Online Evolution for a High-Speed Image Recognition System Implemented On a Virtex-II Pro FPGA

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
University of Tsukuba
Graduate School of Systems
and Information Engineering
1-1-1 Ten-ou-dai, Tsukuba, Ibaraki, Japan
yasunaga@cs.tsukuba.ac.jp

Abstract

Online incremental evolution for a complex high-speed pattern recognition architecture has been implemented on a Xilinx Virtex-II Pro FPGA. The fitness evaluation module is entirely hardware-based in order to increase the speed of the circuit evaluation which uses a large training set (360 images/23040 bytes). The fitness evaluation time for 1000 generations consisting of 16 individuals is 623ms, twice as fast as software fitness evaluation performed on a workstation running at a 30 times higher clock frequency. The rest of the genetic algorithm (GA) runs in software on a PowerPC 405 processor core on the FPGA. The total evolution time for 1000 generations is 1313ms, equivalent to the total time used by the workstation. Resource utilization for the fitness evaluation module is 1393 slices (10%) of a XC2VP30 device.

1 Introduction

Hardware implementation could be important for image recognition systems requiring a low recognition latency or high throughput. Furthermore, if the systems are applied in time-varying environments, and thus need adaptability, online evolvable hardware (EHW) would seem to be a promising approach [10].

One approach to online reconfigurability is the virtual reconfigurable circuit (VRC) method proposed by Sekanina in [7]. This method does not change the bitstream to the FPGA itself, rather it changes the register values of a circuit already implemented on the FPGA, and obtains virtual reconfigurability. This approach has a speed advantage over reconfiguring the FPGA itself, and it is also more feasible because of proprietary formats preventing direct FPGA bitstream manipulation. However, the method requires much logic resources.

Experiments on image recognition by EHW were first reported by Iwata et al in [5]. A field programmable logic array (FPLA) device was utilized for recognition of three different patterns from black and white input images of 8x8 pixels. An EHW road image recognition system has been proposed in [8]. A gate array structure was used for categorizing black and white input images with a resolution of 8x4 pixels. Incremental evolution was applied in order to increase the evolvability.

A speed limit sign recognition system has been proposed in [9]. The architecture employed a column of AND gates followed by a column of OR gates, and then a selector unit. A maximum detector then made it possible to decide a speed limit from 6 categories. Incremental evolution was applied in two ways: each subsystem was first evolved separately, and then in a second step the subsystems were assembled and the selector units were evolved. The input images were black and white and had a resolution of 7x5 pixels.

An EHW face image classifier system, logic design using evolved truth tables (LoDETT), has been presented by Yasunaga et al. [13]. This system is capable of classifying large input vectors into several categories. For a face image recognition task, the input images had a resolution of 8x8 pixels of 8-bit gray scale values, belonging to 40 different categories. In this architecture, the classifier function is directly coded in large AND gates. The classification is based on detecting the category with the highest number of activated AND gates. Incremental evolution is utilized for this system too, where each module for detecting a category is evolved separately. The average recognition accuracy is 94.7% However, evolution is performed *offline* and the final system is synthesized. This approach gives rapid classification in a compact circuit, but lacks run-time reconfigurability.

A system we have developed earlier [3] addresses the reconfigurability by employing a VRC-like array of high-level functions. Online/on-chip evolution is attained, and therefore the system seems suited to applications with changes in the training set. However, the system is limited to recognizing one category out of ten possible input categories. The system uses the same image database as [13] with the same input resolution.

The authors then proposed a novel system in [2] which expands to categorization of all 40 categories from the image database used in [13], while maintaining the on-line evolution features from [3]. A change in the architecture was undertaken to accommodate for the recognition of multiple categories. In LoDETT a large number of inputs to the AND gates could be optimized away offline, during circuit synthesis, and thus save space on the FPGA implementation of the final circuit. The online, run-time reconfiguration aspect of the described architecture led to a different approach, employing fewer, but more flexible, elements. A higher recognition accuracy (96.25%) than the previously described systems was also obtained.

In this paper the architecture proposed and simulated in [2] has been subject for hardware implementation. In particular, the evolutionary part of the system, consisting of a processor core running a GA and a fitness evaluation hardware module, has been implemented to run as a system-on-a-chip (SoC) on an FPGA.

