Intermediate Level FPGA Reconfiguration for an Online EHW Pattern Recognition System

Kyrre Glette, Jim Torresen, and Mats Hovin
University of Oslo
Department of Informatics
P.O. Box 1080 Blindern, 0316 Oslo, Norway
{kyrrehg,jimtoer,matsh}@ifi.uio.no

Abstract

We propose a field programmable gate array (FPGA) implementation for a run-time adaptable evolvable hardware classifier system. Previous implementations have been based on a high-level virtual reconfigurable circuit technique which requires many FPGA resources. We therefore apply an intermediate level reconfiguration technique which consists of using the FPGA lookup tables as shift registers for reconfiguration purposes. This leads to significant resource savings, reducing the classifier circuit size to less than one third of the original implementation. This in turn has made it possible to implement a larger, more accurate classifier than before, giving 97.5% recognition accuracy for a face image application. Experiments also show that a reduction of data element resolution can lead to further resource savings while still maintaining high classification accuracy.

1 Introduction

Pattern recognition systems could benefit from evolvable hardware (EHW) in having a high speed of classification, as well as being contained in a compact and energy-saving circuit. Further, pattern recognition systems could benefit from run-time adaptation, where EHW could have an advantage.

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 evolutionary algorithm (EA) is running on the same chip as the EHW target system. Commercial field programmable gate arrays (FPGAs) would be a good candidate target platform for such systems, beacuse of their availability and reconfiguration capabilities. However, going from a high-level circuit description to an FPGA configuration bitstream is normally a lengthy process which involves invoking design implementation tools on a computer. In addition, manual manipulation of the FPGA configuration bitstream is often unpractical due to undisclosed bitstream formats.

A popular alternative to direct manipulation of the FPGA bitstream has been to implement a circuit with user-defined reconfigurability in the FPGA. Circuit functionality could in this case be modified by for example using registers to control the multiplexing of different outputs. Although providing fast reconfiguration and flexibility through device independency, a disadvantage of this approach is the increased resource utilization overhead associated with having to implement all possible connections and functionality. This method has been termed virtual reconfigurable circuit (VRC) [10] or virtual FPGA [5] in earlier work.

In some cases it is still possible to modify the contents of lookup tables (LUTs) inside the FPGA, by reverse engineering of the bitstream, by utility functions, or even by configuring the LUTs while in a special shift register (SR) mode. The first two approaches can be combined with the use of the Internal Configuration Access Port (ICAP) found in recent Xilinx Virtex FPGAs. This allows for one part of the design to reconfigure another part of the chip with a partial bitstream, and is an interesting possibility for EHW. Work on reverse engineering the bitstream in order to reconfigure LUTs through the ICAP in EHW systems has been performed in [13]. Another EHW approach using the ICAP and Xilinx utility functions in order to reconfigure LUTs can be found in [9]. The third EHW approach shifts configuration data into LUTs by using the SR behavior of Xilinx Virtex LUTs [12].

An early use of EHW for pattern recognition was reported in [6]. Originally, the architecture was applied to character classification but later on used for classification in a prosthetic hand controller [7]. The classifier architecture is a programmable logic array (PLA)-like structure of

AND gates followed by OR gates. The configuration of the architecture was evolved using an EA implemented on the same chip as the classifier, resulting in an online adaptable system. Experiments on two-phase incremental evolution of an EHW architecture applied to prosthetic hand control (PHC) were presented in [11]. A different EHW pattern classification system, Logic Design using Evolved Truth Tables (LoDETT), was presented in [14]. LoDETT allows for high accuracy classification on problems with a much higher number of inputs and outputs. Although providing high classification accuracy, the system lacks the ability of online evolution and relies on a software-based implementation process before the circuit is downloaded to an FPGA. The approach utilizes incremental evolution, i.e., sub-circuits are evolved separately before being assembled into a final system.

An alternative approach, the Functional Unit Row (FUR) architecture, was proposed by the authors in [2]. The system provides high classification capabilities, like the LoDETT system, combined with online evolution. This was applied to face image recognition and slightly higher recognition accuracy than the LoDETT system was achieved. While a large number of inputs to the AND gates in LoDETT can be optimized away during circuit synthesis, the run-time reconfiguration aspect of the online FUR architecture has led to a different approach employing fewer elements. The reconfiguration of the system is VRC-based and implementation details for the on-chip adaptable system are given in [4, 3].

