2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Lookup Table Partial Reconfiguration for an Evolvable Hardware Classifier System

Kyrre Glette
Department of Informatics
University of Oslo
P.O. Box 1080 Blindern, 0316 Oslo, Norway
Email: kyrrehg@ifi.uio.no

Paul Kaufmann
Department of Computer Science
University of Paderborn
Pohlweg 47-49, 33098 Paderborn, Germany
Email: paul.kaufmann@uni-paderborn.de

***Abstract*—The evolvable hardware (EHW) paradigm relies on continuous run-time reconfiguration of hardware. When applied on modern FPGAs, the technically challenging reconfiguration process becomes an issue and can be approached at multiple levels. In related work, virtual reconfigurable circuits (VRC), partial reconfiguration, and lookup table (LUT) reconfiguration approaches have been investigated. In this paper, we show how fine-grained partial reconfiguration of 6-input LUTs of modern Xilinx FPGAs can lead to significantly more efficient resource utilization in an EHW application. Neither manual placement nor any proprietary bitstream manipulation is required in the simplest form of the employed method. We specify the goal architecture in VHDL and read out the locations of the automatically placed LUTs for use in an online reconfiguration setting. This allows for an easy and flexible architecture specification, as well as possible implementation improvements over a hand-placed design. For demonstration, we rely on a hardware signal classifier application. Our results show that the proposed approach can fit a classification circuit 4 times larger than an equivalent VRC-based approach, and 6 times larger than a shift register-based approach, in a Xilinx Virtex-5 device. To verify the reconfiguration process, a MicroBlaze-based embedded system is implemented, and reconfiguration is carried out via the Xilinx Internal Configuration Access Port (ICAP) and driver software.**

I. Introduction

Evolvable Hardware (EHW) is the combination of evolutionary algorithms (EAs) and reconfigurable logic. In addition to being used for the automated design of analog or digital circuits, the paradigm also allows embedded systems to adapt to changing input data distributions and compensate degradation effects in the computational resources [1]. On-chip, run-time adaptable EHW systems usually apply the concept of online evolution, that is, evaluating candidate solutions in the real hardware. Further, some sort of partial reconfiguration of the system is needed, since also the EA is running on the same chip as the EHW target system. Commercial field programmable gate arrays (FPGAs) are a good candidate target platform for such systems, because of their availability and reconfiguration capabilities.

EHW reconfiguration methods: For digital EHW, Xilinx FPGAs are popular for realizing reconfigurable architectures. The reconfigurability of these devices splits roughly into *Full device reconfiguration* and *partial reconfiguration*, where partial reconfiguration is the ability of changing the function of rectangular device areas without interfering with the neighbor elements. Orthogonal to the full and partial reconfiguration is the way the reconfiguration streams are generated. Generally, the methods split into using Xilinx implementation tools for *a priori* generation of the configuration bitstreams, and the development of custom tools for raw bitstream manipulation. The first approach requires a complete high-performance development system, such as a workstation, while the second approach can be made more lightweight and also be executed on an embedded system. As the *a priori* tool-based generation of a full configuration bitstream is very computationally intensive, this approach is impractical for evaluation of candidate circuits consisting of a combination of different functional elements and their connections.

The high level Xilinx modular partial reconfiguration flow allows a coarse-grained division of the system into components that can change, and all potential states of these components can be *a priori* implemented into partial bitstreams. Combinations of these component states can then be instantiated using partial reconfiguration. The drawback of this approach is that the configuration granularity is relatively coarse, and is thus in the context of EHW mostly suitable for modifying architecture parameters [2], [3] or for evolution of systems using high level functions of a certain complexity [4], [5].

The necessity for a more fine-grained reconfiguration approach has motivated the work on computationally efficient bitstream manipulation methods able to run on embedded systems. As the description of the bitstream format is not publicly accessible, reverse engineering has been employed to identify the locations of the lookup table (LUT) configuration bits, whereas the reverse engineering of the routing has been judged too complex and impractical [2]. Therefore, some reconfigurable EHW architectures with a fixed topology and variable LUT functions have been proposed, using Virtex [6], Virtex-II [7], [8], and more recently, Virtex-4 [9] devices. The approaches have achieved fine-grained reconfigurability, but at the cost of the bitstream manipulation effort and a low-level design process with manual placement of FPGA resources, e.g. using the Xilinx FPGA Editor tool. Whereas the most recent device these EHW approaches have supported is the Virtex-4, outside the EHW community there exist reports of direct bitstream manipulation of newer devices, applied to e.g.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

Network-on-Chip [10].

Manipulation of LUT contents can also be done by configuring the LUTs while in a special shift register (SR) mode, from within the implemented circuit. Compared to LUT manipulation using partial bitstreams, SR-based reconfiguration allows for a reduction of the configuration time, and increased design flexibility. This method has been applied to EHW applications in [11]–[13]. However, while on Virtex-II devices all FPGA *slices* (building blocks consisting of LUTs and other basic elements) supported LUT resources to be configured in SR mode, modern devices only offer a limited number of slices supporting this functionality.

