An Online EHW Pattern Recognition System Applied to Sonar Spectrum Classification

Kyrre Glette[1], Jim Torresen[1], and Moritoshi Yasunaga[2]

[1] University of Oslo, Department of Informatics,
P.O. Box 1080 Blindern, 0316 Oslo, Norway,
{kyrrehg,jimtoer}@ifi.uio.no
[2] 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. An evolvable hardware (EHW) system for high-speed sonar return classification has been proposed. The system demonstrates an average accuracy of 91.4% on a sonar spectrum data set. This is better than a feed-forward neural network and previously proposed EHW architectures. Furthermore, this system is designed for online evolution. Incremental evolution, data buses and high level modules have been utilized in order to make the evolution of the 480 bit-input classifier feasible. The classification has been implemented for a Xilinx XC2VP30 FPGA with a resource utilization of 81% and a classification time of 0.5μs.

1 Introduction

High-speed pattern recognition systems applied in time-varying environments, and thus needing adaptability, could benefit from an online evolvable hardware (EHW) approach [1]. One EHW approach to online reconfigurability is the Virtual Reconfigurable Circuit (VRC) method proposed by Sekanina in [2]. 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.

An EHW pattern recognition system, Logic Design using Evolved Truth Tables (LoDETT), has been presented by Yasunaga et al. Applications include face image and sonar target recognition [3,4]. This architecture is capable of classifying large input vectors (512 bits) into several categories. The classifier function is directly coded in large AND gates. The category module with the highest number of activated AND gates determines the classification. Incremental evolution is utilized such that each category is evolved separately. The average recognition accuracy for this system, applied to the sonar target task, is 83.0%. However, evolution is performed *offline* and the final system is synthesized. This approach gives rapid ($<$ 150ns) classification in a compact circuit, but lacks run-time reconfigurability.

A system proposed earlier by the authors addresses the reconfigurability by employing a VRC-like array of high-level functions [5]. 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.

A new architecture was then proposed by the authors to allow for the high classification capabilities of the LoDETT system, while maintaining the online evolution features from [5]. This was applied to multiple-category face image recognition and a slightly higher recognition accuracy than the LoDETT system was achieved [6]. While in LoDETT a large number of inputs to the AND gates can be optimized away during circuit synthesis, the run-time reconfiguration aspect of the online architecture has led to a different approach employing fewer elements. The evolution part of this system has been implemented on an FPGA in [7]. Fitness evaluation is carried out in hardware, while the evolutionary algorithm runs on an on-chip processor.

In this paper the architecture, previously applied to face image recognition, has been applied to the sonar target recognition task. The nature of this application has led to differences in the architecture parameters. Changes in the fitness function were necessary to deal with the higher difficulty of this problem.

The sonar target dataset was presented by Gorman and Sejnowski in [8]. A feed-forward neural network was presented, which contained 12 hidden units and was trained using the back-propagation algorithm. A classification accuracy of 90.4% was reported. Later, better results have been achieved on the same data set, using variants of the Support Vector Machine (SVM) method. An accuracy of 95.2% was obtained in a software implementation presented in [9]. There also exists some hardware implementations of SVMs, such as [10], which performs biometric classification in 0.66ms using an FPGA.

The next section introduces the architecture of the evolvable hardware system. Then, the sonar return-specific implementation is detailed in section 3. Aspects of evolution are discussed in section 4. Results from the experiments are given and discussed in sections 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 (GA) 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 [11,5].

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

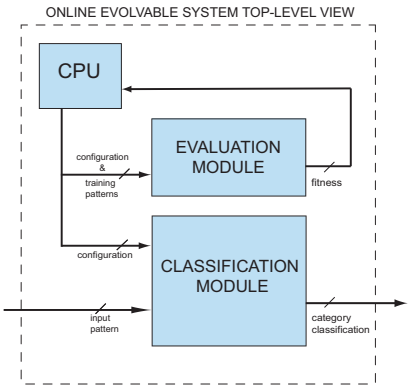

Fig. 1. High level system view.