Another classification architecture has been proposed in [15], the Direct Data Implementation (DDI) classifier, related to the LoDETT architecture. While not being an EHW system, this architecture has been enabled for online adaptation in [8]. In this case, reconfiguration is made possible through the use of the SR behavior of Xilinx Virtex LUTs.

While the FUR system has provided good classification results, resource requirements for implementation have been relatively high due to the VRC-based reconfiguration approach. It is therefore the goal of this paper to investigate the possible advantages of applying the aforementioned SR-based reconfiguration to the FUR classifier.

The next section introduces the FUR architecture. Then, the implementation details are given in Section 3. The experimental results are given and discussed in Section 4. Finally, Section 5 concludes the paper.

2 The online EHW architecture

The FUR EHW architecture, proposed in [2], is designed as a circuit whose behavior and connections can be modified through online reconfiguration. By writing the genome bitstream from the genetic algorithm (GA) to the internal configuration lines, one obtains the phenotype circuit which can then be evaluated.

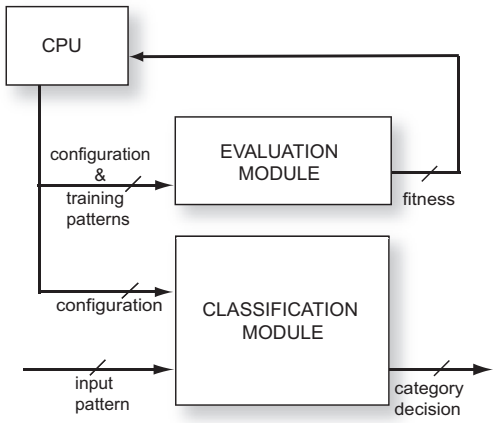

Figure 1. A high-level overview of the on-chip EHW system.

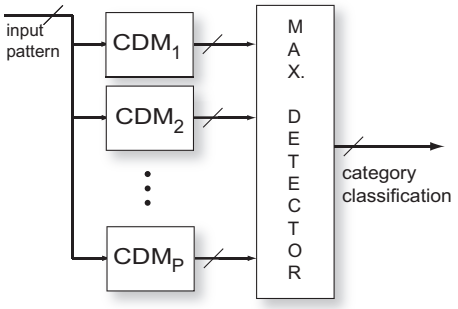

Figure 2. Classification module.

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), 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. 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 GA. The evaluation module has been implemented and described in detail in [4].

The classifier system consists of K category detection modules (CDMs), one for each category C_i to be classified

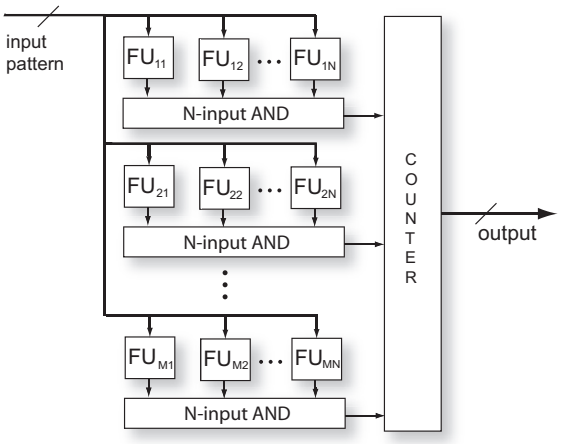

Figure 3. Category detection module.

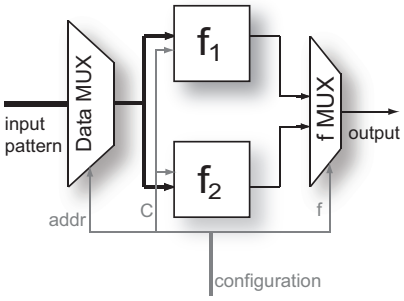

Figure 4. High-level representation of a functional unit.

– 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 from an initial random bitstream, which ensures a variation in the evolved FU rows.