Finally, virtual reconfigurable circuits (VRCs) implemented on top of an FPGA is a popular approach for EHW systems. Reconfiguration is here normally done through writing to registers, which control multiplexers and select outputs from all possible implemented functionality. These architectures can typically be reconfigured within a few cycles, and are thus attractive for applications requiring very fast reconfiguration. Another advantage of the approach is the flexibility of design, not requiring any instantiation of FPGA-specific components. An example is the universal EHW system of Sekanina [14]. The main drawback of this approach is the need to implement all possible functionality and thus the resource utilization overhead can be significant.

EHW Classification Architectures: An early use of EHW for pattern recognition was reported by Higuchi et al. [15]. Their architecture was originally applied to character classification but was later used for classification in a prosthetic hand controller (PHC) [16]. It employed a programmable logic array (PLA)-like structure of AND gates followed by OR gates. Although the results showed a competitive classification rate for evolved circuits compared to artificial neural networks (ANNs), it was noted that the size of the employed data set might be insufficient; this is underlined by the strongly varying classification rates.

Using similar EMG data, Torresen [17] conducted experiments on incremental evolution using an EHW architecture. The two-layered architecture consisted of `AND-OR` matrices followed by a selector layer. The `AND-OR` matrices were evolved in the first step followed by the evolution of the selectors. In addition, the best subsystems from different runs were combined into one system. The results showed that a two-step incremental approach can lead to a better generalization performance and shorter computation times than traditional one-step evolution and ANN.

An approach to online EHW classification on FPGAs can be found in [18], [19], where an on-chip system including a PowerPC processor initiates partial reconfiguration on the classifier sub-system. The system uses a direct bitstream manipulation approach to reconfigure a hierarchical two-dimensional array structure. Good classification accuracies are achieved, however the number of inputs and classes are limited.

A different EHW pattern classification system, Logic Design using Evolved Truth Tables (LoDETT), was presented in [20], [21]. LoDETT allows for high accuracy classification on problems with a much higher number of inputs and classes. However, the system does not implement online evolution and relies on synthesis in software before the circuit is implemented on a field-programmable gate array (FPGA). The approach utilizes incremental evolution; i.e., sub-circuits are evolved separately before being assembled into a final system.

The evolution of evolvable hardware classification architectures for prosthesis control has also been investigated by Kaufmann in [22]. There, the automatic acquisition and reuse of modules have been applied on the digital circuits to evolve accurate signal classifiers faster.

Paper Contributions: In our previous work we have proposed an EHW classifier architecture which handles problems with higher input dimensionality and multiple categories like the LoDETT system, but based on a VRC approach and targeting *online* evolution [23]. The architecture shows good performance; comparisons with traditional state-of-the-art approaches have been performed in [24] and [25], and comparisons with previously proposed EHW architectures show favorable classification accuracy and training speed [26]. However, as the FPGA resource requirements of the VRC implementation have been high, we have also investigated an implementation based on SR-based LUT reconfiguration which led to significantly more efficient resource utilization for Virtex-II Pro devices [13]. With more modern 6-input LUT-based FPGAs available, having limited resources for SR-based reconfiguration, it becomes of interest to re-investigate implementation options for the architecture, and for fine-grained EHW approaches in general.

Thus, in this paper we adopt the SR-based reconfiguration technique to a Virtex-5 device, compare with a VRC implementation, and propose a new implementation based on 6-input LUTs with partial reconfiguration through the Xilinx Internal Configuration Access Port (ICAP). Associated with this implementation, we present a flexible approach to the design and reconfiguration workflow, reducing the need of low-level manipulation associated with the previous EHW work on bitstream based partial reconfiguration. The different partial reconfiguration techniques are then evaluated and compared with a focus on FPGA resource utilization.

The remainder of the paper is organized as follows: The next section introduces the EHW classification architecture and the benchmark applications. Then, the details of the different implementations are given in Section III. The experimental results are given and discussed in Section IV. Finally, Section V concludes the paper.

II. The Evolvable Hardware Classification Architecture

This section presents a high-level description of the EHW classification architecture, originally introduced in [23].

A. Architecture description

A high-level view of the system can be seen in Figure 1. The system consists of three main parts – the classification module, the evaluation module, and the central processing unit (CPU),

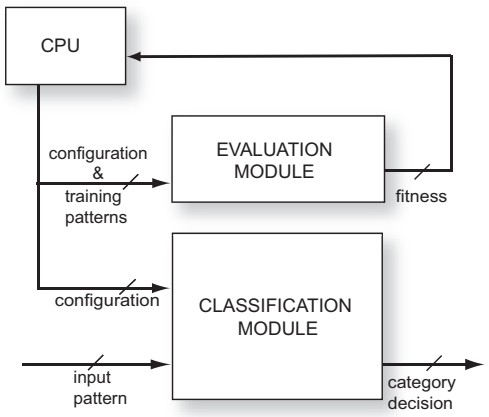

Fig. 1. A high-level overview of the on-chip EHW system.

