Low-Level Flexible Architecture with Hybrid Reconfiguration for Evolvable Hardware

ROLAND DOBAI and LUKAS SEKANINA, Brno University of Technology

Field-programmable gate arrays (FPGAs) can be considered to be the most popular and successful platform for evolvable hardware. They allow one to establish and later reconfigure candidate solutions. Recent work in the field of evolvable hardware includes the use of virtual and native reconfigurations. Virtual reconfiguration is based on the change of functionality by hardware components implemented on top of FPGA resources. Native reconfiguration changes the FPGA resources directly by means provided by the FPGA manufacturer. Both of these approaches have their disadvantages. The virtual reconfiguration is characterized by lower maximal operational frequency of the resulting solutions, and the native reconfiguration is slower. In this work, a hybrid approach is used merging the advantages while limiting the disadvantages of the virtual and native reconfigurations. The main contribution is the new low-level architecture for evolvable hardware in the new Zynq-7000 all-programmable system-on-chip. The proposed architecture offers high flexibility in comparison with other evolvable hardware systems by considering direct modification of the reconfigurable resources. The impact of the higher reconfiguration time of the native approach is limited by the dense placement of the proposed reconfigurable processing elements. These processing elements also ensure fast evaluation of candidate solutions. The proposed architecture is evaluated by evolutionary design of switching image filters and edge detectors. The experimental results demonstrate advantages over the previous approaches considering the time required for evolution, area overhead, and flexibility.

Categories and Subject Descriptors: B.6.1 [**Logic Design**]: Design Styles; B.6.3 [**Logic Design**]: Design Aids; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.4.3 [**Image Processing and Computer Vision**]: Enhancement

General Terms: Design

Additional Key Words and Phrases: Architecture, circuit design, evolvable hardware, image filter, reconfigurable, Zynq

ACM Reference Format:
Roland Dobai and Lukas Sekanina. 2015. Low-level flexible architecture with hybrid reconfiguration for evolvable hardware. ACM Trans. Reconfig. Technol. Syst. 8, 3, Article 20 (May 2015), 24 pages.
DOI: http://dx.doi.org/10.1145/2700414

1. INTRODUCTION

Conventional hardware design methods usually start with the system specification, which describes the intended behavior of the target system. The design process includes several manual and/or automated steps resulting in a hardware implementation.

This work was supported by the European Social Fund (ESF) under the project Excellent Young Researchers at BUT (CZ.1.07/2.3.00/30.0039), the IT4Innovations Centre of Excellence (CZ.1.05/1.1.00/02.0070), and the project Architecture of Parallel and Embedded Computer Systems (FIT-S-14-2297).
Authors' address: R. Dobai and L. Sekanina, Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Bozetechova 1/2, 612 66 Brno, Czech Republic; emails: {dobai, sekanina}@fit.vutbr.cz.

DOI: http://dx.doi.org/10.1145/2700414

Some specifications require an adaptive behavior to be implemented; that is, the functionality of the designed system needs to change over time. One of the most obvious solutions is to implement *all* of the predicted functionalities and switch between them during the system lifetime. Such a switch of functionality can be achieved by altering the content of some control registers. The content stored in the control register can be used for selecting the desired functionality. Multiplexers select the outputs of those system components that implement the given functionality. This kind of system reconfiguration will be called *virtual reconfiguration* throughout the article because reconfiguration is available by the system built on top of the hardware. Reconfiguration is performed virtually and not by an immediate way provided by the hardware.

Another option is available for implementing adaptive systems when a reconfigurable hardware platform, for example, a field-programmable gate array (FPGA), is used. The hardware components and their interconnections can be changed by loading a new hardware configuration into the device configuration interface. In contrast with the previous approach, the implementation of only one functionality is sufficient at a time because the functionality can be later altered (reconfigured). The implementation of a virtual reconfigurational layer is not necessary because a native way is available. This type of reconfiguration will be called *native reconfiguration* in this article.

Evolutionary design is an alternative design approach based on bio-inspired methods. It was shown that in many cases, evolutionary design can autonomously evolve a system meeting the required specification [Sekanina 2012]. The specification can also require support for independent dynamic adaptation and the evolutionary design is then used to ensure the adaptation by a gradual change of the functionality.

Evolvable hardware (EHW) is an umbrella term for hardware systems that either are designed by evolutionary methods or exploit these methods in order to perform self-adaption in the case of a changing environment.

The Zynq-7000 all-programmable (AP) system-on-chip (SoC) is a new reconfigurable platform from Xilinx [2013b]. It offers many useful features, such as a dual-core ARM Cortex A-9 processor and analog-to-digital converters, and can be considered as a promising platform for EHW among the available FPGAs [Dobai and Sekanina 2013b].

In this article, a low-level flexible architecture built for EHW in the Zynq-7000 AP SoC platform is proposed. The programmable logic (PL) of the platform is exploited in order to allow fast evaluation of candidate solutions and adaptability without human intervention. However, this requires a suitable architecture for establishing candidate solutions, and the development of such an architecture is the main objective of this work. The main challenge is the improvement of the speed of reconfiguration and the speed of candidate solution evaluation in comparison with the most recent digital EHW systems. The contributions can be summarized as follows.

(1) Previous EHW methods have considered either virtual *or* native reconfiguration for the establishment of a candidate solution. The work presented in this article uses a hybrid approach merging their advantages (flexibility, speed of evolution, area overhead) while limiting the disadvantages. Although the hybrid approach was first used for EHW in Dobai and Sekanina [2013a], the complete implementation and evaluation including the statistical results and evolved examples are the subject of this work only.
(2) A new low-level flexible architecture is proposed with the goal to reduce the area overhead and the time required for evolution and ensure higher flexibility in comparison with other EHW systems. Previous EHW methods are top-down approaches where the selected set of processing element (PE) operations is implemented in hardware. The selection does not usually consider the hardware platform where it will be implemented, and this results in reduced performance. This work proposes

a bottom-up approach, starting the design at the hardware level, and the required set of PE operations is determined on top of that low-level design.

(3) Both the module-based and difference-based native reconfigurations were exploited for EHW previously. However, to the best of our knowledge, these reconfiguration methods were not compared before. We provide a detailed comparison in the Zynq-7000 AP SoC platform, which can be very useful for the EHW community. We suggest that difference-based reconfiguration should be used for future EHW systems.

The rest of the article is organized as follows. Section 2 contains the related work. The new reconfigurable architecture is proposed in Section 3. In Section 4, the feasibility of the architecture is demonstrated on the problem of image filter and edge detector evolution. Section 5 discusses the achieved experimental results, and Section 6 concludes the article.

2. BACKGROUND AND RELATED WORK

Suppose that the design task is to implement a combinational circuit specified by a truth table. A conventional design method probably will analyze the truth table, extract Boolean functions, and assign hardware components based on the target hardware platform. In contrast, an evolutionary method will consider the available hardware components and will try to select and connect them in the hope that the resulting circuit will fulfill the specification. At first it will not, but after a time, the circuit will come closer and closer to the required specification and the design goal might be achieved. The main objective of an evolutionary design framework is to ensure that the design goal could and will be achieved.

An evolutionary algorithm (EA) is used for managing the evolution. The EA works with a population of candidate solutions. The candidate solution is a combinational circuit in the given design task. Each candidate solution is encoded by a chromosome and represents a possible way to solve the problem. The given solution is evaluated and its fitness is determined. The fitness function measures the suitability of the candidate solution for the given application, that is, how close the functionality is to the desired one. After each candidate solution has been evaluated, another population is established based on the current population by use of bio-inspired operators (e.g., selection, mutation, crossover). The evolution is stopped at some point when a predetermined condition (e.g., maximum number of generations, time of evolution) is met and the solution with the best fitness is proclaimed as the result of the evolution [Greenwood and Tyrrell 2006; Sekanina 2012].

2.1. Comparison of Evolutionary and Conventional Designs

A conventional design method always produces the same solution for the same specification and succeeds given enough design expertise, while evolutionary design is not deterministic. Evolutionary design might or might not result in a solution for the given specification. However, evolutionary design can ensure autonomous adaptive behavior with less human intervention (e.g., during space missions) and lead to solutions that are generally not found when using conventional design methods [Greenwood and Tyrrell 2006; Sekanina 2012]. For example, the design of an adder is rather straightforward because the required functionality can be easily defined; that is, the output value needs to be equal to the sum of the input values. The specification of an image filter is less obvious and can be constructed in various ways; for example, the pixel needs to be replaced by the median value of its neighbors. It cannot be determined which specification is the best. A conventional design method following one of the possible specifications would result in a filter that is a correct solution for the

selected specification, but better filters might exist for other specifications. Evolutionary design based on evaluation of candidate filters using training data can lead to much better filter implementations [Vasicek et al. 2011; Salvador et al. 2011] because the training data can describe the problem in a better way than any other specification by showing how the image should look exactly after the filtering. A successfully evolved filter satisfies the training data approximately only but can provide better results than conventional filters. More details will be given later in this article.

The implementation of adaptive behavior is not a particularly easy task with conventional design methods because it is not always possible to predict all required functionalities. Evolutionary design can handle adaptive behavior more easily. The design framework needs to be implemented, but during operation, the circuit specification (truth table) can be changed and the EA will follow the new design goal and redesign the circuit [Kajitani et al. 2006].