The FUs are the reconfigurable elements of the architecture. This section describes the FU in a general way, and section 3 will describe the implementation details. As seen in Figure 4, each FU behavior is controlled by configuration lines connected to the configuration registers. 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 lines. 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.

3 Implementation

This section first presents the classification applications which will have consequences for the details of the implementation of the classifier system. Then the original VRC-like implementation will be presented, before the new proposed SR-based implementation is explained.

3.1 Applications

This paper will mainly focus on two classification applications: The sonar return and the face image recognition tasks. In addition, an electromyographic (EMG) signal classification task is also discussed. Detailed information about the application of the FUR architecture to these tasks, including the evolutionary training process and classification accuracy, can be found in [2, 3, 1]. For all of the abovementioned tasks the choice of functions for the FU is the same, and can be summarized as follows:

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

In addition, for all of the problems, the input vector consists of 8-bit elements. The sonar return feature vector contains 60 8-bit elements, the image vector contains 64 elements, while the EMG vector contains only 4 elements. For the implementations considered in this paper we will assume the number of input elements to be in the range of the sonar and image tasks.

3.2 Original implementation

A VRC-like implementation has been proposed in [2] and further detailed in [4]. This approach consists of implementing all possible configurations of the circuit in hardware and let configuration register content control the behavior. We will hereafter refer to this approach as the VRC

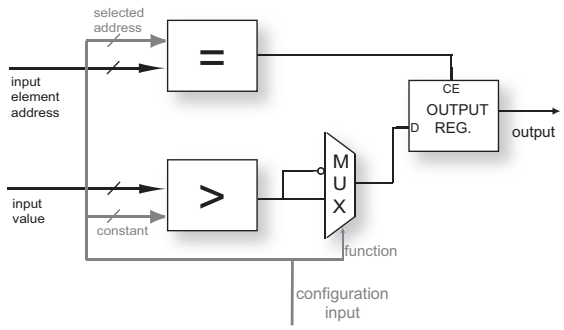

Figure 5. VRC implementation of the FU.

approach, however it should be noted that there could be details which differ from other authors' definition of the term. A high-level implementation for the FU using this technique can be seen in Figure 5. Notice here that the actual configuration registers are not shown. This method abandons the straightforward approach of implementing a large 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, time multiplexing is applied, that is, each data element of the input vector is presented to the circuit sequentially, one per cycle, together with the sequence number (input address) of the given element. This has the advantage of reducing the MUX implementation to a single comparator which checks for equality between the configuration address and the input address. When the addresses are equal, the output of the comparator makes a register store the result of the function part of the circuit, calculated from the selected input element. The drawback of this approach, compared to a standard MUX implementation, is the increased delay: a cycle count which equals to the number of data elements in the input vector. 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.

3.3 Shift register-based implementation

Although the existing VRC implementation has been optimized compared to a straightforward implementation, given the large number of FUs often required for the classification applications, 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 proposed approach instead *embeds* the configuration values together with the functionality into the FPGA LUTs which the circuit is built on. By reconfiguring the actual FPGA LUTs for each virtual circuit reconfiguration, the configuration registers can be omitted, and thereby resources are saved. Further, by not having the configuration values as inputs to the circuit, the total number of inputs to the logic is reduced, allowing for simpler circuits and therefore possible further savings. In addition, in the case of several functions performed by an FU, instead of implementing all possible functions and selecting the desired function's output, only one function would be needed implemented at a time. Note that this requires a general structure which can accommodate all of the functions by reconfiguring the LUTs. It should also be noted that this last advantage is less visible for the FU in the current application, where the LTE output can be found by only inverting the output from the GT circuit.

The approach considered in this paper consists of exploiting the dual nature of LUTs in the Xilinx Virtex series which allow LUTs to function as SRs at the same time as having LUT functionality. This 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 or external interfaces. These dual-mode SR and LUT elements will hereafter be referred to as SRLs. The following sections present how such an SRL-based FU can be constructed based on 4-input SRLs found in Xilinx FPGAs.