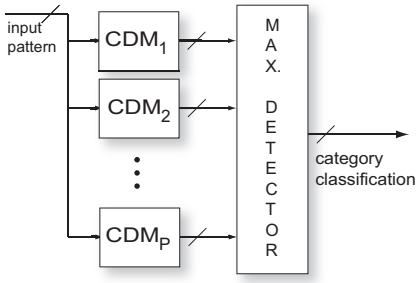

Fig. 2. Classification module.

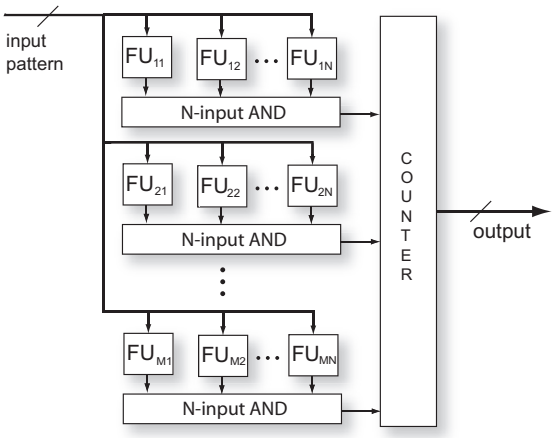

Fig. 3. Category detection module.

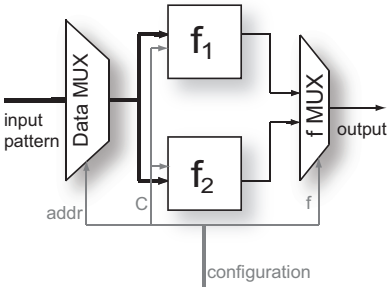

Fig. 4. High-level representation of a functional unit.

all designed to be residing on the same chip. The classification module operates stand-alone except for its reconfiguration which is carried out by the CPU. In a real-world application one would imagine some preprocessing module providing the input pattern and possibly some software interpretation of the classification result. The evaluation module operates in close cooperation with the CPU for the evolution of new configurations, and the evolution happens incrementally on small sub-circuits of the larger classifier module. The evaluation module accepts a configuration bitstring, also called genome, and calculates its fitness value. This information is in turn used by the CPU for running the rest of the evolutionary algorithm (EA). Thus, while the reconfiguration rate in the evaluation module is relatively high, that is, for every new genome to be evaluated, the larger classifier module is only reconfigured infrequently when a new sub-circuit has been evolved.

The classifier system consists of K category detection modules (CDMs), one for each category C_i to be classified—see Figure 2. The input data to be classified is presented to each CDM concurrently on a common input bus. The CDM with the highest output value will be detected by a maximum detector, and the identifying number of this category will be output from the system. Alternatively, the system could also state the degree of certainty of a certain category by taking the output of the corresponding CDM and dividing by the maximum possible output.

Each CDM consists of M "rules" or functional unit (FU) rows—see Figure 3. Each FU row consists of N FUs. The inputs to the circuit are passed on to the inputs of each FU. The 1-bit outputs from the FUs in a row are fed into an N-input AND gate. This means that all outputs from the FUs must be 1 in order for a rule to be activated. The 1-bit outputs from the AND gates are connected to an input counter which counts the number of activated FU rows. As the number of FU rows is increased, so is the output resolution from each CDM. Each FU row is evolved separately from an initial random bitstream, which ensures a variation in the evolved FU rows.

The FUs are the reconfigurable elements of the architecture. As seen in Figure 4, each FU behavior is controlled by a configuration which is dictated by the genome bitstring. Each FU has all input bits to the system available at its inputs, but only one data element (e.g. one byte) of these bits is chosen. One data element is thus *selected* from the input bits, depending on the configuration. This data element, I, is then fed to the available functions. Any number and type of functions could be imagined, but for clarity, in Figure 4 only two functions are illustrated. In addition, the unit is configured with a constant value, C. This value and the input data element are used by the function to compute the output, O, from the unit.

B. Classification Setup

This paper uses two classification benchmarks: The sonar return [27], a two-class problem, and the Olivetti face image

database [28], a 40-class problem. For the selected applications the choice of functions for the FU is the same, based on previous experimentation:

f	Description	Function
0	Greater than (GT)	$O = 1$ if $I > C$, else 0
1	Less than or equal (LTE)	$O = 1$ if $I \leq C$, else 0

The sonar return feature vector contains 60 input data elements, and the image vector has been resized to 8x8 pixels, thus containing 64 elements. In addition, the resolution of the data elements has been scaled to a 6-bit binary representation, which has been demonstrated by earlier experimentation to give adequate results [13]. For the sonar application we use a classifier configuration of $N = 6$, $K = 2$, while the face image application is configured to $N = 8$, $K = 40$.

For the training of the classifier, a genetic algorithm is run independently on each FU row. The fitness score is based on the number of training vectors which are classified correctly for the given row. Each FU is encoded in the genome string with 6, 1, and 6 bits for the feature address, function type, and constant, respectively. Finally, the ensemble of the evolved FU rows constitutes a full classifier. Further details on the evolutionary training setup for the different classification applications can be found in [23], [29].