2.2. Scalability of Evolutionary Design

Although the possibilities offered by EHW are very attractive, there are many limitations that were not fully addressed by previous research [Haddow and Tyrrell 2011]. Probably the most important ones are the problems with the scalability of representation and the scalability of evaluation. The representation defines the number (and granularity) of basic building components. The search space grows exponentially when the number of components increases linearly. This is called the problem with the scalability of representation [Sekanina 2012].

The evaluation of candidate solutions is an essential and the most time-consuming part of evolutionary design. The connection between the number of primary inputs and input stimuli is exponential; when all the input combinations are evaluated for a given candidate solution, even a small increase in the circuit size can result in a very significant increase in the evaluation time. This is known as the problem with the scalability of evaluation [Sekanina 2012].

As the consequence of the mentioned scalability issues, the size of the evolved solutions achievable by current EHW approaches is rather limited [Sekanina 2012] and is considerably smaller than the size of circuits designed by conventional design methods.

2.3. Candidate Solutions in FPGAs

FPGAs are a very popular platform for EHW. First, they provide the reconfigurability required for evolvable solutions. Second, the evaluation of candidate solutions in FPGAs can be several hundred times faster than in hardware simulators running in the most resourceful available desktop computers [Sekanina 2012; Dobai and Sekanina 2013b]. This work is restricted to digital EHWs implemented in FPGAs; therefore, analog EHWs will not be considered in this article.

EHW systems implemented in FPGAs are usually based on Cartesian genetic programming (CGP) because it utilizes the problem representation inspired by FPGAs. CGP uses a two-dimensional array of PEs; the size of this array is constant during the evolution, and this is very advantageous for hardware implementations. Each PE implements one of the operations, which is preselected for the given application [Miller 2011].

The selected operations of the PEs and the way they are interconnected are encoded into the chromosome. Therefore, the chromosome unequivocally describes a larger entity of hardware (candidate circuit), which has a more sophisticated functionality than that of a single PE.

Figure 1 shows an example candidate circuit established by a 3×3 array of PEs. The candidate circuit implements functionality for outputs f and s derived from inputs

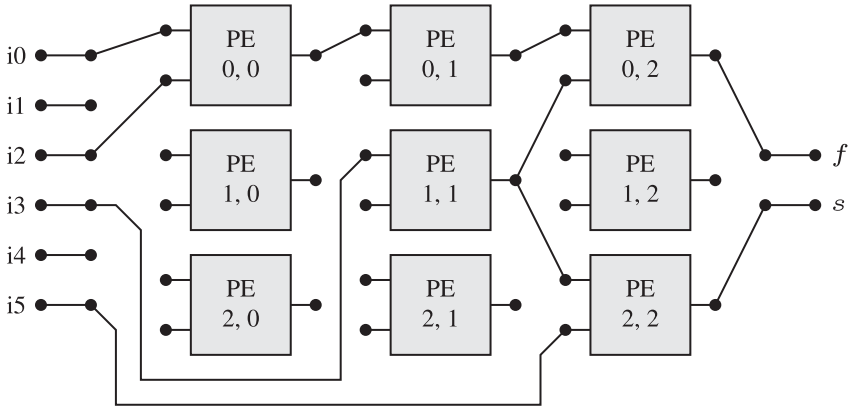

Fig. 1. Example candidate circuit implemented by an array of PEs. The two-dimensional array defines the search space for EA by allowing limited possibilities of interconnections and operations implemented inside the PEs.

i0, ..., i5. The hardware implementation of the PE array determines what kind of interconnections and PE operations are offered. This defines the search space for the EA. Let us assume that the operation of PE at position (1, 1) and (2, 2) is logical negation ($\neg$) and logical conjunction ($\wedge$), respectively. Then, the functionality implemented by the example candidate circuit for output s is $(\neg \text{i3}) \wedge \text{i5}$.

2.4. FPGA-Based EHW Systems

A recent evaluation of the Zynq-7000 AP SoC compared several scenarios of how to implement EHW on this new platform [Dobai and Sekanina 2013b]. According to this evaluation, the EA can be implemented very effectively in the hard processor core of the platform, which is a dual-core ARM Cortex-A9. This gives the advantage of employing more sophisticated EAs and ensuring fast execution. The PL available in the platform can be used to establish and evaluate candidate circuits more effectively than by software.

The most widely used hardware-based accelerator for EHW is called a virtual reconfigurable circuit (VRC) [Sekanina 2003]. The name refers to the fact that it uses virtual reconfiguration. The advantage of the approach is fast reconfiguration because writing into the configuration register can be performed in negligible time. The disadvantages are higher area requirements and lower operational frequency (i.e., slower evaluation of candidate circuits). The high area requirements are caused by the additional multiplexers and the implementation of all functions inside the PEs. VRC can nowadays be considered as the most mature approach for FPGA-based EHW systems [Sekanina 2012; Vasicek and Sekanina 2010].

The good support for native reconfiguration in recent FPGA devices resulted in the adoption of dynamic partial reconfiguration (DPR) for EHW [Upegui and Sanchez 2005; Cancare et al. 2010; Salvador et al. 2011]. DPR uses reconfiguration frames for reconfiguring a set of FPGA resources, and it is not possible to reconfigure only one resource. Candidate solutions can be established in the PL by the use of DPR. This removes the necessity of additional multiplexers and the implementation of unused functionality in the PEs. One might think that this would reduce the area requirements, but the implementation of the support logic for DPR requires some area, and consequently, the overall area requirements become similar to VRC [Salvador et al. 2012]. The real advantage of the native reconfiguration is the simpler PE array with shorter propagational paths without the additional multiplexers. The operational frequency becomes

higher and the evaluation of candidate solutions faster. The disadvantage is that the native reconfiguration of relatively large reconfigurable frames takes considerably longer than the virtual reconfiguration.

The native reconfiguration can be module based [Xilinx 2012] or difference based [Xilinx 2007]. The difference-based approach can be used to make fine-grained changes in the circuit, while the module-based approach reconfigures an entire area (module). The module-based approach for EHW was proposed by Salvador et al. [2011], where a systolic array with static interconnections is used. This removes the problem of handling the interconnections; however, it introduces limitations in comparison with the flexibility of VRCs.

The difference-based approach allows one to identify the positions of reconfigurable resources in the reconfigurable frames that are not documented by the manufacturer, and the EA can work directly with the bitstream in order to perform fine-grained changes in the candidate solution. The approach was proposed by Upegui and Sanchez [2005] and later used by Cancare et al. [2010].

Recently, a hybrid approach was proposed for EHW that adopts the advantages of the native and virtual reconfigurations while limiting their disadvantages [Dobai and Sekanina 2013a]. The name reflects the fact that both types of reconfiguration are necessary to establish a given candidate solution. The interconnections are reconfigured virtually, which keeps the flexibility of the VRC approach. The native approach would be slower for changing the interconnections. The PEs are configured by native reconfiguration, which reduces the area requirements and does not degrade the speed of candidate circuit evaluation. The main disadvantage is that the module-based DPR is used, which is characterized by higher area requirements and slower reconfiguration.

A similar architecture was proposed by Cancare et al. [2012]. It supports the evolution at various levels of hierarchy (e.g., logic gates, functional blocks). In comparison with the work of this article, it specifies the set of functions for PEs and for implementing the EHW by this set of PEs. They did not address the placement of PEs for reducing the reconfiguration time. The reason probably was that the reconfigurable frames in Virtex-4 are significantly smaller than in the Zynq-7000 AP SoC and did not require special attention. The main disadvantage of the method is, like all similar approaches, that it is basically just a hardware-based accelerator and not an EHW built for FPGAs.

Other reconfiguration approaches have also been published [Glette et al. 2009; Bartolini et al. 2013]. Glette et al. [2009] use the shift functionality of lookup tables (LUTs) in order to reduce the area requirements. The chromosome is not stored in the configuration register but shifted immediately into the LUT. Bartolini et al. [2013] exploit a reconfigurable device macro where the functionality can be similarly shifted in. Neither of these methods uses native reconfiguration of the PEs, and furthermore, the PEs do not have a fixed structure and the EHW system needs to be redesigned when a different set of PE functions is required.

3. PROPOSED EHW SYSTEM WITH LOW-LEVEL RECONFIGURABLE ARCHITECTURE

The developed EHW system shown in Figure 2 is based on the concept of an evolutionary design framework implemented in an FPGA. There are three main parts of the system: the processing system (PS), which controls the EHW system; the PL, which enables the establishment and evaluation of candidate solutions; and the external dynamic double data rate (DDR) memory. It should be noted that the PS built around a dual-core processor is an embedded system, which can work without the PL. It is equipped with the controller for the external memory and for interconnecting the PS and PL through the advanced extensible interface (AXI) of the advanced microcontroller bus architecture (AMBA). The evolution is performed as follows.