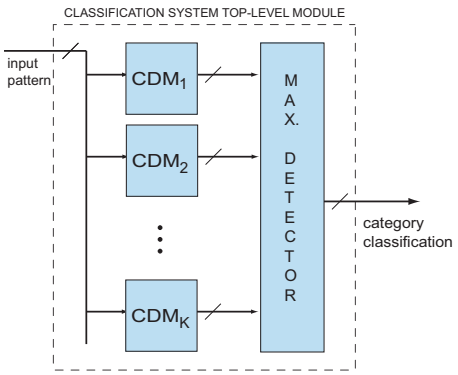

Fig. 2. EHW classification module view.

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 [7].

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.

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 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. To draw a parallel to the LoDETT system, each FU row represents a kernel function. More FU rows give more kernel functions (with different centers) that the unknown pattern can fall into.

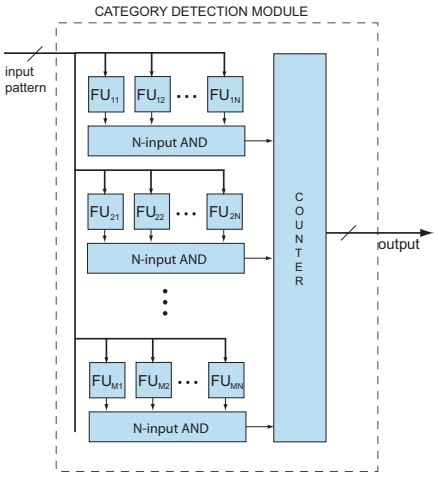

Fig. 3. Category detection module. N functional units are connected to an N-input AND gate.

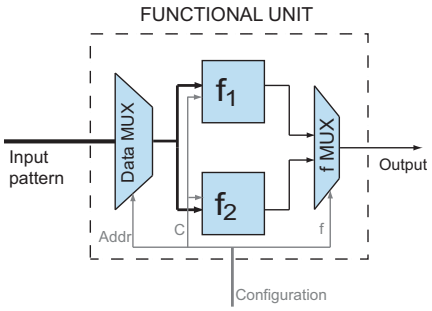

Fig. 4. Functional unit. The data MUX selects which of the input data to feed to the functions f_1 and f_2. The f MUX selects which of the function results to output.

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 sonar classification 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.

The advantage of selecting which inputs to use, is that connection to all inputs is not required. A *direct* implementation of the LoDETT system [4] would have required, in the sonar case, $N = 60$ FUs in a row. Our system typically uses

$N = 6$ 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 [4].

3 Implementation

This section describes the sonar return classification application and the following application-specific implementation of the FU module. The evaluation module, which contains one FU row and calculates fitness based on the training vectors, is in principle equal to the description in [7] and will not be further described in this paper.

3.1 Sonar Return Classification

The application data set has been found in the CMU Neural Networks Benchmark Collection[3], and was first used by Gorman and Sejnowski in [8]. This is a real-world data set consisting of sonar returns from underwater targets of either a metal cylinder or a similarly shaped rock. The number of CDMs in the system then becomes $K = 2$. The returns have been collected from different aspect angles and preprocessed based on experiments with human listeners, such that the input signals are spectral envelopes containing 60 samples, normalized to values between 0.0 and 1.0 – see figure 5. There are 208 returns in total which have been divided

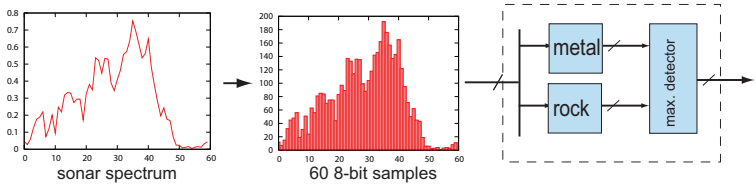

Fig. 5. The sonar return spectral envelope, which is already a preprocessed signal, has its 60 samples scaled to 8-bit values before they are input to the CDMs.

into equally sized training and test sets of 104 returns. The samples have been scaled by the authors to 8-bit values ranging between 0 and 255. This gives a total of $60 \times 8 = 480$ bits to input to the system for each return.

Based on the data elements of the input being 8-bit scalars, 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 the presence or absence of frequencies in the signal, and their amplitude. 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:

[3] http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/

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 data elements and functions above, the application-specific implementation of the FU can be determined. 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 working with data buses as inputs instead of bit signals.