III. Implementation options

This section will present the implementation alternatives for FPGA runtime reconfiguration of the architecture. The original VRC and SR approaches will be presented first, before describing the proposed 6-input LUT based implementation approach. Although we have earlier considered data elements of up to 8 bits, we focus in this paper on an implementation tuned for the FPGA LUT structure. We therefore choose 6 bit data width for the 6-input LUT and VRC approaches, while the SR based approach will use 5 bits.

A. VRC implementation

The VRC approach consists of implementing all possible configurations of the circuit in hardware and let the contents of configuration registers select the current behavior. The design flow is typically HDL-based and device agnostic, and will result in a high number of LUTs required to implement the multiplexing functionality, and a number of registers to store the configuration. A high-level implementation for the FU using this technique can be seen in Figure 5. The implementation of all possible functionality has, given the specified function set, been reduced to a single GT comparator and the selection of its real or inverted output by a MUX element. Note that this is a high-level description which will in turn be synthesized by the design tool to a number of LUTs. The design abandons the straightforward approach of implementing a multiplexer (MUX) for selecting the input elements because of the high number of FPGA resources required for such an implementation, especially when the data element resolution and the number of elements are high. Instead, one data element of the input vector is presented to the circuit per cycle,

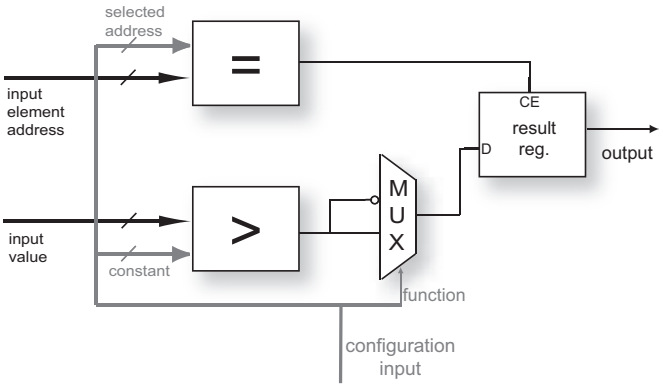

Fig. 5. High-level VRC implementation of the FU. Notice that the configuration registers are not shown.

together with the address of the given element, reducing the MUX implementation to a single comparator which checks for equality between the configuration address and the input address. Although this principle gives a delay equal to the number of data elements in the input vector, it is acceptable for medium-dimensionality input vectors, and will be employed also for the other approaches. Reconfiguration of a VRC-based FU can be performed in as little as one clock cycle, by writing to the configuration registers, if communication overhead is disregarded.

B. Shift register LUT implementation

Although the existing VRC implementation has been optimized compared to a straightforward implementation, the complete classification circuit still requires many FPGA resources. As an example, a typical configuration for the face image application would be N=8, M=8, K=40, which gives a total of 2560 FUs to be instantiated in the classifier. Therefore, it is desirable to further reduce the resource requirements of each single FU.

Instead of storing configuration values in registers which in turn are input as control lines to the VRC, the following approach instead embeds the configuration values together with the functionality into the LUTs which the circuit is built on, thus saving registers and routing resources. Similarly, instead of implementing all possible functions and selecting between the outputs, only one function would be needed implemented at a time. Note that this requires a general LUT structure which can accommodate all desired functions by reconfiguring the LUTs, and also a method to calculate the LUT contents. Such a structure can be designed manually, and there also exists work on automating the process [30].

Some LUTs in Xilinx FPGAs allow LUTs to function as SRs at the same time as having LUT functionality, hereafter referred to as SRLs. This duality makes it possible to shift configuration bits into the LUT, defining the LUT content and thus the behavior, without having to go through the FPGA's normal configuration interfaces such as the ICAP. Our original implementation was based on a Virtex-II Pro device which

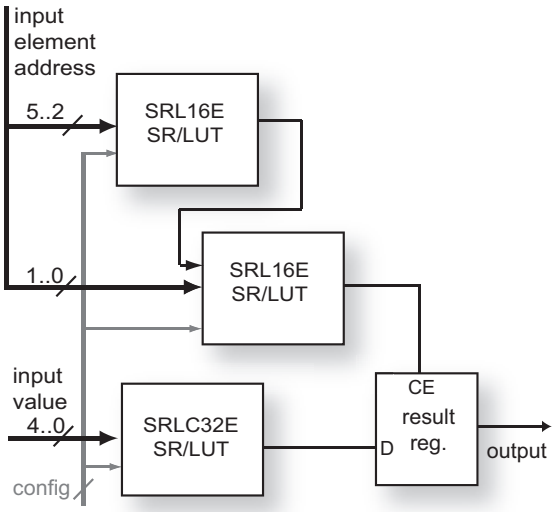

Fig. 6. SRL-based FU implementation.