A large amount of literature exists on conventional face image recognition. A comprehensive survey can be found in [14]. Work on conventional hardware face recognition has been undertaken, based on the modular PCA method [6]. However, the recognition speed (11 ms) is still inferior to the LoDETT system.

The next section introduces the architecture of the evolvable hardware system. Then, the image recognition-specific implementation is presented in section 3. Aspects of evolution are discussed in section 4. Results from the experiments are given and discussed in section 5. Finally, section 6 concludes the paper.

2 The Online EHW Architecture

The EHW architecture is implemented as a circuit whose behaviour and connections can be controlled through configuration registers. By writing the genome bitstream from the genetic algorithm to these registers, one obtains the phenotype circuit which can then be evaluated. This approach is related to the VRC technique, as well as to the architectures in our previous works [1, 3].

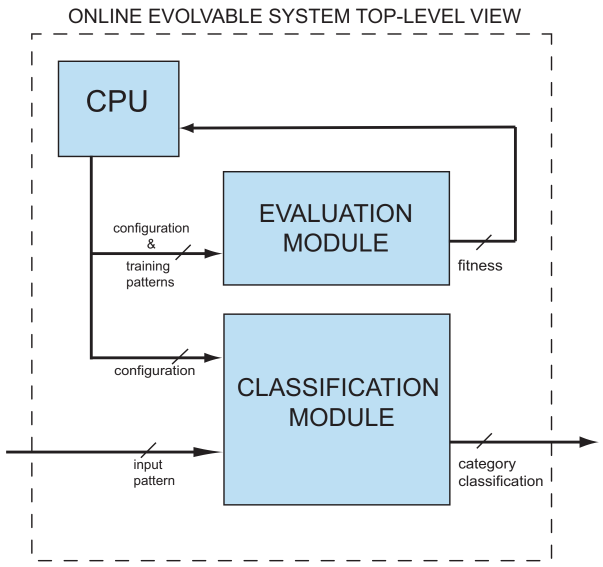

Figure 1. High level system view.

2.1 System Overview

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 CPU. 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, which will be detailed in section 3.3, 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.

2.2 Classification Module Overview

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. In this way, the system could also propose alternative categories in case of doubt.

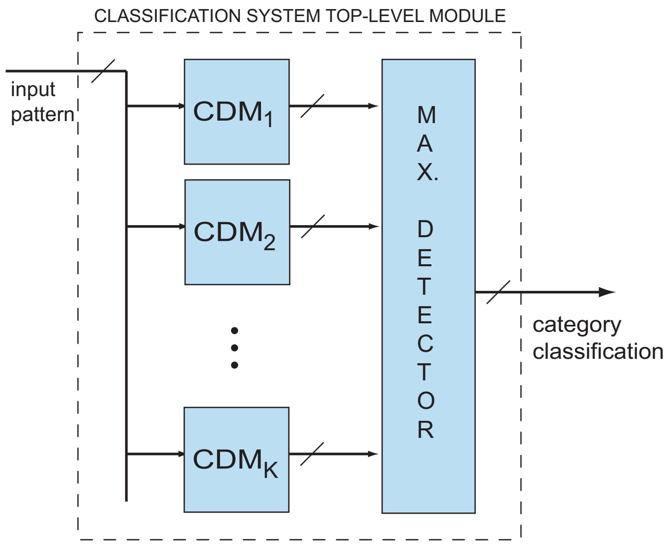

Figure 2. EHW classification module view. The pattern to be classified is input to all of the category detection modules.

2.3 Category Detection Module

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 outputs from the AND gates are connected to an input counter which counts the number of activated FU rows.

2.4 Functional Unit

The FUs are the reconfigurable elements of the architecture. This section describes the FU in a general way, and section 3.2 will describe the application-specific implementation. 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 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. The choice of functions for the image recognition application will be detailed in section 3.1. 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 from the unit.

