, and D. Koch: “A Survey on FPGA Virtualization,” 28th International Conference on FPL, conference number 28, Aug. (2018).
[16] C. Plessel, and M. Platzner: “Virtualization of Hardware-Introduction and Survey,” ERSA, (2004).
[17] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou: “A Fully Pipelined and Dynamically Composable Architecture of CGRA,” IEEE Symposium on FCCM, (2014).
[18] M. Psarakis: “Reliability-Aware Overlay Architectures for FPGAs: Features and Design Challenges,” 2nd International Workshop on OLAF, Monterey, CA, USA, Feb. (2016), (paper number: OLAF/2016/04).
[19] F. Siegle, T. Vladimirova, J. Ilistad, and O. Emin: “Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications,” ACM Computing Surveys (CSUR), Vol. 47, Issue 2, Article 37, Jan. (2015), (DOI: 10.1145/2671181).
[20] F. Eslami, E. Hung, and S. J.E. Wilton: “Enabling Effective FPGA Debug using Overlay: Opportunities and Challenges,” 2nd International Workshop on OLAF, Monterey, CA, USA, Feb. (2016), (paper number: OLAF/2016/08).
[21] D. Koch et al.: “An Efficient FPGA Overlay for Portable Custom Instruction Set Extensions,” FPL, Sept. (2013).
[22] A. K. Jain et al.: “Efficient Overlay Architecture Based on DSP Blocks,” IEEE Symposium on FCCM, May. (2015).
[23] D. Koch, C. Beckhoff, and G.G.F. Lemieux: “An Efficient FPGA Overlay for Portable Custom Instruction Set Extensions,” 23rd International Conference on FPL, (2013), pp. 1-8, (DOI: 10.1109/FPL.2013.6645517).
[24] R. Lysecky, K. Miller, F. Vahid, and K. Vissers: “Firm-core Virtual FPGA for Just-in-time FPGA Compilation (abstract only),” ACM/SIGDA 13th International Symposium on FPGAs, ACM/AFPG’05, ACM, New York, NY, USA, (2005), pp. 271, (DOI: 10.1145/1046192.1046247).
[25] T. Wiersema, A. Bockhorn, and M. Platzner: “Embedding FPGA Overlays into Configurable Systems-on-chip: Reconos meets ZUMA,” Reconfigurable Computing and FPGAs (ReConfig), International Conference on IEEE, (2014), pp. 1-6.
[26] V. Betz, and J. Rose: “A New Packing, Placement and Routing Tool for FPGA Research,” International Workshop on FPL, Springer, (1997), pp. 213-222.
[27] I. Kuon, R. Tessier, and J. Rose: “FPGA Architecture: Survey and Challenges,” Foundations and Trends in Electronic Design Automation, Vol. 2, No. 2, Feb. (2008), pp. 135-253.
[28] Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi: “An Automatic Design and Implementation Framework for Reconfigurable Logic IP Core,” 23rd International Conference on FPL, (2013).
[29] J. Anderson, J. Rose, and V. Betz: “VTR 7.0: Next Generation Architecture and CAD System for FPGAs,” ACM TRET, Vol.7, No.2, Article 6, June. (2014).
[30] Q. Zhao, K. Inoue, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi: “FPGA Design Framework Combined with Commercial VLSI CAD,” IEICE Trans. on Information and Systems, Vol. E96-D, No. 8, Aug. (2013), pp. 1602-1612.
[31] E. Ahmed, and J. Rose: “The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density,” IEEE Trans. on VLSI Systems, Vol. 12, No. 3, March. (2004), pp. 288-298.