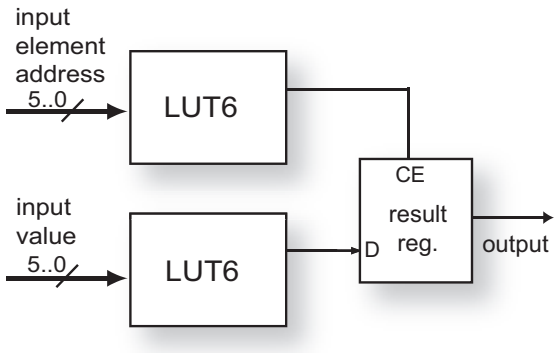

Fig. 7. LUT6-based FU implementation.

allowed all LUTs to be used as 4-input SRLs. However, newer devices (Virtex-5 and above) only allow a limited number of the LUT resources (those located in SLICEM slices) to be configured as SRLs, and the maximum number of inputs is 5, whereas standard LUTs have 6 inputs.

We thus present a modified implementation of the SRL-based FU taking advantage of the increased input width of 5 bits, and set the data width correspondingly, as this will make a resource efficient implementation. The circuit can be seen in Figure 6. For the address comparator part, 6 input bits are needed, and we implement it with two 4-input SRLs, named SRL16E. One of the LUTs checks for equality of the 4 most significant bits (MSBs) of the input and configured addresses, while the second LUT does the same on the 2 least significant bits (LSBs). The second LUT also implements an AND gate to combine the results and can then enable the register to load the result from the function part. The function part of the circuit is implemented using one single 5-input LUT, named SRLC32E, as the data width is tailored to its size. Thus, the function selection f and the constant value C are both embedded into the configuration. Some configuration lines are still needed for the serial shifting of configuration data into the LUTs. Reconfiguration of an SRL-based FU is also fast, disregarding communication overhead the delay is based on the largest SRL size and thus counts 32 cycles.

C. LUT implementation for bitstream partial reconfiguration

The 6-input LUT based circuit is very much similar to the SRL-based circuit, however, all 6 inputs of a LUT can be utilized as opposed to only 5 in the SRL-based approach. This allows encoding the full address comparator into one LUT, hereafter referred to as LUT6, as opposed to needing two LUTs in the SRL approach. Further, a full 6 bits data width can be employed for the function element, whereas for the SRL approach the most practical data width is 5. An illustration of the circuit can be seen in Figure 7. Another difference from the SRL approach is that there are no explicit configuration lines, as the configuration is set using the ICAP. The design and reconfiguration workflow of the approach is outlined in the following steps:

- The design of the circuit is VHDL-based, although explicit instantiation of the LUT6 primitives is required. As the FUs, rows, and CDMs in the architecture are instantiated using `for ... generate` statements, each instantiated LUT will be given a unique numbered name during the implementation process which will be easily identifiable later. It is necessary to apply the `lock_pins` constraint to the LUTs so that the router does not swap the input order of the pins in the routing process.
- When the Xilinx tools have fully implemented the design to an `.ncd` file, we use the `xdl -ncd2xdl` command to generate a text based description of the system. We have written a script which extracts the location and LUT type (A-D) of each of the LUTs from the `.xdl` and combines this with device information from the `xdl -report` command to find the slice type (SLICEM, SLICEL) at the given location. Finally, C code is generated from this information, to be used in an embedded program controlling the reconfiguration.
- The LUTs can then be reconfigured via the ICAP at runtime from e.g. an embedded MicroBlaze processor core, using the generated information to locate the desired LUTs. When using EDK, Xilinx provides the XPS_HWICAP core for interfacing the ICAP, and associated driver software. The most accessible approach to reconfiguring the LUTs is to use the provided `XHwIcap_SetClbBits` function[1] which reads a configuration frame from the ICAP, manipulates the bits corresponding to the specified LUT position, and writes the content back. It would however also be possible to directly manipulate the configuration frame for a more

[1]The supplied bitstream position lookup functions did not function correctly for our Virtex-5 device, but a correction was found at `http://forums.xilinx.com/t5/Embedded-Development-Tools/Issues-about-HWICAP-driver-for-Virtex-5/m-p/285288`

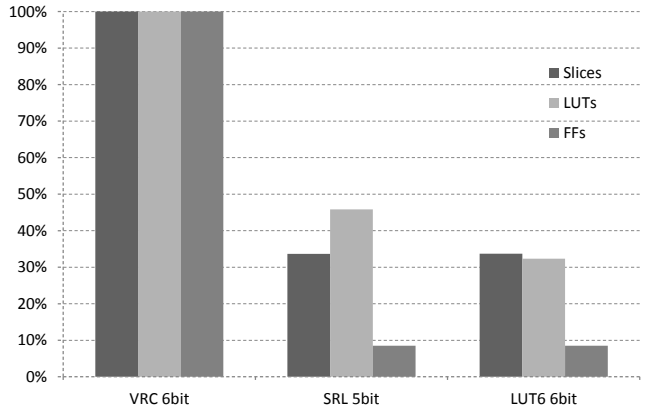

Fig. 8. Relative resource requirements for the sonar classifier

efficient approach.

It should also be noted that reconfiguration may corrupt data which is stored in SRLs or LUT RAM in the same column, and thus care should be exercised so that no such resources from other IPs are overlapping with the EHW architecture. A simple way of prohibiting this is to apply the `area_group` constraint to reserve a specific region of the FPGA for implementation of the EHW architecture only.