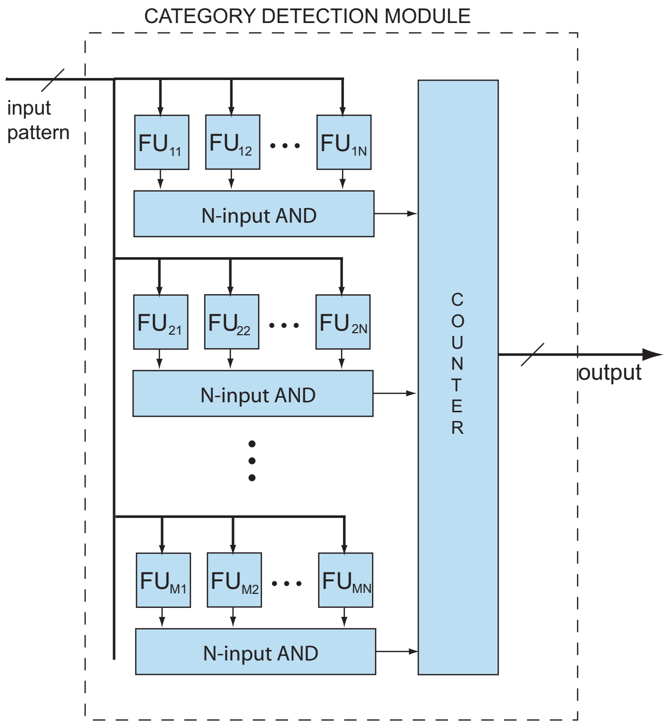

Figure 3. Category detection module. N functional units are connected to an N-input AND gate.

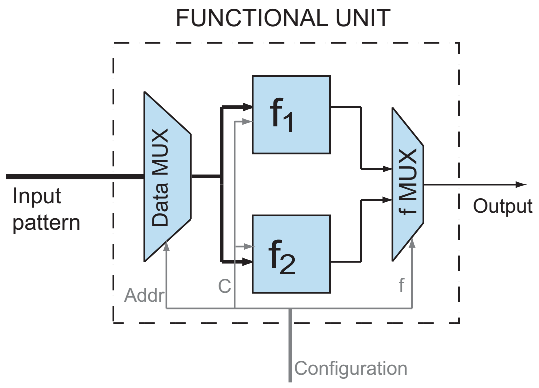

Figure 4. Functional unit. The configuration lines are shown in gray. The data MUX selects which of the input data to feed to the functions f_1 and f_2. The constant C is given by the configuration lines. Finally, the f MUX selects which of the function results to output.

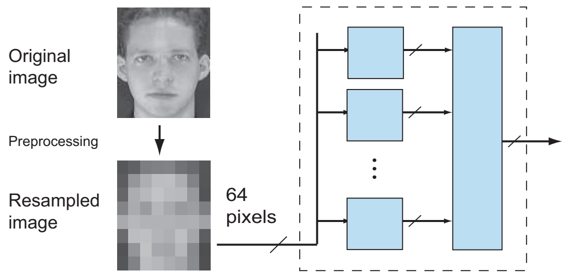

Figure 5. The original images are resampled to a resolution of 8x8 pixels before being fed to the inputs of the classification module.

The advantage of selecting which inputs to use, is that one is not required to connect to all inputs. A *direct* implementation of the LoDETT system [13] would have required, in the image recognition case, $N = 64$ FUs in a row. Our system typically uses $N = 8$ units. The rationale is that not all of the inputs are necessary for the pattern recognition. This is reflected in the *don't care*s evolved in [13].

3 Implementation

This section describes the image recognition application, and the following application-specific implementations of the FU and the evaluation module.

3.1 Face Image Recognition

The pattern recognition system has been applied to face image recognition. The fitness of the face recognition system is based on the system's ability to recognize the correct person from a range of different face images. The images are taken from the AT&T Database of Faces (formerly "The ORL Database of Faces")[1] which contains 400 images divided into 40 people with 10 images each. For each person, images are taken with variations such as different facial expressions and head tilt. The original resolutions of the images were 92x112 pixels, 8-bit gray scale. In our experiment the images were preprocessed by a downsampling to 8x8 pixels, 8-bit gray scale. This was done to reduce noise and the number of inputs to the system. The input pattern to the system is then 64 pixels of 8 bits each (512 bits in total) – see figure 5.

Based on the data elements of the input being 8-bit pixels, the functions available to the FU elements have been chosen to *greater than* and *less than or equal*. Through experiments these functions have shown to work well, and intuitively this allows for detecting dark and bright spots.

