On-Chip Evolution Using a Soft Processor Core Applied to Image Recognition

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

Moritoshi Yasunaga and Yoshiki Yamaguchi
University of Tsukuba
Graduate School of
Systems and Information Engineering
1-1-1 Ten-ou-dai, Tsukuba, Ibaraki, Japan
{yasunaga,yoshiki}@cs.tsukuba.ac.jp

Abstract

To increase the flexibility of single-chip evolvable hardware systems, we explore possibilities of systems with the evolutionary algorithm implemented in software on an on-chip processor. This gives higher flexibility compared to implementing an evolutionary algorithm directly in hardware, since the parameters and behaviour of the algorithm can easily be changed, and complex operators are more feasible to implement. In this paper a Xilinx MicroBlaze soft core processor is used, and the system is implemented in a Xilinx FPGA. A suitable hardware architecture for image recognition has been proposed, and it is applied to a face recognition task. Data buses and higher level functions have been utilized in order to reduce the search space for the evolutionary algorithm. Experiments have been performed on the physical device, with software running in parallel with fitness computation in digital logic. Results show that the MicroBlaze system evolves at half the speed of a Pentium M system running at 17 times the FPGA clock frequency. The distinction of a certain face from others is performed at 94.9% accuracy. In addition, the possibilities for evolutionary adaptation over time are explored by introducing changes in the training set. The system shows ability to adapt to these changes.

1 Introduction

Evolvable hardware (EHW) seems useful for systems submitted to unpredictable, time-varying environments [20, 14]. Such systems will also often be part of embedded applications, and therefore compact, on-chip solutions will be preferred.

There have been undertaken some implementations of single-chip evolution earlier. Kajitani et al have introduced several LSI (Large-Scale Integrated Circuits) devices with evolution performed in hardware [5, 6]. The benefit of such an approach is the evolution speed but the problem is lack of flexibility. This would be important since there are often many degrees of freedom when evolving hardware systems.

Implementing complete evolution in an FPGA has been proposed by Tufte and Haddow in [16]. The evolving design is here implemented in the same device as the evolutionary algorithm. A similar approach is proposed by Perkins et al in [9]. Significant speedup is achieved for non-linear filtering compared to conventional processing. Several custom accelerators in FPGA for solving a protein folding problem have been introduced by Shackleford et al [12].

On-chip evolution using a prototype of the VLSI (Very Large-Scale Integration) POEtic chip has also been reported [10]. A robot controller and logic functions were evolved. The architecture, specialized for the implementation of bio-inspired mechanisms, contains an on-chip custom processor, and a bio-inspired array of building blocks.

Running complete evolution of image filters within an FGA has been reported by Martinek and Sekanina [7]. In this work the evolution (mutation only) is implemented in reconfigurable logic. Image filters were evolved in a few seconds from corrupted and original pictures. This design employs data buses, earlier proposed in [11] and [13], and function-level building blocks, first proposed in [8].

The authors have earlier demonstrated System-On-Chip evolution using an embedded hard processor core in an FPGA [1]. This is accomplished by integrating an evolutionary algorithm running as software on a hard processor IP core with the target EHW implemented in reconfigurable logic. In this paper an alternative approach is explored by using a soft processor core. This allows for portability to a greater range of FPGAs, including cheaper devices.

In our system, all parts of the evolution (except the evaluation of individuals which is implemented in digital logic) are undertaken in software, providing a flexible system for later modifications. This is slower than implementing the evolution in dedicated hardware, but it is expected that the fitness evaluation time will still be the most time consuming part. This balanced software-hardware approach will allow for a low implementation effort while still being able to have a single-chip design, suitable for embedded real-world applications.

The EHW system is applied to a face image recognition task. Experiments on image recognition by EHW were first reported by Iwata et al in [4]. An FPLA device was utlized for recognition of black and white images. An EHW face image classifier system has been developed by Yasunaga et al in [21], in which the classifier function is directly coded in large AND gates. Evolution is performed offline and the final system is synthesized. This approach gives rapid classification in a compact circuit, but lacks the run-time reconfigurability. Another classifying system has been proposed in [15], also employing AND gates, in combination with OR gates. These systems have in common the selection of one category from a set of candidate categories.

