A Signal Processing Architecture based on Evolving Digital Logic Gates

Jim Torresen
Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: jimtoer@ifi.uio.no

Abstract— **Evolvable Hardware (EHW) is a new method for designing electronic circuits. In this paper, the scheme is applied for the design of a novel digital signal processing architecture. It is based on incremental evolution of digital logic circuits. That is, to improve evolvability, subcircuits are initially evolved. The architecture is applied as a prosthetic hand controller. By applying the proposed method, the best performance achieved is substantially better than that obtained by an artificial neural network.**

1 Introduction

There exist a set of signal classification tasks where there is a need for adaptation. To make this possible, automatic schemes for adaptation of electronic circuits would have to be developed. One such technique – inspired by natural evolution, is called evolvable hardware (EHW). Instead of manually designing a circuit, only input/output-relations are specified. The circuit is automatically designed using an adaptive algorithm like Genetic Algorithms (GA). The proposed signal classification architecture is a general one. However, to show how well it performs, this paper contains the results from classifying human signals for prosthetic hand control.

To enhance the lives of people who have lost a hand, prosthetic hands have existed for a long time. These are operated by the signals generated by contracting muscles – named electromyography (EMG) signals, in the remaining part of the arm [1]. By using EHW it is possible to make the *controller* itself adapt to each disabled person. The controller is constructed as a signal classification hardware which maps input patterns to desired actions of the prosthetic hand. Adaptable controllers have been proposed based on neural networks [2]. These require a floating point CPU or a neural network chip. EHW based controllers, on the other hand, use a few layers of digital logic gates for the processing. Thus, a more compact implementation can be provided making it more feasible to be installed inside a prosthetic hand.

One of the main problems in evolving hardware systems seems to be the limitation in the chromosome string length [3], [4]. A long string is normally required for solving a complex problem. However, a larger number of generations is required by the evolutionary algorithm as the string increases. This often makes the search space becoming too large. Thus, work has been undertaken to try to diminish this limitation. One promising approach, incremental evolution of EHW was first introduced in [5] for a character recognition system. The approach is a divide-and-conquer on the evolution of the EHW system, and thus, named *increased complexity evolution*. It consists of a division of the *problem* domain together with incremental evolution of the hardware system. Evolution is first undertaken individually on a set of basic units. The evolved units are the building blocks used in further evolution of a larger and more complex system. The benefits of applying this scheme is both a *simpler* and *smaller* search space compared to conducting evolution in one single run. The goal is to develop a scheme that could evolve systems for complex real-world applications.

In this paper, it is applied to evolve a signal classification architecture applied to prosthetic hand control. An EHW architecture as well as how incremental evolution is applied are described. The goal of the design is to provide an architecture with a high generalization performance. This is to make it a strong alternative to methods like artificial neural networks.

The next two sections introduce the concepts of the architecture. Then, results are given in section 4 with conclusions in section 5.

2 Prosthetic Hand Control

The research on adaptable controllers presented in this paper is based on designing a controller providing six different motions in three different degrees of freedom: Open and Close hand, Extension and Flection of wrist, Pronation and Supination of wrist. Such a complex controller could probably only be designed by *adapting* the controller to each dedicated user. The data set consists of the same motions as used in earlier work [6], and it is collected by Dr. Kajitani at Electrotechnical Laboratory in Japan. Some of the initial results using this data set can be found in [7].

The absolute value of the EMG signal is integrated for 1 s and the resulting value is coded by *four* bits. To improve the performance of the controller it is beneficial to be using several channels. In these experiments *four* channels were used in total, giving an input vector of 4 x 4 = 16 bits.

The *output* vector consists of one binary output for each hand motion, and therefore, the output vector is coded by *six* bits. For each vector only *one* bit is "1". Thus, the data set is collected from a disabled person by considering one motion at a time. For each of the six possible motions, a total of 50 data vectors are collected, resulting in a total of: 6 x 50 = 300 vectors. Further, *two* such sets were made, one to be

used for evolution (training) and the others to be used as a separate test set for evaluating the best circuit *after* evolution is finished.

3 An Architecture for Incremental Evolution

In this section, the proposed classifier architecture is described. This includes the algorithm for undertaking the incremental evolution.