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

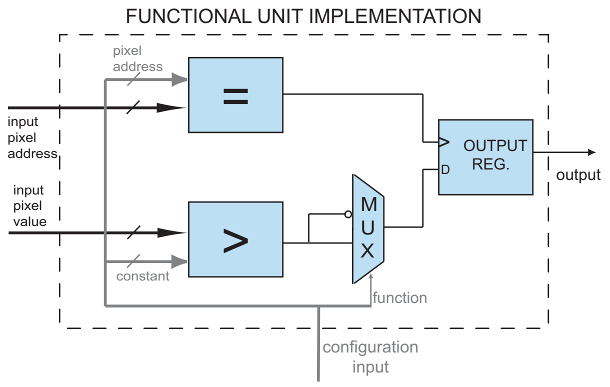

Figure 6. Implementation of the FU for the image recognition case.

Combined use of these functions for the same pixel makes it possible to define an intensity range. The constant is also 8 bits, and the input is then compared to this value to give *true* or *false* as output. This can be summarized as follows, with I being the selected input value, O the output, and C the constant value:

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

3.2 Functional Unit Implementation

Based on the choice of input data elements and functions above, the application-specific implementation of the FU can be determined. See figure 6. As described in the introduction, the VRC technique used for reconfiguration of circuits has the disadvantage of requiring much logic resources. This especially becomes the case when one needs to select the input to a unit from many possible sources, which is a common case for EHW. This is an even bigger problem when one works with data buses as inputs instead of bit signals.

Instead of using large amounts of multiplexer resources for selecting one 8-bit pixel from 64 possible, we have opted for a "time multiplexing" scheme, where only one bit is presented to the unit at a time. The 64 pixels are presented sequentially, one for each clock cycle, together with their identifying image address. The FU checks the input pixel address for a match with the pixel address stored in the configuration register, and in the case of a match, the output value of the FU is stored in a memory element. This method thus requires a maximum of 64 clock cycles before an FU has selected its input value, but saves the use of logic resources for multiplexing.

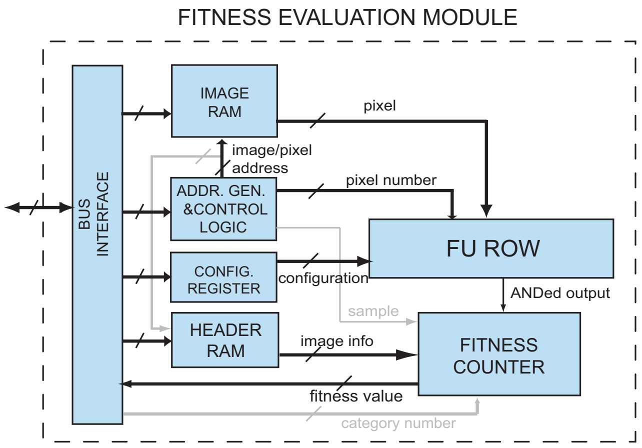

Figure 7. Evaluation module. For clarity, only one FU row is illustrated. The CPU has access to the various registers and RAMs through the on-chip bus and the bus interface.

The pixel input value is then used for comparison with the constant value C stored in the configuration. Since the two functions *greater than* and *less than or equal* are opposite, only a *greater than*-comparator is implemented, and the function bit in the configuration decides whether to choose the direct or the negated output.

3.3 Evaluation Module

The evaluation module computes the fitness for an FU row, which is the phenotype for an *individual* in the evolution. The reason for this will be explained in section 4.2. After being configured through the configuration register – see figure 7, the FU row needs to be fed with all training vectors in the training set, in this case all the training face images of all categories. When using 9 face images for training in each of the 40 categories, this equals 360 images of 64 pixels. In this implementation these images are preloaded into the image RAM, which consists of Xilinx Block RAM (BRAM), local to the evaluation module. This is done in order to facilitate the bursting of pixels to the FU row. An extra BRAM block, the header RAM, contains image header info, i.e. which category the current image belongs to. One could also put the training images in external RAM as long as the throughput is guaranteed.

When the configuration register has been written to by the CPU, the evaluation can be started. The address generator generates addresses for the image and header RAM blocks. The control logic tells the fitness counter when to sample the output of the FU row for fitness calculation. This happens each time after the 64 pixels in one image have been cycled. Finally, when all of the images have been cycled through, an interrupt is sent to the CPU and the calculated fitness value is presented for readback. The details of the fitness calculation will be presented in section 4.3.

Note that the FU row can be duplicated in order to achieve parallel fitness evaluation of different individuals in a generation. This requires a separate configuration register and fitness counter for each FU row, but the pixels are being fed simultaneously to all of the rows. In our implementation we have implemented 8 FU rows in parallel, thus allowing for simultaneous fitness evaluation of 8 individuals.