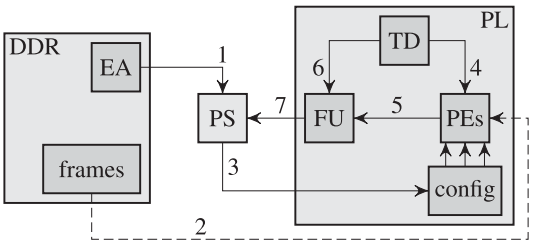

Fig. 2. Proposed EHW system implemented in the Zynq AP SoC. The EHW system is controlled by the PS for which the program code (EA) is stored in the external memory. The frames generated during runtime are also stored in the memory before they are used for reconfiguration. The establishment and evaluation of candidate solutions are performed inside the PL.

(1) The EA is executed by the PS, which manages the evolution. The evolution is started with a randomly generated population of candidate solutions. Each candidate solution is established in the PL by native and virtual reconfigurations in order to evaluate it.
(2) The native reconfiguration uses the reconfigurable frames stored in the external memory in order to set the functions inside the PEs. The EA modifies these frames based on the chromosome prior to reconfiguration. The reconfiguration uses direct memory access (DMA) and the processor configuration access port (PCAP), which is a new feature of the Zynq platform.
(3) The interconnections between the PEs are changed by virtual reconfiguration based on the content of the chromosome. This is executed by writing into the configuration register through the AXI bus.
(4) The training data (TD) is applied sequentially to the inputs of the established candidate solution.
(5) The response of the candidate solution is captured for each training input and evaluated by the fitness unit (FU).
(6) The evaluation takes into account the expected and the actual responses of the candidate solution, and the fitness value is computed gradually while all the prepared TD are used sequentially.
(7) The fitness value is transferred back to the PS through the AXI bus. The establishment and evaluation of another candidate solution follow.

After all candidate solutions are evaluated in the PL, a new generation is generated by bio-inspired operators (e.g., selection and mutation). The candidate solutions are evaluated in the PL. This process is repeated until the end of the evolution, which can be, for example, a desired number of generations.

Although the developed EHW system was implemented in a Zynq-7000 AP SoC, other FPGA platforms could be used also. The memory and AXI controllers, the configuration interface, or even the processor would be necessary to implement in the PL depending on the available features of the given FPGA platform.

The evolutionary design framework is designed for a given application, and this determines the used TD and array of PEs. After the framework has been deployed into the Zynq-7000 AP SoC, it can be used for evolving solutions for the given application. It is certainly a more flexible approach than a conventional design method because another and completely different solution can be evolved by modifying only the used TD. However, the domain of application can be changed only by the redesign of the EHW system because different PEs are needed with different bit precisions, and the TD should be replaced also.

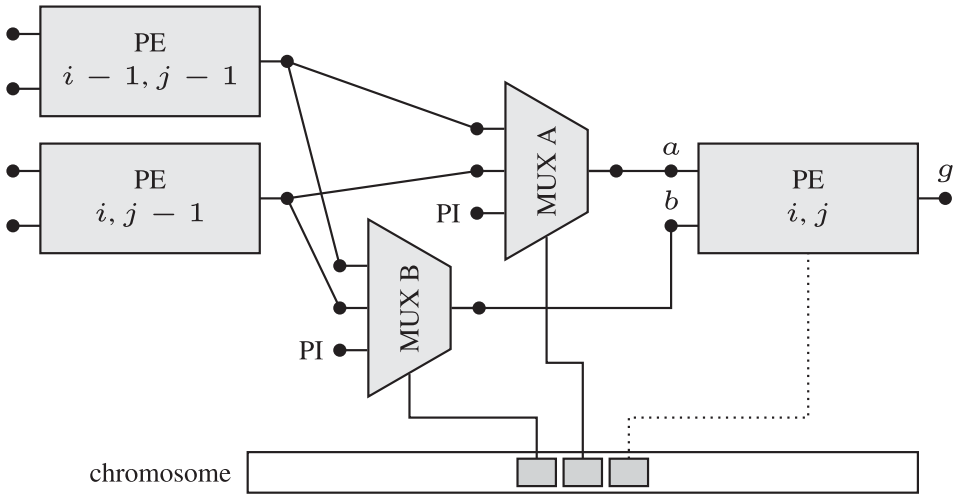

Fig. 3. Interconnections and PE functions determined by the chromosome. A given PE can be interconnected with any PE from the previous PE column or any primary input (PI). The interconnections are implemented by multiplexers (MUXs). The PE function is modified by native reconfiguration. Frames are generated based on the corresponding positions in the chromosome.

The size of the PE array can be set based on the expected size of the solution, and the size of the solution can be derived experimentally for the target application. A too small PE array would prevent a successful evolution or the quality of solutions would be limited. A relatively large array, where only a small portion of PEs are used, will have a negative impact on the time of evolution because the search space will become too big; this is known as the problem with the scalability of representation.

Figure 3 shows how the functionality and interconnections of a PE are influenced by the chromosome. The PE in the ith row and jth column has inputs a and b. These inputs can be connected to any PE output from the previous $(j-1)$-th column by means of multiplexers (MUX A and MUX B). Furthermore, these multiplexers allow connections to the primary input (PIs) of the candidate solution. These PIs are the inputs of the circuit that is under evolution. The selection of interconnections is based on the appropriate bit positions in the chromosome and the PE interconnections are changed by virtual reconfiguration. Another part of the chromosome determines the functionality of the given PE. However, this is not implemented by multiplexers. The EA prepares the configuration bitstream based on these bits and the PE operations are changed by native reconfiguration. Output g is used as one of the inputs to the multiplexers of the next $(j+1)$-th PE column.

The primary output (PO) of the candidate solution is selected as the output of one of the PEs in the last PE column. The smallest candidate solution can be constructed from only one PE, while the largest one will contain almost all the PEs. In the last column, not all PEs are used when the number of POs is less than the PEs in a column. The PE array is the "sandbox" of the EA where the EA can find the solution best suited for the given specification. The search space is determined by the possible interconnections and operations of PEs offered by the PE array.

3.1. Proposed Processing Element

Recent experiments with the Zynq-7000 AP SoC identified the high reconfiguration time as the main disadvantage for EHW systems [Dobai and Sekanina 2013a, 2013b].

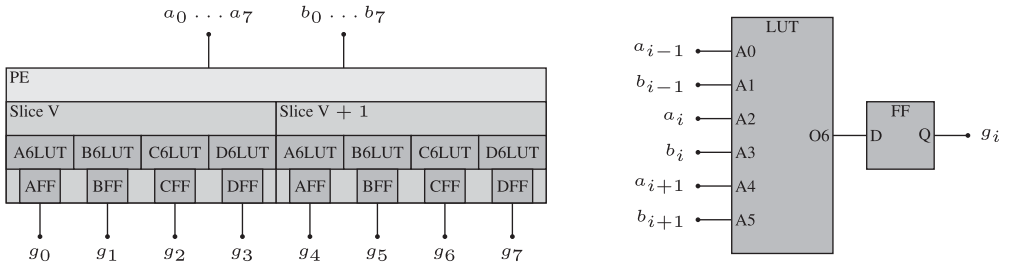

Fig. 4. Proposed PE implemented by FPGA resources. An output bit is generated by one LUT and stored in an FF. The inputs of the LUTs are connected in a way that they allow the implementation of various types of operations.

The PL of the platform is compatible with other 7-series Xilinx FPGAs. The reconfigurable frames of Virtex-5 and Virtex-4 are 60% and 68% smaller than those of the Zynq platform, respectively [Xilinx 2012]. Since the speed of the reconfiguration has not improved with the introduction of newer platforms, the reconfiguration time in the Zynq platform is obviously higher by the amount proportional to the increase of the frame size.

These first experiments motivated the research for developing a new architecture for EHW, which reduces the impact of the reconfiguration time and at the same time ensures higher flexibility than that of the previous EHW approaches.

Previous methods select a set of PE functions and implement them in hardware. The selection is usually made based on the types of operations frequently used by conventional design methods for similar applications. These functions do not usually reflect the hardware in which they are implemented. These implementations have various propagational times, and the worst of them will set the operational frequency of the EHW system. Furthermore, the area overhead and the reconfiguration time are similarly not considered. We refer to this method as the top-down design flow of EHW.

In this article, a bottom-up design flow is proposed. The design starts at the hardware level and on top of that design the EHW is built. The inner structure of PE is developed without taking into account the set of PE functions. Figure 4 shows the developed PE where a, b are the inputs; $g = (g_7, \ldots, g_0)$ is the output; A6LUT, B6LUT, C6LUT, and D6LUT are six-input LUT; AFF, BFF, CFF, and DFF are flip-flop (FFs); LUT inputs are denoted as A0, ..., A5 and the output as O6; and FF input is D and the output is Q.

A LUT together with an FF generates the ith bit of the output, and a Xilinx slice with four LUT and four FFs is used for four output bits. It should be noted that the implementation of an arbitrary number of output bits can be achieved by using the required amount of LUT, FFs, and slices. The role of the FFs is to create pipelined processing. The A, B, C, and D designations of the LUTs and FFs determine the exact position inside the slice. The PE can be extended by additional slices in the direction from the bottom to the top of the device. Slice V + 1 is the slice that is immediately above slice V. Therefore, the bits from the least significant to the most significant one are generated by programmable resources in the same direction as the slices, from the bottom to the top. The orientation and the direction of these resources were developed in order to reduce the routing overhead when such PEs are very closely placed to each other and are mutually interconnected. Furthermore, the placement allows one to consistently determine where the given functionality is implemented in the hardware. These topics will be addressed later in the article.