3.3.1 Address comparator

The VRC-based address comparator for time multiplexing, described in Section 3.2, needs the same number of input bits both from the configuration register which holds the configured address and the input address which holds the address of the current input element. However, with the SRL-based approach, the configured address can be stored in the comparator circuit itself, removing half of the inputs. Therefore, given a 6-bit addressing scheme, 6 inputs are needed. It is possible to implement the address comparator with 2 4-input SRLS, as seen in Figure 6. 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). A MUX element present in the FPGA slice, which functions as an AND gate, checks that both of the equality tests are true before the result from the function part of the FU can be stored in the output register. Two 1-bit configuration inputs are needed per SRL, one for data and one for configuration control.

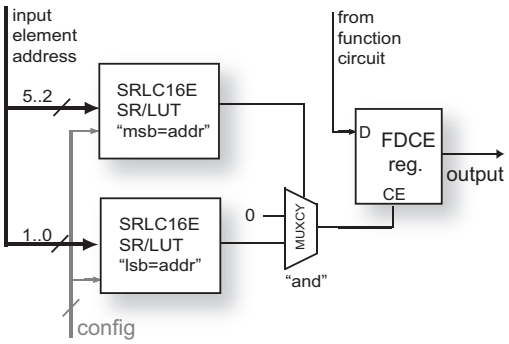

Figure 6. SRL-based address comparator.

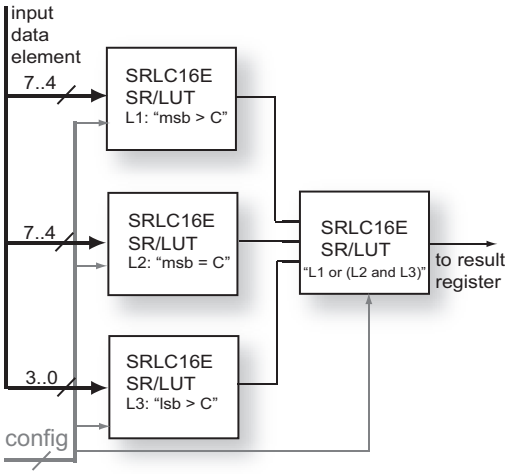

Figure 7. SRL-based 8-bit function circuit.

3.3.2 Function element

The part of the circuit which generates the functionality of the FU has two configuration parameters: the function selection f and the constant value C. While the VRC approach can be optimized according to the description in Section 3.2, it is still necessary to have the configuration parameters as inputs, giving 17 inputs in total.On the other hand the SRL approach needs, as C and f can be embedded in the circuit, only the selected 8-bit input value I as input. A circuit is therefore proposed, which performs an 8-bit GT/LTE comparison, using a compound of 4 4-input LUTs – see Figure 7. The first LUT, $L1$, checks if $I > C$ for the MSBs, in which case it is not necessary to check the LSBs. The next two LUTs, $L1$ and $L2$, checks the second possible case for which I could be greater, if $I = C$ for the MSBs and $I > C$ for the LSBs. The final LUT then combines these two cases to give the GT function, or its inverse, the LTE function. If one considers only 4-bit precision, that is, using 4-bit values for I and C, by using the SRL approach one could fit the entire functionality into one 4-input LUT.

4 Experiments and results

This section describes the results of the implementations and experiments which have been undertaken. In these implementations we consider only the classifier module and not the evaluation module or other parts controlling the EA. Details about evolution and the evaluation module can be found in [4].

4.1 Implementation results

The SRL-based FU has been implemented in both 8-bit and 4-bit precision configurations. The Xilinx Virtex-II Pro has been chosen as the target device since it allows all LUTs to be instanciated as SRLs. The systems have been described in VHDL with explicit instantiation of device-specific primitives, however no further low-level efforts, such as floorplanning, have been performed. The FUs have then been assembled into a FU row and further a complete classifier module for the sonar application, with a classifier configuration of $N = 6$, $M = 20$, and $K = 2$. For comparison, the VRC-based equivalents have also been implemented, and resource utilization for all configurations have been measured.