IV. Experiments and results

This section describes the results of the conducted implementations and experiments. In these implementations we consider only the classifier module and not the evaluation module or other parts controlling the EA.

A. FPGA resource utilization

For the implementation experiments, a Xilinx ML505 evaluation board fitted with a Virtex-5 XC5VLX50T device has been employed. The FUs have been implemented for the VRC, SRL, and LUT6 approaches as described in Sec. II. Further, the FUs have been assembled in rows and then a complete classifier module for the sonar application, with $M = 20$. Note that in these results we do not consider the max detector part of the classifier, or any logic related to reconfiguration such as a CPU or ICAP module. The resource utilization for all approaches are reported in detail in Table I, and a visualization of resource requirements relative to the VRC implementation of the whole classifier can be seen in Figure 8. The results are reported from a full implementation process in Xilinx ISE 14.2, with default parameters. "FFs" refers to the number of flip flops used. The LUT6 approach uses 32% and 70% of the LUTs used by the VRC and SRL approaches, respectively, however on the slice level the SRL and LUT6 approaches both utilize 34% of the number of slices used by the VRC approach.

B. Maximum classifier configuration

After investigating the amount of resources required for a given configuration of the classifier, we wanted to investigate the maximum configuration possible to pack in a given device.

TABLE II
Maximum classifier rows for an XC5VLX50T device, resource usage, and classification accuracies.

Approach	Max. rows / CDM	Slices	Accuracy
Face			
VRC	5	7044	93.3%
SRL	3	1775	87.8%
LUT6	20	6904	97.0%
Sonar			
VRC	130	6999	86.1%
SRL	90	1992	84.8%
LUT6	550	7156	86.3%

This perspective also allows us to observe the effect of the limited number of LUTs available in shift register mode for the SRL approach. We targeted the XC5VLX50T device, where 27% of the slices offer SRL functionality (SLICEM), and searched for the maximum possible configuration for each of the approaches. Configurations were tested in increments of 1 and 10 rows, M, per CDM for the face and the sonar application, respectively, until the place and route process was unable to fit the design. The results are reported in Table II, together with the number of slices used for the given configuration. We also report the classification accuracies corresponding to these configurations, found through evolution in a software simulation. The reported numbers are average classification accuracies based on 10 runs, each using 10-fold cross validation, for the face image data set, and 100 runs on a separate training and test set for the sonar application. The results show that, in the sonar case, the LUT6 approach makes it possible to fit an architecture 4.2 times larger than when using VRC, and 6.1 times larger than when using SRL.

C. Verification and reconfiguration speed

As a verification of the proposed approach, we have implemented an embedded system using Xilinx EDK 14.2, for the Xilinx ML505 board. The system contains a MicroBlaze soft CPU core, an XPS_HWICAP core for internal reconfiguration, a custom designed PLB core containing a LUT6-based classifier circuit, and some utility cores. The system clock frequency was set to 100MHz, limited by the ICAP core frequency. The FU row classifier was designed as described in Section III-C, and the positions of the automatically placed LUTs were extracted from the final generated .ncd file. A C program running on the MicroBlaze used this information to configure and reconfigure the LUTs of the classifier, and apply stimuli for classification. Using the Xilinx supplied ICAP driver function `XHwIcap_SetClbBits`, which reads and writes back a full frame, the reconfiguration time for a single LUT was measured to be 1.96 ms. In the case of the sonar classifier, the reconfiguration time for a full row would then be 23.54 ms.

D. Discussion

The implementation results in Table I clearly show that both the SRL-based and the LUT6-based implementations

TABLE I
FPGA ELEMENTS REQUIRED FOR DIFFERENT IMPLEMENTATIONS OF THE SONAR CLASSIFIER.

	VRC 6 bit			**SRL 5 bit**			**LUT6 6 bit**		
Implementation	LUTs	FFs	Slices	LUTs	FFs	Slices	LUTs	FFs	Slices
FU	6	1	4	3	1	3	2	1	2
FU + conf. regs.	7	14	7	NA	NA	NA	NA	NA	NA
Whole classifier	1773	3410	1287	813	290	433	573	290	434

offer significant resource savings compared to a VRC-based implementation of the architecture. These numbers are explained by both reducing the number of inputs needed to the circuit by storing the configuration in the LUT contents, and also by saving register resources which are needed to store the configuration in the VRC case. From the same resource utilization perspective, the LUT6 approach has an advantage over the SRL approach by being able to take advantage of all 6 inputs to the LUTs, thus saving a LUT in the address comparison part, and also having higher precision (6 vs. 5 bits) in the function part. However, the extra LUT required by the SRL approach does not generate any slice utilization overhead in this case, as shown in Table I.

As EHW applications often can benefit from a low-level hardware specification of its components, the increased functionality of the wide LUTs in the modern FPGA architectures can simplify the design process. Using fewer LUTs also facilitates the routing and should allow for specification of larger EHW structures. The results from Table II show that the LUT6 approach packs well in the FPGA, as the maximum possible rows is in the order of 4 times as large as with VRC, even though the resource utilization results from Table I only would suggest a threefold improvement.