The ith output bit is generated based on the ith, $(i+1)$-th, and $(i-1)$-th bits of a and b. When $i = 0$, the index of the most significant bit is considered as $i - 1$. Similarly, 0 is used for $i + 1$ when i is the index of the most significant bit. The inner structure of

the proposed PE allows one to implement various PE functions using not only bitwise operations but also shift in both directions, which can imitate to some limit the carry part of arithmetic operations. For example, $g = a \vee b$ can be implemented as $g_i = a_i \vee b_i$ for $i \in \{0, \ldots, 7\}$, and $g = a/2$ as $g_i = a_{i+1}$ for $i \in \{0, \ldots, 6\}$ and $g_7 = 0$.

Because the set of PE functions is not considered during the design of the inner structure of the PE in the proposed bottom-up design flow, not every operation over the operands a and b can be implemented. For example, $g = a + b$ cannot be implemented by the proposed PE in such a way that the functionality will be an exact adder. However, an approximated, almost correct adder can be used and, as the experimental results will demonstrate, the evolved solution will still be good. It should be noted that the EA develops unconventional solutions with incomprehensible interconnections. The proposed method takes this idea further by using approximated PE functions and leaves the workaround to the EA.

The proposed PE is very flexible. A six-input LUT is initialized with 64 bits, and the PE function can be changed at the bit level. This means that an eight-bit PE can implement $2^{8\times64}$ different configurations. PEs of other EHW systems usually implement only 12 to 16 operations [Vasicek et al. 2011; Salvador et al. 2011; Sekanina 2012]. This opens a whole new spectrum of possibilities. For example, the EA can change the set of PE functions at runtime, or it is possible to start the evolution at coarse granularity (mutate PEs) and later move to finer granularity (mutate only parts of PEs). These possibilities were absent in previous EHW systems.

The next advantage of the proposed PE is the simpler inner structure in comparison with PEs designed with the top-down approach. First, this reduces the overall area requirements since usually a lot of PEs are used in an EHW system. Second, smaller PEs will fit into smaller reconfigurable areas, and this will reduce the time consumed by native reconfiguration. Third, the critical propagational path inside the PE contains only one LUT, which will result in higher operational frequency and, consequently, faster candidate evaluation and reduced time of evolution.

The PE shown in Figure 4 was proposed for applications that tolerate approximations (e.g., image processing). However, the idea of bottom-up EHW design can be successfully applied to other domains of application. For example, when the application does not tolerate approximation, the CARRY4 instance of Xilinx slices can be connected to the LUTs, which will allow for implementing arithmetic operations without approximation. Or even digital signal processing (DSPs) can be included in the PE in order to support more complex operations. The main idea of the proposed bottom-up approach is to assemble PEs from resources available in FPGAs and consequently determine the set of PE operations that will be used by the EA for evolving solutions. This will result in a hardware implementation that supports faster candidate evaluation through higher operational frequency and faster reconfiguration because of simpler PEs. Both of these will contribute to faster evolution in comparison with the previous top-down approach.

3.2. Interconnections and Arrangements of Processing Elements

The PE arrangement of the proposed architecture is shown in Figure 5. The PEs are placed below each other starting with PE1 at the top boundary of the clock region. The last one is PE25 at the bottom boundary. There is no empty space between them; that is, all the LUTs are used in the given column. The next one, designated as PE26, is placed at the bottom, and the following PEs are inserted in the opposite direction. The PEs are organized always in the opposite direction in comparison with the previous column. The reason for this orientation is the reduction of the routing overhead required for connecting the PEs. The implementation of selected interconnections influences which PEs are connected, but in general, it can be assumed that the interconnection of PEs logically close to each other is more probable than those that are logically more distant;

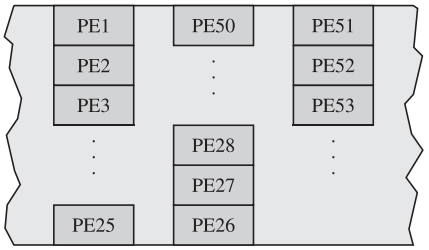

Fig. 5. PEs aligned to the clock region boundaries. There is no space between two rows of PEs in order to reduce the native reconfiguration time. The empty space between the PE columns is reserved for the resources required for virtual reconfiguration.

for example, PE25 and PE26 will probably be connected, while PE25 and PE50 will probably not be.

It should be noted that the PE arrangement in the clock region does not correspond to the arrangement in the PE array of CGP. For example, if a PE array has four rows, then the first column is placed as PE1, . . . , PE4, the following column as PE5, . . . , PE8, and so on while the column in the clock region is filled, and afterward, the placement continues in the next clock region column.

A clock region column of PEs occupies a full reconfigurable area. The dense placement of PEs together with their small size ensures that the highest number of PEs can be reconfigured at once and this limits the reconfiguration time; the reconfiguration of a lower number of PEs in the clock region column would take the same time. Additional columns can be placed in other clock regions without any consequence on the reconfiguration time.

A horizontal space between the clock region columns is left empty in order to leave space for the implementation of the PE interconnections. A small space should be used as the starting point and increased until the timing constraints become satisfied; that is, the design can be routed and the desired operational frequency is achieved. Again, neither of these spaces has negative influence on the performance of the EHW system.

The reconfiguration of PEs is performed by native reconfiguration. Virtual reconfiguration would be able to perform this task less effectively in terms of area overhead and operational frequency. On the other hand, the interconnections are reconfigured virtually, which is faster than the native reconfiguration. Native reconfiguration of the interconnections would not introduce any relevant advantage into the proposed architecture.

The virtual reconfiguration of interconnections is performed by writing into the configuration register. Consequently, this register sets the multiplexers that select the desired interconnections. Similarly to other VRC implementations, only neighbor PE columns can be interconnected. This limits the size of the multiplexers. The proposed architecture uses additional FFs at the output of the multiplexers. This introduces a negligible area overhead but ensures higher operational frequency. The placement of these multiplexers is not shown in Figure 5 because they are placed by the commercial routing tool. They are not reconfigured natively, and therefore, the placement is not constrained.

The proposed architecture was developed for the Zynq-7000 AP SoC platform but can be used in other 5-, 6-, or 7-series Xilinx FPGA devices that also have six-input LUTs. The only difference is that fewer than 25 PEs would be used in a clock region column because they have smaller clock regions. In devices with LUTs of smaller size, the architecture can be used by sacrificing some of the inputs in the proposed PE.

The bit-precision of PEs has an influence on the number of PEs in the clock region column. Twenty-five PEs fit into it when the PE is constructed by two slices (8-bit PEs). However, this can be changed easily, and consequently, the number of PEs changes.

3.3. Implementation in the Zynq Platform

The implementation of the proposed architecture can be performed by the design tools of Xilinx. The architecture described by a hardware description language is synthesized, placed, and routed, and finally, the bitstream is generated. Consequently, the bitstream is used for deployment in the PL. The implementation of the EA together with the control and communication with the PL are implemented in the C programming language.

An important issue that should be discussed is how the placement performed by Xilinx tools can coexist with the proposed placement described in this article. The placer tool can employ the synthesized entities to any free reconfigurable resources. The only exceptions are the proposed PEs. The placer is instructed by means of design constraints [Xilinx 2013a] to place the PEs in the way it is proposed in this article. The rest of the design can be placed where the placer might consider it appropriate.

The proposed placement is ensured by three types of design constraints:

(1) Constraint BEL is used to select the desired LUTs and FFs shown in Figure 4.
(2) Keeping the input connections of PE LUTs is enforced by the LOC_PINS constraint.
(3) The PEs are placed to their desired position shown in Figure 5 thanks to the LOC constraint.

3.4. Difference-Based Reconfiguration of the Proposed Processing Elements

The dense placement and reconfiguration of PEs in Figure 5 is possible thanks to the adoption of the difference-based native DPR. It will be demonstrated that this approach is more suitable for EHW than the module-based DPR. VRCs are not considered here because they could handle the proposed PEs only at the cost of unacceptable high area overhead.

The reconfigurable frames are the smallest addressable items in the device configuration address space. Therefore, all the native reconfigurations must operate with whole frames [Xilinx 2012]. The frames are aligned to the clock regions with the same height. This means that reconfigurable resources in a clock region column can be reconfigured by several frames. The change of only one reconfigurable resource implies that the whole column also is reconfigured because whole frames always are reconfigured. Therefore, the reconfiguration of a clock region column containing only one PE would take the same time as when the column full of PEs is reconfigured. This is the reason the proposed architecture contains so densely placed PEs aligned to the clock region boundaries.