4 Evolution

This section describes the evolutionary process. The GA implemented for the experiments follows the Simple GA style [4]. The algorithm is written to be run on the PowerPC 405 core in the Xilinx Virtex-II Pro (or better) FPGAs [1]. Allowing the GA to run in software instead of implementing it in hardware gives an increased flexibility compared to a hardware implementation.

The GA associates a bit string (genome) with each individual in the population. For each individual, the fitness evaluation circuit is configured with the associated bit string, and training vectors are applied on the inputs. By reading back the fitness value from the circuit, the individuals can be ranked and used in selection for a new generation. When an individual with a maximum possible fitness value has been created, the evolution run is over and the bit string can be used to configure the operational classification circuit.

4.1 Genome

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

Pixel address (6 bit)	Function (1 bit)	Constant (8 bit)

This gives a total of $B_{unit} = 15$ bits for each unit. The genome for one FU row is encoded as follows:

FU_1(15b)	FU_2(15b)	...	FU_N(15b)

The total amount of bits in the genome for one FU row is then, with $N = 8$, $B_{tot} = B_{unit} \times N = 15 \times 8 = 120$. In the implementation this is rounded up to 128 bits (4 words).

4.2 Incremental Evolution of the Category Detectors

Evolving the whole classification system in one run would give a very long genome, therefore an incremental approach is chosen. Each category detector CDM_i is evolved separately, since there is no interdependency between the different categories. This is also true for the FU rows each CDM consists of. Thus, the evolution can be performed on one FU row at a time. This significantly reduces the genome size.

One then has the possibility of evolving CDM_i in M steps before proceeding to CDM_{i+1}. However, we evolve only *one* FU row in CDM_i before proceeding to CDM_{i+1}. This makes it possible to have a working system in K evolution runs (that is, $1/M$ of the total evolution time). While the recognition accuracy is lower with only one FU row for each CDM, the system is operational and improves gradually as more FU rows are added for each CDM.

4.3 Fitness Function

A certain set of the available vectors, V_t, are used for training of the system, while the remaining, V_v, are used for verification after the evolution run. Each row of FUs is fed with the training vectors ($v \in V_t$), and fitness is based on the row's ability to give a positive (1) output for vectors v belonging to its own category ($C_v = C_i$), while giving a negative (0) output for the rest of the vectors ($C_v \neq C_i$).

In the case of a positive output when $C_v = C_i$, the value A is added to the fitness sum. When $C_v \neq C_i$ and the row gives a negative output (value 0), 1 is added to the fitness sum. The other cases do not contribute to the fitness value. The fitness function F for a row can then be expressed in the following way, where o is the output of the FU row:

$$F = \sum_{v \in V_t} x_v \qquad \text{where } x_v = \begin{cases} A \times o & \text{if } C_v = C_i \\ 1 - o & \text{if } C_v \neq C_i \end{cases} \tag{1}$$

For the experiments, a value of $A = 64$ has been used. This emphasis on the positive matches for C_i has shown to speed up the evolution.

4.4 Genetic Algorithm Implementation

Since the fitness evaluation is performed by the hardware evaluation module, one can save time by letting the evaluation module do fitness evaluation of some individuals while other individuals are created by the CPU. When one new individual has been created, the CPU can send it to the evaluation module, and continue creating the next individual. The evaluation module will then send an interrupt when the evaluation is done, and a new individual can be sent for evaluation.

5 Results

This section presents the results of the experiments undertaken. A complete software simulation and details about the recognition accuracy and architecture parameters have been reported in [2]. The following results concentrate on the implementation of the online evolution system. The hardware implementation was verified by feeding the evaluation module and the software simulator the same genome, and reading back the fitness value. An evolution run was also performed on the online system, with a maximum fitness value individual as the result. In the experiments a population size of 16 was used.

Resource	Used	Available	Percent
Slices	1393	13696	10
Slice Flip Flops	1419	27392	5
4 input LUTs	1571	27392	5
18Kb BRAMs	17	136	12

Table 1. Post-synthesis device utilization for the 8-row evaluation module (except the IPIF bus interface) implemented on the XC2VP30.