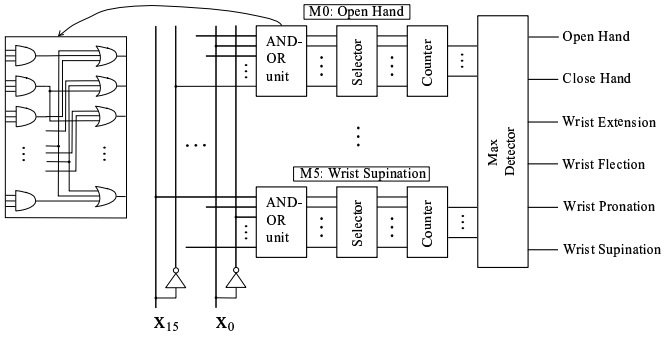

Fig. 1. The Digital Gate Based Architecture of the Prosthetic Hand Controller

The architecture is illustrated in Fig. 1. It consists of one subsystem for *each* of the six prosthetic motions. In each subsystem, the binary inputs $x_0 \ldots x_{15}$ are processed by a number of different units, starting by the AND-OR unit. This is a layer of AND gates followed by a layer of OR gates. Each gate has three inputs. The outputs of the OR gates are routed to the Selector. This unit selects *which* of these outputs that are to be counted by the succeeding counter. That is, for each new input, the Counter is counting the number of *selected* outputs being "1" from the corresponding AND-OR unit. Finally, the Max Detector outputs which counter – corresponding to *one* specific motion, is having the largest value. Each output from the Max Detector is connected to the corresponding motor in the prosthesis. If the Counter having the *largest* value corresponds to the correct hand motion, the input has been correctly classified. One of the motivations for introducing the selectors is to be able to adjust the *number* of outputs from each AND-OR unit in a flexible way. A scheme, based on using multi-input AND gates together with counters, has been proposed earlier [8]. However, the architecture used in this paper is distinguished by including OR-gates, together with the selector units involving incremental evolution. The incremental evolution of this system can be described by the following steps:

1. **Step 1 evolution.** Evolve the AND-OR unit for each subsystem *separately* one at a time. Apply *all* vectors in the training set for the evolution of each subsystem. There are no interaction among the subsystems at this step, and the fitness is measured on the output of the AND-OR units.
2. **Step 2 evolution.** Assemble the six AND-OR units into one system as seen in Fig. 1. The AND-OR units are now fixed and the *Selectors* are to be evolved in the assembled system – in one common run. The fitness is measured using the same training set as in step 1 but the evaluation is now on the output of the Max Detector.
3. The system is now ready to be applied in the prosthesis.

In the first step, subsystems are evolved separately, while in the second step these are evolved together. The motivation for evolving separate subsystems – instead of a single system in one operation, is that earlier work has shown that the evolution time can be substantially reduced by this approach [5], [9].

The layers of AND and OR gates in one AND-OR unit consist of 32 gates each. This number has been selected to give a chromosome string of about 1000 bits which has been shown earlier to be appropriate. A larger number would have been beneficial for expressing more complex Boolean functions. However, the search space for evolution could easily become too large. For the step 1 evolution, each gate's *inputs* are determined by evolution.

As described in the previous section, the EMG signal input consists of 16 bits. Inverted versions of these are made available on the inputs as well, making up a total of 32 input lines to the gate array. The evolution is based on gate level building blocks. However, since several output bits are used to represent one motion, the signal resolution becomes increased from the two binary levels.

For the step 2 evolution, each line in each selector is represented by *one* bit in the chromosome. This makes a chromosome of 32 x 6 bits= 192 bits. If a bit is "0", the corresponding line should *not* be input to the counter, whereas if the bit "1", the line *should* be input.

Fitness Measure

In step 1 evolution, the fitness is measured on all the 32 outputs of each AND-OR unit. As an alternative experiment, we would like to measure the fitness on a *limited* number (16 is here used as an example) of the outputs. That is, each AND-OR unit still has 32 outputs but – as seen in Fig. 2, only 16 are included in the computation of the fitness function:

$$\text{Fitness} = \sum_{i=1}^{16} \text{Output OR gate } i \qquad (1)$$

The 16 outputs not used are included in the chromosome and have *random* values. That is, their values do not affect the fitness of the circuit. After evolution, all 32 outputs are applied for computing the performance:

$$\text{Performance} = \sum_{i=1}^{32} \text{Output OR gate } i \qquad (2)$$

Since 16 OR gates are used for fitness computation, the "fitness measure" equals 16. In the figure, gate 1 to 16 are used for the fitness function. However, in principle any 16 gates out of the 32 can be used. Other numbers than 16 were tested in experiments but 16

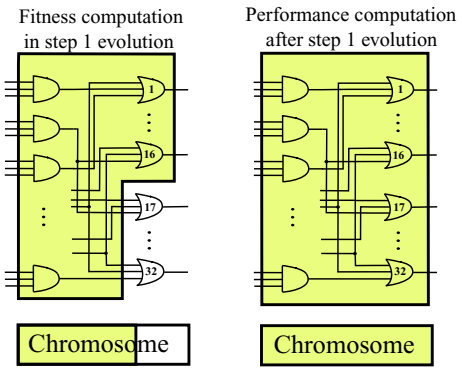

Fig. 2. A "Fitness Measure" Equal to 16

showed to give the best performance results and was used in the following reported experiments.