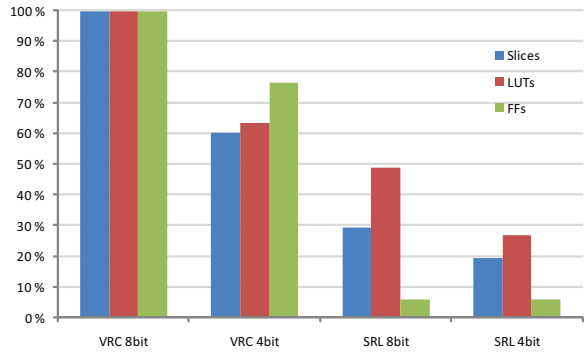

Figure 8. Relative resource requirements for the sonar classifier

A detailed resource utilization report can be seen in Table 1, and a visualization of resource requirements relative to the the 8-bit precision VRC implementation of the whole classifier can be seen in Figure 8. Note that the numbers reported have been taken from the synthesis report of the Xilinx synthesis tool, which is an estimate of the final utilization, with default optimization parameters. The exception to this is the ”Full impl.” row which reports the utilization after a full implementation process for an XC2VP30

Table 1. FPGA elements required for different implementations of the sonar classifier.

	VRC 8 bit			VRC 4 bit			SRL 8 bit			SRL 4 bit		
Implementation	LUTs	FFs	Slices	LUTs	FFs	Slices	LUTs	FFs	Slices	LUTs	FFs	Slices
Address comp.	4	0	2	4	0	2	2	0	1	2	0	1
Function part	9	1	5	3	1	2	4	1	2	1	1	1
Whole FU	13	1	7	7	1	4	6	1	3	3	1	2
FU+conf. regs.	13	16	14	8	12	8	NA	NA	NA	NA	NA	NA
Whole classifier	3280	4080	3560	2080	3120	2150	1600	240	1051	889	240	697
Full impl. VP30	3288	4048	3926	2086	3112	2975	1840	240	1287	1129	240	820

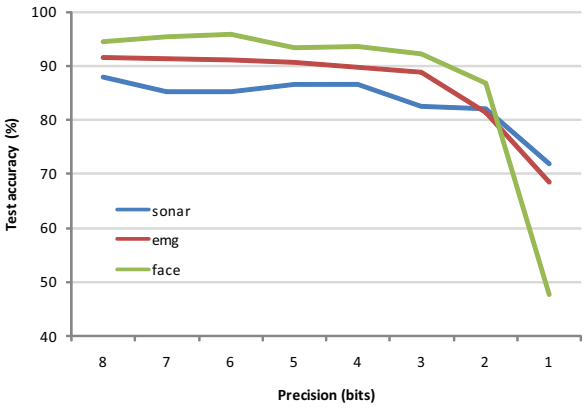

Figure 9. Classification accuracy for various applications as a function of data precision.

FPGA, including added overhead such as routing. The row "FU+conf. regs." refers to the VRC implementation including the register flip flops for storing the configuration. "FFs" refers to the number of flip flops used.

Configuration of the SRLs is in this case implemented as parallel over one FU row, which means that with 6 FUs and 6 SRLs per FU there are 36 parallel configuration bits which are shifted into a FU row per cycle, and the total configuration time per row is 16 cycles. The VRC implementation can be configured in one cycle, but 15 configuration bits are required per FU and this gives a configuration word size of 90 bits per FU row.

4.2 Data element precision effects

The effect of the data element precision on the recognition accuracy was investigated. A number of simulation experiments were run, with the precision varying from 1 to 8 bits on all three classification applications. The resulting test set classification accuracies have been plotted in Figure 9. Further experimental setup details can be found in [2, 3, 1], although some details differ, such as the maximum number of generations allowed for the EA.

Table 2. Implementation configurations for the face and sonar classifiers.

		VRC		SRL	
Config.	Acc.	Slices	Device	Slices	Device
Face					
8x8x4	93.5%	22864	VP50	7281	VP20
8x8x8	95.4%	37904	VP100	11121	VP30
8x18x8	**97.5%**	85304	NA	25267	VP70
Sonar					
6x20x4	86.6%	2150	VP4	697	VP2
6x20x8	87.8%	3560	VP7	1051	VP2
6x58x8	91.4%	10324	VP20	3052	VP7