The image recognition task proposed in this paper will be restricted to recognizing one face out of a set of candidates. That is, the system reports an input as either a positive match to the trained face category, or a negative match. Such an application could be imagined for an image-based lock for mobile devices, or cameras trained to recognize one particular person.

The next section introduces the architecture and the implementation of the on-chip EHW system, including both hardware and software aspects. Results from the implementation are given and discussed in Section 3. Finally, Section 4 concludes the paper.

2 The On-Chip Evolution System

This section details the hardware and software architecture of the evolvable hardware system. The evolvable hardware system is entirely implemented on one FPGA chip. The core modules are the processor, the RAM and the target EHW module. The evolutionary algorithm, stored in RAM, runs on the embedded processor, and the target EHW module is used for fitness evaluation. This module is built as a reconfigurable hardware module which accepts a bit string as configuration.

2.1 System Overview

The architecture consists of a set of modules interconnected with buses, as seen in Fig. 1. The MicroBlaze soft processor core provided by Xilinx is central in the system. It employs a 32-bit, 3-pipeline stage RISC architecture and is optimized for implementation in the Xilinx FPGAs. It is user configurable, allowing for use of various interfaces and functionality according to system needs and constraints. Amongst others, hardware division and multiplication, and a floating point unit (FPU), are supported [18].

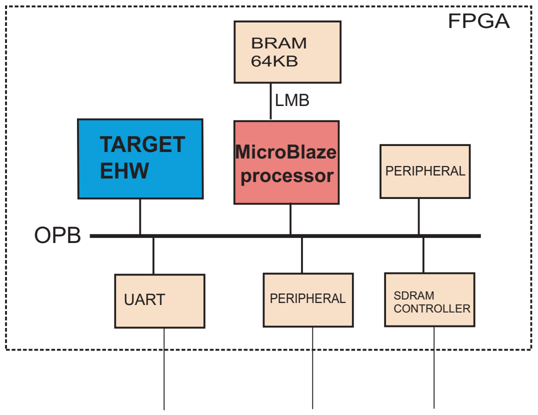

Figure 1. Example hardware architecture including our target EHW.

The bus system is a part of IBM's CoreConnect architecture [19]. The On-chip Peripheral Bus (OPB), with a data width of 32 bits, is used to connect the processor and the peripherals. The Processor is connected to dual-port SRAM, called Block RAM (BRAM), using a dedicated Local Memory Bus (LMB). This bus features separate 32-bit wide channels for program instructions and program data, using the dual-port feature of the BRAM. The LMB makes single-cycle access of BRAM possible.

The target evolvable hardware is at the moment connected to the OPB. This will be detailed in section 2.2. Various on-chip peripherals are also connected to the OPB, including a UART for RS-232 serial communications. An SDRAM controller can also be connected to the OPB should more memory be needed.

The on-chip system is built using the Xilinx Embedded Development Kit (EDK) [17]. EDK is a collection of Intellectual Property (IP) cores and tools for building embedded systems on FPGAs. The hardware and software parts of the system can be specified parametrically through various configuration files, and net lists and libraries are automatically generated.

2.2 Target Evolvable Hardware

The target EHW is implemented as an OPB slave peripheral module – see Fig. 2. Interfacing with the OPB bus has been simplified by the use of a Xilinx IP Interface core (IPIF). This provides a simpler interface standard, the Xilinx IPIC, for the user module.

Control and configuration of this module are undertaken through register write operations. Genome values are written to registers which are again connected to the configuration inputs of the functional unit array. Registers are also

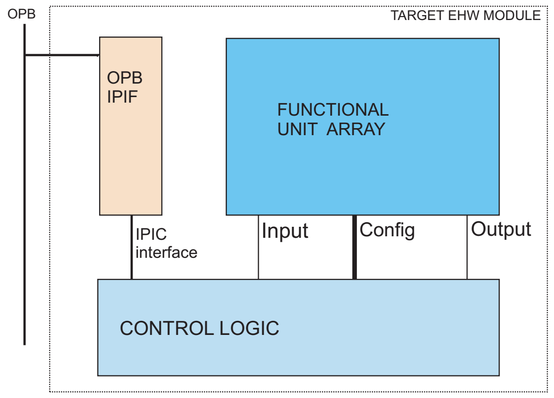

Figure 2. The architecture of the target EHW system.