Resource	Used	Available	Percent
Slices	145	13696	1.1
Slice Flip Flops	170	27392	0.6
4 input LUTs	115	27392	0.4

Table 2. Post-synthesis device utilization for one 8-FU row (including configuration registers and fitness counter).

5.1 Device Utilization and Clock Speed

The evaluation module, with 8 FU row instances for parallel evaluation of individuals, each with $N = 8$ units, was implemented on a Virtex-II Pro XC2VP30 device. The device utilization can be seen in table 1. These numbers do not include the device utilization of the Xilinx IPIF bus interface, used to simplify the connection to the on-chip bus. The device utilization of one single FU row was also measured, see table 2. From these numbers one can deduct the resources required for any increase or decrease in fitness evaluation parallelism. If one wants to sacrifice evaluation speed for smaller size, the resource usage for a minimal evaluation module (containing one FU row) is 371 slices, that is, 2% of the device.

Synthesis of the evaluation module reports a maximum clock frequency of 131MHz. On the evaluation board the PowerPC core runs at 300MHz, while the rest of the system on the FPGA runs at 100MHz.

5.2 Evolution Speed

	EHW	Xeon	$\frac{\text{Xeon}}{\text{EHW}}$
GA clock freq.(MHz)	300	3000	10
Fit. clock freq.(MHz)	100	3000	30
GA time(ms)	926	9	0.01
Fit. eval. time(ms)	623	1323	2.12
Total time(ms)*	1313	1323	1.01

Table 3. Speed comparison of the online EHW and Xeon 5160 workstation systems. GA time indicates the evolution time except the time used for fitness evaluation. The last column indicates the ratio between the second and first columns.

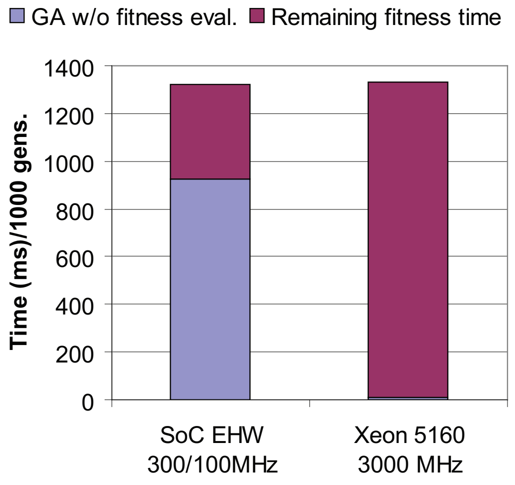

Figure 8. Evolution speeds. The fitness time for the online system only reflects the time the CPU had to wait for the evaluation module.

The speed of an evolution run of 1000 generations was measured for the online EHW system and a state-of-the-art Intel Xeon 5160 workstation for speed comparison. The results can be seen in table 3. The total evolution time for the two systems is almost equal, even though the Xeon workstation runs at 10 to 30 times the clock speed of the online system. Time is spent on different tasks in the two systems. This is illustrated in figure 8. Almost all of the time is spent on fitness evaluation in the workstation, while the time needed for this is lower for the online system, because of the hardware implementation. The online system on the other hand spends a large portion of the time running the rest of the GA, since this part is not hardware optimized. This is much slower than the workstation due not only to the PowerPC's lower clock frequency, but also slower caches (100MHz) and in general a less powerful processor architecture. In addition the GA is not fully optimized for the integer-only PowerPC.

It is interesting to note from table 3 that the total evolution time of the online system is lower than the sum of the time spent on fitness evaluation and the time spent on the rest of the evolution. This is because some of the fitness evaluation is performed in the hardware module in parallel with the software GA running on the processor.

5.3 Discussion

By implementing the time multiplexing scheme for dealing with the high number of inputs to each FU, we have saved a large amount of multiplexer resources on the FPGA. The main drawback of the time multiplexing scheme is the number of cycles required, in this case 64, before each FU has the desired input. This is compensated for by adding several FU rows in parallel. In our experiment, having a parallelism of 8 FU rows made the system evolve solutions as fast as a state-of-the-art workstation, running at much higher frequencies and having a much higher power consumption. This implies that a system has an average total evolution time of 140s for an entire system [2], although the system would be operational (with a lower accuracy) in a much shorter time.