4.3 Size and performance considerations

We have investigated the possible benefits of the reduced resource utilization of the new implementations. The benefits can be in the form of possibilities for larger FUR systems in the same FPGA, e.g. CDMs with more rows, or in the form of systems requiring less resources than before. This will in turn have an impact on the recognition accuracy of the system, or the smallest possible device on which the system can be installed. The results of these investigations have been summarized in Table 2, which suggests some possible implementation configurations and presents their test set classification accuracy and resource requirements. The three numbers identifying the configuration represent N, M, and the data element bit precision, respectively. The device column indicates the smallest possible device the system can fit into from the Xilinx Virtex-II Pro XC2VP range, given the slice count estimates. The "NA" for the face image VRC 8x18x8 configuration indicates there is no device which is large enough to accomodate the system. It would also be possible to fit the smallest configurations into lower-cost devices, such as the Spartan-3A range.

The 6x20x4 sonar configuration would require 59% of the slices of an XC3S400A device in the case of a VRC implementation, while it would require 38% of the slices of an XC3S200A device with an SRL implementation. Note that the Spartan-3A range only has 50% SLICEM slices, which are the slices containing SRL LUTs. Therefore the SRL-based implementation can only make use of 50% of the Spartan-3A slices for reconfiguration purposes. The highest classification accuracy of 91.4% for the sonar application is the same results as was reported in [3], while the accuracy of 97.5% for the face image application is higher than the 96.3% previously achieved in [2].

4.4 Discussion

It is clear from the implementation results that the SRL-based configurations offer significant savings in terms of FPGA resources. The number of LUTs required is halved going from the 8-bit VRC configuration to the 8-bit SRL configuration, which can be primarily be explained by the omission of the configuration register inputs in the SRL-based circuits. More importantly, the number of slices needed for the SRL version is reduced to one third of the slices needed for the VRC version, which can be explained by the configuration registers in the VRC approach requiring several slices. Going from 8-bit precision to 4-bit precision further reduces the resource requirements of the SRL approach, due to the simpler function part fitting in only one LUT. With applications requiring 16 or fewer input elements it would also be possible to fit the address comparator part into one LUT, opening for a possible implementation using only one slice per FU. This becomes even more interesting when considering the increased LUT size of newer FPGA devices, making it possible to have a higher number of input elements as well as a higher precision while still using only one slice per FU.

Although using a full MUX implementation instead of the time multiplexing scheme was not considered in this paper due to the large number of data elements in the input vectors, this could be an interesting option for smaller input vectors. Also in this case it should be possible to save resources by configuring the LUTs directly with the MUX select value.

While the 16 cycles reconfiguration time of the SRL approach is longer than in the case of the VRC approach, it is still an acceptably short time given that the rest of the classifier module can be operational while one row is being reconfigured. This is also compatible with the incremental evolution principle applied by the system in which one FU row is evolved at a time, giving a gradual replacement of the complete system. For the evaluation module, which contains only the row under evaluation by the EA, one can consider employing the VRC technique instead, giving fast reconfiguration for the fitness evaluation. This could be useful if the reconfiguration time is critical for the total fitness evaluation time, although in many cases this would be dominated by time spent processing the training data. Also note that in practice there may be increases in reconfiguration time for both approaches, as the VRC configuration word size of 90 for one row may be too large for the configuration bus and that the configuration words for the SRL approach should be calculated from the original genome of 90 bits.

The results from the precision experiment show that although a higher number of bits give a higher level of classification accuracy, the biggest performance drops become visible when the precision is reduced to as little as 3 or even 2 bits, depending on the application. This shows that it is very possible to consider for instance 4-bit precision configurations of the classifiers, saving resources while still maintaining high classification accuracy.

It is interesting to notice from Table 2 that by adopting the SRL-based implementation and reducing the data precision it is possible to make the classifier fit into significantly smaller devices than before, opening up for cost savings or better possibilities for integration with other systems. In addition it is possible to implement larger configurations of the systems which would not earlier fit into a single chip, resulting in better classification accuracies than what was possible before.