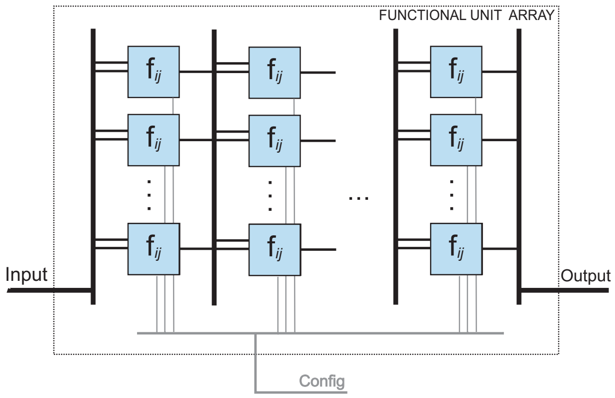

Figure 3. The architecture of the functional unit array subsystem.

provided for feeding the EHW with inputs and for storing the outputs.

2.2.1 Functional unit array

The functional unit array (FUA), see Fig. 3, is a general structure used for EHW. It is based on the principle that the configuration of the FPGA itself is not changed, but a virtual circuit which is implemented on top of it can be reconfigured. Hence the names ”Virtual FPGA” [3] or ”Virtual Reconfigurable Circuit” [11] have been proposed for circuits building on the same principles. The behavior of the FUA is achieved by writing configuration data to registers which in turn control the functionality of each unit, $f_{i,j}$ in the array. A fixed set of functions and connections to other units' outputs are available in each functional unit. The configuration lines control multiplexers which select which inputs and functions to use. The system bears resemblance to the VRC in [11].

Our FUA consists of a fixed-size array of functional units. The array consists of C columns of R units from input to output. Each unit has I inputs, each of which can be connected to any output in the previous column. The unit's output is a result of any of F functions. The function of each unit and its inputs are configurable. They are determined by evolution, in the way that each individual's binary genome is sent to the FUA and mapped to the configuration lines. Fitness is then calculated by feeding a number of input vectors on the inputs of the first column, and reading the results from the outputs of the last column. The array is constructed in a pipelined fashion, that is, registers are connected to the outputs of each layer. Currently, this is not exploited for fitness evaluation. Only one input vector is evaluated at a time.

#	Description	Function
0	Saturated Add	$O = A + B$, FF if $(A + B) > FF$
1	High Threshold	$O = FF$ if $A > C_2$, else 0
2	Range	$O = FF$ if $C_1 < A < C_2$, else 0
3	Greater	$O = FF$ if $A > B$, else 0
4	Bitwise AND	$O = A$ AND B
5	Bitwise OR	$O = A$ OR B
6	Average	$O = (A + B) >> 1$
7	Half	$O = A >> 1$

Table 1. Functions used by units in the image recognition task. Inputs are A and B, ouput is O. C_1 and C_2 are constants available to each unit.

2.2.2 Image recognition application

The FUA has been applied to an image recognition application. For this, 8-bit wide databuses are used as inputs and outputs of the units, which is also the data width of one pixel from the input image. The array has $R = 8$ rows and $C = 5$ columns. There are $I = 2$ inputs and there are $F = 8$ functions available for each unit. The functions are summarized in table 1. The specific functions are chosen because they are believed to be useful for an image recognition task. The threshold and range functions give a possibility to discriminate pixels based on their intensity value. Combined with the saturated adder, threshold element-like functionality, as seen in artificial neural networks, can be achieved. In addition, two global constants, C_1 and C_2, are available to each unit. These are coded in the genome for each individual as 8-bit values.