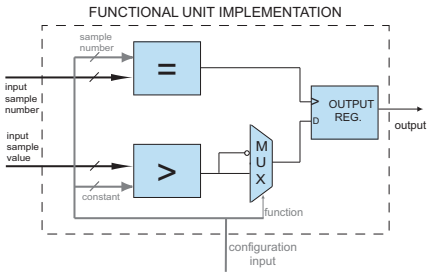

Fig. 6. Implementation of the FU for the sonar spectrum recognition case.

Instead of using a large amount of multiplexer resources for selecting one 8-bit sample from 60 possible, we have opted for a "time multiplexing" scheme, where only one bit is presented to the unit at a time. See figure 6. The 60 samples are presented sequentially, one for each clock cycle, together with their identifying sample number. The FU checks the input sample number for a match with the sample number 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 60 clock cycles before an FU has selected its input value.

The sample input value is 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.

4 Evolution

This section describes the evolutionary process. Although the base mechanisms are the same as in [6], there are important changes to the fitness function.

The GA implemented for the experiments follows the Simple GA style [12]. The algorithm is written to be run on the PowerPC 405 hard processor core in the Xilinx Virtex-II Pro (or better) FPGAs [11], or the MicroBlaze soft processor

core available for a greater number of FPGA devices [5]. 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 (or the maximum limit of generations has been reached), the evolution run is over and the bit string can be used to configure a part of the operational classification circuit.

4.1 Genome

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

Spectrum sample 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 $N = 6$, $B_{tot} = B_{unit} \times N = 15 \times 6 = 90$. In the implementation this is rounded up to 96 bits (3 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 can be evolved separately, since there is no interdependency between the different categories. This is also true for the FU rows each CDM consists of. Although the fitness function changes between the rows, as will be detailed in the next section, the evolution can be performed on one FU row at a time. This significantly reduces the genome size.

4.3 Fitness Function

The basic fitness function, applied in [6], can be described as follows: 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 the 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 ($C_v \neq C_i$).

In the case of a positive output when $C_v = C_i$, the value 1 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 basic fitness function F_B for a row can then be expressed in the following way, where o is the output of the FU row:

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

While in the face image application each FU row within the same category was evolved with the same fitness function, the increased variation of the training set in the current application made it sensible to divide the training set between different FU rows. Each FU row within the same category is evolved separately, but by changing the fitness function between the evolution runs one can make different FU rows react to different parts of the training set. The extended fitness function F_E can then be expressed as follows:

$$F_E = \sum_{v \in V_t} x_v \qquad \text{where } x_v = \begin{cases} o & \text{if } C_v = C_i \text{ and } v \in V_{f,m} \\ 1 - o & \text{if } C_v \neq C_i \end{cases} \tag{2}$$

Where $V_{f,m}$ is the part of the training set FU row m will be trained to react positively to. For instance, if the training set of 104 vectors is divided into 4 equally sized parts, FU row 1 of the CDM would receive an increase in fitness for the first 26 vectors, if the output is positive for vectors belonging to the row's category (i.e. "rock" or "metal"). In addition, the fitness is increased for giving a negative output to vectors not belonging to the row's category, for all vectors of the training set.

5 Results

This section presents the results of the implementation and experiments undertaken. The classification results are based on a software simulation of the EHW architecture, with has identical functionality to the hardware proposed. Results from a hardware implementation of the classifier module are also presented.

5.1 Architecture and Evolution Parameters

The architecture parameters N and M, that is, the number of FUs in an FU row and the number of FU rows in a CDM, respectively, have been evaluated. From experiments, a value of $N = 6$ has shown to work well. Increasing the number of FU rows for a category leads to an increase in the recognition accuracy, as seen in figure 7. However, few FU rows are required before the system classifies with a relatively good accuracy, thus the system could be considered operational before the full number of FU rows are evolved. It is worth noting that the training set classification accuracy relatively quickly rises to a high value, and then has slow growth, compared to the test set accuracy which has a more steady increase as more FU rows are added.

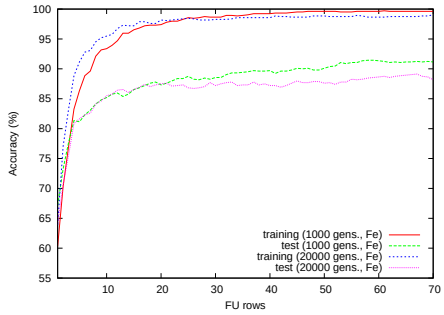

Fig. 7. Average classification accuracy on the training and test sets as a function of the number of FU rows per CDM. $N = 6$. Results for generation limits of 1000 and 20000.