In the module-based DPR flow, a reconfigurable module is given by an area selected by the user in the Xilinx floor-planning tool. This module contains everything in the selected area (i.e., reconfigurable resources and routing). However, the effectively reconfigured area accommodates everything that lays below and above this module in the clock region column because the reconfiguration works with whole frames. There are several ways to reconfigure PEs by the module-based DPR:

(1) A large reconfigurable module can be selected with several PEs in it. The reconfiguration time would be almost as short as in the case of the difference-based DPR; more frames are used because the width of the module cannot be smaller than one configurable logic block (CLB). However, a different bitstream should be prepared for each possible combination of PE functions. If the number of PE functions is X and the module contains 25 PEs, then X^{25} bitstreams are required. This is

unacceptable even for a small number of PE functions, and the proposed PEs implement a very large number of functions.

(2) A similarly large reconfigurable module can be selected with only one PE in it. This reduces the number of required bitstreams to the number of PE functions. The proposed flexible PEs cannot be handled because a lot of bitstreams would still be required. Let's assume for the sake of argument that there are only a few PE functions. Then, the reconfiguration time would still be the same as in the case when 25 PEs are reconfigured at once; that is, the establishment of candidate solutions would take 25 times longer.

The previous discussion assumed only one module. However, there are more modules and all of them require their own set of bitstreams because the location of the module is encoded into the bitstream. Since the functionality of modules is equivalent, it is possible to reduce the number of bitstreams by module relocation. Relocation ensures that the bitstream can establish the same functionality in more modules. There is a lot of previous research in this direction [Ichinomiya et al. 2012; Otero et al. 2010]. Since the module is relocated, it cannot contain implementation of other parts of the system, not even routing. This is the reason the reconfigurable module cannot be smaller than the height of the clock region. The placer of the commercial design flow would probably use those resources and the system would become nonfunctional after the module relocation. Using reconfigurable modules aligned to the clock region column still does not resolve the problem of external routes passing through the module. One needs to ensure and manually remove such violations. Considering that in an EHW system there are lots of PEs and interconnections, such manual intervention is a very hard and time-consuming process. Moreover, a small change in the EHW system or in the set of PE functions would require one to repeat this manual process.

Another disadvantage of the module-based DPR approach is that bus macros or proxy LUTs are placed at the boundaries of modules. This reduces the operational frequency and causes slower candidate evaluation. The packing density is also degraded [Xilinx 2012].

The difference-based reconfiguration requires direct bitstream manipulation for which the knowledge of configuration bit positions is necessary, and this is not documented by the manufacturer. The determination of this knowledge is easier than the bitstream relocation. Moreover, the achieved knowledge can be reused when the EHW system changes, but it is always necessary to repeat the manual steps of the module-based approach.

The proposed PE requires the reconfiguration of LUTs. The identification of positions for the LUT configuration bits can be achieved by following the steps described in Xilinx [2007] using either the Xilinx ISE design tools or the Vivado tools for newer devices. It is possible to identify the required knowledge for setting the selected LUT to the desired function by performing small changes and comparing the bitstreams. It is sufficient to achieve this knowledge for only one LUT because the placement of LUT configurations in the bitstream and the proposed arrangement of PEs are fairly regular and the positions in the bitstream can be computed based on the placement of LUTs in the clock region column.

The difference-based DPR predominates the module-based approach for the requirements of EHWs in all aspects. The area requirements are smaller, the reconfiguration is faster and more flexible, the design flow is easier, no relocations are needed, and the performance is not degraded. As the experimental results will demonstrate, the proposed architecture with virtual and difference-based reconfigurations is superior to previous EHW methods.

4. CASE STUDY: EVOLUTIONARY CIRCUIT DESIGN OF IMAGE FILTERS

The proposed architecture with hybrid reconfiguration was evaluated by evolutionary circuit design of image filters, which is one of the most frequently used case studies [Sekanina 2012; Vasicek and Sekanina 2010; Salvador et al. 2011] for EHW systems and provides a relatively good ground for objective comparison. The evolution of image filters requires a relatively high computational effort [Dobai and Sekanina 2013b] and demonstrates the advantages of an FPGA platform. Furthermore, the evolution of image filters gives acceptable solutions because the human eye cannot detect small errors in an image. On the other hand, the evolution of an adder is successful only when the evolved solution has 100% functionality.

The image filter is a combinational circuit. The (original noisy) pixel and the neighbor pixels are the filter inputs. The filter generates the pixel value (filter output) that should be used instead of the original pixel. The neighbor pixels are provided in order to help the filter in this generation. The image is filtered by processing all of the pixels together with the neighbors.

An evolutionary framework for developing image filters needs an array of PEs where the candidate solutions are established, a pair of training images, and a fitness unit for measuring the quality (fitness) of the solutions. A pair of training images is necessary because the filtered image has to be compared with a reference image in order to compute the fitness. It should be noted that this pair of training images influences the functionality of the resulting solutions. The EA will try to evolve a solution with the available PEs, which will generate outputs closest to the reference data; that is, the filtered image will be very similar to the reference image. It is possible to evolve a filter for a completely different purpose by changing only the training images, which is one of the main advantages of an EHW system.

Filters for 8-bit grayscale images are considered; therefore, 8-bit PEs are used. A typical PE array was selected containing eight columns and four rows. Column no. 1 was mapped at PE1, . . . , PE4; column no. 2 at PE5, . . . , PE8; . . . ; column no. 7 at PE25, . . . , PE28; and column no. 8 at PE29, . . . , PE32. This means that only two clock region columns are used and two partial bitstreams are required. The second column is not fully utilized but the remaining resources can be used for the implementation of other parts of the EHW system because those resources will not be changed in the bitstream.

A kernel of 3×3 pixels was chosen, which implies nine inputs for the filter: one is the current pixel and the other eight are the neighbors. The PE inputs can be connected to one of the nine filter inputs or one of the four PE outputs from the previous PE column; that is, only neighbor columns can be interconnected. For implementing the interconnections, 16-to-1 multiplexers can be used because $8 < 9 + 4 < 16$. A slice with four 6-input LUT can implement a 16-to-1 multiplexer, and therefore, the propagational depth of the implementation of interconnections is only one slice. The multiplexer output is captured in an FF. The outputs of the proposed PEs are also stored in FFs, and the propagational depth of PEs is also one slice. This means that the established candidate solution will be able to operate at high frequency because the critical propagational path will also be equal to one slice because all the propagational paths are one slice long.

The PE array with the interconnecting multiplexers is pipelined by the FFs. The case study considering eight columns has 16 pipeline stages and in each column both the PEs and multiplexers are pipelined.

The candidate solution is described by the chromosome. The chromosome encodes the selected PE functions and interconnections. A PE function is encoded by 8×64 bits, where 8 is the PE precision and 64 is the number of LUT configuration bits. Therefore, the native reconfiguration is performed based on 8 columns $\times$ 4 rows $\times$ 8×64

configuration bits. These bits are copied into the partial bitstreams prior to reconfiguration. The encoding of two input connections of a PE is accomplished by 2×4 bits, where 4 is the number of control inputs of a 16-to-1 multiplexer. In consequence, the virtual reconfiguration is performed by writing 8 columns $\times$ 4 rows $\times$ 2×4 bits into the configuration register. When a switching image filter is used [Vasicek et al. 2011], there are by 2×2 more bits, considering 2 bits for selecting the filtered pixel and 2 bits for the filter switch; an example switching image filter will be provided later in the article.

The TD set includes a training image pair: an image with noise and another one that can be considered to be noise free. The task of the EA is to develop a filter able to suppress the noise; that is, the training image with the noise becomes very similar to the noise-free training image after its pixels are processed by the filter. The similarity between the filtered and the noise-free images, which represents the quality of the solution, is measured by the fitness. The fitness is determined by the following equation:

$$\text{fitness} = \sum_{i=1}^{c} \sum_{j=1}^{r} |y(i, j) - r(i, j)|,$$

where c is the number of image columns, r is the number of image rows, $y(i, j)$ is the filtered pixel, and $r(i, j)$ is the noise-free (reference) pixel. As can be expected, the fitness measures the absolute difference between the filtered and the noise-free images. The lower the fitness value is, the better is the filter. An ideal filter able to suppress all the noise in the image would achieve the fitness value 0. One can argue that a higher value for a better fitness would be more reasonable. However, this would mean an additional transformation of the range and would offer no real advantage. The used fitness function is a simplified version of quality measures like peak signal-to-noise ratio (PSNR), mean squared error, or mean difference per pixel [Sekanina 2012; Miller 2011] but requires much lower computational effort.

Sequential pixel processing is assumed in all the experiments; that is, the images are filtered by processing only one pixel at a time.

An image sequencer [Sekanina 2012] is used to sequentially prepare the input kernels. This requires a $(3 - 1) \times c + 3$-long first-in first-out pipeline, where 3 is the kernel size and c is the number of image columns. The latency caused by the pipelined implementation is negligible and it represents a delay only at the beginning of the processing because the pixels are sent into the beginning of the pipeline, while the last pixels from the previous processing leave it at the other end.