The input images have a resolution of 8x8 pixels, 64 in total, while each column in the FUA consists of 8 units,

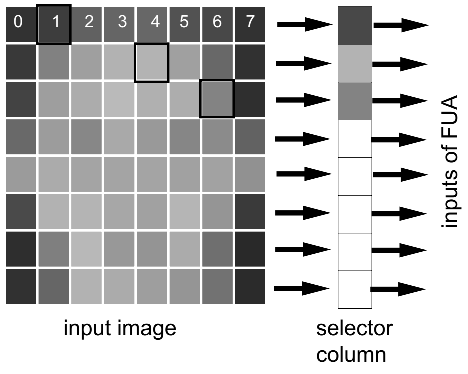

Figure 4. The genome selects one pixel from each row of the 64-pixel picture, giving 8 pixels for the first column of the FUA.

capable of selecting from 8 inputs. To save genome size and circuit space, as well as simplifying the design, a ”selector column” is introduced. Basically this imposes a restriction of only letting the unit in one row select a pixel input from the corresponding row of 8 pixels in the image. Thus, only 3 bits are needed code for a pixel from each row, which gives a total of 24 bits of the genome for the entire selector column. The selected pixels are then passed on to the inputs of the regular FUA. See Fig. 4.

This functionality could have been implemented in hardware by having the first column of the FUA be populated with special selector units, or hard-coded ”Add $A, 0$” units (using already defined functionality in the standard units), each of them connected to a corresponding row of image pixels. However, in the current implementation, this selection is done in software on the MicroBlaze, as this lets us transfer only 8 instead of 64 pixels over the data bus for each vector. Depending on the source of the data vectors this can be moved to hardware in future versions.

The last column of the array gives 8 8-bit outputs. However, only the 8-bit output of the topmost functional unit is used for the classification of the image.

The encoding of each functional unit in the genome string is as follows:

Function (3 bit)	Input 1 (3 bit)	Input 2 (3 bit)

This gives a total of $U = \log_2 F + I \times \log_2 R = 9$ bits for each unit. The entire genome is encoded as follows:

C_1,C_2(16b)	Sel. col.(24b)	$f_{0,0}$(9b)	...	$f_{4,7}$(9b)

The total amount of bits in the genome is then $2 \times 8 + R \times \log_2 R + C \times R \times U = 400$.

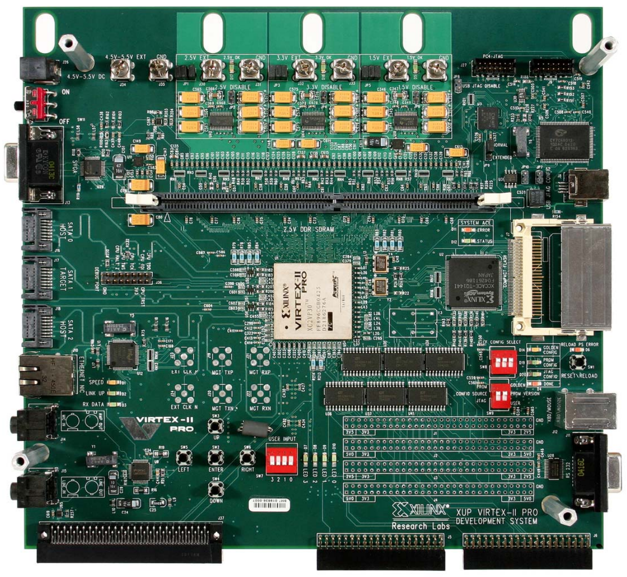

Figure 5. The prototype board with the Virtex-II Pro FPGA. Source: Xilinx

2.3 The Hardware Platform

The design is synthesized for a Xilinx Virtex-II Pro (XC2VP30) – see Fig. 5. This device contains 30,816 logic cells, 2,448 Kbit BRAM, and two PowerPC 405 (PPC) embedded processors. The FPGA is situated on a Xilinx XUP Virtex-II Pro development board, which also contains a configuration PROM and various useful interfaces.

2.4 The Genetic Algorithm Implementation on the MicroBlaze

A Genetic Algorithm (GA) was implemented to run on the MicroBlaze processor. 64KB of BRAM was used as a combined instruction and data memory. The program was written in C and compiled and linked using the MicroBlaze version of the GNU GCC compiler tools. However, the limited amount of RAM makes it necessary to omit the use of most standard C library functions. The code was developed with verification and simulation on a PC workstation in mind. It is therefore possible to compile the program both for the MicroBlaze using GCC and for a PC workstation using Microsoft Visual C, with code differences only for low-level functionality.