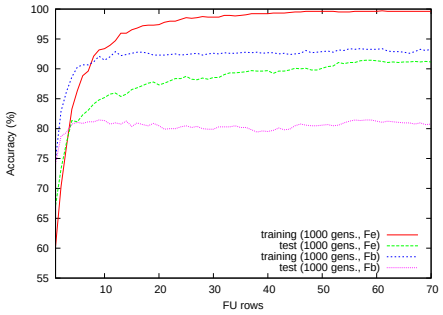

Fig. 8. Average classification performance obtained using fitness functions F_B and F_E. $N = 6$.

For the evolution experiments, a population size of 32 is used. The crossover rate is 0.9. Linear fitness scaling is used, with 4 expected copies of the best individual. In addition, elitism is applied. The maximum number of generations allowed for one evolution run is 1000. By observing figure 7 one can see that this produces better results than having a limit of 20000 generations, even though this implies that fewer FU rows are evolved to a maximum fitness value.

The classification accuracies obtained by using the basic fitness function F_B, and the extended fitness function F_E with the training set partitioned into 4 parts of equal size, have been compared – see figure 8. The use of F_E allows for a higher classification accuracy on both the training and the test set.

5.2 Classification Accuracy

10 evolution runs were conducted, each with the same training and test set as described in section 3.1, but with different randomized initialization values for the genomes. The extended fitness function F_E is used. The results can be seen

FU rows (M)	training set	test set
20	97.3%	87.8%
58	99.6%	91.4%

Table 1. Average classification accuracy.

in table 1. The highest average classification accuracy, 91.4%, was obtained at $M = 58$ rows. The best system evolved presented an accuracy of 97.1% at $M = 66$ rows (however, the average value was only 91.2%). The results for $M = 20$, a configuration requiring less resources, are also shown. These values are higher than the average and maximum values of 83.0% and 91.7% respectively obtained

in [4], although different training and test sets have been used. The classification performance is also better than the average value of 90.4% obtained from the original neural network implementation in [8], but lower than the results obtained by SVM methods (95.2%) [9].

5.3 Evolution Speed

In the experiments, several rows did not achieve maximum fitness before reaching the limit of 1000 generations per evolution run. The average number of generations required for each evolution run (that is, one FU row) was 853. This gives an average of 98948 generations for the entire system. The average evolution time for the system is 63s on an Intel Xeon 5160 processor using 1 core. This gives an average of 0.54s for one FU row, or 4.3s for 8 rows (the time before the system has evolved 4 FU rows for each category and thus is operational).

A hardware implementation of the evaluation module has been reported in [7]. This was reported to use 10% of the resources of a Xilinx XC2VP30 FPGA, and, together with the GA running on an on-chip processor, the evolution time was equivalent to the time used by the Xeon workstation. Similar results, or better because of software optimization, are expected for the evolution in the sonar case.

5.4 Hardware Implementation

An implementation of the classifier module has been synthesized for a Xilinx XC2VP30 FPGA in order to achieve an impression of speed and resource usage. The resources used for two configurations of the system can be seen in table 2. While the $M = 58$ configuration uses 81% of the FPGA slices, the $M = 20$

Resource	$M = 20$	$M = 58$	Available
Slices	3866	11189	13696
Slice Flip Flops	4094	11846	27392
4 input LUTs	3041	8793	27392

Table 2. Post-synthesis device utilization for two configurations of the 2-category classification module implemented on an XC2VP30.

configuration only uses 28% of the slices. Both of the configurations classify at the same speed, due to the parallel implementation. Given the post-synthesis clock frequency estimate of 118MHz, and the delay of 63 cycles before one pattern is classified, one has a classification time of 0.5μs.

5.5 Discussion

Although a good classification accuracy was achieved, it became apparent that there were larger variations within each category in this data set than in the face image recognition data set applied in [6]. The fitness function was therefore extended such that each row would be evolved with emphasis on a specific part of

the training set. This led to increased classification accuracy due to the possibility for each row to specialize on certain features of the training set. However, the partitioning of the training set was fixed, and further investigation into the partitioning could be useful.