This could be an interesting approach to improve the generalisation of the circuit. Only the OR gates in the AND-OR unit are "floating" during the evolution since all AND gates may be inputs to the 16 OR gates used by the fitness function. The 16 "floating" OR-gates then provide additional combination of these *trained* AND gates.

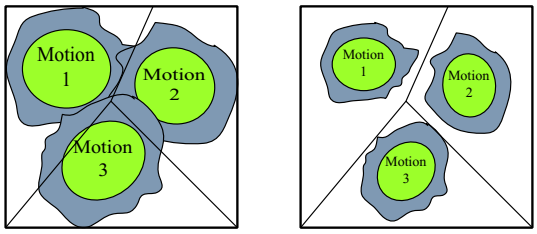

Fig. 3. Illustration of Noise Added to a) A Plain Signal and b) A Pre-Processed Signal

Another way to look at this is that the "floating" gates provide "noise". However, the noise is not added to the plain input but to a preprocessed and *improved* signal (output from the AND gates) as illustrated in Fig. 3. The inner circle for each motion indicates the *training set* domain, with the outer circle indicating the added generalisation obtained by adding "noise".

In a) the signal is not pre-processed and adding noise makes the interference among classes worse while in b) it improves the generalisation rather than introducing interference. The step 2 evolution will be evolving the ratio of noise in the final system by adjusting the number of selector bits set for gates 1 to 16 compared to for gates 17 to 32.

Fitness Function

The fitness function is important for the performance when evolving circuits. For the step 1 evolution, the fitness function – applied for each AND-OR unit separately, is as follows for the motion m ($m \in [0,5]$) unit:

$$F_1(m) = \frac{1}{s} \sum_{j=0}^{50m-1} \sum_{i=1}^{O} x + \sum_{j=50m}^{50m+49} \sum_{i=1}^{O} x + \frac{1}{s} \sum_{j=50m+50}^{P-1} \sum_{i=1}^{O} x$$

$$\text{where } x = \begin{cases} 0 & \text{if } y_{i,j} \neq d_{m,j} \\ 1 & \text{if } y_{i,j} = d_{m,j} \end{cases}$$

where $y_{i,j}$ in the computed output of OR gate i and $d_{m,j}$ is the corresponding target value of the training vector j. P is the total number of vectors in the training set ($P = 300$). As mentioned earlier, each subsystem is trained for one motion (the middle expression of F_1). This includes outputting "0" for input vectors for other motions (the first and last expressions of F_1).

The s is a scaling factor to implicitly emphasize on the vectors for the motion the given subsystem is assigned to detect. An appropriate value ($s = 4$) was found after some initial experiments. The O is the number of outputs included in the fitness function and is either 16 or 32 in the following experiments (referred to as "fitness measure" in the previous section).

The fitness function for the step 2 evolution is applied on the complete system and is given as follows:

$$F_2 = \sum_{j=0}^{P-1} x \qquad \text{where}$$

$$x = \begin{cases} 1 & \text{if } d_{m,j} = 1 \text{ and } m = i \text{ for which } \max_{i=0}^{5}(Ctr_i) \\ 0 & \text{else} \end{cases}$$

This fitness function counts the number of training vectors for which the target *output* being "1" *equals* the *id* of the counter having the maximum output (as mentioned earlier only *one* output bit is "1" for each training vector).

The Evolutionary Algorithm

The simple Genetic Algorithm (GA) – given by Goldberg [10], was applied for the evolution with a population size of 50. For each new generation an entirely new population of individuals is generated. Elitism is used, thus, the best individuals from each generation are carried over to the next generation. The (single point) crossover rate is 0.8, thus the cloning rate is 0.2. Roulette wheel selection scheme is applied. The mutation rate – the probability of bit inversion for each bit in the binary chromosome string, is 0.01.

Various experiments were undertaken to find appropriate GA parameters. The ones that seemed to give the best results were selected and fixed for all the experiments. This was necessary due to the large number of experiments that would have been required if GA parameters should be able vary through all the experiments. The preliminary experiments indicated that the parameter setting was not a major critical issue.

The proposed architecture fits into most FPGAs. The evolution is undertaken off-line using software simulation. However, since no feed-back connections are used and the number of gates between the input and output is limited, the real performance should equal the simulation. Any spikes could be removed using registers in the circuit.

For each experiment presented, four different runs of GA were performed. Thus, *each* of the four resulting circuits from step 1 evolution is taken to step 2 evolution and evolved for four runs.