As the MicroBlaze can be configured with a FPU, some of the code uses floating point. However, if FPGA resource usage is critical, the FPU should be removed. This implies software emulation of floating point, which should be avoided in order to reduce code size and execution speed. Conversion to fixed point arithmetic could then be considered. Dynamic memory management is not supported for the MicroBlaze. Allocation of memory for data structures such as population and individuals is handled manually.

The GA implemented for this experiment follows the Simple GA style, given by Goldberg [2]. Fitness scaling has been implemented, including linear scaling. A fitness-proportionate selection scheme is implemented through the use of a roulette wheel mechanism. The individuals are sorted with the qsort algorithm. For mutation, instead of having one probability of mutation for every bit in the genome, a quicker solution has been adopted. The number of mutations, n, for the whole genome is calculated by a random lookup in a 10-position array. Then, n random places are mutated (bit-flipped) in the genome. This is more efficient than performing a check for every bit if a mutation should occur or not.

2.5 Fitness Function and GA Parameters

The fitness of the face recognition system is based on the system's ability to recognize one face from a range of different faces. The images are taken from the "Olivetti Research Laboratory Database of Faces"[1]. The original resolutions of the images were 92x112 and there were 400 faces divided over 40 people, giving 10 face images per person. In our experiment the images were downsampled to 8x8 pixels and 10 categories were used, giving a total of 100 64-pixel vectors. These are stored in BRAM.

90 of the vectors were used for training of the system, while the remaining 10 were used for verification after the evolution run. Each configuration of the hardware is fed with the 90 training vectors (vec), and fitness is based on the system's ability to give a positive output for the 9 image vectors belonging to the right person, and a negative output for the rest. Output values from the system are compared to target values d, which are positive ($d = 1$) for vectors belonging to the right person, and negative for all other vectors ($d = 0$). An output value of FF(hexadecimal) from the topmost functional unit in the last column is considered a positive output ($o = 1$), while anything else is considered a negative output ($o = 0$).

The fitness function can be expressed in the following way:

$$F = \sum_{vec} x \qquad \text{where } x = \begin{cases} 0 & \text{if } o \neq d \\ 1 & \text{if } o = d = 0 \\ 4 & \text{if } o = d = 1 \end{cases} \tag{1}$$

For each input image vector the computed output o is compared to the target value d. If these equal and the value is

[1]http://www.cl.cam.ac.uk/Research/DTG/attarchive/facedatabase.html

Resource	Used	Available	Percent
Slices	2990	13696	21
Slice Flip Flops	1037	27392	3
4 input LUTs	5488	27392	20

Table 2. Post-synthesis device utilization for the EHW module implemented on the XC2VP30.

negative (0) then 1 is added to the fitness function. On the other hand, if they equal and the value is equal to one, 4 is added. In this way, an emphasize is given to the outputs being positive. This has shown to be important for getting faster evolution of well performing circuits. The function sums these values for the training image vectors (vec).

For the evolution, a population size of 30 is used. Elitism is used, thus, the best individuals from each generation are carried over to the next generation. The (single point) crossover rate is 0.9, thus the cloning rate is 0.1. A roulette wheel selection scheme is applied, and linear scaling is used. The mutation rate is expressed as a probability for a certain number, n, of mutations on each genome. The probabilities are as follows:

n	0	1	2	3
$p(n)$	0	$\frac{7}{10}$	$\frac{2}{10}$	$\frac{1}{10}$

3 Results

This section presents and discusses the results of our implementation and experiments.

3.1 Device utilization and clock speed

The total resource usage for the system, including the target EHW, the MicroBlaze, bus structure and peripherals, is 5113 slices, equalling 38% of the XC2VP30. The system was implemented to run at 100MHz. It is possible that higher frequencies are attainable, up to a limit of around 150MHz. The MicroBlaze has a maximum frequency of 150MHz on the Virtex-II Pro, and the target EHW has a post-synthesis maximum estimate of 148MHz. Table 2 shows the amount of logic used for the target EHW module. A maximum of 21% of the FPGA's total resources are used by this module. The post-synthesis resource usage of the MicroBlaze processor is 1731 slices, of which 33% is used for the FPU.