5. EXPERIMENTAL RESULTS

The EHW system for switching image filter evolution was implemented in an XC7Z020 Zynq-7000 AP SoC device using the proposed low-level flexible architecture with hybrid reconfiguration. Eight columns and four rows of PEs and a training image size of 128×128 were considered. The functions implemented by the PEs are shown in Table I. This list of functions was previously used for switching image filters by Vasicek et al. [2011] and was selected based on functions frequently used in conventional filter implementations. PE functions containing only basic logic operations could be a starting point for any application, but using more complex functions inspired by functions used in conventional solutions can speed up the evolution because the search space will be smaller. As the adopted list of PE functions demonstrates, any function can be implemented by the proposed flexible PEs to some degree, which is expressed as correctness in the table. The correctness shows how close the implemented PE function is to the desired one. A 100% correctness means an exact implementation, while a lower value indicates some degree of approximation. This measure was determined based on

Table I. Eight-Bit Functions Over Operands *a*, *b* Implemented by PEs

Code	Function	Description	LUT functions	Correctness
0	255	constant	0 – 7: 0xFFFFFFFFFFFFFFFF	100%
1	a	identity	0 – 7: 0xF0F0F0F0F0F0F0F0	100%
2	$255 - a$	inversion	0 – 7: 0x0F0F0F0F0F0F0F0F	100%
3	$a \gg 1$	right shift by 1	0 – 6: 0xFFFF0000FFFF0000 7: 0x0000000000000000	100%
4	$a \gg 2$	right shift by 2	0: 0xC88988888DCC8C88 1 – 6: 0x70000000FF2F0000 7: 0x0000000000000000	93%
5	$a + b$	addition	0: 0x0F08E3B000000C00 1 – 6: 0xAF0AA77AA11AAF0A 7: 0xA11AA77AA77AA77A	87%
6	$a +^s b$	addition with saturation	0: 0xEFFEFFFEFFFEAFFE 1 – 6: 0xFFFFFFFAFFFA8FDA 7: 0xFFFAFFFAFFFAFFFA	97%
7	$(a + b) \gg 1$	average	0: 0xF044FFFFFFFF040 1 – 6: 0xF8800FFF0FFFF880 7: 0xF880F880F880F880	97%
8	$max(a, b)$	maximum	0: 0xD10080000000D000 1 – 6: 0xFFF0FF00F0F0FFF0 7: 0xFFF0FFF0FFF0FFF0	97%
9	$min(a, b)$	minimum	0: 0xFFF6FFF6FFC7FFF4 1 – 6: 0xF000F0F0FF00F000 7: 0xF000F000F000F000	97%
10	if $a > 127$ then b else a	conditional selection	0: 0xFB59FFFEFF55FB50 1 – 6: 0xFD00F0F0FF00F0B0 7: 0xF000F000F000F000	95%
11	$\lvert a - b \rvert$	absolute difference	0: 0x0030059002200420 1 – 6: 0x0DB00DD00BB00DB0 7: 0x0DB00DB00DB00DB0	95%

The correctness is not always 100% for the desired function because the PEs have fixed inner structure.

the accumulated absolute difference between the current and the desired responses of the PE considering all the $2^{2\times8}$ combinations of the two 8-bit PE inputs. The PE can be modified when approximations are not tolerated by the domain of application; for example, CARRY4 instances present in Xilinx slices can be added into the PEs.

Table I also contains the LUT functions that were used in order to achieve the required functionality. The hexadecimal numbers are the initialization values of LUTs and the number (or the interval) before the colon denotes the bit position(s) in the PE.

It should be noted that the proposed method is very flexible. An 8-bit version of the proposed PE allows one to have $2^{8\times64}$ possible configurations, where 64 is the number of configuration bits of a six-input LUT. Although only the functions from Table I were used in the experiments, it would be possible to replace them or extend the list during runtime by a more sophisticated EA. Such replacement in the case of the previous EHW approaches would require one to redesign the whole system. The proposed architecture is time saving when one wants to experiment with various lists of PE functions. This can be done manually and will be automated in the future. The support for this kind of flexibility in the previous approaches is hardly imaginable. A VRC with a lot of functions inside the PEs would be extremely large and probably would not fit into an FPGA device. The top-down approach using DPR would require at

Table II. Resource Utilization in XC7Z020

	Available	VRC		Proposed DPR		Module-Based DPR	
Flip-flops	106,400	3,492	3.3%	2,373	2.2%	27,717	26.0%
Total LUTs	53,200	6,340	11.9%	3,899	7.3%	16,443	30.9%
Logic LUTs	53,200	6,276	11.8%	3,028	5.7%	15,584	29.3%
Shift LUTs	17,400	64	0.4%	859	4.9%	859	4.9%
Block RAMs	140	58	41.4%	18	12.9%	18	12.9%

least $2^{8\times64}$ bitstreams to be stored in unavailable large external memories considering the case when one PE is implemented in a reconfigurable module.

5.1. Programmable Resource Utilization

Table II shows the resource utilization of the implementation in the column "Proposed DPR" and compares it with the VRC [Dobai and Sekanina 2013a] and module-based DPR implementations. As it can be observed, the area required by the proposed method is smaller than for the VRC. This is caused mainly by the fact that the PEs are not multiplexed in the proposed method but replaced by native reconfiguration. The reduction in block random access memory (RAM) usage is the result of more efficient control unit implementation. The utilization of LUTs in the role of shift registers has been increased in order to achieve higher performance. This is not a disadvantage of the proposed method because the total LUT utilization decreased.

The module-based DPR has a similar policy about the replacement of PEs, but relatively large areas are reserved for them. This is a limitation of the module-based reconfiguration flow of Xilinx. The resources of the reserved areas should be considered also because these PEs cannot share the implementation of other parts because the reconfiguration frames are relocated in order to limit the number of bitstreams stored in memory. The required area for the PE array is 8 columns $\times$ 4 rows $\times$ 50 CLBs = 1,600 CLBs = 12,800 LUTs and 25,600 FFs. The area required for the PE array in the proposed architecture is only 8 columns $\times$ 4 rows $\times$ 8 FFs and LUTs = 256 FFs and LUTs, which can share frames with each other and even with other parts of the design. A fair comparison like this, to the best of our knowledge, was not provided previously.

The results in Table II show that the proposed method has lower area overhead in comparison with previous work. It should be noted that the Zynq-7000 AP SoC uses the PCAP for performing the reconfiguration. In other platforms where the traditional internal configuration access port (ICAP) is used, an additional area overhead is required to implement the support for the reconfiguration [Salvador et al. 2012].

5.2. Statistical Evaluation of the Evolution

The simple $(1 + \lambda)$ evolutionary strategy was considered in the experiments with a randomly initialized first population where λ candidate solutions are created in each generation from the best solution of the previous generation using mutations [Miller 2011]. The LUT contents were not evolved but selected from Table I. The chromosome size was 388 bits: 8 columns $\times$ 4 rows of PEs were used; 2 $\times$ 4 bits select the interconnections for the two inputs of a PE, 4 bits determine the function of a PE, and 2 bits are used for selecting the filtered pixel and 2 bits for the filter switch. Figure 6 shows the influence of the population size λ and the number of mutations on the quality of the evolved solutions. The quality is measured by the fitness; a mutation means that a random PE function or interconnection is changed. Each box plot represents 30 independent runs of EA. In the graph on the left, one run consists of 400,000 candidate filter evaluations and five mutations per chromosome. The results demonstrate

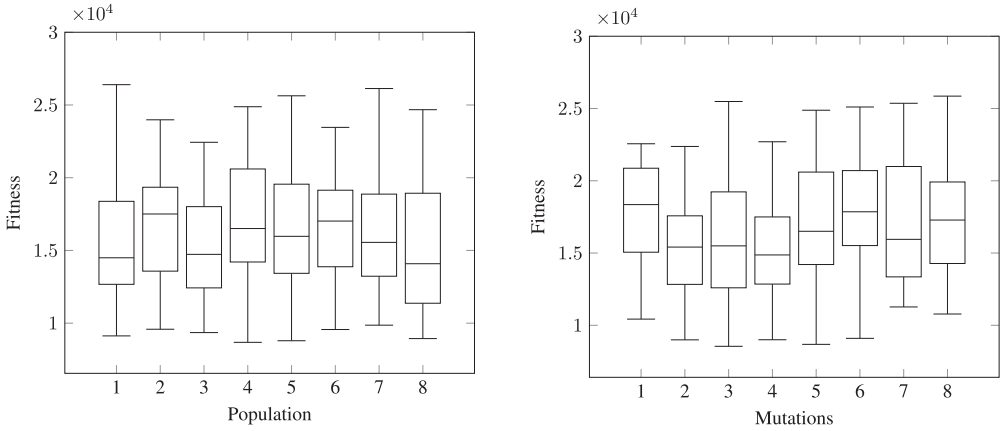

Fig. 6. Influence of population size and number of mutations on the fitness. The population with constant number of candidate evaluations has a negligible impact. The fitness of an average individual seems to improve with the mutation rate converging to 4.

that the population size has negligible influence on the quality of evolution. The lower quartile seems to be improving when approaching four from both lower and higher values, but the middle quartile and the median do not have a global minimum. The median is distributed around the fitness value 1.5×10^4. The main reason is that a simple EA is used without any parallel processing and it does not seem to matter if more candidates are used in the population or more generations are considered for the predefined number of evaluations.

The graph on the right in Figure 6 shows the result of experiments considering 100,000 generations and population of four individuals. The results indicate that the quality of solutions improves when the number of mutations converges to four. The lower quartile, middle quartile, and median all improve when changing the number of mutations from lower or higher values in the direction of four mutations.