Motion	Selector 1-16	Selector 17-32	# 1's
0	0 0 0 1 0 0 1 0	1 1 1 1 1 0 1 0	0 1 1 1 0 1 1 1	0 0 0 1 1 1 0 1	18
1	1 1 1 1 1 1 0 0	0 0 0 1 1 1 1 1	1 0 1 1 0 1 0 1	1 0 1 0 1 0 0 0	19
2	1 0 0 1 1 0 1 0	0 0 1 1 0 0 0 0	1 1 0 1 0 1 1 1	1 0 0 1 0 0 0 0	14
3	0 0 1 1 0 1 1 1	1 1 0 0 1 0 0 0	0 1 1 0 0 0 1 1	1 1 1 0 0 1 1 1	18
4	0 1 0 1 1 0 1 1	1 1 0 1 0 1 1 1	0 0 1 1 1 0 1 1	0 1 0 1 0 0 0 1	19
5	1 1 1 0 1 1 1 0	0 1 0 1 1 1 1 1	0 1 0 1 1 0 1 1	1 0 0 1 1 0 0 0	20

TABLE I

The selector settings of the best performing circuit.

4 Results

This section reports the experiments undertaken to search for an optimal configuration of the prosthetic hand controller.

The circuit evolved with the best *test set* performance obtained 67% correct classification (the maximum training set performance was 76.3%.). The circuit had a 60.7% test set performance after step 1 evolution (evaluated with all 32 outputs of the subsystems). The step 2 evolution provided a substantial increase up to 67%. Other circuits didn't perform that well, but the important issue is that it has been shown that the proposed architecture provides the *potential* for achieving high degree of generalization.

A feed-forward neural network was trained and tested with the same data sets. The network consisted of (two weight layers with) 16 inputs, 40 hidden units and 6 outputs. In the best case, a test set performance of 58.8% correct classification was obtained. The training set performance was 88%. Thus, a higher training set performance but a lower test set performance than for the best EHW circuit. This shows that the EHW architecture holds good generalisation properties.

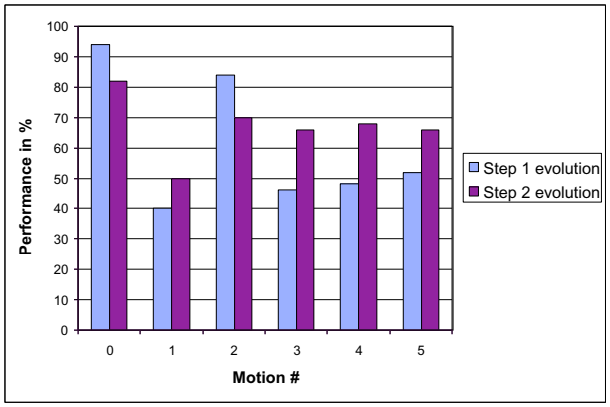

Fig. 4. The test set performance for each motion for the best circuit evolved.

In Fig. 4, the performance of each motion of the best circuit is compared for step 1 and step 2 evolution, respectively. For some motions the performance after step 1 evolution is higher than that achieved after step 2 evolution. However, after step 2 evolution the performance of the *worst* performing motions are improved. The latter would be most important to provide the best usability of the prosthesis.

Table I shows the selector settings of the best performing circuit. It is interesting to observe that in average about half of the selector lines are connected ("1") while the other half is not – see the righthand column. For each of the motions the number of selectors connected varies from 14 to 20.

5 Conclusions

In this paper, an EHW architecture for signal classification has been introduced. The best circuit evolved shows a much better performance than what was obtained by artificial neural networks. The results illustrate that this is a promising approach for evolving systems for signal classification applications.

References

[1] R.N. Scott and P.A. Parker, "Myoelectric prostheses: State of the art," *Journal of Medical Engineering and Technology*, vol. 12, no. 4, pp. 143–151, July-August 1988.

[2] S. Fuji, "Development of prosthetic hand using adaptable control method for human characteristics," in *Proc. of Fifth International Conference on Intelligent Autonomous Systems*, 1998, pp. 360–367.

[3] W-P. Lee, J. Hallam, and H.H. Lund, "Learning complex robot behaviours by evolutionary computing with task decomposition," in *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, A. Birk and J. Demiris, Eds., vol. 1545 of *Lecture Notes in Artificial Intelligence*, pp. 155–172. Springer-Verlag, 1997.

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

[5] J. Torresen, "A divide-and-conquer approach to evolvable hardware.," in *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, M. Sipper et al., Eds., vol. 1478 of *Lecture Notes in Computer Science*, pp. 57–65. Springer-Verlag, 1998.

[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)*, 1999, pp. 182–187.

[7] 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)*, vol. 2210 of *Lecture Notes in Computer Science*, pp. 1–13. Springer-Verlag, 2001.

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

[9] J. Torresen, "Scalable evolvable hardware applied to road image recognition.," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn et al., Ed. July 2000, pp. 245–252, IEEE Computer Society, Silicon Valley, USA.

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