Moreover, when considering the results from Table II it becomes evident that the SRL approach is limited by the number of available SLICEMs, and that even the VRC approach can fit a larger classifier configuration. Although the remaining resources could be used for other IP cores, the SRL approach does not seem to be suitable for implementation of large EHW architectures on modern Xilinx FPGAs.

Although the proposed partial reconfiguration implementation is low-level compared to the device-agnostic VRC implementation, the design process described here is relatively simple. Instantiation of LUTs in VHDL and then automatic place and route is more convenient, flexible, and requires less expert knowledge than earlier EHW approaches, describing manually placed components with e.g. the FPGA Editor [2]. For more complex EHW architectures, several LUTs may have to be combined in order to achieve the desired functionality, and the complexity may prove to be a design challenge. In this case specialized tools such as TMAP, which calculates LUT contents based on parameter values, would be helpful [30].

The current approach is based on manipulation of a partial bitstream to reconfigure the LUTs. While this can be achieved through Xilinx driver functions for the XPS_HWICAP core, the method is somewhat inefficient given that it is necessary to read and write back one full configuration frame per LUT. Therefore, if higher reconfiguration speed is required, an improvement to the current proof-of-concept reconfiguration method would be to write custom code to directly manipulate the bitstream. Although the exact placement of LUT contents in the bitstream is not documented, several works demonstrate the feasibility of such an approach [2], [10]. Moreover, another possible way of speeding up the partial reconfiguration would be to use a custom ICAP core which optimizes and/or overclocks the ICAP interface [31].

Note that in the current application the reconfiguration speed of the full classifier module is not critical, since the EA is applied to a single FU row at a time, in a separate evaluation module. This single row under evaluation could be quickly reconfigured using the VRC implementation, as it is only a fraction of the full classifier and is not required to be compact in the same way. Therefore, although the proof-of-concept implemented here does not present an effort to achieve high-speed reconfiguration, the approach has the advantage of being readily available and may be sufficient for applications where reconfiguration speed is not critical.

The classification accuracies provided are intended to demonstrate some relationships between different configurations and classification performance, but the effort here has not been focused on maximizing classifier performance. The low difference in moving from 130 to 550 rows of the sonar classifier indicates that using the LUT6 approach it would be possible to fit this classifier configuration in a much smaller FPGA and still achieve good results. Moreover, the results from the face application indicate that, on the current device, only the LUT6 approach would fit a configuration which can give competitive results.

Given the current fit of the function part in only one 6-input LUT, it would be interesting to investigate if higher discrimination performance could be achieved by evolving the entire contents of the LUT instead of following the currently pre-defined function. This would however come at the cost of having 64 instead of 7 bits to evolve for the function part, and would thus increase the search space significantly.

V. CONCLUSIONS AND FUTURE WORK

In this work we have presented a dynamically reconfigurable evolvable hardware classifier architecture by means of partial reconfiguration of 6-input LUTs. This reconfiguration technique is intriguing, as it has multiple benefits over other reconfiguration types: Compared to the VRC implementation, the proposed approach allows for a significantly more compact implementation. Compared to the SRL-based reconfiguration, which only gives limited access to the FPGA LUTs, partial reconfiguration allows for a full utilization of the FPGA LUT

resources, both in terms of the number of resources as well as the number of LUT inputs. While the VRC approach still has advantages in terms of design flexibility and fast reconfiguration, the design process of the proposed approach is significantly simplified compared to a low-level manual placement of resources, and no reverse engineering of the configuration bitstring format is required.

In our next steps we will validate our implementation on the current Xilinx 7 series, and also investigate improving the reconfiguration times. Here, we will implement own methods for efficient bitstream manipulation and investigate the speeds of a complete evolvable system, considering the genotype processing times and ICAP interface bandwidths. Another path for future investigation would be to study the performance of the approach for other EHW architectures.

REFERENCES

[1] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu, "Real-world applications of analog and digital evolvable hardware," *IEEE Trans. Evolutionary Computation*, vol. 3, no. 3, pp. 220–235, 1999.

[2] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and microsystems*, vol. 29, no. 5, pp. 211–223, 2005.

[3] J. Torresen, G. A. Senland, and K. Glette, "Partial reconfiguration applied in an on-line evolvable pattern recognition system," in *NORCHIP, 2008.* IEEE, 2008, pp. 61–64.

[4] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "A fast reconfigurable 2d hw core architecture on fpgas for evolvable self-adaptive systems," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*. IEEE, 2011, pp. 336–343.

[5] R. Dobai and L. Sekanina, "Towards evolvable systems based on the xilinx zynq platform," in *2013 IEEE International Conference on Evolvable Systems (ICES)*, ser. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE Computational Intelligence Society, 2013, pp. 89–95.

[6] P. Haddow and G. Tufte, "Bridging the genotype-phenotype mapping for digital FPGAs," in *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*, 2001.

[7] A. Upegui, "Dynamically reconfigurable bio-inspired hardware," Ph.D. dissertation, Ecole Polytechnique Fdrale de Lausanne (EPFL), 2006, thesis No. 3632.