3.2 Evolution speed

The speed of an evolution run of 1000 generations was measured for the on-chip EHW system and a Pentium M PC

	EHW	PC	$\frac{PC}{EHW}$
Clock speed(MHz)	100	1700	17
Evolution speed(ms)	10914	4993	0.46

Table 3. Evolution speeds on the on-chip EHW and PC systems. The last column indicates the ratio between the first and second columns.

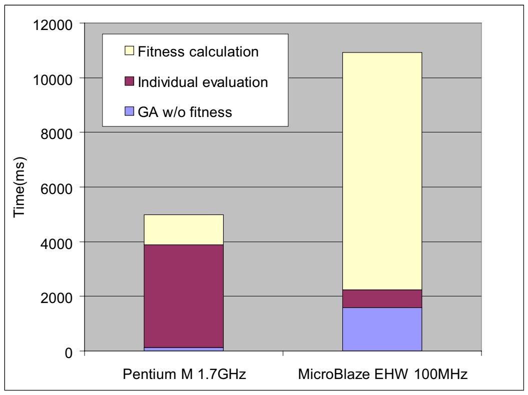

Figure 6. Evolution speeds.

for speed comparison. As can be seen in Table 3, the evolution runs faster on the PC. However, although the evolution runs 2.2 times faster, the processor runs at 17 times the clock frequency of the MicroBlaze system. This is mainly due to the fact that the evaluation of individuals is carried out in hardware on the on-chip EHW system whereas it is simulated in the PC system. In order to better analyze the speed of different parts of the evolution process, three different measures were made. The first measure indicates the time used by the GA without any form of fitness evaluation, ie. all the fitness values were set directly to 0 in the program. The second measure indicates the time spent on *evaluation* of the individuals. That is, the time used for calculating outputs from inputs to the FUA. The third measure is the time used for fitness calculation. This consists of the calculation of a fitness value based on the training vectors and outputs of the FUA, as well as pixel selection and transfer to the FUA. The results are summarized in Fig. 6. It is clear that the sum of the operations related to fitness are the most time-consuming parts for both of the systems. The bottleneck in the PC program is clearly the evaluation of individuals, which is simulated in software. The execution time for this part is greatly reduced in the on-chip system. If the target EHW module increases in size or complexity, this difference will become even more significant. The hardware system's bottleneck is the overhead associated to the fitness determination of each individual. This is because pixel selection and transfer over the system bus, as well as the readback of the FUA output and comparison with the target value, is done for each training vector. Thus, the low program execution speed and bus transfer rate contribute to making the fitness overhead much slower than the hardware fitness evaluation.

3.3 Image Recognition Performance

Individuals with maximal fitness values were evolved after an average of 1133 generations over 20 evolution runs. A solution with maximal fitness value was found in every run. The average evolution time is then 12.4 seconds. For verification, the last vector in each category was used. This means that there were 10 vectors to test the accuracy, where the evolved system would need to produce a '1' for one of these, and '0' for the others. Due to the lack of more verification vectors, the position of the vector within the category was changed, and the remaining vectors were used for training vectors for a new evolution run. This was repeated 10 times, until all of the vectors in a category had served as a verification vector. The accuracy over all the outputs from all the evolution runs is 94.9% correct outputs. Also, 7 of these 10 evolution runs produced individuals which gave correct outputs for all vectors.

3.4 Real-time adaptation

To explore possibilities for using the system with applications where the environment is changing over time, an adaptation experiment was carried out. In a real-world application one would imagine the training set changes, by introducing new face images of either the target face category or the other category. It could also be imagined that some training vectors would have to be removed, based on timestamps for example.

We simulated such a change in the training set by removing one vector and adding another vector from the target face category every 500 generations. However, as there were only 9 images available for training in each category, 8 images were used for the training set and 1 image cycled as the not used image. To introduce some more change, all the pixels in the added vector were multiplied by four.

The experiment was run for 5000 generations. The results can be seen in Fig. 7. By observing the fitness value of the best individual of each population, one can get an impression of the system's adaptability. Some of the training set changes are clearly observable by a drop in fitness, while other changes do not affect the fitness value. The recover time from the fitness value drops seems to vary from around 50 to 500 generations, which is equivalent to 0.5 to 5.5 seconds of run time on the MicroBlaze system.