The implementations have targeted a Virtex-II Pro device, which is advantageous to an SRL-based approach because all of the device's LUTs can be used as SRLs. Newer devices from Xilinx do not offer 100% of the LUTs available as SRLs, such as the Spartan3 and Virtex4 devices which have 50% SLICEMs and the Virtex5 around 25% SLICEMs. This may give less efficient implementations, however it could be that other parts of the system could make use of the other slices. A way to overcome the reduced SRL ratio would be to reconfigure the LUTs in another fashion, through the internal ICAP reconfiguration port. This would make it possible to achieve reconfiguration of all LUTs in the classifier module, without having to use the SRL interface. However, this approach is more low-level and one has to keep track of the precise FPGA location of every LUT. Thus, although giving potentially higher resource utilization, this low-level reconfiguration method would require a higher implementation effort.

One could sum up the different levels of internal reconfiguration with VRC being the high-level approach, SRL being an intermediate and ICAP the low-level approach. VRC can be specified in a high-level, device-independent way, giving the possibility for fast reconfiguration but also being the least efficient in terms of FPGA resource utilization. The SRL requires some explicit primitive instantiation while the layout on FPGA can be given less attention. This approach gives a much better resource utilization, given that

the FPGA LUTs can be used as SRLs. The ICAP is the most device-specific approach and would be required in order to maximize resource utilization on newer devices. Although a high level of attention to the FPGA details would be required, this approach also offers the potential for a very efficient circuit implementation through the use of 6-input LUTs in the newer devices.

5 Conclusions and future work

An intermediate-level FPGA partial reconfiguration approach was applied to the implementation of the FUR pattern recognition EHW architecture, resulting in significantly reducing the resource requirements. This economization makes it possible to deploy the FUR architecture to smaller FPGA devices and thus save costs. Alternatively, a FUR configuration with more rows can fit into the same space as before, increasing recognition accuracy. This was demonstrated by the FPGA implementation of a face image classifier with a higher accuracy than previously achieved. In addition it was discovered that the precision of the architecture can be reduced from 8 to 4 bits without a large loss in classification accuracy, thus making it possible to further reduce the FPGA resource usage. Future work includes experimenting with an even lower level of partial reconfiguration by means of the ICAP port in order to make full use of the resources in newer FPGA devices. This would be important in the process of optimizing the architecture for low-cost devices such as the Xilinx Spartan range.

Acknowledgment

This work was supported by the Research Council of Norway through the project *Biological-Inspired Design of Systems for Complex Real-World Applications* under project number 160308/V30.

References

[1] 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)*, volume 5216 of *Lecture Notes in Computer Science*, pages 22–33. Springer, 2008.

[2] 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)*, volume 4448 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2007.

[3] 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)*, volume 4684 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.

[4] K. Glette, J. Torresen, and M. Yasunaga. Online Evolution for a High-Speed Image Recognition System Implemented On a Virtex-II Pro FPGA. In *Proceedings 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 463–470, Los Alamitos, CA, USA, 2007. IEEE CS Press.

[5] 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.

[6] 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*, volume 1062 of *LNCS*, pages 118–135. Springer, April 1996.

[7] 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. In *Proceedings 2nd International Conference on Evolvable Systems (ICES)*, volume 1478 of *LNCS*, pages 1–12. Springer, 1998.

[8] 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)*, pages 169–176. IEEE CS Press, 2008.

[9] 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.

[10] 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)*, pages 161–168, 2000.

[11] J. Torresen. Two-Step Incremental Evolution of a Digital Logic Gate Based Prosthetic Hand Controller. In *Proceedings 4th International Conference on Evolvable Systems (ICES)*, volume 2210 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2001.

[12] 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*, volume 3038 of *Lecture Notes in Computer Science*, pages 1249–1256, May 2004.

[13] A. Upegui. *Dynamically Reconfigurable Bio-inspired Hardware*. PhD thesis, Ecole Polytechnique Fdrale de Lausanne (EPFL), 2006. Thesis No. 3632.

[14] M. Yasunaga, T. Nakamura, and I. Yoshihara. Evolvable Sonar Spectrum Discrimination Chip Designed by Genetic Algorithm. In *Systems, Man and Cybernetics*, volume 5, pages 585–590. IEEE, 1999.

[15] M. Yasunaga, T. Takami, and I. Yoshihara. Image Recognition Hardware Using FPGAs for Nano-Second Range Recognition Speed. *IEICE Transactions on Information and Systems*, 84(10):2280–2292, 2001.