The experiments also showed that by increasing the number of FU rows per category, better generalization abilities were obtained. The fact that the generalization became better when the evolution was cut off at an earlier stage, could indicate that a CDM consisting of less "perfect" FU rows has a higher diversity and is thus less sensible to noise in the input patterns.

The main improvement of this system over the LoDETT system is the aspect of on-line evolution. As a bonus the classification accuracies are also higher. The drawback is the slower classification speed, 63 cycles, whereas LoDETT only uses 3 cycles for one pattern. It can be argued that this slower speed is negligeable in the case of the sonar return application, since the time used for preprocessing the input data in a real-world system would be higher than 63 cycles. In this case, even an SVM-based hardware system such as the one reported in [10] could be fast enough. Although the speed is not directly comparable since the application differs, this system has a classification time of 0.66ms, roughly 1000 times slower than our proposed architecture. Therefore, the architecture would be more ideal applied to pattern recognition problems requiring very high throughput. The LoDETT system has been successfully applied to genome informatics and other applications [13,14]. It is expected that the proposed architecture also could perform well on similar problems, if suitable functions for the FUs are found.

6 Conclusions

The online EHW architecture proposed has so far proven to perform well on a face image recognition task and a sonar return classification task. Incremental evolution and high level building blocks are applied in order to handle the complex inputs. The architecture benefits from good classification accuracy at a very high throughput. The classification accuracy has been shown to be higher than an earlier offline EHW approach. Little evolution time is needed to get a basic working system operational. Increased generalisation can then be added through further evolution. Further, if the training set changes over time, it would be possible to evolve better configurations in parallel with a constantly operational classification module.

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. Yao, X., Higuchi, T.: Promises and challenges of evolvable hardware. In Higuchi, T., et al., eds.: Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96. Volume 1259 of Lecture Notes in Computer Science. Springer-Verlag (1997) 55–78
2. Sekanina, L., Ruzicka, R.: 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. (2000) 161–168
3. Yasunaga, M., et al.: Evolvable sonar spectrum discrimination chip designed by genetic algorithm. In: Proc. of 1999 IEEE Systems, Man, and Cybernetics Conference (SMC'99). (1999)
4. Yasunaga, M., Nakamura, T., Yoshihara, I., Kim, J.: Genetic algorithm-based design methodology for pattern recognition hardware. In Miller, J., et al., eds.: Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000. Volume 1801 of Lecture Notes in Computer Science., Springer-Verlag (2000) 264–273
5. Glette, K., Torresen, J., Yasunaga, M., Yamaguchi, Y.: 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), Los Alamitos, CA, USA, IEEE Computer Society (2006) 373–380
6. Glette, K., Torresen, J., Yasunaga, M.: An online EHW pattern recognition system applied to face image recognition. In et al., M.G., ed.: Applications of Evolutionary Computing, EvoWorkshops2007: EvoCOMNET, EvoFIN, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC, EvoTransLog. Volume 4448 of Lecture Notes in Computer Science. Springer-Verlag (2007) 271–280 To appear.
7. Glette, K., Torresen, J., Yasunaga, M.: Online evolution for a high-speed image recognition system implemented on a Virtex-II Pro FPGA. (2007) Accepted to The Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007).
8. Gorman, R.P., Sejnowski, T.J.: Analysis of hidden units in a layered network trained to classify sonar targets. Neural Networks **1**(1) (1988) 75–89
9. Frieß, T.T., Cristianini, N., Campbell, C.: The Kernel-Adatron algorithm: a fast and simple learning procedure for Support Vector machines. In: Proc. 15th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (1998) 188–196
10. Choi, W.Y., Ahn, D., Pan, S.B., Chung, K.I., Chung, Y., Chung, S.H.: SVM-based speaker verification system for match-on-card and its hardware implementation. Electronics and Telecommunications Research Institute journal **28**(3) (2006) 320–328
11. Glette, K., Torresen, J.: 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. Springer-Verlag (2005) 66–75
12. Goldberg, D.: Genetic Algorithms in search, optimization, and machine learning. Addison–Wesley (1989)
13. Yasunaga, M., et al.: Gene finding using evolvable reasoning hardware. In A. Tyrrel, P.H., Torresen, J., eds.: Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES'03. Volume 2606 of Lecture Notes in Computer Science. Springer-Verlag (2003) 228–237
14. Yasunaga, M., Kim, J.H., Yoshihara, I.: 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, New York, NY, USA, ACM Press (2000) 116–125