Having 8 rows in parallel for fitness evaluation does not require very much resources on the FPGA used in our experiments, but it could be desirable to move the design to smaller devices, or achieve even higher evolution speeds. One approach which could increase the evaluation speed further would be to add a smaller local memory to each FU row in the evaluation module, containing the current image used for fitness training. The FUs could then, in turn, look up the desired pixels in this memory. This would take, in the case of 8 FUs, only 8 cycles, which is 8 times faster than 64 cycles. This would however require a constant refill of this local memory so that the next image is always ready, which means that the memory needs to be dual-ported and have a bandwidth of 64 bits. In Xilinx Virtex-II Pro FPGAs this could be accomplished by using two 32-bit wide BRAM blocks per row. Applying this to the entire classification system may require much RAM resources, but it could be possible using this solution only for the fitness evaluation module, and it should therefore be investigated.

The analysis of the evolution speed of the online system reveals that much time is spent on the GA algorithm in software. Fitness sorting, selection and scaling should be optimized, and it is expected that this would result in an important speed increase. Moreover, further tuning of the software/hardware parallelism should be applied.

Although the classifier module has not yet been fully implemented in hardware, the results of the implementation so far indicate that the classification time estimate of less than 1μs for one image, suggested in [2], will hold given that there are enough resources on the target device.

Future work includes completion of the classification module and the entire on-chip adapting system, as well as investigating the above-mentioned alternative for faster fitness evaluation. In addition, other applications for the system should be investigated. Suitable applications should be pattern recognition problems requiring high recognition speed. The LoDETT system has been successfully applied to genome informatics and other applications [11, 12]. It is expected that the architecture described in this paper could perform well on similar problems, if suitable functions for the FUs are found.

6 Conclusions

Online evolution has been implemented on an FPGA for a complex and high-speed pattern recognition system. Still, as the evolution follows an incremental scheme, few FPGA resources are needed for evolution compared to the size of the entire system. The functionality has been verified and the evolution speed is equivalent to a software implementation on a high-end workstation. The degree of parallelism of fitness evaluations, and thus evolution speed, can be adjusted at the cost of more resources. Further optimization of the software is possible and needed. On-chip evolution makes the system suitable for continuous adaptation in an embedded application where power consumption and size requirements are important.

Acknowledgment

The research is funded by the Research Council of Norway through the project *Biological-Inspired Design of Systems for Complex Real-World Applications* (proj. 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] K. Glette, J. Torresen, and M. Yasunaga. An online ehw pattern recognition system applied to face image recognition. In M. G. et al., editor, *Applications of Evolutionary Computing, EvoWorkshops2007: EvoCOMNET, EvoFIN, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC, EvoTransLog*, volume 4448 of *Lecture Notes in Computer Science*, pages 271–280. Springer-Verlag, 2007.

[3] K. Glette, J. Torresen, M. Yasunaga, and Y. Yamaguchi. On-chip evolution using a soft processor core applied to image recognition. In *Proc. of the First NASA /ESA Conference on Adaptive Hardware and Systems (AHS 2006)*, pages 373–380, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

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

[5] 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.

[6] H. Ngo, R. Gottumukkal, and V. Asari. A flexible and efficient hardware architecture for real-time face recognition based on eigenface. In *Proc. of IEEE Computer Society Annual Symposium on VLSI*, pages 280–281. IEEE, 2005.

[7] L. Sekanina and R. Ruzicka. Design of the special fast reconfigurable chip using common F PGA. In *Proc. of Design and Diagnostics of Electronic Circuits and Sy stems - IEEE DDECS'2000*, pages 161–168, 2000.

[8] J. Torresen. Scalable evolvable hardware applied to road image recognition. In J. L. et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 245–252. IEEE Computer Society, Silicon Valley, USA, July 2000.

[9] J. Torresen, W. J. Bakke, and L. Sekanina. Recognizing speed limit sign numbers by evolvable hardware. In *Parallel Problem Solving from Nature*, volume 2004, pages 682–691. Springer Verlag, 2004.

[10] 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.

[11] M. Yasunaga et al. Gene finding using evolvable reasoning hardware. In P. H. A. Tyrrel and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 228–237. Springer-Verlag, 2003.

[12] M. Yasunaga, J. H. Kim, and I. Yoshihara. The application of genetic algorithms to the design of reconfigurable reasoning vlsi chips. In *FPGA '00: Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, pages 116–125, New York, NY, USA, 2000. ACM Press.

[13] 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.

[14] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458, 2003.