5.3. Time Required for Evolution

The time required for the evolution was fairly constant in the experiments because the hardware implementation operates on constant frequency similarly to the native and virtual reconfigurations, and mutations can be executed in negligible time by the software implementation of EA compared to all operations of the evolution because a mutation involves only the generation of a random number. In all of the runs where 400,000 candidate solutions were considered independently of the population size and number of mutations, the total evolution time was 46 seconds (less than 1 second consumed by the virtual reconfiguration, 21 seconds by the native reconfiguration, 22 seconds by the candidate evaluation, and 2 seconds by the EA). This means that the proposed architecture is able to process 8,700 candidate solutions in each second.

A comparable and highly optimized VRC [Vasicek and Sekanina 2010] with four candidate solutions evaluated in parallel, which can be probably considered as the fastest VRC implementation, evaluates 25,000 candidates in each second. The device resource utilization of the proposed method indicates that there are enough free resources to implement parallel evaluation. The proposed method with four fitness unit and four PE arrays should reach over 30,000 candidate circuit evaluations.

The proposed method even outperforms the methods based on DPR; 3,100 evaluations were reported in Salvador et al. [2011] for a larger training set. A Virtex-5 device was used, which has 60% smaller reconfigurable frames [Xilinx 2012]. This means that

their implementation in the Zynq-7000 AP SoC would be slower than 3,100 per second as the consequence of higher reconfiguration times.

The proposed method is able to outperform other implementations because it has the highest operational frequency to the best of our knowledge. Thanks to the proposed simple PEs, the operational frequency of the implementation is 300MHz. The second-fastest implementation operates at 250MHz [Salvador et al. 2011], while the VRC implementations usually achieve around 100MHz only as the result of either the longer propagational delays introduced by additional multiplexers [Vasicek and Sekanina 2010] or the less effective PEs of the top-down design approach [Bartolini et al. 2013].

The native DPR reconfiguration still represents a considerable part of the evolution time (21 out of 46 seconds); however, this was already extremely reduced by the use of the difference-based DPR flow and the dense placement of PEs in the proposed architecture. Virtual reconfiguration would be faster as it is indicated by the 1 second out of 46 required for setting the interconnections; however, the gain in the operational frequency is more dominant, and therefore, the overall performance of the proposed method is higher.

The proposed method is more advantageous in comparison with the module-based approach considering the amount of required configurations. The proposed method uses only two bitstreams because the PEs are arranged into two clock region columns. Each bitstream contains 21,056 bits. The bitstream size of the module-based approach is 124,512 considering the Zynq-7000 AP SoC platform because it contains the configuration data for not just the LUTs. Moreover, 32 such bitstreams are required to establish a candidate solution considering the same size of the PE array because only one PE can be in one reconfigurable area for that approach. Since the speed of the configuration is the same, the proposed approach is able to establish the candidate solutions 95 times faster. Moreover, 32 PEs do not occupy two full clock region columns. If the full occupation with 50 PEs would be considered, then the reconfiguration would become 148 times faster. It should be noted that in the module-based approach, the memory requirements for storing the configuration bitstreams are much higher: $124{,}512 \times 12$, where 12 is the number of functions, while only $21{,}056 \times 2$ is required for the proposed approach, which has considerably better flexibility than just 12 functions.

Previous difference-based approaches [Upegui and Sanchez 2005; Cancare et al. 2010] would be able to perform faster native reconfiguration because older FPGA devices have smaller reconfigurable frames. The proposed work is aimed at the new Zynq-7000 AP SoC, which can otherwise be considered a good platform for EHW. The impact of the reconfiguration time is limited in the proposed architecture by the dense placement of PEs.

5.4. Evolved Switching Image Filter

Figure 7 shows an evolved switching image filter that was selected as the best one from 30 independent runs considering 100,000 generations, population of four candidates, and five mutations per chromosome. Image kernel of 3×3 pixels was used and the arrangement of the input pixels i0 . . . i8 in the figure indicates their orientation, where the current pixel is i4 and the others are the neighbor pixels. The filter switch s decides whether i4 is passed directly to the filter output y or is replaced by the filtered pixel f. Only the most significant bit of the filter switch is used as the control input to the multiplexer. Both y and f are evolved together by the EA. The interconnections of PEs are accomplished by multiplexers and are not shown in the figure. The numbers inside the PEs indicate the type of the implemented function according to the code designations from Table I. It should be noted that there is not much sense in the interconnections and the selected functions. The inner structure of a conventional solution might offer some hints about what the solution does; for example, for a median filter, it is possible

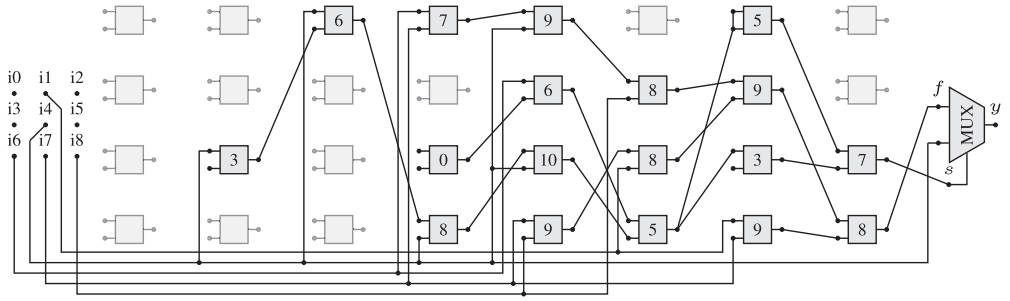

Fig. 7. An evolved switching image filter that has seemingly random interconnections. This demonstrates that EHW is able to develop unconventional solutions.

to discover that it computes the median value of the pixels. An evolved filter does not offer such a hint. It is working but it cannot be determined why. This can be considered as the ability of EAs to find unconventional and incomprehensible solutions.

Figure 8 shows the ability of the selected filter to suppress "salt and pepper" noise with indication of PSNRs. The left column contains the original (correct) images, the middle one the images with noise, and the right one the images after the filtering by the evolved filter. A Lena benchmark image of size 128 $\times$ 128 with 5% noise was used in the experiments as the training image. The results demonstrate that the evolved filter can handle as well other images and a higher level of noise than those considered during the evolution.

The evolved solution can also be reused as an ad hoc filter without the implementation of the full EHW system. The use of the new PCAP of the Zynq-7000 AP SoC allows one to completely "erase" the EHW system from the PL and reconfigure it with only the evolved filter. In this case, the establishment of the evolved solution without the EHW systems requires only 17 PEs. Figure 7 shows 18 used PEs, but one of them implements a constant value, and therefore, it can be eliminated. The multiplexers for setting the interconnections are not required and the static routing shown in the figure can be used, but an 8-bit multiplexer should be used as the filter switch and pipelining of the input pixels is also required. An operational frequency of 300MHz or higher can be expected similarly to the implementation used in the experiments; assuming an image of size 1,920 $\times$ 1,080, this would result in a processing speed of at least 145 images per second because the PEs create a pipeline, and a pixel is processed in each clock cycle with a negligible latency only at the beginning of the filtering. Since the area requirements of the evolved filter are very low, the processing speed can be multiplied at low cost by using several copies of the filter.

5.5. Adaptive Evolution

An interesting property of EHWs is the ability to adapt in the case of a changing environment. The specification of the desired digital circuit can change, and the EHW system can evolve a solution for the new specification.

This ability of the proposed EHW system is demonstrated by evolving an edge detector. Previously, the noise filter was evolved by using a noisy image for training. An edge detector will be evolved by changing this training image to an image showing the result of a reference edge detection. Figure 9 shows that the replacement of training images resulted in the development of a digital circuit able to detect edges. It should be noted that nothing else was changed or modified besides the training images. The implemented EHW system in the PL and the parameters of EA remained the same. The evolution of the edge detector was possible without any modification of the EHW

Lena image with 5% noise used as the training image (PSNR 18.29dB and 30.82dB)

Goldhill image with 5% noise (PSNR 18.66dB and 30.12dB)

Bridge image with 10% noise (PSNR 15.54dB and 25.21dB)

Camera image with 15% noise (PSNR 13.58dB and 23.27dB)

Fig. 8. Image filtering of "salt and pepper" noise by an evolved filter.

Fig. 9. Edge detection achieved by replacing the training data. The evolved edge detector is able to produce the image shown on the right when the center image is applied pixel by pixel to the inputs of the detector. The left image was used as the training reference.

system because edge detection is possible with a similar circuit using a 3×3 filtering kernel.

A conventional design approach not using EHW would require one to redesign the hardware system after each change of the specification or prepare several partial bitstreams in advance. These partial bitstreams could be used to achieve similar adaptive behavior. However, each desired specification needs to be known in advance because a bitstream is necessary for each such specification. Furthermore, the bitstreams are stored in limited memory resources, and as a consequence, only a limited number of bitstreams can be prepared.

A conventional design method usually does not tolerate faults in the hardware. The bitstream deployed to a location with a fault will not produce the expected functionality. EHW systems can partially tolerate faults because they work with the hardware platform. The presence of a fault will influence the fitness value, and consequently, a solution will be evolved that produces a good fitness value even with the fault [Salvador et al. 2011].