[8] S. Lynch, "A platform for intrinsic evolution of digital circuits on Virtex II pro," B.E. Electronic and Computer Engineering Project Report, National University of Ireland, Galway, 2006.

[9] F. Cancare, M. D. Santambrogio, and D. Sciuto, "A direct bitstream manipulation approach for virtex4-based evolvable systems," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 853–856.

[10] C. H. Hoo and A. Kumar, "An area-efficient partially reconfigurable crossbar switch with low reconfiguration delay," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 400–406.

[11] G. Tufte and P. C. Haddow, "Biologically-inspired: A rule-based self-reconfiguration of a virtex chip." in *Proc. of International Conference on Computational Science 2004*, ser. Lecture Notes in Computer Science, vol. 3038, May 2004, pp. 1249–1256.

[12] H. Kawai, Y. Yamaguchi, M. Yasunaga, K. Glette, and J. Torresen, "An adaptive pattern recognition hardware with on-chip shift register-based partial reconfiguration," in *Proceedings International Conference on Field-Programmable Technology (ICFPT)*. IEEE CS Press, 2008, pp. 169–176.

[13] K. Glette, J. Torresen, and M. Hovin, "Intermediate level FPGA reconfiguration for an online EHW pattern recognition system," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*. IEEE, 2009, pp. 19–26.

[14] L. Sekanina and R. Ruzicka, "Design of the Special Fast Reconfigurable Chip Using Common FPGA," in *Proceedings of the IEEE Conference on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2000, pp. 161–168.

[15] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, B. Manderick, and T. Furuya, "Evolvable Hardware and its Applications to Pattern Recognition and Fault-Tolerant Systems," in *Towards Evolvable Hardware: The evolutionary Engineering Approach*, ser. LNCS. Springer, 1996, vol. 1062, pp. 118–135.

[16] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iwata, D. Keymeulen, and T. Higuchi, "A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI," ser. LNCS, vol. 1478. Springer, 1998, pp. 1–12.

[17] J. Torresen, "Two-Step Incremental Evolution of a Digital Logic Gate Based Prosthetic Hand Controller," ser. LNCS. Springer, 2001, vol. 2210, pp. 1–13.

[18] D. B. Bartolini, F. Cancare, M. Carminati, and D. Sciuto, "Hera: Hardware evolution over reconfigurable architectures," in *Computing in Heterogeneous, Autonomous' N'Goal-Oriented Environments (CHANGE), 2011 1st International Workshop on*. IEEE, 2011, pp. 1–8.

[19] F. Cancare, D. B. Bartolini, M. Carminati, D. Sciuto, and M. D. Santambrogio, "On the evolution of hardware circuits via reconfigurable architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 22, 2012.

[20] M. Yasunaga, T. Nakamura, and I. Yoshihara, "Evolvable Sonar Spectrum Discrimination Chip Designed by Genetic Algorithm," in *Systems, Man and Cybernetics*, vol. 5. IEEE, 1999, pp. 585–590.

[21] M. Yasunaga, T. Nakamura, I. Yoshihara, and J. Kim, "Genetic Algorithm-based Design Methodology for Pattern Recognition Hardware," in *Proceedings 3rd International Conference on Evolvable Systems (ICES)*, ser. Lecture Notes in Computer Science, vol. 1801. Springer, 2000, pp. 264–273.

[22] P. Kaufmann, *Adapting Hardware Systems by Means of Multi-Objective Evolution*. Logos Verlag.

[23] K. Glette, J. Torresen, and M. Yasunaga, "An Online EHW Pattern Recognition System Applied to Face Image Recognition," in *Proceedings Applications of Evolutionary Computing (EvoWorkshops2007)*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4448, pp. 271–280.

[24] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen, and B. Sick, "Classification of electromyographic signals: Comparing evolvable hardware to conventional classifiers," *IEEE Transactions On Evolutionary Computation*, vol. 17, pp. 46–63, 2013.

[25] K. Glette, P. Kaufmann, C. Assad, and M. T. Wolf, "Investigating evolvable hardware classification for the biosleeve electromyographic interface," in *Proceedings of the 2013 IEEE International Conference on Evolvable Systems*. IEEE Press, 2013, pp. 73–80.

[26] K. Glette, J. Torresen, P. Kaufmann, and M. Platzner, "A Comparison of Evolvable Hardware Architectures for Classification Tasks," in *Proceedings 8th International Conference on Evolvable Systems (ICES)*, ser. Lecture Notes in Computer Science, vol. 5216. Springer, 2008, pp. 22–33.

[27] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets." *Neural Networks*, vol. 1, no. 1, pp. 75–89, 1988.

[28] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*. IEEE, 1994, pp. 138–142.

[29] K. Glette, J. Torresen, and M. Yasunaga, "An Online EHW Pattern Recognition System Applied to Sonar Spectrum Classification," in *Proceedings 7th International Conference on Evolvable Systems (ICES)*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4684, pp. 1–12.

[30] K. Bruneel and D. Stroobandt, "Automatic generation of run-time parameterizable configurations," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. IEEE, 2008, pp. 361–366.

[31] S. G. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced icap hard macro," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 174–180.