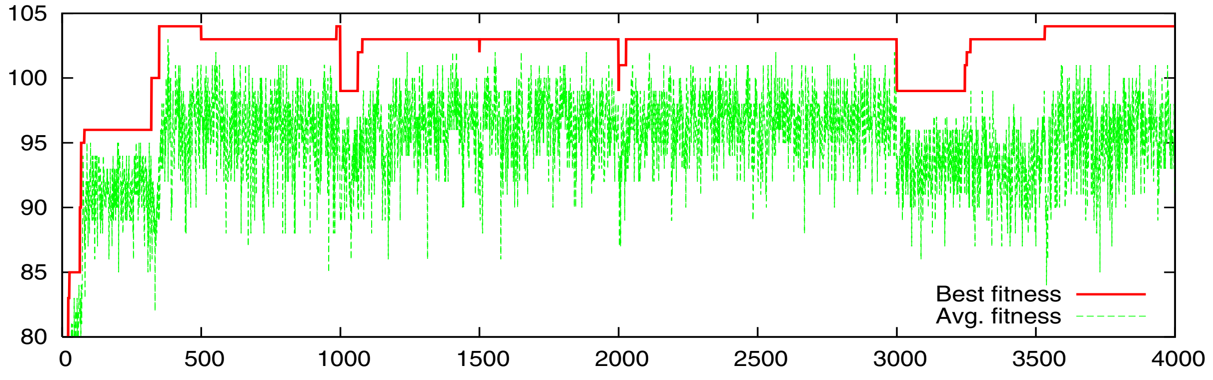

Figure 7. Population fitness over generations, showing the fitness of the best indiviudal and the population average fitness. Temporary drops in fitness can be observed when the training set is modified. Notice that the y-axis only shows fitness values from 80 to 105. 104 is the maximum possible fitness value.

3.5 Discussion

The FPGA resource usage of the system is not a problem with the FPGA used in our experiments. But for low-cost applications, it would be desirable to use a smaller, less expensive FPGA device. As [11] points out, the disadvantage of a virtual circuit approach is the high implementation cost. Much of this cost comes from interconnect between the units and multiplexers for selecting the inputs to each unit. Other connection schemes should be explored. At the expense of a slight speed loss, a possibility could be to use a kind of addressing or broadcasting scheme for sending the data to the next layer of units. The MicroBlaze processor can be configured to take up less resources by removing some functionality. A removal of the FPU would give the most significant decrease in resource usage. This does not necessarily have to imply a performance penalty if the program code is written to avoid the use of floating point instructions.

Using 12.4 seconds on average to evolve a system with maximum fitness, the speed of the on-chip system is currently acceptable. It should also be noted that systems with relatively high fitness values are available earlier in the evolution runs, which are usable until a better solution has been found. However, as future tasks may be more complex to evolve, a speed increase would be desirable. As pointed out in section 3.2, there is a large overhead associated with fitness calculation in software. If feeding of the training vectors to the FUA and fitness value calculation would be moved to hardware, a significant speed increase would be possible. The FUA would achieve a much higher throughput, and the pipelined nature of the design could be exploited. In this case the total execution time for the on-chip evolution would tend towards the execution time used by only the GA. In such a case the on-chip system could perform at twice the speed, or better, than the PC system. However, an increased degree of hardware specialization comes at the prize of a higher implementation effort and increased resource usage. In general, performance needs for the application should be considered up against implementation costs. A speed performance comparison with the system described in [1] should be the subject of future work.

A face recognition performance of 94.9% is acceptable, but the accuracy measure might not reflect real-world performance. Since only one category of faces is to be recognized, more verification vectors for this category are desirable. This would give a clearer picture of the accuracy of the system. In future work, it could also be explored how this system would perform when expanded to a multi-category classification system, like the ones seen in [21] and [15]. In that case, the current FUA would be duplicated to the amount of categories desired, and the amount of high output values from the last column in each FUA could be used as input to a max detector for determination of a category.