An EHW system is able to develop new solutions and can cope with changing inputs or even altered circuit specification, as was demonstrated by the successful adaptation to edge detection after noise filtering. This represents a clear advantage over conventional design methods; that is, EHW is more flexible in a way. However, the adaptability has certain limits. When a more significant change is required (e.g., migration from image processing to network traffic analysis), the EHW system needs to be redesigned.

6. CONCLUSIONS

EHW can evolve solutions that are hard to find when conventional design methods are used, ensure flexible adaptation in the case of a changing environment, and tolerate faults. The nondeterministic behavior and the higher time required for evolution are the disadvantages of EHW, which can be partially limited by the use of FPGAs.

The Zynq-7000 AP SoC is a new FPGA platform equipped with two hard processor cores. These features could be exploited in order to develop faster EHW systems and explore new domains of application. However, EHW needs a custom-tailored architecture for deployment in FPGAs.

In this article, a new low-level flexible architecture with hybrid reconfiguration was proposed for EHW. The approach merges the advantages of the virtual and native reconfigurations. The use of virtual reconfiguration for setting the interconnections in the candidate solutions ensures the flexibility of VRCs and removes the burden from the native reconfiguration, which would perform this task less effectively. The native

reconfiguration is used to reconfigure the flexible PEs, which are the building blocks of the proposed architecture. The virtual reconfiguration would be able to handle these PEs only at the cost of very high area overhead.

A unique bottom-up design flow was introduced for EHW starting at the hardware level by using PEs assembled from reconfigurable resources. This ensures good flexibility by supporting a very large number of PE functions without the need to redesign the whole EHW system. These PEs are arranged in the Zynq-7000 AP SoC in such a way that the number of influenced configurable frames is limited. This keeps the reconfiguration time and the area overhead at an acceptable level.

The proposed flexible PE allows one to implement a very large number of functions. The set of PE functions can be replaced at runtime without the need to redesign the EHW system. This ensures good reusability when one wants to experiment with various settings of PEs.

The proposed hybrid approach and the reconfigurable architecture were evaluated by evolutionary design of image filters and edge detectors. The experimental results demonstrate advantages over the previous work considering the time required for evolution, area overhead, flexibility, and reusability.

It was demonstrated by experiments in the Zynq-7000 AP SoC platform that for EHW, the difference-based native reconfiguration should be used, which excels in reconfiguration time, area overhead, memory requirements, and reusability over the module-based reconfiguration.

Future work will be conducted to exploit the proposed architecture for a wider spectrum of EHW systems, mainly larger digital combinational circuits. Although the achieved results are very promising, there are many possibilities offered by the Zynq-7000 AP SoC that can be further exploited. The analog-to-digital converter could be used for exploring new domains of application. The processors could give fertile ground for more sophisticated EAs and reduce the scalability problem of evolutionary design. The EA used in the experiments uses only one of the available processor cores. The use of both cores offers possibilities such as reconfiguration, candidate evaluation, and candidate preparation executed in parallel.

REFERENCES

Davide B. Bartolini, Matteo Carminati, Fabio Cancare, Marco D. Santambrogio, and Donatella Sciuto. 2013. HERA project's holistic evolutionary framework. In *Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW'13)*. 231–238. DOI: http://dx.doi.org/10.1109/IPDPSW.2013.110

Fabio Cancare, Davide B. Bartolini, Matteo Carminati, Donatella Sciuto, and Marco D. Santambrogio. 2012. On the evolution of hardware circuits via reconfigurable architectures. *ACM Transactions on Reconfigurable Technology Systems* 5, 4, Article 22 (Dec. 2012), 22 pages. DOI: http://dx.doi.org/10.1145/2392616.2392620

Fabio Cancare, Marco D. Santambrogio, and Donatella Sciuto. 2010. A direct bitstream manipulation approach for Virtex4-based evolvable systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS'10)*. 853–856. DOI: http://dx.doi.org/10.1109/ISCAS.2010.5537429

Roland Dobai and Lukas Sekanina. 2013a. Image filter evolution on the Xilinx Zynq platform. In *Proceedings of the NASA / ESA Conference on Adaptive Hardware and Systems (AHS'13)*. IEEE Circuits and Systems Society, 164–171. DOI: http://dx.doi.org/10.1109/AHS.2013.6604241

Roland Dobai and Lukas Sekanina. 2013b. Towards evolvable systems based on the Xilinx Zynq platform. In *Proceedings of the 2013 IEEE International Conference on Evolvable Systems (ICES'13)*. IEEE Computational Intelligence Society, 89–95. DOI: http://dx.doi.org/10.1109/ICES.2013.6613287

Kyrre Glette, Jim Torresen, and Mats Hovin. 2009. Intermediate level FPGA reconfiguration for an online EHW pattern recognition system. In *Proceedings of the NASA / ESA Conference on Adaptive Hardware and Systems (AHS'09)*. IEEE Computer Society, 19–26. DOI: http://dx.doi.org/10.1109/AHS.2009.46

Garrison Greenwood and Andrew M. Tyrrell. 2006. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley-IEEE Press, Piscataway, New Jersey.

Pauline C. Haddow and Andy M. Tyrrell. 2011. Challenges of evolvable hardware: Past, present and the path to a promising future. *Genetic Programming and Evolvable Machines* 12, 3 (2011), 183–215. DOI:http://dx.doi.org/10.1007/s10710-011-9141-6

Isamu Kajitani, Masaya Iwata, and Tetsuya Higuchi. 2006. A GA hardware engine and its applications. In *Evolvable Hardware*, Tetsuya Higuchi, Yong Liu, and Xin Yao (Eds.). Springer US, 41–63. DOI:http://dx.doi.org/10.1007/0-387-31238-2_3

Yoshihiro Ichinomiya, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. 2012. A bitstream relocation technique to improve flexibility of partial reconfiguration. In *Algorithms and Architectures for Parallel Processing (Lecture Notes in Computer Science)*, Vol. 7439. Springer, Berlin, 139–152. DOI:http://dx.doi.org/10.1007/978-3-642-33078-0_11

Julian F. Miller. 2011. *Cartesian Genetic Programming*. Springer, Berlin. DOI:http://dx.doi.org/10.1007/978-3-642-17310-3

Andres Otero, Angel Morales-Cas, Jorge Portilla, Eduardo de la Torre, and Teresa Riesgo. 2010. A modular peripheral to support self-reconfiguration in SoCs. In *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. 88–95. DOI:http://dx.doi.org/10.1109/DSD.2010.100

Lukas Sekanina. 2012. Evolvable hardware. In *Handbook of Natural Computing*, Grzegorz Rozenberg, Thomas Back, and Joost N. Kok (Eds.). Springer, Berlin, 1657–1705. DOI:http://dx.doi.org/10.1007/978-3-540-92910-9

Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina. 2011. Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support. In *Proceedings of the 2011 NASA/ESA Conference on Adaptive Hardware and Systems*. 184–191. DOI:http://dx.doi.org/10.1109/AHS.2011.5963934

Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina. 2012. Implementation techniques for evolvable HW systems: Virtual VS dynamic reconfiguration. In *Proceedings of the 2012 22nd International Conference on Field Programmable Logic and Applications (FPL'12)*. 547–550. DOI:http://dx.doi.org/10.1109/FPL.2012.6339376

Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Lukas Sekanina, and Teresa Riesgo. 2011. Fault tolerance analysis and self-healing strategy of autonomous, evolvable hardware systems. In *Proceedings of the 2011 International Conference on ReConFigurable Computing and FPGAs*. 164–169. DOI:http://dx.doi.org/10.1109/ReConFig.2011.37

Lukas Sekanina. 2003. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In *Evolvable Systems: From Biology to Hardware (Lecture Notes in Computer Science)*, Vol. 2606. Springer, Berlin, 186–197. DOI:http://dx.doi.org/10.1007/3-540-36553-2_17

Andres Upegui and Eduardo Sanchez. 2005. Evolving hardware by dynamically reconfiguring Xilinx FPGAs. In *Evolvable Systems: From Biology to Hardware (Lecture Notes in Computer Science)*, Vol. 3637. Springer, Berlin, 56–65. DOI:http://dx.doi.org/10.1007/11549703_6

Zdenek Vasicek, Michal Bidlo, Lukas Sekanina, and Kyrre Glette. 2011. Evolutionary design of efficient and robust switching image filters. In *Proceedings of the 2011 NASA/ESA Conference on Adaptive Hardware and Systems*. 192–199. DOI:http://dx.doi.org/10.1109/AHS.2011.5963935

Zdenek Vasicek and Lukas Sekanina. 2010. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics* 29, 6 (2010), 1359–1371.

Xilinx. 2007. Difference-Based Partial Reconfiguration XAPP290 (v2.0).

Xilinx. 2012. Partial Reconfiguration User Guide UG702 (v14.4).

Xilinx. 2013a. Constraints Guide UG625 (v14.5).

Xilinx. 2013b. Zynq-7000 All Programmable SoC Overview DS190 (v1.3).

Received July 2013; revised July 2014; accepted September 2014