Real-time adaptation to a changing environment is a goal for our on-chip evolutionary systems. The initial experiments of introducing changes to the training set seem promising, as the system seems to regain a high fitness in short time. But in order to explore adaptation further, a larger training set is desirable for the face recognition application. In a real-world application, one target EHW module configured with the best individual could be used as an operational circuit, while a second target EHW module is used for fitness evaluation for the evolving population. One example of an adapting application could be a camera with an integrated chip, assigned to detect a certain person. Over time the system could receive new images of the person, or images of other persons which should not be detected, to be added to the training set.

4 Conclusions

We have presented a system-on-chip EHW system using a soft IP core processor for running the evolutionary algorithm. This has shown reasonable performance combined with flexibility for experimentation. The configurability of the MicroBlaze core makes such a combination interesting for applications in low-cost systems. The EHW architecture proposed utilizes data buses and higher level functions in order to reduce the search space. The inputs of the FUA and the functions available to the units are adapted to the image recognition task. These measures make it possible to use evolution for a relatively complex task. The experimental results indicate that the system is suitable for further experiments and development of real-time on-chip adaptation.

Acknowledgments

The research is funded by the Research Council of Norway through the project *Biological-Inspired Design of Systems for Complex Real-World Applications* (project no 160308/V30).

References

[1] K. Glette and J. Torresen. A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device. In *Evolvable Systems: From Biology to Hardware. Sixth International Conference, ICES 2005*, volume 3637 of *Lecture Notes in Computer Science*, pages 66–75. Springer-Verlag, 2005.

[2] D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison–Wesley, 1989.

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

[4] M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi. A pattern recognition system using evolvable hardware. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 761–770. Springer-Verlag, September 1996.

[5] I. Kajitani et al. A myoelectric controlled prosthetic hand with an evolvable hardware lsi chip. *Technology and Disability*, 15(2):129–143, 2003.

[6] I. Kajitani, T. Hoshino, N. Kajihara, M. Iwata, and T. Higuchi. An evolvable hardware chip and its application as a multi-function prosthetic hand controller. In *Proc. of 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 182–187, 1999.

[7] T. Martinek and L. Sekanina. An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. *Lecture Notes in Computer Science*, 2005(3637):76–85, 2005.

[8] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. Hardware evolution at function level. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 62–71. Springer-Verlag, September 1996.

[9] S. Perkins, P. Porter, and N. Harvey. Self-contained spatially-structured genetic algorithm for signal processing. In J. Miller et al., editors, *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pages 165–174. Springer-Verlag, 2000.

[10] D. Roggen, Y. Thoma, and E. Sanchez. An evolving and developing cellular electronic circuit. In J. Pollack, M. Bedau, P. Husbands, T. Ikegami, and R. A. Watson, editors, *ALife9: Proceedings of the Ninth International Conference on Artificial Life*, pages 33–38, Boston, MA, 2004. MIT Press.

[11] L. Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. *Lecture Notes in Computer Science*, 2003(2606):186–197, 2003.

[12] B. Shackleford et al. A high-performance, pipelined, FPGA-based genetic algorithm machine. *Journal of Genetic Programming and Evolvable Machines*, 2(1):33–60, 2001.

[13] J. Torresen. Exploring knowledge schemes for efficient evolution of hardware. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*.

[14] J. Torresen. Possibilities and limitations of applying evolvable hardware to real-world application. In R. Hartenstein et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, volume 1896 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 2000.

[15] J. Torresen. Two-step incremental evolution of a digital logic gate based prosthetic hand controller. In *Evolvable Systems: From Biology to Hardware. Fourth International Conference, (ICES'01)*, volume 2210 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2001.

[16] G. Tufte and P. C. Haddow. Prototyping a ga pipeline for complete hardware evolution. In *1st NASA / DoD Workshop on Evolvable Hardware (EH '99)*, pages 18–25, 1999.

[17] Xilinx Inc. *Embedded System Tools Reference Manual*, February 2005.

[18] Xilinx Inc. *MicroBlaze Processor Reference Guide*, May 2005.

[19] Xilinx Inc. *Processor IP Reference Guide*, February 2005.

[20] X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, pages 55–78. Springer-Verlag, 1997.

[21] M. Yasunaga, T. Nakamura, I. Yoshihara, and J. Kim. Genetic algorithm-based design methodology for pattern recognition hardware. In J. Miller et al., editors, *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pages 264–273. Springer-Verlag, 2000.